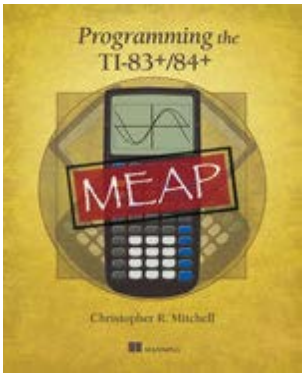
MANNING PUBLICATIONS



Programming the TI-83+/84+
By Christopher R. Mitchell

With the ability to display text and numbers back to the user, you now need a way to get feedback back from the user in the form of numbers, strings, and other types of data. This article based on chapter 2 from Programming the TI-83+/84+, teaches you two TI-BASIC commands that let you do just that.

Input from Users: The Prompt and Input Commands

A program that can only display things to users is not very useful at all unless it also has a way of getting feedback, or input, from users. In computer programs, you provide input by moving the mouse around, by clicking, by typing things on the keyboard. In much the same way, calculator program users can type in numbers, strings, and other variables for programs to use. This article will teach you about the `Prompt` command and how it can be used to get numbers from the user. It will also introduce the more powerful `Input` command, which provides more control over what you, as a programmer, display to a user who's running your program when you ask them for input. I'll show you how you can also get non-numerical input such as strings, and how you can build a simple conversational program with these commands.

The simplest task is using the `Prompt` command to get a number from the user and save it into a variable, so I'll present that first.

Prompting for numbers

Many programs, both games and utilities written for educational purposes, need a way to get numbers from the user. Of the two commands that the calculator has for the purpose, we'll first examine `Prompt`, the command you've already seen. You can find Prompt under `[PRGM][▶][2]`. After the command, you must put at least one variable that `Prompt` will ask the user to type in. This can be a numeric ("real") variable such as B or R or N, which you'll see in this article. It can also be other types such as strings, lists, or matrices. Let's jump right into a very simple two-line program that prompts the user to type the variable `X`, then squares it, and displays the resulting value. The source code for this program looks as follows:

```
PROGRAM:PRMPTSQR
:Prompt X
:Disp X²
```

You can perform math within a `Disp` statement, squaring `X` before displaying it rather than needing to square `X`, store the result back in `X`, and then display that. You can see the output of this program for a sample input of `X=5` in figure 1.

Figure 1 Testing program PRMPTSQR, which prompts the user for a value of X and then displays that value squared

There's really not much to this command, as you can see. It simply displays the `X=?` prompt at the left margin of the calculator's screen, then flashes the cursor and waits for the user to type in a number. When the user does so and presses `[ENTER]`, the program continues at the next line. `Prompt` is a blocking input command, which means that your program cannot continue to the next line of the program until the user types in their number and presses `[ENTER]`.

`Prompt` has only one other feature that you'll find handy because it is a very simple command to use. For example, I used `Prompt` to get three numbers at once in `prgmQUAD`—the three coefficients needed for a quadratic formula. The naïve way to ask the user for variables A, B, and C, might look as follows:

```
:Prompt A
:Prompt B
:Prompt C
```

Luckily, since every extra command is wasted space that makes a program bigger, we have a way to compress these three commands down to a single command:

```
Prompt A,B,C
```

Notice that there are no spaces between the commas or the variables in that command. To see this technique being used for something slightly more useful, glance at `prgmPRMPTPWR`. This program will ask the user for two numbers, P and Q, and then raise P to the Qth power.

```
PROGRAM:PRMPTPWR
:Prompt P,Q
:Disp P^Q
```

The `PRMPTPWR` program is demonstrated in figure 2. Here, 10 is raised to the 5th power, returning the value `P^Q = 10^5 = 100000`. Notice that the `P=?` and `Q=?` prompts appear on two lines, just as if two separate `Prompt` commands had been used to build the program.

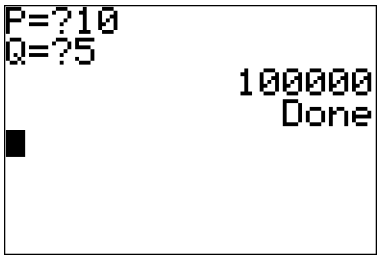


Figure 2 A program that prompts for two different variables, P and Q, and raises P to the Qth power

For `PRMPTPWR`, we start to see a shortcoming of the `Prompt` command in action. Although the name of the program hints that it will deal with powers or exponents in some way, it's not really clear from the program itself what P and Q represent. The best programs of any sort, including calculator programs, will be intuitive and understandable to their users even if the users haven't bothered to read the manuals for the programs. Therefore, it would be better to add a way for the program to explain itself to the user. You could add the following line to the beginning of the program, for instance:

```
:Disp "RAISES P TO QTH","POWER
```

However, instead you might want to give the user more explicit instructions to replace the text `P=?` and `Q=?`. The `Input` command will give you exactly that power.

Fancier Input for numbers and strings

The `Prompt` command lets you ask the user to type in a value and then stores it in a variable, as you have seen, but it has one notable shortcoming. As a programmer, you cannot control what text appears when a `Prompt` command occurs. If you're prompting for variable A, the prompt will always be `A=?`. This is not always very helpful. If, for example, you want the user to type in his or her age, the `A=?` text is not going to provide many clues that that is what the user is expected to type. If you explore the commands in the three tabs of the `PROGRAM ([PRGM])` menu shown in figure 3, you may notice the `Input` command right before the `Prompt` command.

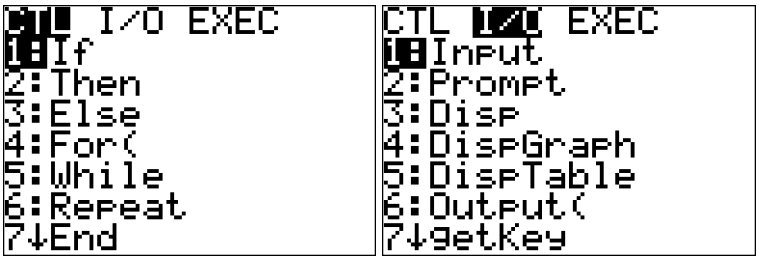


Figure 3 CTL and I/O, the first two tabs of the PRGM commands menu, accessed by pressing [PRGM] from inside the program editor. The CTL tab contains control flow commands. I don't show you the EXEC tab because its contents are different on each calculator: it lists the program your calculator holds, like the [EXEC] tab of the original PRGM menu.

For the sake of curiosity, try replacing the `Prompt` in my previous program `PRMPTSQR` (shown in figure 1 and again in figure 5) with the `Input` command, and see what happens.

```
PROGRAM:INPTSQR
:Input X
:Disp X²
```

You can see the output of the `INPTSQR` program in figure 5 in comparison to the similar `PRMPTSQR` program. As you may notice, the output of `INPTSQR` that it displays before asking the user to type a number is not quite as descriptive as the output of the `Prompt` command, which at least lets users know the name of the variable into which the value they're typing will be stored. This may seem like a disadvantage in many cases since your program is giving the users even less information about what it expects them to type in. You could use `Disp` to display a line of text explaining what the user needs to enter before the Input, but then you could still use the `Prompt` command, so what's the point?

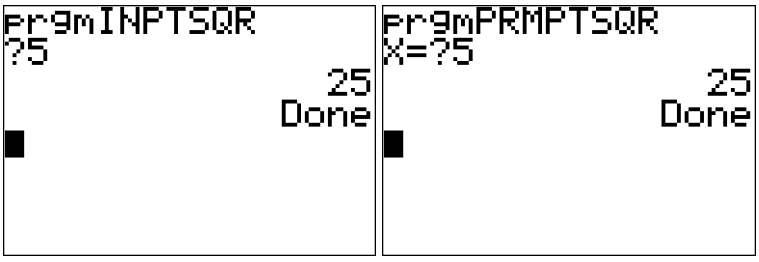


Figure 4 Using Input (left) versus Prompt (right) to ask the user for a number to be squared. The Prompt command always displays the name of the variable being stored, while Input does not.

The point of the `Input` command is that you can make it display anything you want on the same line as where the user types in input. Just as `Output` lets you display two different things on the same line, `Input` lets you do

output and input on the same line. Used with a string to display before the user types their input, `Input` takes the following arguments:

```
:Input "STRING",Variable
```

The calculator will display the string `"STRING"`, then immediately afterwards wait for the user to type in a value that will be store into the variable `Variable`. `Variable` can be a real (number) variable like `A`, `M`, `X`, or `θ`, a string like `Str3`, or even a list like L3 or a matrix such as `[B]`.

Besides the `INPTSQR` program you just saw, which demonstrates how you can simply replace a `Prompt` command with an `Input` command, you will see two more examples of `Input` in this article. First, I'll show you how to add a descriptive line to be displayed before the user can type in their input, which can be used to explain to the user what they should type. I'll end the article with a more elaborate program that calculates the slope of a line, using the `Input` command to get a pair of `(X,Y)` coordinates for two points on the line from the user.

To demonstrate what sort of things you can make `Input` do, I will first show you a version of the `INPTSQR` program that asks `"SQUARE OF?"` before the user can type in a number, as shown in the source code for `INPTSQR2` below:

```
PROGRAM:INPTSQR2
:Input "SQUARE OF?",X
:Disp X²
```

When executed on your calculator, the program's output should resemble figure 5. You could put an extra space after the question mark on the first line of the program to separate the `"SQUARE OF?"` query and the number the user enters to make the program a little clearer, but you can see that we've already made a substantial improvement to the program as far as explaining to users what the program expects from them.

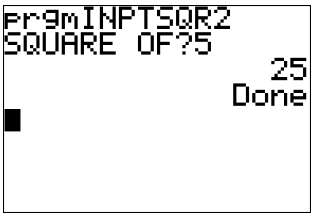


Figure 5 Using the Input command to display a line of text before the user can type something in. As you can see, Input allows you to customize the text displayed before the space where the user can type in their input, be it a number, string, or other datatype. Here, that string is "SQUARE OF?", and the user has typed the number 5.

These smaller examples are a perfect way to introduce the `Input` command or indeed any command since they show how the particular command works with little other code around it to confuse matters. Here, I'll present a program that uses the `Input`, `Disp`, and `ClrHome` commands to calculate and display the slope of a line.

Using Input: calculating slope

The final math program of this article asks the user for the coordinates of a first point and the coordinates of a second point and displays the slope of the line that joins those two points. You'll see that I've used a few optimization, including omitting a closing parenthesis that is immediately followed by the end of the line as well as two quotation marks that also immediately precede the end of two line. If you're not familiar with a slope of a line, it describes how steep it is; the larger the slope, the steeper the line. It is calculated from the quotient of how far the line rises over a certain horizontal distance divided by that horizontal distance or, in common math-class parlance, *rise over run*. Figure 6 demonstrates the concept of a line's slope, calculated from any two points on the line.

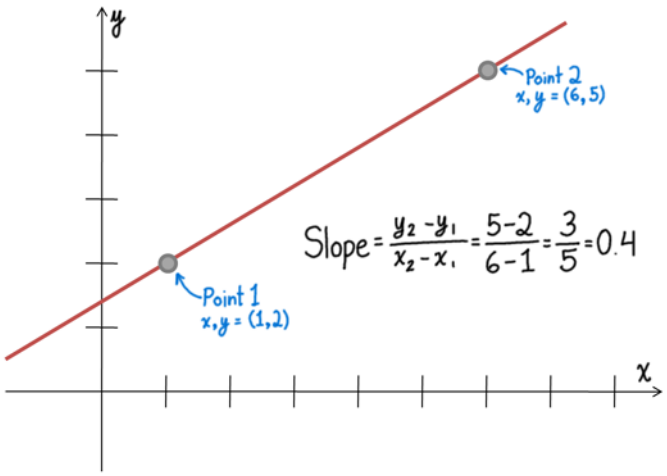


Figure 6 Calculating the slope of a line. Divide the vertical distance between the two points (the *rise*) by the horizontal distance (the *run*) to get a value for the slope. A horizontal line has a slope of zero, a diagonal line has a slope of 1, and a vertical line has a slope of infinity.

In the following program, the `(X,Y)` coordinates of the first point are stored in variables `(A,B)`, and the `(X,Y)` coordinates of the second point are in `(C,D)`. The *rise* is the change in `Y` between the two points, or `(D-B)`, and the *run* is the change in `X` between the two points, or `(C-A)`. The slope is then `(D-B)/(C-A)`, which you can see calculated on the last line of `prgmSLOPE` with the same formula as in figure 6.

```
PROGRAM:SLOPE
:ClrHome
:Disp "FIRST POINT
:Input "X1: ",A
:Input "Y1: ",B
:Disp "SECOND POINT
:Input "X2: ",C
:Input "Y2: ",D
:Disp "SLOPE:",(D-B)/(C-A
```

You can run this by finding `SLOPE` in the `[PRGM]` menu, pasting `prgmSLOPE` to the homescreen by pressing `[ENTER]`, and running it by pressing `[ENTER]` a second time. For example, the horizontal line going through `(1,1)` and `(5,1)` will have a slope of 0, since its `Y` changes by 0 as it travels 4 horizontally; the results of the program for those two values are shown in figure 7. In this figure, a calculator-generated graph of the line in question is shown next to the output of the `SLOPE` program for your edification.

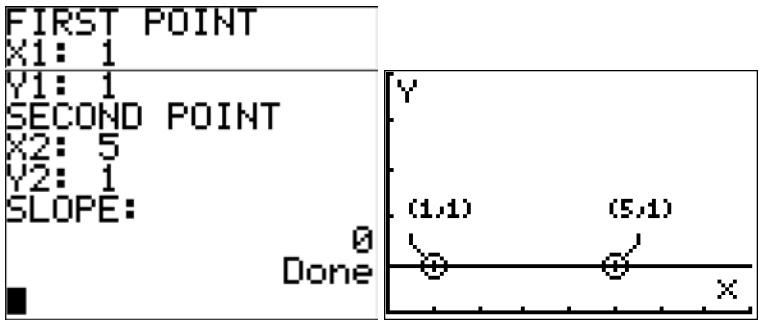


Figure 7 Testing prgmSLOPE on the line passing through points (1,1) and (5,1), shown on the left. The program correctly calculates a slope of zero (a horizontal line), proven by the graph drawn at right.

The program can easily calculate less trivial (obvious) slopes of tilted lines. The diagonal line passing through `(-3,-3)` and `(4,4)` rises 7 while it runs 7, so its slope should be `7/7=1`. You can see this accurate result from the `SLOPE` program in figure 8.

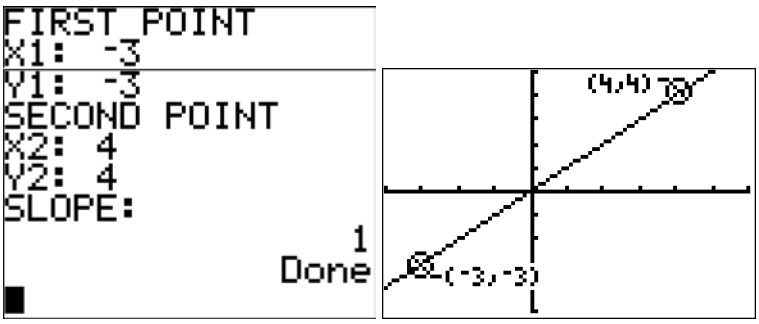


Figure 8 Calculating the slope of the line between (-3,-3) and (4,4) using the SLOPE program, at left. A graph of the line is at right.

Once again, a graph is shown at the right side of figure 9 that demonstrates the results of the `SLOPE` program. You can see how this sort of combination of input, calculation, and output could be expanded into all sorts of mathematical and scientific solvers.

Now that you've seen how `Input`, `Prompt`, `Output`, and `Disp` can be used for a few math programs, how about a simple "conversation" program that lets you talk to your calculator?

Exercise: making conversation

Since you now know how to get input from the user and display that input (or some modified form of it, such as the input with additional math performed on it) on the screen, you can start using your knowledge to make more complex programs. In this exercise, you will make a small program that pretends to chat with the user. It will ask users to type in their name, their age, and then will greet them by name and repeat their age back to them. A transcript from a sample conversation with the program should look something like this:

```
HELLO, WHAT IS YOUR NAME?<User types name and presses [ENTER]>
YOUR AGE?<User types age and presses [ENTER]>
HELLO, <Name>
YOU ARE <Age> YEARS OLD
```

Your assignment

Your task is to write a program called `CONVO` that can hold a conversation like this. You will definitely want to use `Input` in this program, and you'll also need `Output` and/or `Disp`. For the age, you can use a simple numeric variable such as `A`, `X`, `D`, and the like.

What's a string variable?

There are ten string variables, `Str1` through `Str9` and `Str0`. Each one can hold any sequence of characters enclosed in quotes, like "`HELLO`" or "`3.1415`" or "`THIS IS A VERY, VERY LONG SENTENCE THAT WASTES SPACE.`" Just as numbers can be stored into numeric variables and those variables can be used as if they were numbers, strings can be stored into `Str` variables, and the `Str` variables then used as if they were the strings themselves. For example, this code:

```
:Disp "HELLO
```

is equivalent to storing "`HELLO`" into a string variable and then using that instead:

```
:"HELLO"→Str6
:Disp Str6
```

Although we haven't discussed them much before, we will use a `String` variable in the `CONVO` program, specifically `Str1` (short for `String 1`). In your program, `Str1` will be used with `Input`, so that once the user types a value to be stored into the string, it can in turn be used for display. You can type `Str1` with `[VARS][7]1: Str1`.

To give you a bit of a visual idea of how this program might look when run, take a look at figure 9. The user has typed in `Kerm` for his name and `24` as his age. Good luck!

Figure 9 Running the CONVO program. Here, the user enters KERM as the string for his name and number 24 as his age.

Solution

The source code for one possible solution to `CONVO` program is shown below. It consists of six lines of code, all of which you should be able to easily type into your calculator by now. As noted, the one thing you might not be easily able to find is the `Str1` token, which is under `[VARS][7]1: Str1`.

```
PROGRAM:CONVO
:ClrHome
:Disp "HELLO, WHAT IS"
:Input "YOUR NAME?",Str1
:Input "AGE?",A
:ClrHome
:Disp "HELLO, ",Str1,"YOU ARE",A,"YEARS OLD"
```

Once you type this program in and run it, you should see something resembling figure 9. The program will first ask you to type in your name; you should be able to type anything at this prompt. When you press `[ENTER]`, it will ask you for your age, which must be a number. If you type anything other than a number, you will get an error message from the calculator's operating system. In that case, you can choose the quit the program or, if you select `2: Goto` at the error screen, you will have an opportunity to retype your age.

After you type your age and press `[ENTER]`, the program will greet you by name and state your age, as seen at the right side of figure 9. As you can see, the output is correct, and the `Input`-based prompts are very descriptive, showing the strings `"YOUR NAME?"` and `"AGE?"` right before the user has an opportunity to type each of those items in. If this program has one shortcoming, it is that the output shown on the right side of figure 10 is less than neat.

You could go further and try to tidy up the lines of text that the program outputs at the end. The most obvious change would be to put the user's age right next to the string `"YOU ARE"`, rather than all the way over at the right edge of the screen one line down. In fact, if you did that, you might even be able to fit the word `"YEARS"` on the same line. This fix is particularly easy because we know that the user's age probably has two digits in it and definitely has one, two, or three digits. Therefore, if we put four spaces between `"ARE"` and `"YEARS"`, we know we'll have room for a space, then one or two digits, and then the word `"YEARS"`. Let's try that out and see what happens.

The program named `CONVO2` below makes this change. Notice that the `Output` command is used for all of the age-related display now instead of `Disp`, although `Disp` is still used for the `"HELLO, <Name>"`.

```
PROGRAM:CONVO2
:ClrHome
:Disp "HELLO, WHAT IS"
:Input "YOUR NAME?",Str1
:Input "AGE?",A
:ClrHome
:Disp "HELLO, ",Str1
```

```
:Output(3,1,"YOU ARE
:Output(3,9,A
:Output(3,12,"YEARS
:Output(4,1,"OLD
```

You can see the output of this program in figure 10 for two possible age inputs, one with two digits, and one with one digit. As you can see in both cases, we have left enough space for the age to fit neatly between "ARE" and "YEARS". Unfortunately, if the user enters a one-digit age, there's still an ugly extra space between the age and "YEARS". You can use conditional statements to execute different commands based on values that the user types in, so that you could make the word "YEARS" be one space to the left if the user enters a one-digit age.

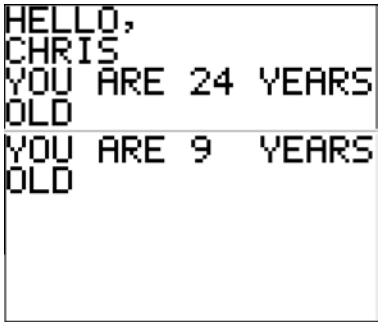


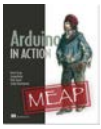
Figure 10 Using the Output command to turn the messy CONVO program into the neater CONVO2 program. It tidily displays both one- and two-digit ages in a sentence.

You now have enough knowledge to put together simple programs with input and output. You can make simple math solvers and perhaps even a few small fun programs.

Summary

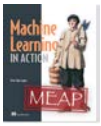
You have now seen how to create basic programs that can use `Disp` and `Output` to write text and numbers on your calculator's screen, and `Input` and `Prompt` to get the user to enter values to be stored in variables. With these two sets of commands, you can write a great number of simple programs, especially programs that ask a user for a set of values, and then solve equations based on the numbers the user inputs.

Here are some other Manning titles you might be interested in:



Arduino in Action
Martin Evans, Joshua Noble, Mark Sproul, and Jordan Hochenbaum

The Well-Grounded Java Developer
Benjamin J. Evans and Martijn Verburg



Machine Learning in Action
Peter Harrington

Last updated: March 13, 2012