

TI-Basic 89 Programming □

This page is designed to give someone with no programming experience an insight into the TI-89's version of TI-Basic.

For other versions of TI-BASIC, See TI-Basic Programming.

Contents

Introduction

Purpose

The purpose of this section of the TI-Basic Wikibook is to explore the capabilities of the TI-89 series (TI-89/Titanium, TI-92/+, Voyage 200) with the built-in interpreter. It is recommended to read through the guide from the beginning in order to build a strong understanding of the language and its applications. This section is devoted entirely to the Motorola 68k version of TI-Basic, which is more powerful than the Z80 version. For information on TI-Basic for the Z80 series, return to TI-Basic Programming and select the TI-84 version.

Overview

TI-Basic is a simple interpreted language used on Texas Instruments graphing calculators, which can be used to rapidly automate equations or algorithms. However, as an interpreted language, it is generally not fast enough for a game, so for anything requiring fast processing, assembly language is required. This is, however, outside the scope of this Wikibook. Even though the language is slow, for most purposes it will serve admirably.

Differences between Versions of TI-Basic

- The version of TI-Basic for the 83-type calculators (73 Explorer, 83+/SE, 84+/SE) is less powerful, and also less integrated into the calculator itself. However, it is still powerful enough to automate most functions.
- The version for the 89-series (89/Titanium, 92/+, Voyage 200), is more powerful and more heavily integrated into the calculator. For this section of the TI-Basic Wikibook, this is the version that will be focused upon.

Output

How to display words, numbers and more to the user

Disp

`Disp`, **I/O(F3):2**, is a command which will display a number, string, equation or other type of variable/literal given as arguments. `Disp` can accept an unlimited number of arguments separated by commas (although it is recommended to keep the number of arguments under 7 because after seven, for each consecutive argument the screen pushes everything else up passed the top of the screen, even the seventh pushes the first line up a slight amount), and will display each argument following the previous line.

Syntax: Disp

```
:Disp [arg][,arg2][,arg3][,arg4][,...argN]
```

- Where *arg*, *arg2*, *arg3*, *arg4* and all other arguments through *argN* are optional parameters. The number of arguments is only limited by the calculator memory. The arguments can be either a literal or variable of any type.
- For each argument, `Disp` displays the argument on the next line, starting from where the cursor initially was located. If the display reaches the seventh line, the display will 'scroll' the rest of the screen up so that the argument can be displayed.
- `Disp` displays all arguments left-aligned
 - if what is being displayed contains more than 26 characters string will "run off the screen", resulting in a display of only a part of the 27th character and none of the ones beyond that being shown.
- If no argument is specified, the command does nothing.

Disp 5+7

Ex: Display Lines

Assuming you have stored the value 5 into x (via `5→x`) and a blank I/O screen and you executed this program,

```
:Disp "HELLO WORLD",52,x+1
```

you would have the following on your I/O screen:

```
HELLO WORLD
52
6
```

Ex: Empty Lines

```
:Disp "", "", "HELLO WORLD"
```

```
HELLO WORLD
```

Ex: Truncated

```
:Disp "This line too long for one line"
```

```
This line too long for one
```

Pause

Pause, **Control(F2):Transfers(8):1**, displays an argument, then pauses execution afterwards.

Syntax: Pause

```
:Pause [arg]
```

- Where *arg* is either a literal or any variable type which `Pause` will display then pause execution of the rest of the program until the enter key is pressed
 - `Pause` will display *arg* on the next line, starting from where the the cursor initially was located. If the display reaches the seventh line, the display will 'scroll' the rest of the screen up so that the argument can be displayed
 - `Pause` will always display *arg* left-aligned
 - If an argument is too long to be displayed, the argument will be truncated and an arrow will be displayed indicating you may scroll left/right to read the entirety of the line.
- If no argument is specified, the command merely pauses the execution until enter is pressed without displaying anything

Ex: Hello

```
:Pause "Hello World!"
```

*It should be noted that the program execution would pause until enter is pressed.

You may get the ! symbol by pressing **2nd+Math(5):Probability(7):1, or **Diamond+Divide(+)** on the TI-89 or **2nd+W** on the TI-92 Plus and Voyage 200.

Ex: Pause

```
:Pause
```

*It should be noted that the program execution would pause until enter is pressed.

Output

Output, **I/O(F3):6**, allows the display of an argument in a location other than the next line. The item to be displayed is outputted to the specified coordinates supplied to the function. It is useful for formatted display.

Syntax: Output

```
:Output row,col,arg
```

- Where *row* is a number (can be positive or negative) which determines the row location (vertically, in pixels) *arg* is to be displayed.
- Where *col* is a number (can be positive or negative) which determines the column location (horizontally, in pixels) *arg* is to be displayed
- Where *arg* is the argument to be displayed. This can be a number, string or list.
 - arg* is displayed from left to right
 - if *arg* doesn't fit on one line, the parts that don't fit are cut out.
 - Each new `Output` command used overwrites the previous ones.

Ex: Basic

```
:Output 24,0,"HELLO WORLD"
```

```
HELLO WORLD
```

*HELLO WORLD is moved down two rows (24 pixels is the equivalent of two rows).

Ex: Too Long

```
:Output 0,100,"This is too long"
```

```
This is to
```

Return

Return, **Control(F2):Transfers(8):2**, displays an argument on the home screen, or just returns to the home screen if no argument is supplied.

Syntax: Return

```
:Return [arg]
```

- Where *arg* is any literal or variable.
- This command will return the argument to the home screen only if used in a Function.
 - You can check to see whether you have a program or a function by the line under the title. If it says `Prgm`, then it is a program. If it says `Func`, then it is a function.
- arg* will always be right-aligned
- When used without *arg*, whether in a Program or a Function, it will terminate the running of the code.

Ex: Return

Let's say you stored the value 5 in *x* (using `5→x`) before executing the function. For the sake that this can only be used in a function, the appropriate tags and a name are supplied for this example.

```
temp()
:Func
:Return 2x+1
:EndFunc
```

```
•temp()          11
```

Input

How to receive words, numbers and more from the user

Input, **I/O(F3):3**, is a command which will display a string and wait for the user to input a value. The value will then be stored into the specified variable once ENTER is pressed. The behavior of this command changes depending on how many parameters are passed into it, explained below.

Syntax: Input

```
:Input [string],[var]
```

- Where *string* is the optional string to display, and *var* is the optional variable to store the value to
- If *string* is provided, the I/O screen will display that string on a new line, then wait for user input.
- If *string* is not provided, the I/O screen will display a ? on a new line, then wait for user input.
- If neither *string* nor *var* is provided, it will display the graph screen and wait for the user to select a coordinate (x value of coordinate is stored into *xc*, and the y value into *yc*)
- Providing only *string* but not *var* results in an error ("Argument must be a variable name", *errornum* 140)

Ex: With String

```
:Input "X Value",x
```

```
X Value
5
```

*The value of 5 would be stored into x

Ex. Without String

```
:Input x
```

```
?  
5
```

*The value of 5 would be stored into x

Ex: Without Arguments

```
:Input
```

InputStr

`InputStr`, **I/O(F3):4**, is much like `input`, but the values inputted into this command will always be a string (so no quotation marks are needed).

Syntax: InputStr

```
:InputStr [string],var
```

- Where *string* is an optional string to display instead of ?
- Where *var* is the variable the string inputted will be stored into

Ex: InputStr

```
:InputStr "What is your name",name
```

```
What is your name  
George
```

*Pressing ENTER would store "George" into the variable *name*.

Prompt

`Prompt`, **I/O(F3):5**, prompts the user to input a number of variables. While this can be used to input more than one variable with one command, it does not give you the flexibility to change what is displayed on screen.

Syntax: Prompt

```
:Prompt var[,var2][,var3][,...varN]
```

- Where *var2* through *varN* are all optional arguments. Arguments must be valid variable names.
 - Amount of arguments limited only by available memory
- For each argument, Prompt will wait until a value is entered and ENTER pressed before asking for the next one.

Example: Prompt

```
:Prompt x,y
```

```
x?  
5  
y?  
7
```

*This would store 5 into x and 7 into y

Request

`Request`, **I/O(F3):1:2**, puts a pop-up box on the current screen and allows the user to input a string into it, and then stores that string into a variable. `Request` can be used as a stand-alone command or as part of a `Dialog` block (more on those later).

Syntax: Request

```
:Request string,var
```

- Where *string* is the string to be displayed before the input box
- Where *var* is the variable the string gets stored into
- `Request` automatically turns a-lock on, so the alpha button must be pressed once before numbers and other symbols may be entered

Ex: Request

```
:Request "name",x
```

PopUp

`PopUp`, **I/O(F3):1:3**, displays a pop up window with a number of choices for the user to choose between (passed in as a list). Then, the *number* of the choice the user selects will be stored into the variable given.

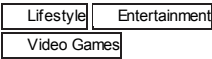
Syntax: PopUp

```
:PopUp itemlist,var
```

- Where *itemlist* is a list of strings that will appear in the popup.
- Where *var* is the variable that the number of the choice will be stored to.

Example: Correct Equation

```
:Dialog
:Title "Choose Correctly"
:DropDown "Choose the equation",{ "1+2=2","1/2=2","1-2=2","1*2=2"},x
:EndDlog
```



DropDown

`DropDown`, **I/O(F3):Dialog(1):4**, displays a pop up window with (a) pull down menu(s) with a number of choices for the user to choose between (passed in as a list). Then, the *number* of the choice the user selects will be stored into the variable given. The `DropDown` command must be used in a `Dialog...EndDlog`, **I/O(F3):Dialog(1):5**, block. Placing multiple `DropDown`s will not return multiple pop up windows but will put all menus in the same window. `DropDown` can be paired with other dialog options such as `Request` or `Title`

Syntax: DropDown

```
:DropDown "Title",{"itemlist"},var
```

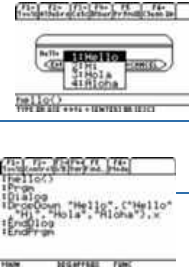
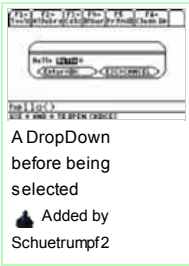
- Where *Title* is the title of the pull down menu
- Where *itemlist* is a list of strings that will appear in the popup.
- Where *var* is the variable that the number of the choice will be stored to.

Example: Correct Equation

```
:Dialog
:Title "Choose Correctly"
:DropDown "Choose the equation",{ "1+2=2","1/2=2","1-2=2","1*2=2"},x
:EndDlog
```


The *Title* will appear in the top of the pop up window

Passing in Arguments



Passing in arguments is vital to functions, and can be helpful in programs as well. In order to get an argument to be passed in, it must be declared in the opening parenthesis of the program editor. Then, when calling the program (by typing `prgmname()` in the home screen), you put the declared arguments into the parenthesis, like so: `prgmname(arg)`. Note: for all of the following examples, the name of the function is "temp" (and it is a function).

Syntax

What using the code results in
 Added by Schuetrumpf2

```
temp([var1][,var2][,var3][,...varN])
```

- Where *var1* through *varN* are all valid variable names
 - The amount of variables passed into the function is limited only by the amount of available memory
- The variables passed in are local variables, as in they disappear the moment program execution stops (unlike from the previous ways of input, where the variables stick around until they are deleted).

Example: Temp

```
temp(x,y,z)
:Func
:Return x^2+(y*z)/100
:EndFunc
```

Let's say you call this program like so: `temp(12,4,52)`. The following will appear on your home screen:

```
*temp(12,4,52)      146.08
```

Custom Tool Bars

It is possible to create your own personal toolbar at the top of your TI-89 where you can place whatever programs and/or functions that you want for easy access. To create a toolbar, you must create a `Custom...EndCustm`, **Control(F2):7**, block. Each tab is a `Title`, **I/O(F3):Dialog(1):7**, and each item within the tab is an `Item`, **I/O(F3):Dialog(1):8**.

Syntax


```
Custm()
:Custom
:Title "String"
:Item "var1"
:Item "varN"
:EndCustm
:CustmOn
```

- Where *Title* is each tab
- Where *Item* is each object in the tab
- Where *CustmOn* changes from normal toolbar to the new one. *CustmOn* must be typed.

Example

```
Custm()

Prgm
Custom
Title "Solve("
Title "Math"
Item "Factor("
Item "Expand("
EndCustm
CustmOn
EndPrgm
```

Editing the example custom program
 Added by Schuetrumpf2

Conditional Functions

Learn about controlling the flow of a program using conditionals such as If

If

If, **Control(F2):1**, is a command which will determine whether or not the next line of code is executed.

Syntax: If

Using custom program
Added by Schuetrumpf2

```
:If condition
```

- Where *condition* is an expression that resolves to be either *true* or *false*
 - If *condition* is true, then the code in the next line is executed. Otherwise that code is skipped.

Ex: If

```
:5→x  
:If x=5  
:Disp "True"
```

```
True
```

This page uses content from **Wikibooks** . The original book was at TI-Basic 89 Programming. The authors of this book were Skizzerz, Jimmyatic, T3thys::ben, and D1rohrer2003. As with TI-BASIC Wiki, the text of Wikibooks is available under the Creative Commons Attribution-Share Alike License.

Categories: TI-Basic 89 Programing | TI-Basic Wikibook | Reference | [Add category](#)