Basic Coding for math
with the
TI-84 Plus CE Graphing Calculator

Lisa Conzemius

T^3 Regional Instructor

Detroit Lakes High School

Detroit Lakes, MN

701-212-5065 Cell

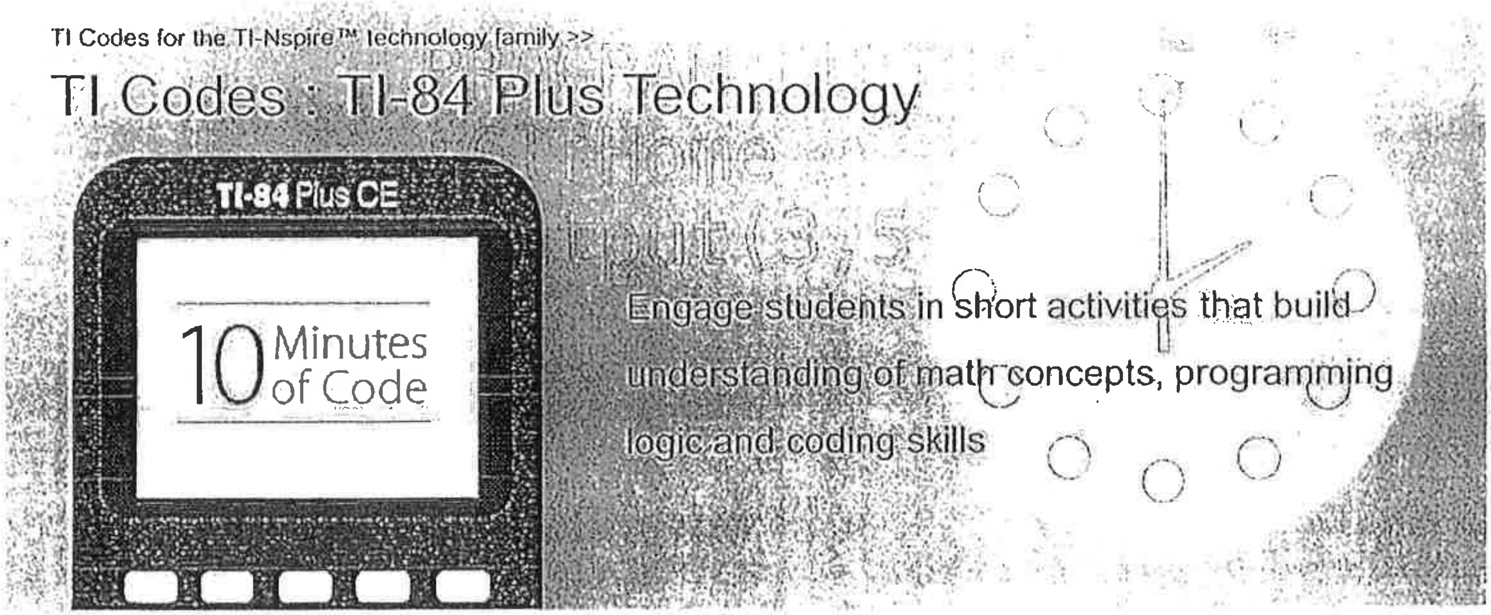
218-847-1163 School

zemilarson@gmail.com home email

lconzemius@detlakes.k12.mn.us school email

TI Codes for the TI-Nspire™ technology family >>

TI Codes : TI-84 Plus Technology



Engage students in short activities that build understanding of math concepts, programming logic and coding skills

Unit 1: Program Basics and Displaying on Screen

Unit 2: Using Variables

Unit 3: Conditional Statements

Unit 4 Loops

Unit 5: Graphics

10 Minutes of Code

Unit 1: Program Basics and Displaying on Screen

Skill Builder 1: Using program editor and syntax

This is the first of three 'Skill Builders' in Unit 1. At the end of this unit, you will use the skills you have learned in these Skill Builders to create a more complex program. This is your first lesson in learning to code with TI Basic.

TI Basic is a programming language that can be used to program on the TI calculators. While the structure and syntax (grammar) of TI Basic is simpler than other modern languages, it provides a great starting point for learning the basics of coding. Let's get started!

Objectives:

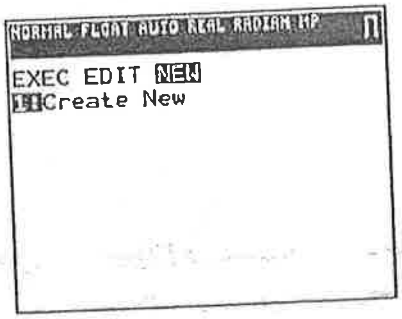
- Use the TI Basic Program Editor to create and run a simple program.
- Use the program menus to select and paste commands into a program.
- Run a program.

Teacher Tip: B.A.S.I.C. is one of the original programming languages that was designed for teaching and learning programming. It is an acronym of Beginners' All-purpose Symbolic Instruction Code. TI Basic is based upon this language.

Turn on your TI-84 Plus CE and press the PRGM **key.**

Select **NEW** using the arrow keys.

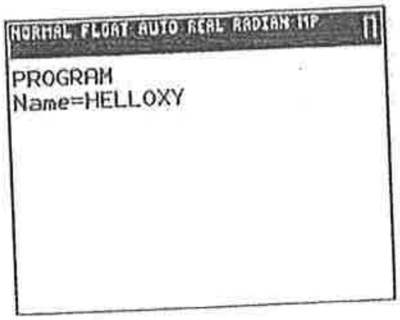
Select **Create New** by pressing ENTER.



Name your program.

Our program name will be **HELLOXY**. It can be any legal name*. Press ENTER after typing the name. You are now in the Program Editor. Each line begins with the colon character (:).

**A legal name must: be up to 8 characters long, start with a letter, include only uppercase letters and numbers, with no spaces; and be unique.*



PROGRAM
Name=HELLOXY

Teacher Tip: If you use a name that has already been used then you will edit that program rather than create a new one.

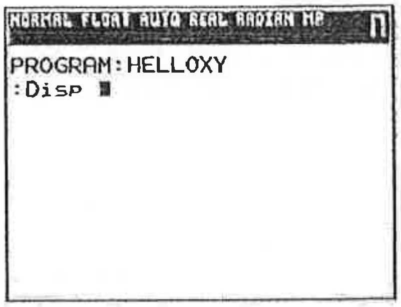
This program will display a simple message on the home screen of your calculator.

©2015 Texas Instruments Incorporated

Selecting a programming command from the Program Menu.

The PRGM key now contains new menus that contain the commands that are used in TI Basic. If you want to use one of the commands, you *must* select it from this menu rather than type it on the screen.

1. Press the PRGM key
2. Choose the **I/O** menu using the arrow keys. This menu contains all of the commands affecting Input and Output.
3. Select **Disp**. The word will be pasted into your program at the current cursor position. The **Disp** command will display something on the HOME screen.



```
PROGRAM:HELLOXY
:Disp ▮
```

Teacher Tip: You cannot type in the programming commands. The commands are also not editable. All keywords in a program are selected from the menus. The text displayed is actually just a readable symbol (token) for the programming command.

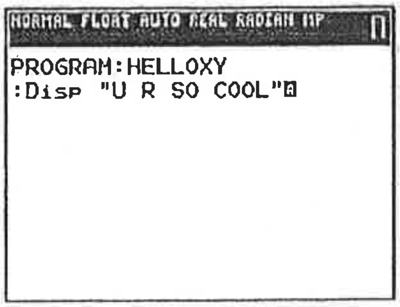
Teacher Tip: Some of the other keys on the calculator behave differently while using the Program Editor. The MATH key allows you to select one of the math functions to use in a program. The CATALOG key contains a list of ALL calculator functions in alphabetical order. The MODE key allows you to select a mode setting so that the program will change the mode to that setting. There are other keys that exhibit similar behavior.

Teacher Tip: Some keys, such as Y= or GRAPH, will take you out of the Program Editor and into its own environment. Fear not. Just press PRGM > EDIT to get back to editing the program that you select from the list.

Type a greeting in double quotation marks.

This greeting is called a *string*, which is a group of characters that are "strung together".

- Your string must start and end with quotation marks. Without the quotes, the program thinks you mean something completely different.
- Make your life easier: Press 2nd ALPHA to turn on the alpha-lock while you type in the string.



```
PROGRAM:HELLOXY
:Disp "U R SO COOL"▮
```

Teacher Tip: The CLEAR key clears an entire line of code. It cannot be undone.

Teacher Tip: To insert a blank line place the cursor at the beginning or end of a line and press [INSERT] (2^{nd} DEL) and then press ENTER.

Teacher Tip: To delete a line press CLEAR and then press DEL. There is no copy and paste.

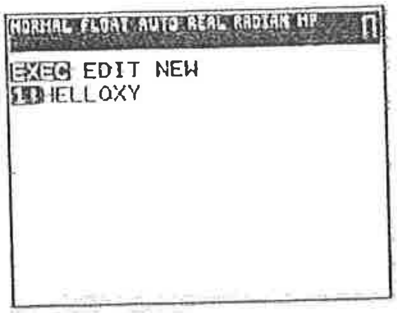
Your program is complete! Let's run it now. There is no need to 'save' with TI Basic; the program is preserved as you type it in. That's why we named the program first.

10 Minutes of Code

To run the program:

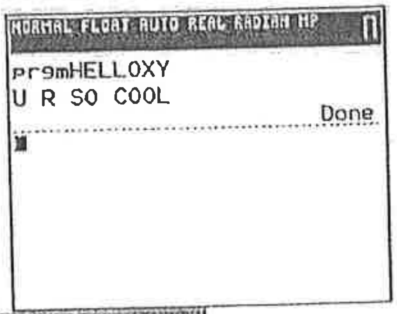
Press [quit] ([2nd][MODE]) to return to the HOME screen.

1. Press [PRGM].
2. Under the **EXEC** ('execute') menu, select your program.
3. Press [ENTER] to paste the program name on the HOME screen.
4. Press [ENTER] again to begin the run.



```
NORMAL FLOAT AUTO REAL RADIAN MP
EXEC EDIT NEW
1:HELLOXY
```

Your text message is displayed on the HOME screen.



```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmHELLOXY
U R SO COOL
                            Done
```

You can edit your program, too:

1. Press [PRGM].
2. Choose the **EDIT** menu using the arrow keys.
3. Select your program and press [ENTER].

Teacher Tip: Pressing [ENTER] after a program finishes running will re-run the program (because [ENTER] processes the last command entered on the HOME screen).

Teacher Tip: If a program generates an ERROR message then there is something wrong in the program. There are two options under the error: 1:Quit and 2:Go To. Quit takes you to the HOME screen and Go To takes you into the Program Editor to the place in the program where the error occurred. This may or may not be the actual place that causes the error.

Teacher Tip: A common error is SYNTAX. Syntax is a synonym for grammar; there's something wrong with the *structure* of the statement.

Teacher Tip: To delete a program use the Memory Management utility (2^{nd} + > Memory Mgmt/Delete…). Select PRGM… and press the delete key on the program you wish to remove. You will see an 'Are you sure?' warning message.

©2015 Texas Instruments Incorporated

Unit 1: Program Basics and Displaying on Screen — Skill Builder 2: Editing programs; clearing the screen

In this second of the three Skill Builders in Unit 1 you will practice editing a simple program and learn how to clear the HOME screen of your TI-84 Plus CE. We will use and add to the same program that you started in Skill Builder 1.

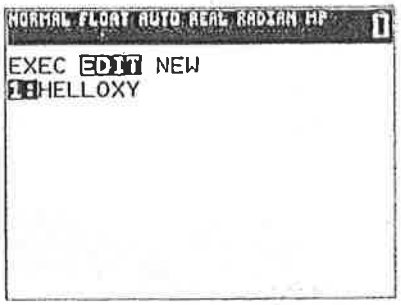
Objectives:

- Use the TI Basic Program Editor to add to and edit a simple program.
- Use the program menus to select and paste commands into a program.
- Use simple editing features to insert and delete things.
- Learn how to clear the HOME screen.

Turn on your TI-84 Plus CE and press the [PRGM] key.

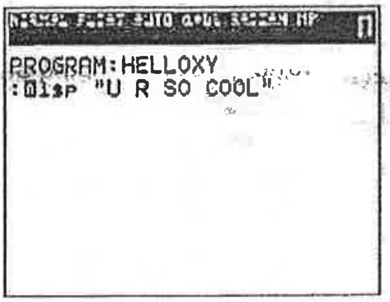
Select **EDIT** using the arrow keys.

Select the program you started earlier. We used **HELLOXY** in that lesson so our screen shows that program name.



```
EXEC EDIT NEW
1:HELLOXY
```

In the Program Editor the cursor is blinking at the beginning of the first statement in the program. Use the arrow keys to move the cursor placement

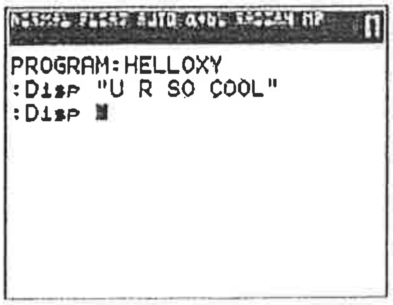


```
PROGRAM:HELLOXY
:Disp "U R SO COOL"
```

We are going to *edit* this program and add more Disp statements to it.

Move the cursor to the *end* of the first line of the program and press [ENTER]. A second colon will appear. This is the second line of the program.

1. Press the [PRGM] key.
2. Choose the **I/O** menu using the arrow keys. This menu contains all of the commands affecting Input and Output.
3. Select **Disp** again. The word will again be pasted into your program at the current cursor position as shown in the figure to the right.



```
PROGRAM:HELLOXY
:Disp "U R SO COOL"
:Disp 
```

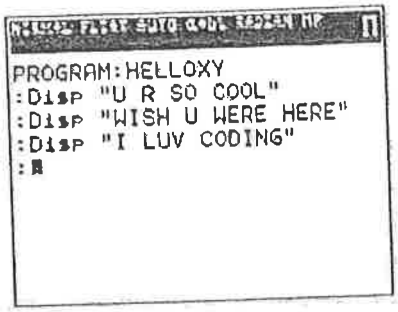
Teacher Tip: Editing a program is done using the arrow keys, [DEL] and [INS]. The cursor indicates the current editing state: a blinking block is overwrite; a blinking underscore is insert; and blinking 'A' is alpha or alpha-lock.

In these documents, calculator keys in square brackets, such as [INS], indicate a 2nd function feature of a key. In the early lessons we indicate the full 2nd function keystroke but eventually transition to using just the 2nd function notation.

5

©2015 Texas Instruments Incorporated

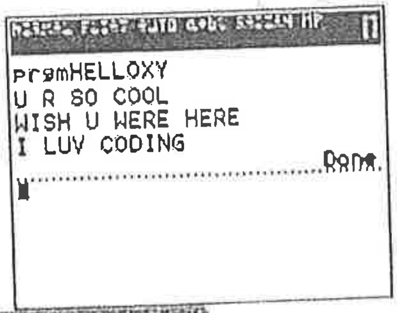
pe another message in quotation marks. Remember to press [2nd] [ALPHA] to turn on 'alpha-lock' while you type in the string. Note the change in the appearance of the cursor when alpha-lock is active.



```
PROGRAM:HELLOXY
:Disp "U R SO COOL"
:Disp "WISH U WERE HERE"
:Disp "I LUV CODING"
:
```

Press [ENTER] again at the end of the second line and add more Disp statements. You can add as many statements as you like, but it's possible that you could add so many that the resulting text won't fit on the screen all at once.

Your program is complete. Let's run it by pressing [quit] and selecting it from the EXEC menu of the [PRGM] key.

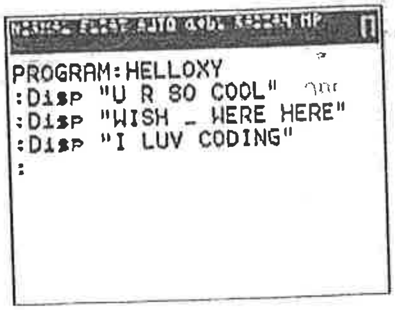


```
prgmHELLOXY
U R SO COOL
WISH U WERE HERE
I LUV CODING
                 Done
```

Teacher Tip: The number of lines on the HOME screen depends on which TI-84 model you are using. The TI-84 Plus has 8 lines. The TI-84 Plus C and CE have 10 lines. Displaying on the *bottom* line also forces a return on that line so the screen will scroll once.

diting your program

To change the "U" in the second statement to "YOU", use the arrow keys to place the cursor on the U, press [INS] ([2nd] [DEL]) so that you see a blinking underscore cursor. Type the Y and the O characters, then press an arrow key.



```
PROGRAM:HELLOXY
:Disp "U R SO COOL"
:Disp "WISH _ WERE HERE"
:Disp "I LUV CODING"
:
```

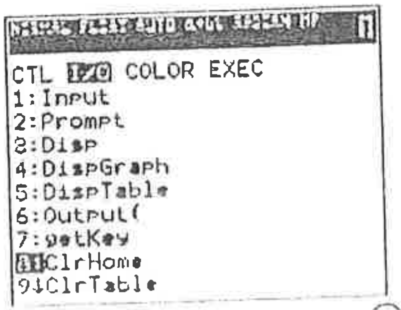
To delete a character, press the [DEL] key on the character.

To clear an entire statement, press the [CLEAR] key anywhere in the statement. This clears the line of code and leaves a blank line (a colon with nothing after it). Blank lines have no effect on the running of the program; they are ignored. If you want to delete the blank line, you can press [DEL] while the cursor is on the blank line.

When you are done (or if you just want to test what you have so far), press [quit] and run the program.

Clearing the HOME screen

The **ClrHome** statement clears the HOME screen but we want this statement to be at the *top* of the program.

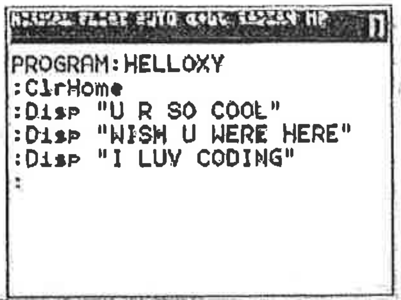


```
CTL I/O COLOR EXEC
1:Input
2:Prompt
3:Disp
4:DispGraph
5:DispTable
6:Output(
7:getKey
8:ClrHome
9↓ClrTable
```

1. While editing your program place your cursor at the top of the program (on the "D" of the first **Disp** statement).
2. Press [INS] and then press [ENTER] to make a new, blank line above the **Disp** statement.
3. Press the up arrow key to place your cursor on that blank line.

⑥

4. Press [PRGM] and use the right arrow key to see to the I/O menu and select the **ClrHome** statement.
5. Quit the editor and run the program. You will see your text displayed on a clean HOME screen.



```
PROGRAM:HELLOXY
:ClrHome
:Disp "U R SO COOL"
:Disp "WISH U WERE HERE"
:Disp "I LUV CODING"
:
```

Teacher Tip: There is no 'undo' feature in this editor. All keystrokes are stored. If a mistake is made in editing then the statement will have to be reconstructed. So be careful when editing.

Recall that the [CLEAR] key deletes the entire line that the cursor is on so be careful about using that key!

10 Minutes of Code

Unit 1: Program Basics and Displaying on Screen

Skill Builder 3: Output to the HOME screen

In this third of the three Skill Builders in Unit 1 you will practice editing a simple program and learn how to place text anywhere on the HOME screen using the **Output** statement.

Objectives:

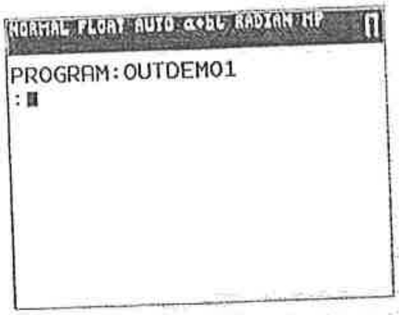
- Use the TI Basic **Output** statement to control the position of text displayed on the HOME screen.
- Use the **Pause** statement to prevent a program from ending too soon.

Teacher Tip: This lesson introduces two new statements: **Output(** and **Pause**. Emphasize that learning this programming language is a building process and that previously learned statements should not be forgotten. The **Output(** statement takes three arguments and a closing parenthesis. The **Pause** statement has an optional argument that we will not deal with now.

Turn on your TI-84 Plus CE and press the [PRGM] **key.**

Select **NEW** using the arrow keys.

Press [ENTER] to create a new program and enter a name for the program. We will use OUTDEMO1 here. Note that to type the digit "1" we have to press [ALPHA] to turn alpha mode *off*.



```
PROGRAM:OUTDEMO1
:■
```

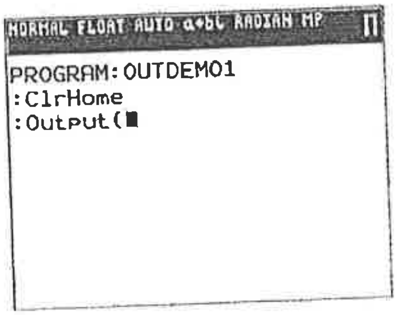
Using Output(Statements

Start your program with the **ClrHome** statement from the [PRGM] key **I/O** menu. Remember to press [ENTER] at the end of the line.

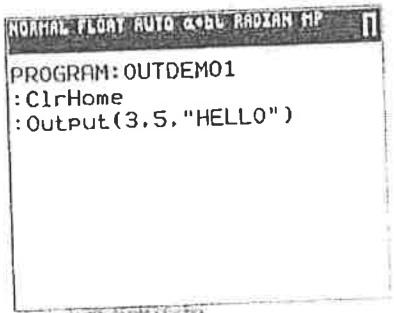
Next, select the **Output(** statement from the same menu.

The HOME screen is divided into an invisible grid of characters. The Output statement will place your text starting at one of these grid positions using the line number and column number of that position. The upper left corner is line 1, column 1.

So, after the open parenthesis in the Output statement, type **3,5,"HELLO")** Remember to close the parentheses.



```
PROGRAM:OUTDEMO1
:ClrHome
:Output(■
```



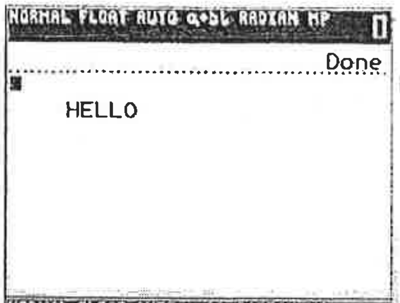
```
PROGRAM:OUTDEMO1
:ClrHome
:Output(3,5,"HELLO")
```

Teacher Tip:(3, 5, "HELLO") means that the letter "H" will be displayed at line 3, column 5 of the HOME screen. The TI-84 Plus HOME screen has 8 lines and 16 columns. The color calculators each have 10 lines and 26 columns due to the higher resolution of the screens.

Let's run the program. The result should look like the screen to the right if you used the same values.

©2015 Texas Instruments Incorporated

Notice how 'Done' appears on the top line even though "HELLO" is displayed on the third line. The **Output(** statement has no effect on the current cursor position. The **ClrHome** statement positions the cursor in the top left of the screen, so the 'Done' is then displayed on the top line.

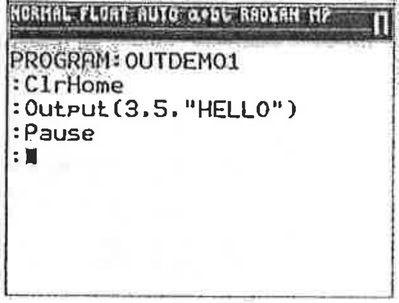


```
                         Done

   HELLO
```

Adding the Pause Statement

Edit your program and add the **Pause** statement below the **Output(** statement.

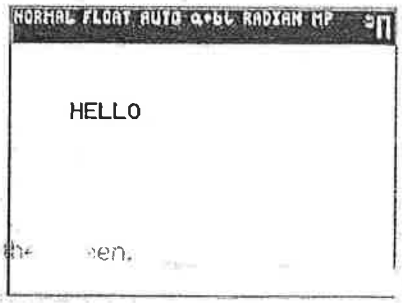
You will find **Pause** on the [PRGM] key **CTL** menu. **CTL** is short for 'Control' and the menu contains statements that *control* the behavior of the program.



```
PROGRAM:OUTDEMO1
:ClrHome
:Output(3,5,"HELLO")
:Pause
:
```

Run the program again and see what happens. See that the 'Done' is now missing?

Look closely in the top right of the screen and you'll see the 'busy' indicator. That's the **Pause** statement at work. The program is paused at this point and the user must press [ENTER] to continue. Then the 'Done' message appears at the top of the screen.



```
   HELLO
```

Teacher Tip: The color calculators have a spinning circle 'busy signal' in the upper right corner of the screen next to the battery icon. The TI-84 Plus has a blinking 'worm' (vertical segment) in the upper right corner of the screen. These are an indication to 'Press enter to continue.'

Adding More Output(Statements to Your Program

Test to see what happens when there's not enough room to display the message on the line.

Answer: The text wraps around to the beginning of the next line.

Add a statement at the bottom of the program so that you end the program with a clear screen.

Answer: Add another ClrHome statement after the Pause statement.

Application 1: Create a title screen

this Application for Unit 1 you will make use of the statements learned in this unit to build one (or more) title screen(s).

Objectives:

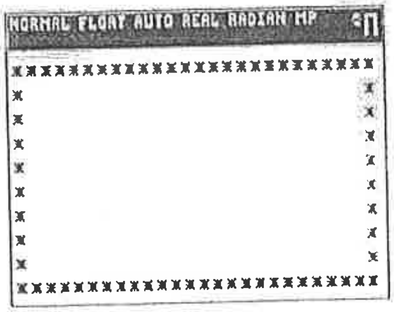
- Use the TI Basic statements learned in Unit 1 to build a title screen for a larger program.

Part 1:

Use **Disp** statements to display a border of asterisks around the screen. Use an **Output(** statement to display the bottom line because **Disp** will scroll the whole screen. Remember to clear the HOME screen first and to *pause* the program at this point.

With the *empty* border showing, when the user presses [ENTER], the text from Part 2 below will appear in the *middle* of the screen.

Note: TI-84 Plus users will have a different number of stars on the screen due to the different screen dimensions.

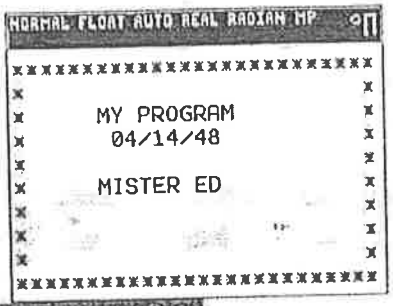


```
**************************
*                        *
*                        *
*                        *
*                        *
*                        *
*                        *
*                        *
*                        *
**************************
```

Part 2:

Use **Output(** statements to display a title, date, and author information centered in the border. The screen to the right is not very good because the text is not neatly centered.

After the text is displayed the user will need to press [ENTER] again to continue. At this point the program should clear the screen and then end.



```
**************************
*                        *
*        MY PROGRAM      *
*         04/14/48       *
*                        *
*        MISTER ED       *
*                        *
*                        *
*                        *
**************************
```

Teacher Tip: The number of asterisks and spaces in the first part of the program is important. Students need to carefully calculate the amount of text needed and the lines of code will wrap around onto the next screen line. The bottom line of the screen needs an **Output(** statement because **Disp** forces an automatic return, and that would force the screen to scroll up one line.

Sample Answer:

```
ClrHome
Disp "**************************"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Disp "*                        *"
Output(10,1,"**************************")
Pause
Output(3,10,"MY PROGRAM")
Output(4,11,"04/14/48")
Output(6,10,"MISTER ED")
Pause
```

⑩

```
ClrHome
```

©2015 Texas Instruments Incorporated

Unit 2: Using Variables and Expressions

Skill Builder 1: Prompting for a variable

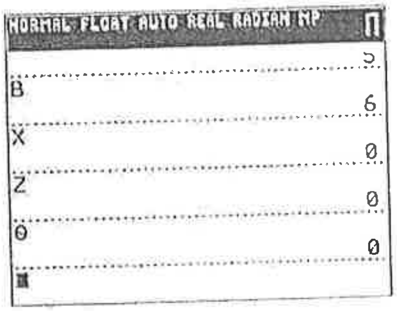
In the first lesson for Unit 2 you will learn about the **Prompt** statement to make your programs interactive, using variables to hold numeric values, evaluating and storing results of mathematical expressions, and using **Disp** and **Output** statements to show the results of stored computations.

Objectives:

- Use the TI Basic **Prompt** statement to assign a value to a variable.
- Know the difference between mathematical variables and computer variables.
- Perform calculations within **Disp** statements.
- Use **Output** statements to produce meaningful, readable results.

Real Variables

- The TI-84 Plus has 27 built-in variables that are used to store numeric values.
- The values can be *real* (decimal) numbers or *complex* numbers.
- The variable names are the letters A through Z and the letter θ ('theta').
- All variables contain a value. If a value is not assigned then the default value is 0 (zero).
- The values remain stored even when the calculator is turned off.
- If RAM is reset then all the values are set to 0.
- The HOME screen at the right shows some variables (on the left) and their *current* values (on the right). Yours may differ.

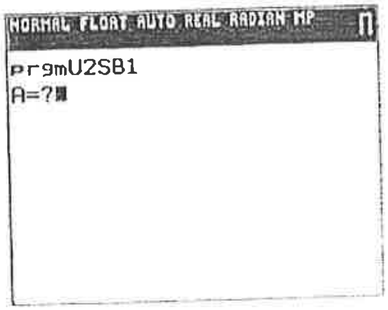


```
NORMAL FLOAT AUTO REAL RADIAN MP
                              5
B
                              6
X
                              0
Z
                              0
θ
                              0
▮
```

Teacher Tip: There are many types of variables in a TI-84 Plus. Among them are real and complex, lists, matrices, Y-vars (the graphing functions of all types), programs, apps, appvars, strings, pictures (PICS), and on the TI-84 Plus CE, background Images. This lesson deals only with the *real* variables (which can also store *complex* values). For the list of variable types look at the Mem Mgmt/Del section of the [MEM] menu.

The Prompt Statement

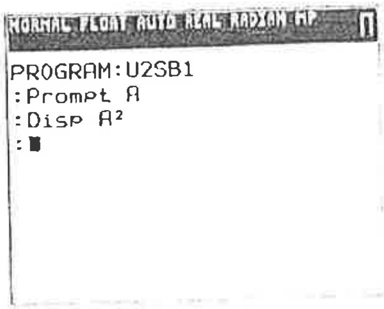
- The **Prompt** statement is followed by one or more variable names that ask the user to enter a value for a variable.
- It is called '**Prompt**' because when you run the program, it displays the name of the variable and a question mark.



```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmU2SB1
A=?▮
```

Programming with Prompt

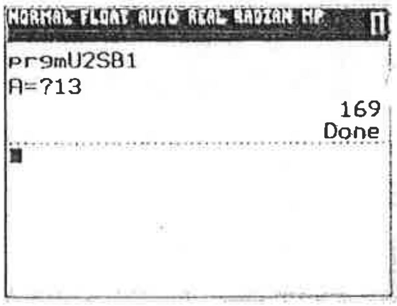
1. Start a new program.
2. For first statement of the program use the **Prompt** statement found in the [PRGM] I/O menu.
3. After the **Prompt** command type the name of the variable you want your program to use. In this program we will use the letter A.
4. Use the **Disp** statement to display the square A^2; type A then the [x^2] key.
5. Quit the editor and run the program.
6. After the "A=?" prompt, type any number



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:U2SB1
:Prompt A
:Disp A²
:▮
```

©2015 Texas Instruments Incorporated

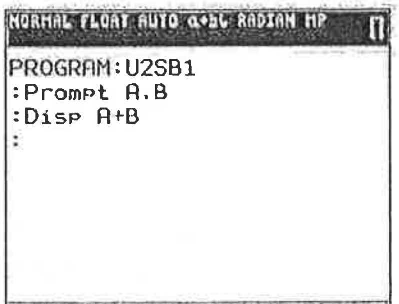
7. The program displays the square of that number and ends.



```
prgmU2SB1
A=?13
                         169
                        Done
```

Entering Multiple Values with Prompt

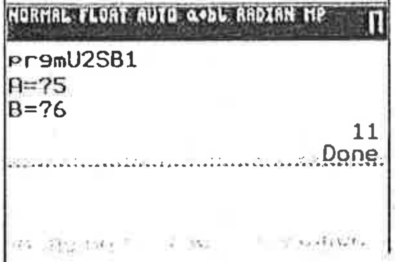
1. Edit the program from above.
2. Add ,B to the **Prompt** statement.
3. Change the **Disp** statement so that it displays the sum **A+B**
4. Run the program again.



```
PROGRAM:U2SB1
:Prompt A,B
:Disp A+B
:
```

Notice the two prompts? The **Prompt** statement asks for a value for each variable separately.

This is a very simple, efficient program requiring only two statements, but these two statements are doing a lot of work!



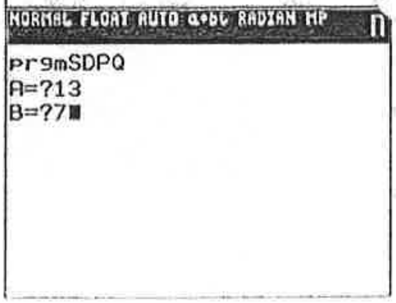
```
prgmU2SB1
A=?5
B=?6
                          11
                        Done
```

Using Output(Instead of Disp

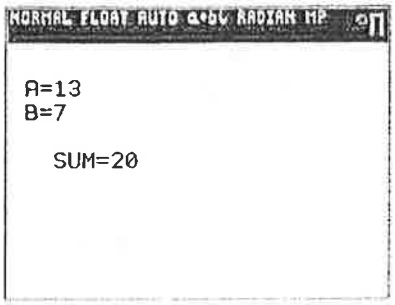
Recall that you can improve the output of programs using **Output(** rather than **Disp** to show the original values entered *and* the results *properly labeled*. Just put the calculation right in the Output statement. You try it.

Example: **Output(5,7,A+B)** show the value of A+B on line 5 beginning in column 7.

- To the right are two screens of a running program, one showing the **Prompt** section and one showing the **Output** section. Can you do better?
- Remember to include **Pause** and **ClrHome** statements at the right moments in the program to keep the screen neat.



```
prgmSDPQ
A=?13
B=?7
```



```
 A=13
 B=7

   SUM=20
```

You cannot output two items with one **Output** statement. The message "SUM=" and the sum **A+B** must be output using separate statements. Screen position is important!

Note: you'll find the "=" ('equals' sign) on the Test menu (2nd MATH).

©2015 Texas Instruments Incorporated

Teacher Tip: Suggest trying other mathematical expressions in the **Disp** statement. We'll get into storing (assigning) values to variables later in this unit (see Unit 2, Skill Builder 3).

Remember that the TI-84 Plus (non-color) screen has 16 characters per line and 8 lines, so the choice of Output positions may vary. The TI-84 Plus C and CE (color) screens have 26 characters per line and 10 lines.

14

Unit 2: Using Variables and Expressions

Skill Builder 2: Inputting a value for a variable

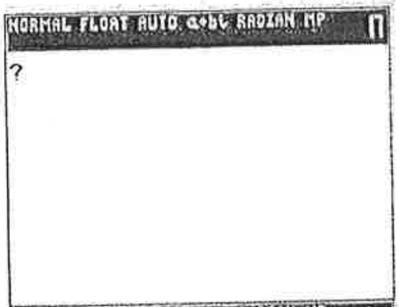
In this second lesson for Unit 2 you will learn about the different forms of the **Input** statement.

Objectives:

- Use the TI Basic **Input** statement to assign a value to a variable.
- Perform calculations within **Disp** statements.
- Use the GRAPH screen to get input to two variables at once.

The *Simple* Input Statement

The **Input** statement is followed by *only one* variable name to ask the user to enter a value for that variable. Unlike **Prompt**, **Input** only places a question mark on the screen as you can see in the program example at the right.



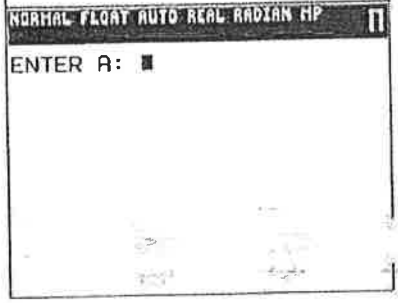
```
?
```

The *Improved* Input Statement

This type of **Input** statement can display a custom message that is displayed before waiting for a value for the variable. The structure of the Input statement with a message is:

Input "YOUR MESSAGE HERE",V

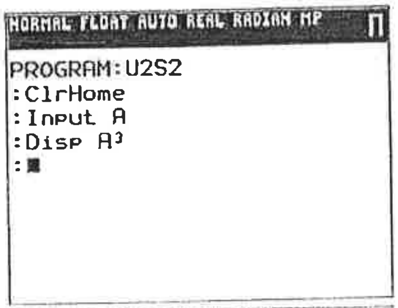
Note: This statement does not provide any question mark or other punctuation, so if you want one then it must be included inside the message.



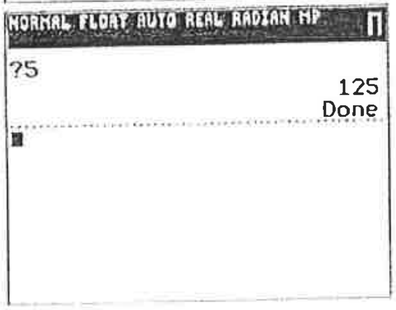
```
ENTER A: ▮
```

Programming with *Simple* Input

1. Start a new program.
2. For the first statement of the program use the **Input** statement found on the [PRGM] I/O menu.
3. After the **Input** command type the name of the variable you want your program to use. Here we use the variable **A**.
4. Use the **Disp** statement to display the cube, A^3; type the **A** then use the [MATH] menu to get the cute, small 'cubed' exponent.
5. Quit the editor and run the program.
6. After the "?" type any number and press [ENTER].
7. The program displays the cube of the entered number and ends.



```
PROGRAM:U2S2
:ClrHome
:Input A
:Disp A³
:▮
```



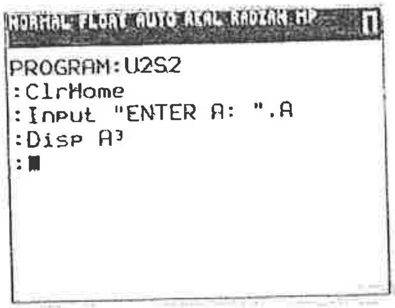
```
?5
                          125
                         Done
▮
```

©2015 Texas Instruments Incorporated

10 Minutes of Code

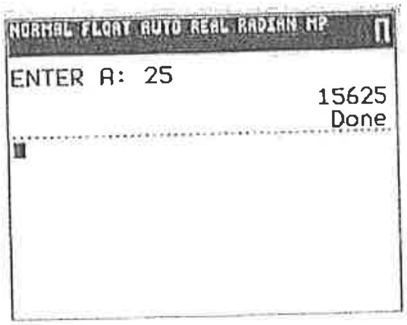
Programming with *Improved* Input

1. Edit the program you started earlier.
2. Place the cursor on the variable after the word **Input**.
3. Press [INS].
4. Type a message to display. Remember to use [A-LOCK] and quotation marks.
5. Include a punctuation mark at the end of the message (inside the quotes).
6. Place a comma after the closing quote and before the variable.
7. Keep the **Disp** statement that displays A^3.



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:U2S2
:ClrHome
:Input "ENTER A: ",A
:Disp A³
:
```

8. Quit the editor and run the program.
9. After the message type any number and press [ENTER].
10. The program displays the cube of that number and ends.



```
NORMAL FLOAT AUTO REAL RADIAN MP
ENTER A: 25
                  15625
                   Done
```

Using Input without a Variable

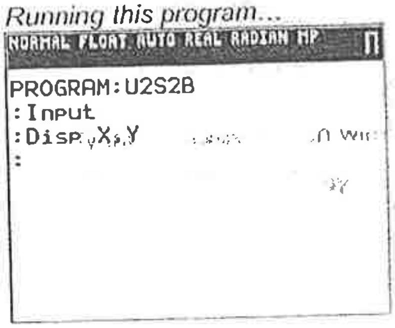
If you use the **Input** statement *without* a variable then the program will display the GRAPH screen with a free-floating cursor.

When you press [ENTER] the program continues and the variables **X** and **Y** contain the values that you pointed to on the GRAPH screen!

You can then use these two variables in the rest of your program.

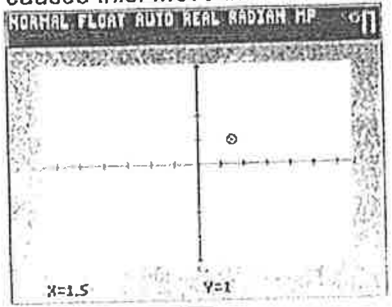
The intent of this feature is to let you input values for **X** and **Y** 'graphically'. Cool, eh?

Running this program...

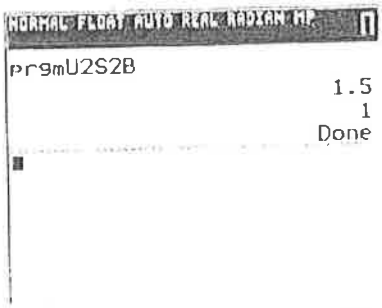


```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:U2S2B
:Input
:Disp X,Y
:
```

causes this. Move the cursor...



and press [ENTER] *to see this...*



```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmU2S2B
                    1.5
                      1
                   Done
```

©2015 Texas Instruments Incorporated

Teacher Tip: When we get into graphics programming in Unit 5 this special feature will come in very handy!

17

Unit 2: Using Variables and Expressions

Skill Builder 3: Expressions and Storing

In this third lesson for Unit 2 you will learn about using expressions and storing values in variables within programs.

Objectives:

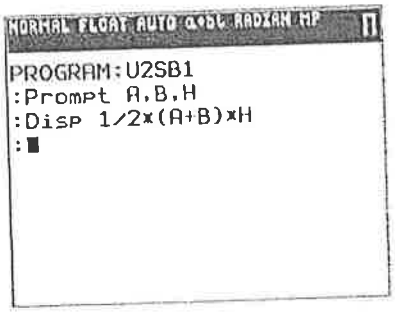
- Learn about programming mathematical expressions.
- Understand order of operations.
- Realize the difference between mathematical variables and computer program variables.
- Evaluate **expressions.**
- Store the results of expressions in variables.

Expressions

Items such as **A^2** and **A+B** are called *expressions*. Expressions can be found in mathematical formulas. For example, the **formula** for the area of a triangle is $A = \frac{1}{2} \times B \times H$. The **expression** is $\frac{1}{2} \times B \times H$.

A program evaluates an expression using the current values of all variables and gives the result as a numeric value. Expressions are evaluated using the *algebraic order of operations*.

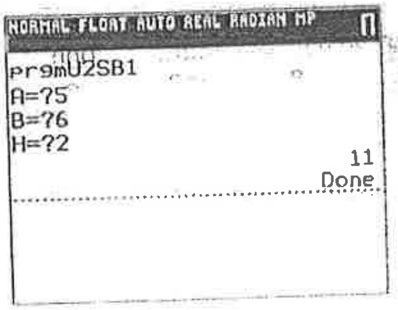
Try the program to the right which computes the area of a *trapezoid* with bases **A** and **B** and height **H**.



```
PROGRAM:U2SB1
:Prompt A,B,H
:Disp 1/2*(A+B)*H
:▮
```

You cannot use variables such as **B1** and **B2**. The calculator computes these as the expressions $B \times 1$ and $B \times 2$ and may cause an error when used incorrectly.

You also cannot use variables with more than one letter such as **AB**. As stated earlier, this means $A \times B$. This is called *implied* multiplication because the multiplication sign between the variables is 'implied' or assumed.



```
prgmU2SB1
A=?5
B=?6
H=?2
                    11
                  Done
```

Mathematical Expressions and Computer Expressions

While there are many similarities in the *appearance* of expressions in mathematics and computer programs there are also important differences. The most significant difference is that in a mathematical expression the variables stand for 'unknown' numbers and are replaced with numbers when needed. In a computer expression the variables are *names* for numbers.

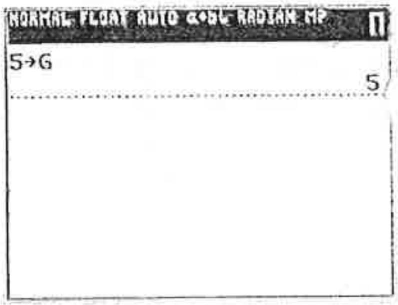
In mathematics we use formulas to state a relationship between things such as area and lengths. In programs we use the expressions to do the calculations and the variables' values are used to compute a result. When we *write* the program we type the expression but when we *run* the program the expression is computed and creates a result to be used later.

Teacher Tip: One of the more confusing statements in a program for beginners is x+1→x. It's pretty clear in the syntax of TI-Basic that 1 is being added to the variable x and then being stored in variable x, so the x on the left and the x on the right represent different values. This statement is known as a 'counter' because each time it is processed (in a loop) it increases x by 1. But in other languages such as B.A.S.I.C and Lua (and many more) the statement appears as x=x+1 which is clearly a false assertion in a mathematical sense but is perfectly good in a program!

18

©2015 Texas Instruments Incorporated

Storing Values in Variables: The Assignment Statement



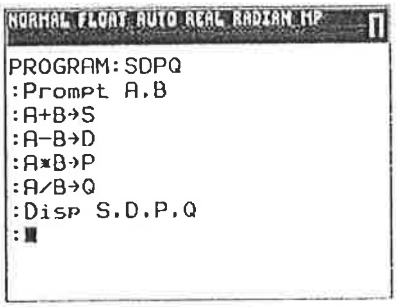
```
NORMAL FLOAT AUTO a+bi RADIAN MP
5→G
                                   5
```

The [STO▸] operator is used to **store** (assign) the result of an expression into a variable.

Pressing the [STO▸] key always displays the symbol → .

After pressing [ENTER], the HOME screen displays the result of the expression *and* the variable G now contains the value **5**. In front of the arrow must be a **value** or an **expression** that produces a **value**. This is called the *assignment statement* because it assigns a value to a variable. The symbol after the arrow must be a **variable**.

Programming with Assignment Statements



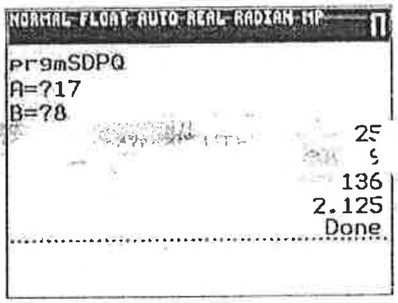
```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:SDPQ
:Prompt A,B
:A+B→S
:A-B→D
:A*B→P
:A/B→Q
:Disp S,D,P,Q
:▮
```

Let's write a program that asks you to enter two numbers, stores their sum, difference, product, and quotient in four variables and then displays the results.

1. Start a new program. Our program name is **SDPQ**.
2. Prompt for two variables **A** and **B**.
3. Store the four expressions in four *other* variables **S**, **D**, **P**, and **Q**.
4. Display **S**, **D**, **P** and **Q**.
5. Run the program.

Enter a value for **A** and another value for **B**.

Be sure to use numbers for which you can confirm that the calculations were performed correctly! This is known as 'testing' the program.

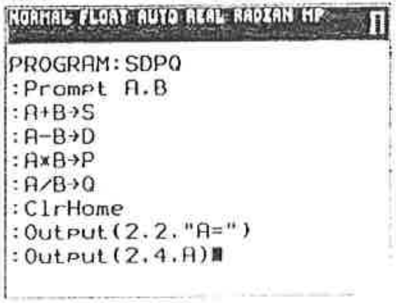


```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmSDPQ
A=?17
B=?8
                                  25
                                   9
                                 136
                               2.125
                                Done
```

Note: TI Basic's assignment statement is unique in that the expression comes first, then the 'store' operator, then the variable. This makes it easy to read from left to right. In most other languages the order is the opposite, such as S=A+B. This is backwards because the computer evaluates the expression on the right first and then stores the result in the variable on the left.

Teacher Tip: There are many advantages to storing values in variables. Using variables makes it easier to display the result. It also allows the variable to be used in subsequent calculations. If you find you are using the same number in a lot of different places in your program then you can store the number in a variable and use it everywhere that it occurs in your program.

Making Improvements



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:SDPQ
:Prompt A,B
:A+B→S
:A-B→D
:A*B→P
:A/B→Q
:ClrHome
:Output(2,2,"A=")
:Output(2,4,A)▮
```

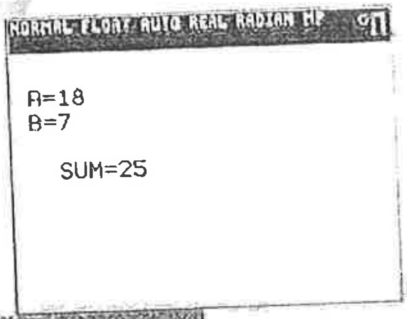
You can improve the program using **Output(** rather than **Disp** to show the original values entered *and* the four results *properly labeled*. You try it!

To the right is a snippet of code and a screen of the partially completed program running. There's still some work to do here so we'll leave it to you to complete the program.

Remember to include a **Pause** statement at the end of the **Output** statements to prevent the 'Done' message from spoiling the display.

19

Remember that the **TI-84 Plus** and the **TI-84 Plus C/CE** have different HOME screen sizes (as well as different GRAPH screen sizes) so plan accordingly. The TI-84 Plus HOME has 16 characters per line and 8 lines, while TI-8 Plus C/CE HOME has 26 characters per line and 10 lines.



```
A=18
B=7

   SUM=25
```

Teacher Tip: Challenge students to experiment with other mathematical formulas and use assignment statements to store the result to make it easier to **Display** or **Output** the results. In the Application for this unit students are given three mathematical formulas to compute.

Unit 2: Using Variables and Expressions

Application 2: Computing a Formula

In this Application for Unit 2 you will write programs to evaluate some mathematical formulas.

Objectives:

- Use the TI Basic statements learned in Unit 2 to write a program that evaluates a formula.

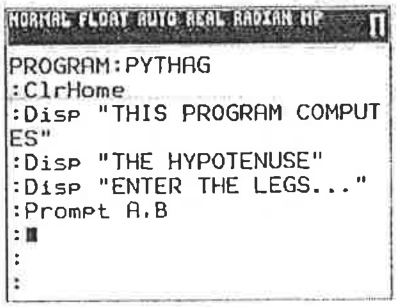
The Pythagorean Theorem

In a right triangle with legs A and B and hypotenuse C,

$$A^2 + B^2 = C^2$$

Write a program that asks the user to enter the lengths of the legs then computes the length of the hypotenuse and nicely displays all three values.

Note: You first have to solve the formula above for C.


```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:PYTHAG
:ClrHome
:Disp "THIS PROGRAM COMPUT
ES"
:Disp "THE HYPOTENUSE"
:Disp "ENTER THE LEGS..."
:Prompt A,B
:■
:
:
```

Heron's Formula

Heron's Formula determines the area of any triangle using only the lengths of the three sides of the triangle, A, B, and C. It is usually stated in two parts:

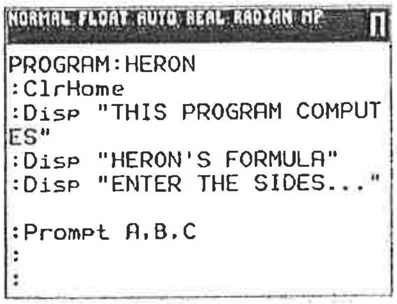
$S = (A + B + C)/2$ is the 'semi-perimeter' (half the perimeter) of the triangle

$A = \sqrt{S*(S-A)(S-B)(S-C)}$ is the area of the triangle

Write a program that asks the user to enter the lengths of the three sides of a triangle and then computes the area and displays (Outputs) the sides and the area on a pretty screen.

Note: It's possible for the user to enter three numbers that cannot be the sides of any triangle. What will happen when the user enters invalid values?

Teacher Tip: Heron's formula will fail (NON-REAL answer) when the three values are impossible for a triangle (the Triangle Inequality). This can be accounted for by changing the complex MODE to a+bi.


```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:HERON
:ClrHome
:Disp "THIS PROGRAM COMPUT
ES"
:Disp "HERON'S FORMULA"
:Disp "ENTER THE SIDES..."

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

The Quadratic Formula

If a quadratic equation is of the form $Ax^2 + Bx + C = 0$ then the roots of the equation are found by...

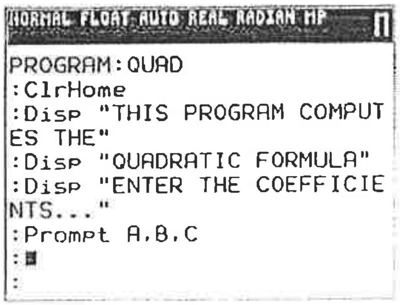
First, determining the discriminant:

$$D = B^2 - 4AC$$

And then the two roots are:

$$R1 = (-B + \sqrt{D})/(2A)$$

$$R2 = (-B - \sqrt{D})/(2A)$$


```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:QUAD
:ClrHome
:Disp "THIS PROGRAM COMPUT
ES THE"
:Disp "QUADRATIC FORMULA"
:Disp "ENTER THE COEFFICIE
NTS..."
:Prompt A,B,C
:■
:
```

Write a program that asks the user to enter the three coefficients of the quadratic equation, A, B, and C and nicely displays the coefficients and the two roots of the equation.

©2015 Texas Instruments Incorporated

Note: You cannot use R1 and R2 as variables! Use something else.

What could possibly go wrong with this program?

Teacher Tip: Here are program listings for each assignment. The important steps are the formula calculations. The Output positions should be fine on any TI-84, but the TI-84 C/CE may use different values depending on the appearance desired.

The Pythagorean Theorem

Answer:

```
prgmPYTHAG
ClrHome
Disp "THIS PROGRAM COMPUTES"
Disp "THE HYPOTENUSE"
Disp "ENTER THE LEGS..."
Prompt A,B
√(A²+B²)→C

ClrHome
Output(3,5,"A= ")
Output(3,8,"A)
Output(4,5,"B= ")
Output(4,8,B)
Output(6,5," HYPOTENUSE = ")
Output(6,16,C)
Pause
ClrHome
```

Heron's Formula

Answer:

```
prgmHERON
ClrHome
Disp "THIS PROGRAM COMPUTES"
Disp "HERON'S FORMULA"
Disp "ENTER THE SIDES..."
Prompt A,B,C
(A+B+C)/2→S
√(S(S-A)(S-B)(S-C))→D

ClrHome
Output(3,5,"A= ")
Output(3,8,A)
Output(4,5,"B= ")
Output(4,8,B)
Output(5,5,"C= ")
Output(5,8,C)
Output(7,5,"AREA= ")
Output(7,11,D)
Pause
ClrHome
```

The Quadratic Formula

Answer:

```
prgmQUAD
ClrHome
Disp "THIS PROGRAM COMPUTES
THE"
Disp "QUADRATIC FORMULA"
Disp "ENTER THE COEFFICIENTS..."
Prompt A,B,C
B²-4AC→D
(-B+√(D))/(2A)→R
(-B-√(D))/(2A)→S

ClrHome
Output(3,5,"A= ")
Output(3,8,A)
Output(4,5,"B= ")
Output(4,8,B)
Output(5,5,"C= ")
Output(5,8,C)
Output(7,5,"ROOT1= ")
Output(7,12,R)
Output(8,5,"ROOT2= ")
Output(8,12,S)
Pause
ClrHome
```

22

Unit 3: Conditional Statements

Skill Builder 1: Conditions and the If... Statement

In this first lesson for Unit 3 you will learn about conditions and the introduction to the **If** statements available in TI Basic.

Objectives:

- Learn about conditions.
- Use the 'simple' **If…** statement to conditionally process another statement.

If…Then statements are used to process a block of statements only when a *condition* is true or false. Before visiting the **If…Then** collection of statements, let's get an idea of just what a *condition* is.

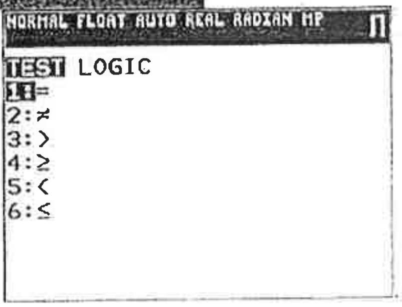
Teacher Tip: Relational and logic operators also have an order of operations and fit into the arithmetic order of operations as well.

Here's a complete list of the TI-Basic order:

Priority Level	*Functions*
1	*Functions that precede their argument (such as √(or sin(), except for negation*
2	*Functions that follow their argument (such as 2 or !)*
3	*^ and ×√*
3.5	*Negation*
4	*nPr and nCr*
5	*Multiplication, division, and implied multiplication*
6	*Addition and subtraction*
7	*The relational operators =, ≠, <, >, ≤, ≥*
8	*The logic operator and*
9	*The logic operators or and xor*
10	*Conversions such as ►Frac*

Conditions and the [TEST] Menu

Conditions are expressions that evaluate to 'true' or 'false'. Such expressions are either true <u>or</u> false; they cannot be both or neither. The relational operators and the logical operators are both found on the [TEST] menu (2nd MATH). The TEST menu contains the **relational operators.** The LOGIC menu contains the **logical operators.** The = sign is used to form a condition, not an assignment.



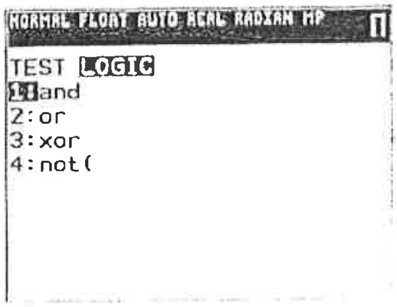
```
NORMAL FLOAT AUTO REAL RADIAN MP
TEST LOGIC
1:=
2:≠
3:>
4:≥
5:<
6:≤
```

Examples of some conditions:

3>5   XY>0   X=5 or Y=5

X+4>X B^2-4AC=0   X/2=int(X/2)

XøY   X>0 and Y>0   not(X>0)



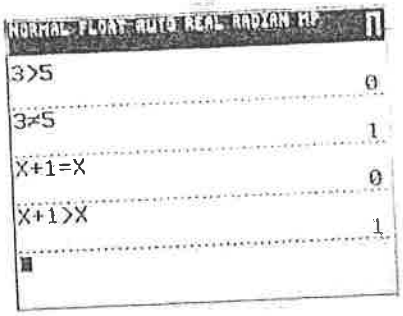
```
NORMAL FLOAT AUTO REAL RADIAN MP
TEST LOGIC
1:and
2:or
3:xor
4:not(
```

©2015 Texas Instruments Incorporated

10 Minutes of Code

nditions on the HOME Screen

You can enter conditions right on the HOME screen to see how they are computed.



```
NORMAL FLOAT AUTO REAL RADIAN MP
3>5
                                0
3≠5
                                1
X+1=X
                                0
X+1>X
                                1
```

Observe that 1 stands for *true* and 0 stands for *false*.

Note: when you use a variable in a condition the calculator evaluates it using the current value stored in the variable.

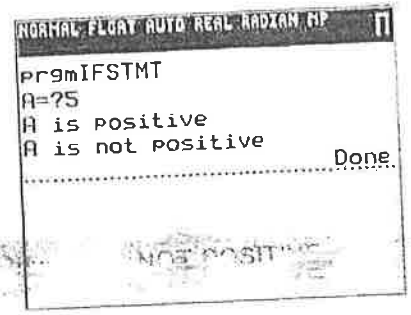
Teacher Tip: We introduce the 'primitive' or 'simple' If statement below because it is an easy way of conditionally executing only one statement (if that's all that is needed). But the more versatile If… Then statement discussed after is preferred because it is clearer to a reader of the program what the programmer is trying to do. The If <condition> (without Then) processes only the next statement when the condition is true, otherwise it is skipped.

Programming with the 'Simple' If… Statement

Try this program:

```
:Prompt A
:If A>0
:Disp "A IS POSITIVE"
:Disp "A IS NOT POSITIVE"
```

[If is in the PRGM CTL menu. '>' is in the [TEST] menu]



```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmIFSTMT
A=?5
A is positive
A is not positive
                             Done
```

Run the program several times entering both positive and non-positive numbers and observe the output. What can you learn?

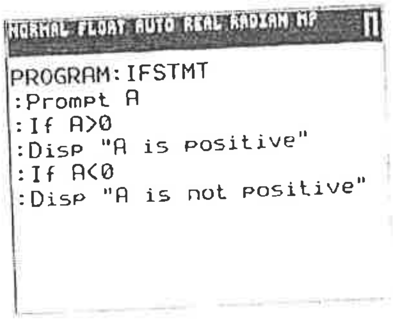
When the condition A>0 is true, the statement that follows If is executed, otherwise it is simply skipped. But the statement that displays "A IS NOT POSITIVE" is <u>always</u> executed, which is not correct! See the screen at the right. We'll fix this soon.

This 'simple' If… is a concise way of skipping **one** statement based on a condition (when it is FALSE).

Editing the If… Statement

Let's correct the program above by adding another If…

1. Place the cursor on the second **Disp.**
2. Press ï and press ENTER to insert a blank line.
3. On that blank line add **If A<0.**



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:IFSTMT
:Prompt A
:If A>0
:Disp "A is positive"
:If A<0
:Disp "A is not positive"
```

Quit and run the program several times using both positive and negative numbers and 0, too!

es your program work correctly in all cases? If, not, try to fix the problem.

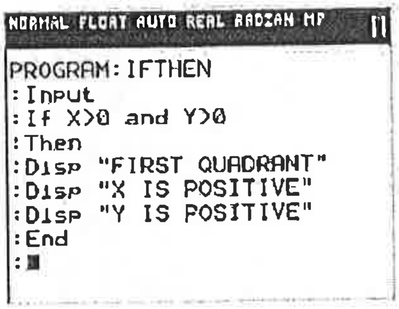
Unit 3: Conditional Statements **Skill Builder 2: If...Then...End and Compound Conditions**

In this second lesson for Unit 3 you will learn about a much better conditional structure and compound conditions.

Objectives:

- Examine the If...Then...End structure.
- Make compound conditions with the logical operators.
- Write a program using the If...Then...End structure that examines the regions of the coordinate plane.

The If...Then...End structure



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:IFTHEN
:Input
:If X>0 and Y>0
:Then
:Disp "FIRST QUADRANT"
:Disp "X IS POSITIVE"
:Disp "Y IS POSITIVE"
:End
:
```

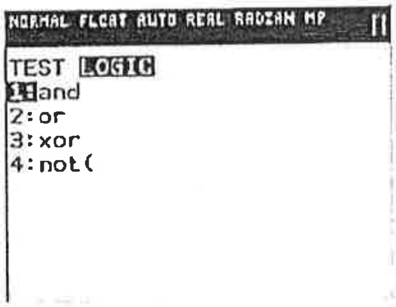
TI Basic has a unique **If...Then** structure that makes use of the **End** keyword to control the statements that form the block of code that will be processed when the condition is true. It looks like this:

If *<condition>*
Then
<true block: do these statements when the <condition> is true
End

Note:
If is followed by some <condition>.
***Then** is immediately below **If**, set on a line by itself.*
There are one or more statements in the <true block>.
***End** indicates the end of the **Then** block and the statements below **End** will be processed.*
End is not the end of the program! It is the End of the If...Then...End structure.

Teacher Tip: This statement is preferred over the previous If statement because it is easier to read. The 'block' can be a single statement (or even no statement at all). Then and End appear on their own lines in the program. The block can also include another If statement. Each If Then requires a corresponding End. But we discuss nested structures later on.

Compound Conditions



```
NORMAL FLOAT AUTO REAL RADIAN MP
TEST LOGIC
1:and
2:or
3:xor
4:not(
```

Compound conditions involve more than one relational expression. The logical operators **and**, **or**, **xor** and **not(** are found on the [TEST] LOGIC menu. These operators allow you to build compound conditions.

Examples:

- X>0 **and** Y>0 is true when both X and Y are positive
- X>0 **or** Y>0 is true when either X or Y is positive (or both)
- **not(**X>0 **and** Y>0) is true when either X or Y is not positive
  it means the same as X<=0 **or** Y<=0
- X>0 **xor** Y>0 is true when either X or Y is positive *but not both*
  it means the same as... X>0 **or** Y>0 **and not(**X>0 **and** Y>0)

xor stands for 'exclusive or' and is true when either part is true but not both parts.
You cannot 'string together' the relational operators: 2<A<3 is interpreted to mean "A is between 2 and 3" and must be coded as 2<A **and** A<3. The logical operators have an order of operations just like the arithmetic operators +, - , *, and /.

A<0 **or** A<5 **and** A>2 means A can be negative or between 2 and 5.
and is processed before **or** (similar to 'multiplication before addition').

©2015 Texas Instruments Incorporated

Teacher Tip: Some practice evaluating logical conditions would be helpful here. If you've never done it, truth tables make a good learning activity.

A	B	A and B	A or B	not(A)
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Programming with If...Then...End Statements

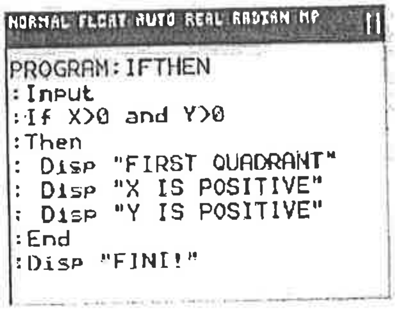
Try the **IFTHEN** program to the right.

Note: ***Input*** *has no variable. This is a special feature of TI-Basic. Recall from Unit 2 that the GRAPH screen will appear so that you can move the cursor anywhere and press [ENTER] to set values for X and Y.*

'and' is on the [TEST] **LOGIC** *menu.*

Then *is on a line by itself right below* ***If***

End *is the bottom of the 'true' block (the set of statements that are executed when the condition is true). It is not the end of the program.*



```
PROGRAM:IFTHEN
:Input
:If X>0 and Y>0
:Then
: Disp "FIRST QUADRANT"
: Disp "X IS POSITIVE"
: Disp "Y IS POSITIVE"
:End
:Disp "FINI!"
```

Complete the Program

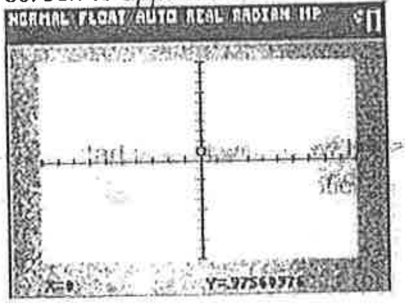
A graph has several named regions: Quadrants I, II, III, and IV and the positive and negative *x* and *y* axes. Let's write a program that allows the user to select a point on the GRAPH screen and then the program will tell where the point lies using those names.

We'll start you off with a few **If** statements and you can finish the rest:

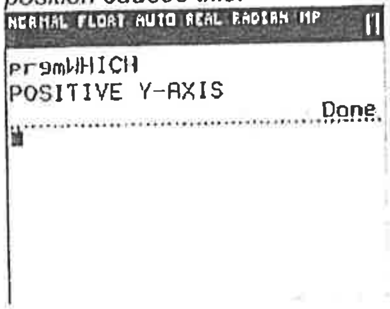
```
Input              notice, no variable!
Disp X,Y
If X>0 and Y>0
Then
Disp "FIRST QUADRANT"
End
If X=0 and Y>0
Then
Disp "POSITIVE Y-AXIS"
End
If X<0 and Y>0
Then
Disp "SECOND QUADRANT"
End
.
.
.
```

You should have eight **If** structures (for the four quadrants and the four half-axes).

Running the program cause this screen to appear…



…and pressing enter at that position causes this:



```
prgmWHICH
POSITIVE Y-AXIS
                     Done
```

20

Sample Answer:

[Notice, no variable]

```
Input
Disp X,Y

If X>0 and Y>0
Then
Disp "FIRST QUADRANT"
End

If X=0 and Y>0
Then
Disp "POSITIVE Y-AXIS"
End

If X<0 and Y>0
Then
Disp "SECOND QUADRANT"
End

If X<0 and Y=0
Then
Disp "NEGATIVE X-AXIS"
End

If X<0 and Y<0
Then
Disp "THIRD QUADRANT"
End

If X=0 and Y<0
Then
Disp "NEGATIVE Y-AXIS"
End

If X>0 and Y<0
Then
Disp "FOURTH QUADRANT"
End
```

27

©2015 Texas Instruments Incorporated

In this third lesson for Unit 3 you will learn about another form of the **If** statement in TI Basic and the value of understanding numeric algorithms.

Objectives:

- See the structure of the **If...Then...Else...End** statement.
- Learn the test for 'whole-ness' (is a value an integer?).
- What is an **Algorithm?**

Teacher Tip: This lesson introduces students to the concept of an **algorithm**. This principle is critical to successful programming. While we do not delve too deeply into an algorithmic approach, the idea of having a plan before writing a program is crucial, since writing a program first requires an understanding of the task and knowledge of the programming commands and their purpose. Then we can devise an algorithm to solve the problem.

The three forms of the **If** statement are concluded in this lesson with a discussion of **If...Then...Else...End**. The syntax is important: each of the four keywords appear as separate statements in the program and only the **If** can have something follow it (the *condition*).

About If...Then...Else...End

In the previous activity you learned about the **If...Then...End** statement. There are times when we'll need to take two different courses of action depending on a condition. The structure of this new **If...Then...Else...End** statement is similar:

```
If   <condition>
Then
    <True block>
Else
    <False block>
End
```

Note:

Then, **Else**, *and* **End** *are on line by themselves.*

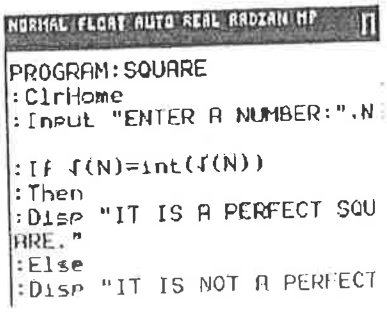
The <True block> is the set of statements that will be executed when the <condition> is true.

The <False block> is the set of statements that will be executed when the <condition> is not true.

Therefore, one or the other of these two blocks will be executed.

Programming with If...Then...Else...End

We'll write a program to tell whether or not an entered number is a perfect square. A 'perfect square' is the square of an integer, such as 25 (5^2). The method used here is to take the square root of the number and test to see if it is an integer. The program listing is at the right.



```
PROGRAM:SQUARE
:ClrHome
:Input "ENTER A NUMBER:",N

:If √(N)=int(√(N))
:Then
:Disp "IT IS A PERFECT SQU
ARE."
:Else
:Disp "IT IS NOT A PERFECT
```

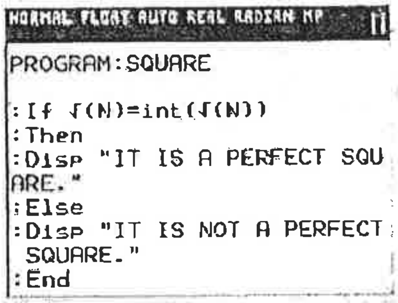
Note:

Use the [√] key on the keypad for the square root symbol.

If, **Then**, **Else**, *and* **End** *are all on the* [PRGM] CTL menu.

The **int(**) *function is on the* [MATH] NUM *menu. The function returns an integer, so, for example,* int(6.56) → 6   int(9.999) → 9   int(-2.01) → -3

Try your own **int(**) examples on the HOME screen. Remember to close parentheses for the **int(**) function and the square root function. You'll only use UPPERCASE letters in your **Disp** statements on the calculator. You can enter lowercase letters through **TI-Connect CE**.



```
PROGRAM:SQUARE

:If √(N)=int(√(N))
:Then
:Disp "IT IS A PERFECT SQU
ARE."
:Else
:Disp "IT IS NOT A PERFECT
 SQUARE."
:End
```

Teacher Tip: Suggest to students that they could improve the output of the program by displaying the number entered in the sentence rather than just the word 'It'. This would require the use of **Output(**) rather than **Disp**.

Algorithms

Techniques such as that used in this exercise are known as '**algorithms**'. An **algorithm** is a procedure or formula for solving a problem.

A recipe for baking a cake is an algorithm. If you follow the recipe you will have cake. All mathematical formulas, such as the formula of a triangle (A=B*H/2) are algorithms: they give you a method to determine a new value based on existing values. Algorithms such at the 'perfect square' technique above are important problem-solving tools. Learning common computer algorithms such as this will make your programming experience very rewarding.

Here's a box cake recipe…

1. Prepare cake batter per directions on box.
2. Bake as directed on package—two cake layers.
3. Cool in pans 10 minutes.
4. Remove from oven to wire racks.
5. Let cool completely.
6. Beat pudding mixes and milk with whisk 2 minutes.
7. Immediately spread over tops of cakes.
8. Stack cake layers.
9. Frost with whipped cream.
10. Enjoy!

As in a program, the baker follows the steps from beginning to end. At the end there's a delicious cake to enjoy. When a computer follows the steps in a program (the algorithm) the desired result is achieved. And, in fact, there's a branch of computer science that deals with PROVING that an algorithm will give the desired result. It's similar to proving mathematics theorems.

Teacher Tip: Here's an example of a computer algorithm:

To round a number to the nearest whole number use:

A = int(A+0.5)

Why does it work? When the decimal part of A is less than 0.5, adding 0.5 to the number keeps the number lower than the next higher integer. Then the int( ) function truncates the number at the decimal point, leaving just the integer part.

Here's an example:

suppose A=3.4   A+0.5 = 3.9   int(3.9) = 3 (rounds down)

©2015 Texas Instruments Incorporated

suppose A=3.7 A+0.5 = 4.2 int(4.2) = 4 (rounds up)

But of course, the TI-84 also has a built-in round( ) function on the MATH NUM menu!

©2015 Texas Instruments Incorporated

Unit 3: Conditional Statements

Application: The Zodiac

In this application for Unit 3 you will be developing a program that can tell the user in what sign of the zodiac an entered date lies.

Objectives:

- Work with date conversions.
- Use **If** statements to determine the sign of the zodiac for a given date.
- Work with **String** variables.

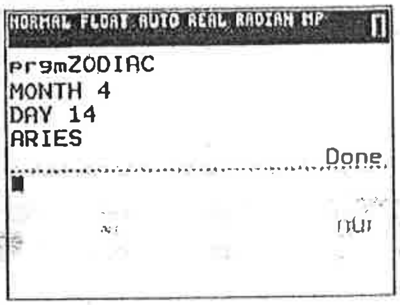
The Zodiac

In astronomy and astrology the **zodiac** is a division of the sky into 12 equal regions. The regions are named by the constellations that are approximately within these regions. The Babylonians developed this division around 1,000–500 B.C. They began their calendar year with the vernal equinox (the first day of Spring), hence the first sign of the zodiac is Aries and covers the period from March 21 to April 20.

Teacher Tip: The Zodiac is a legitimate astronomical convention as well as an astrological one. Students can research the Zodiac online.

The Program

In this lesson we'll write a program that lets a user enter a month and a day and then the program displays the sign of the zodiac for that date. This program utilizes many **If...Then** statements.



```
prgmZODIAC
MONTH 4
DAY 14
ARIES
                          Done
```

The user will enter the month number and day number according to our calendar, and the program will convert the date into the Babylonian format (March = 1, April = 2, … , January = 11, February = 12). The tests for each sign can be complex. For example:

If (M=1 and D>20) or (M=2 and D<21)

To simplify the programming we'll develop a single numeric value (a *code*) to represent the dates. This code makes it easier to write the **If** statements rather than have to deal with more complex expressions involving **and**s and **or**s.

Teacher Tip: Writing pseudocode is a form of programming that does not depend on any particular programming language. Writing in pseudocode first means that the programmer can then implement the algorithm in any programming language.

Pseudocode

When developing a large program it is often helpful to start with a plain-English outline of the program. This is called 'pseudocode' because it is not written using a specific programming language. The outline is written in a way that makes it easy to convert to a programming language later on.

Here's an outline (pseudocode) of the zodiac program:

```
   PROGRAM: ZODIAC
Enter the Month          user enters a number from 1 to 12
Enter the Day            user enters a number from 1  to 31
```

©2015 Texas Instruments Incorporated

10 Minutes of Code

(The zodiac signs begin with Aries so we make March the first month and January and February are months 11 and 12.)

Subtract 2 from the Month.

If the Month is less than 1 then add 12 to the Month.

Create a one-number Code for the date (combine Month and Day into one number, the *Code*, whose first one or two digits are the month and whose last two digits are the day). We accomplish this by multiplying the month by 100 and add the day. For example: July 4 is Babylonian Month 5, Day 4 so the Code is 100*5+4 = 504.

> Teacher Tip: When we multiply a number by 100 we add two zeros to the end of the number. Then, when adding the day value, the leftmost 1 or 2 digits are the month and the day is the right two digits. This puts the date into a neat numerical form for comparison to the zodiac signs boundary dates.

Multiply the Month by 100 and add the Day and then store the result in the Code variable.

> Teacher Tip: String variables store a collection of characters (words) rather than numbers. The TI-84 Plus CE has ten special variables that can store strings. See the notes later in this document.

Store "Invalid" in a **String*** variable. This variable will be used later to display the sign or the word *Invalid*. See ***Strings** below.

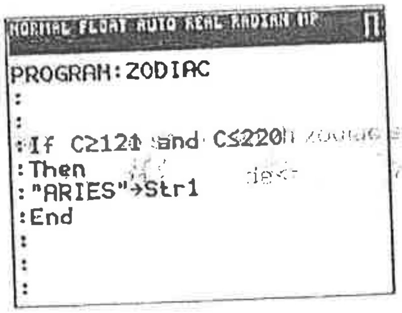
Now check to see in which zodiac sign the Code belongs:

if Code >=121 and Code<=220 *this represents the days from March 21 to April 20*

then

store "Aries" in the string variable **Note the quotation marks!**

end



```
PROGRAM:ZODIAC
:
:
:If C≥121 and C≤220
:Then
:"ARIES"→Str1
:End
:
:
:
```

Write one of these structures for each of the twelve signs of the zodiac.

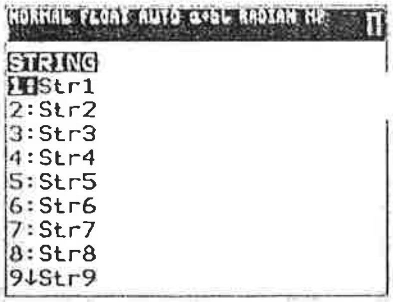
After the 12 **If** structures the string variable will contain either "INVALID" or one of the signs, so...

Display the string variable.

Here are the dates for each sign:

Aries:	March 21–April 20
Taurus:	April 21–May 21
Gemini:	May 22–June 21
Cancer:	June 22—July 22
Leo:	July 23—August 22
Virgo:	August 23—September 23
Libra:	September 24—October 23
Scorpio:	October 24—November 22
Sagittarius:	November 23—December 21
Capricorn:	December 22—January 20
Aquarius:	January 21—February 19
Pisces:	February 20—March 20

©2015 Texas Instruments Incorporated

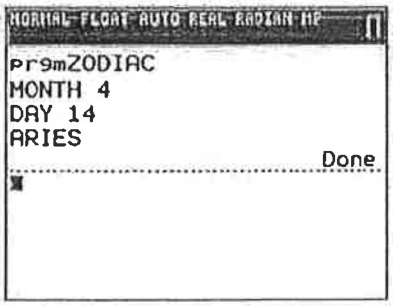


***Strings**

This program stores a *string of characters* such as "INVALID" in a **String** variable. The TI-84 has 10 string variables for you to use. To access these variable names press the VARS key and select the **String…** menu. Be sure to use only one string variable for all the signs of the zodiac.

Your Task

Write the program **ZODIAC** that meets the above specifications. The program should produce something like what is displayed here.



```
prgmZODIAC
MONTH 4
DAY 14
ARIES
                    Done
```

Caution: *Pisces* takes a special condition!

Extension:

The program does not check to see if the month and day entered are legal dates. Add **If** statements after the input section to make sure that the values entered are 'legal'.

Tip: Remember, some months have 30 days, some have 31, and one has only 29. Also, there is no month 13 or 14. See what happens if you enter 14 for the month.

33

Sample Answer (without the 'extension'):

PROGRAM: ZODIAC

If C≥623 and C≤723

©2015 Texas Instruments Incorporated

10 Minutes of Code

```
Input "MONTH ",M
Input "DAY ",D

M-2→M
If M<1
Then
M+12→M
End

100M+D→C
"INVALID"→Str1

If C≥121 and C≤220
Then
"ARIES"→Str1
End
If C≥221 and C≤321
Then
"TAURUS"→Str1
End
If C≥322 and C≤421
Then
"GEMINI"→Str1
End
If C≥422 and C≤522
Then
"CANCER"→Str1
End
If C≥523 and C≤622
Then
"LEO"→Str1
End
Then
"VIRGO"→Str1
End
If C≥724 and C≤823
Then
"LIBRA"→Str1
End
If C≥824 and C≤922
Then
"SCORPIO"→Str1
End
If C≥923 and C≤1021
Then
"SAGITTARIUS"→Str1
End
If C≥1022 and C≤1120
Then
"CAPRICORN"→Str1
End
If (C≥1121 and C≤1219)
Then
"AQUARIUS"→Str1
End
If (C≥1219 and C≤1229) or (C≥101 and C≤120)
Then
"PISCES"→Str1
End

Disp Str1
```

(34)

Unit 4: Skill Builder 1: Loops and the For(...) loop

In this first lesson for Unit 4 you will learn about the loop concept and the structure and use of the **For(...)** loop.

Objectives:
- Understand loops.
- Use the For(...) loop to generate a list of values.

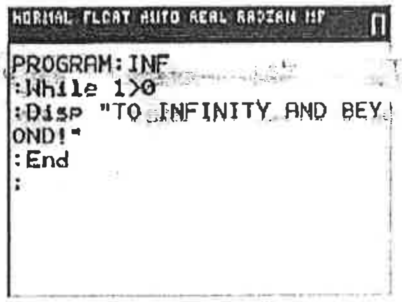
Teacher Tip: There are three fundamental loops in TI-Basic: For, While, and Repeat. A loop structure gives a program the ability to process a set of statements over and over either iterating over a sequence of values (as in the For loop) or until a specific condition is met (or not) as in **While** and **Repeat**. In this unit each lesson deals with just one of these structures. It is also possible to use the archaic **Lbl** and **Goto** statements to make a loop but that leads to bad habits and could result in program errors if not done properly, so we avoid referencing these statements at all. However, the **Lbl** statement is used and, in fact, required in conjunction with the **Menu** statement to design custom menus within a program. For conditional statements and loops, though, the **Goto** statement is not needed at all.

Programs can become complicated because it becomes necessary to mix in all the control structures, (If statements and loops) in a program to satisfy more complex algorithms. This is what makes programming FUN!

Loops

A loop is a method of repeating a set of statements. All programming languages have at least one looping structure. The loop structure has a way of going backwards in a program to a previous spot. TI-Basic has three different types of loops. An infinite loop never ends.

To 'break' (stop) a running program, press [ON]. You'll see the options 'Quit' and 'Goto'. 'Quit' returns you to the HOME screen and 'Goto' takes you into the Program Editor to the spot where the program stopped.

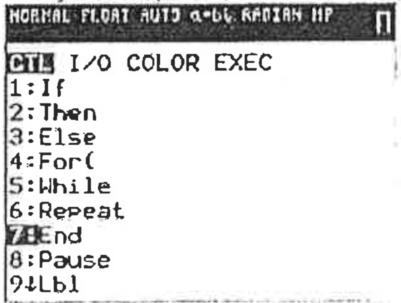


```
PROGRAM:INF
:While 1>0
:Disp "TO INFINITY AND BEY
OND!"
:End
:
```

An *infinite loop*. Why?

The Three TI-Basic Loops are:

For( ) ... **End**
While <condition is true> ... **End**
Repeat <*until* condition is true> ... **End**

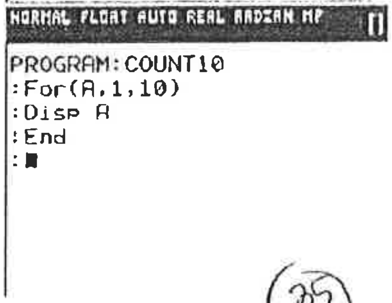


This rest of this lesson deals only with the **For(**) loop.

The For(...) Loop

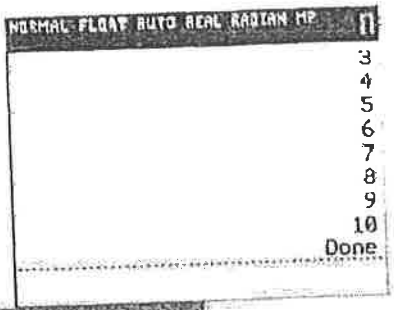
Structure: For(*variable, starting value, ending value*)
loop body
End

Example:
```
For(A,1,10)
    Disp A        ← loop body
End
```



```
PROGRAM:COUNT10
:For(A,1,10)
:Disp A
:End
:
```

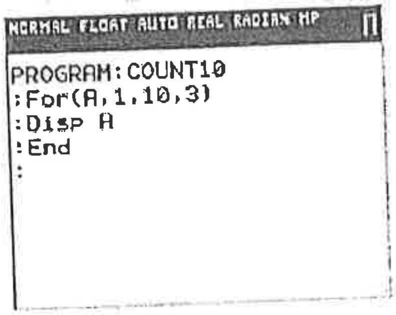
©2015 Texas Instruments Incorporated



```
NORMAL FLOAT AUTO REAL RADIAN MP
                                 3
                                 4
                                 5
                                 6
                                 7
                                 8
                                 9
                                10
                              Done
```

Note:
The **For( )** *statement requires a variable (the loop control variable), a starting value and an ending value, separated by commas. The starting and ending values can be variables. The loop body can be as many statements as needed but should not change the loop control variable. The loop will run from your starting value to your ending value with a standard increment of 1.*

Teacher Tip: The loop ends when the variable *exceeds* it's ending value. If you Disp A after the loop above ends you will see that its value is 11, not 10.

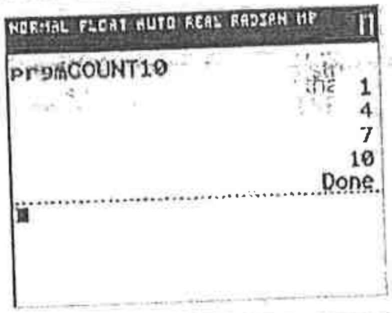


```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:COUNT10
:For(A,1,10,3)
:Disp A
:End
:
```

For Loop With Increments Other Than 1

There is an optional fourth argument for the **For( )** statement: the *increment*. The increment is the value by which the loop control variable increases with each iteration of the loop. The default value is 1.

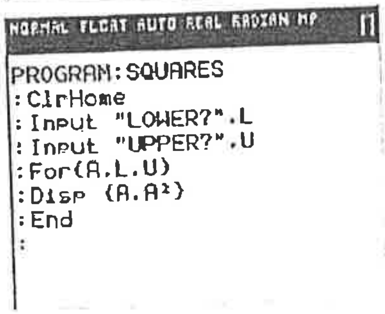
Teacher Tip: When the loop increment is negative the loop ends when the loop variable is *less than* the ending value.



```
NORMAL FLOAT AUTO REAL RADIAN MP
prgmCOUNT10
                                 1
                                 4
                                 7
                                10
                              Done
```

For(A,1,10,3) starts with **A**=1, then adds 3 to **A** each time the loop repeats. The loop stops when **A** is greater than 10. The increment can be a negative number.

For(B,10,0,-1) counts down from 10 to 0.



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:SQUARES
:ClrHome
:Input "LOWER?",L
:Input "UPPER?",U
:For(A,L,U)
:Disp {A,A²}
:End
:
```

Programming with For(...)

Let's write a program that displays a table of numbers and their squares. The user can enter the lower and upper bounds of the range of numbers. The tricky part is to **Display** the pairs of numbers on the same line! We can do this using **lists**.

Note:
L *and* **U** *are used to represent* **L**ower *and* **U**pper. *The* **For(**) *statement uses the values of* **L** *and* **U**. *The list brackets are on the parentheses keys. Press* 2nd (*and* 2nd) *for the brackets.*

Teacher Tip: Displaying a list is a convenient method to display more than one value on a line on the HOME screen.

Run the program and enter a lower and an upper bound for the table.

If the lists go by too quickly then consider adding a **Pause** statement after the **Disp** statement and before the **End** statement.

30

©2015 Texas Instruments Incorporated

Challenge:

Use an **If … Then… End** structure to **Pause** every 5 pairs of numbers. Recall the divisibility technique from the previous unit.

Sample Answer:

```
:If (A-L+1)/5=int((A-L+1)/5)
:Pause "Press Enter"
```

©2015 Texas Instruments Incorporated

10 Minutes of Code

it 4: Loops

Skill Builder 2: The While...End loop

In this second lesson for Unit 4 you will learn about the **While…End** loop. We'll compare it to the **For…** loop and even show why it is more powerful and versatile than the **For…** loop.

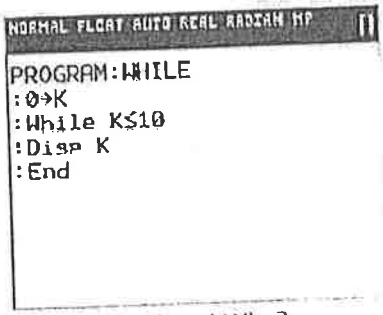
Objectives:

- Learn the structure of the **While…End** loop.
- Compare it to the **For…End** loop.
- See how it is used to ensure valid input values.

The While… End Loop

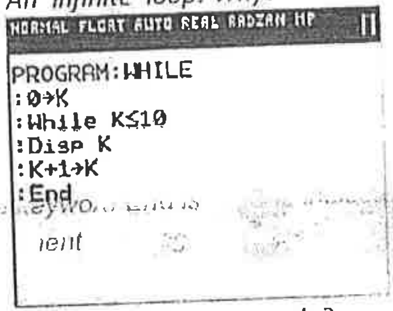
The **While…End** loop will continue looping as long as its <condition> is True. It looks like this:

```
While <condition>
    <loop body>
End
```



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:WHILE
:0→K
:While K≤10
:Disp K
:End
```

An 'infinite' loop! Why?



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:WHILE
:0→K
:While K≤10
:Disp K
:K+1→K
:End
```

What will this program do?

Notes

The <condition> is a logical expression such as X>0.

The <loop body> is any set of statements, including other loops and **If** *structures.*

It is processed whenever the <condition> is True.

The keyword **End** *is used to indicate the bottom of the <loop body>. At the* **End** *statement the program loops back to the* **While** *statement and tests the <condition> again.*

'Initialize' the **While**'s condition: establish a value so that the condition is properly established as True or False. If the initial condition is False the loop is completely skipped. If the condition is True then the loop body is processed. The **0→K** at the top of this program sets the initial condition to False. Without it there's no way of knowing what will happen because any value could have been stored in the variable **K** before the program runs.

Somewhere in the <loop body> there should be a statement that will have an effect on the <condition> so that the loop will eventually end and statements after the loop will be processed. Usually this statement is near the bottom of the <loop body>. **K+1→ K** ensures that eventually **K** will be greater than 10.

Teacher Tip: It's important to stress that the **While** loop might not be processed at all. In the next lesson we discuss the **Repeat** loop which is always processed at least once. This is a subtle but important distinction.

There are three components to building a successful While loop: ***Initialize, test, and change***:

- *Initialize* a variable.
- *Test* a condition based on that variable.
- *Change* the variable so that eventually the condition becomes false so that the loop will terminate.

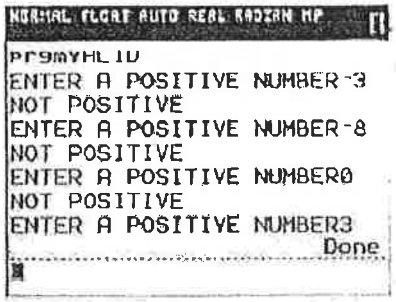
©2015 Texas Instruments Incorporated

Checking for Valid Input with While...End

We'll write a section of a program (a 'code segment') that makes sure that the user enters a positive number, tells her when that entry is invalid and to enter another value in its place.

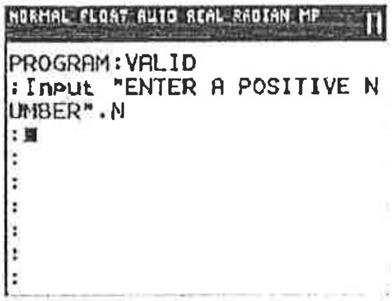
The output of the code segment is shown to the right with some non-positive numbers entered to see the effect.
See if you can write this without peeking at the next page!

Note: TI-84 Plus users may have to use a shorter message as the screen is not as wide.



```
prgmVALID
ENTER A POSITIVE NUMBER⁻3
NOT POSITIVE
ENTER A POSITIVE NUMBER⁻8
NOT POSITIVE
ENTER A POSITIVE NUMBER0
NOT POSITIVE
ENTER A POSITIVE NUMBER3
                       Done
```

We start with an **Input** statement with a message to get a value from the user…

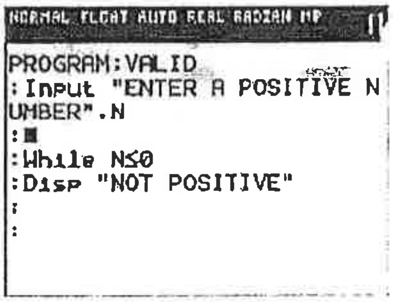


```
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:
:
:
:
:
:
```

Teacher Tip: This *initializes* N.

Then begin the **While** loop, checking to see if the value entered is negative. If it is, then we'll **Display** an error message.

The loop body will be entered only when N is negative, otherwise the entire loop body is skipped.

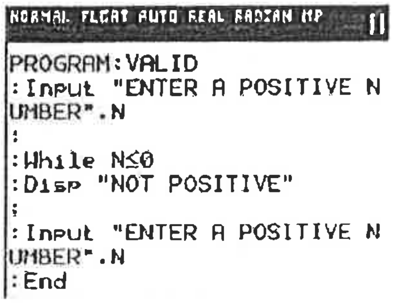


```
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:While N≤0
:Disp "NOT POSITIVE"
:
:
```

Teacher Tip: This sets up the condition to *test* involving N.

Finally, use the **Input** statement again at the bottom of the <loop body> to request another value from the user and then **End** the <loop body>.

This code segment will ask for a value from the user and make sure that the entry is a positive number. Below this code segment will be more statements to process the number entered.



```
PROGRAM:VALID
:Input "ENTER A POSITIVE N
UMBER",N
:
:While N≤0
:Disp "NOT POSITIVE"
:
:Input "ENTER A POSITIVE N
UMBER",N
:End
```

Teacher Tip: This gives an opportunity to change the value of N.

©2015 Texas Instruments Incorporated

10 Minutes of Code

t 4: Loops

Skill Builder 3: The Repeat...End loop

In this third lesson for Unit 4 you will learn about the **Repeat...End** loop. We'll compare it to the **For...** loop and even show why it is more powerful.

Objectives:

- Learn the structure and logic of the **Repeat...End** loop.
- Compare it to the **While...End** loop.
- See how **Repeat...End** is used in programming the Fibonacci sequence.

Teacher Tip: Although the Repeat condition is written at the top of the loop it is not actually tested until the bottom of the loop! A Repeat loop therefore is always executed at least once.

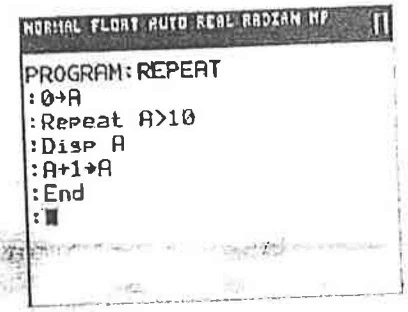
There are both While and Repeat loops in the language because they behave slightly differently.

The Repeat... End Loop

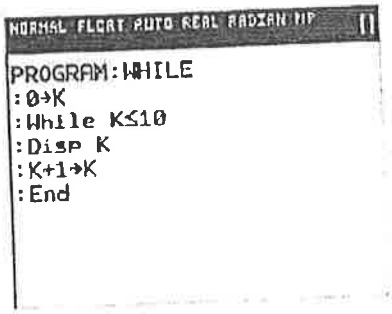
The **Repeat...End** loop will continue looping as long as its <condition> is False. This is the exact *opposite* if the **While** loop behavior and that's not the only difference! It looks like this, which looks pretty much the same as the **While** structure:

```
Repeat <condition>
  <loop body>
End
```

The programs on the right have the same output. What are the differences?



```
PROGRAM:REPEAT
:0→A
:Repeat A>10
:Disp A
:A+1→A
:End
:■
```



```
PROGRAM:WHILE
:0→K
:While K≤10
:Disp K
:K+1→K
:End
```

The <condition> is a logical expression such as **X>0.**

The <loop body> is any set of statements, including other loops and **If** structures. **It is processed once** and then continues **until** the <condition> is true so it behaves more like this:

```
Repeat
<loop body>
until <condition> is true End
```

But there's no 'until' keyword in TI Basic. It's implied. Even if the <condition> is true to start with, the loop body is still processed once because the condition is tested at the bottom of the loop.

In a **While** loop it's a great idea to 'initialize' the variable(s) that your loop relies upon. In a **Repeat** loop this will happen inside the loop body so initializing is not necessary.

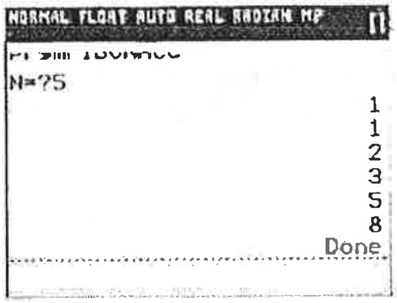
As with **While**, somewhere in the <loop body> there should be a statement that will have an effect on the <condition> so that the loop will eventually end and statements after the loop can be processed. Usually this statement is near the bottom of the <loop body>.

©2015 Texas Instruments Incorporated

Programming the Fibonacci Sequence

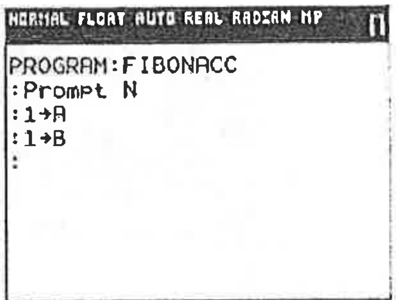
Let's write a program that displays the *Fibonacci* sequence up to a certain value. You can research the *Fibonacci* sequence if you've never heard of it.

The output of the program is shown to the right. Can you write this program without peeking below?



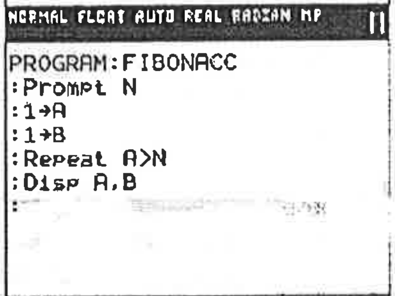
```
N=?5
                                   1
                                   1
                                   2
                                   3
                                   5
                                   8
                                Done
```

We start with a **Prompt** statement to get an upper bound value from the user. The first two Fibonacci numbers are 1 and 1 so we'll store those values in the variables **A** and **B**. These variables are going to be used to calculate the rest of the Fibonacci numbers (up to **N**).



```
PROGRAM:FIBONACC
:Prompt N
:1→A
:1→B
:
```

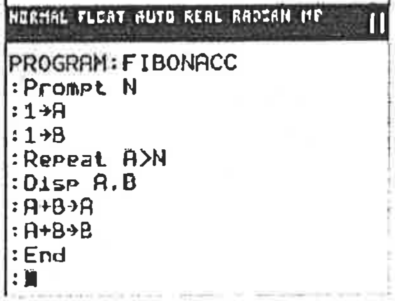
Then we begin the **Repeat** loop using the condition **A>N**, meaning '*until* A is greater than N'. In the loop we first display the current two numbers.



```
PROGRAM:FIBONACC
:Prompt N
:1→A
:1→B
:Repeat A>N
:Disp A,B
:
```

Finally, we calculate the next two Fibonacci numbers and **End** the loop.

These last two loop statements show that A+B is being stored in both A and B and it appears that A and B are getting the same value. **This is not the case!** Try it with 1 and 1. The first statement stores 1+1 in A, making A 2. The second statement stores 2+1 in B making it 3. Try it yourself - 'play computer'!



```
PROGRAM:FIBONACC
:Prompt N
:1→A
:1→B
:Repeat A>N
:Disp A,B
:A+B→A
:A+B→B
:End
:
```

Run the program with several different input values. Does it behave as planned?

Try changing the **Repeat** condition to **B>N**. What is the effect? How can we modify the program to display the 'correct' set of numbers, stopping when exactly the largest Fibonacci number less than N has been displayed?

©2015 Texas Instruments Incorporated

it 4: Loops **Application: Warning Notices**

This Application (WARNING NOTICES) makes use of loops to enter as many values as needed and, as an extension, checks for valid values entered and uses If statements to display an appropriate message.

Objectives:

- Learn about Counter and Accumulator statements.
- Use a loop in a program to get an undetermined amount of data.
- Use a 'flag' value to terminate a loop.

Teacher Tip: This project illustrates the tendency toward complexity when developing a piece of software. *Nesting* relates to the idea of putting one control structure inside another. As explained below, the OS knows which End goes with which statement.

Nested Structures

- *Nesting* is the programming technique of placing one control statement inside another. The term is derived from the idea of placing one cardboard box inside another in order to save space.
- It's important to put one *complete* structure *completely* inside a block of another in order to avoid errors.
- The program listing at the right shows an **If** structure inside the **Else** block of another **If** structure. Notice the multiple uses of the **End** statement; the computer knows which **End** belongs with which **If**.
- The indenting is for instructional purposes only. You cannot indent lines in TI-Basic.
- The program first tests to see if **A<0**. If it is, the program displays "A is negative!" and nothing else. But when **A** is not negative then the **underlined** square root calculation, another **If** statement, and the **Disp** statement are all executed.
- The programmer places **If**s inside loops and loops inside **If**s to accomplish more complex tasks as the program requires.

```
prgmSQUARE
Prompt A
If A<0
Then
    Disp "A is negative!"
Else
    √(A)→S
    If S=int(S)
    Then
      Disp "A is a perfect square."
    Else
      Disp "A is not a perfect
square."
    End
    Disp "Its square root is",S
End
```

Summary of the three loops:

For(*var, start, end*)	**While** *condition is true*	**Repeat** *until condition is true*
End	**End**	**End**

For(is used when 'counting' or processing an arithmetic sequence of values (iteration).
While is used when you might be able to skip the loop body completely.
Repeat is used when you are certain that you want the loop body to run at least once.

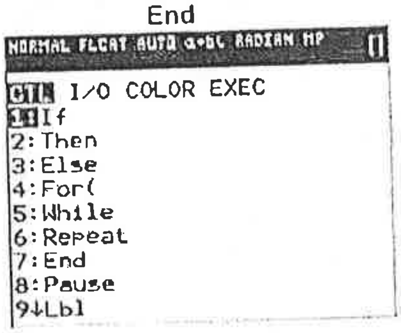
42

About End

The keyword End is used for all the multi-line control structures:

If …	**If …**	**For(…)**	**While …**	**Repeat …**
Then	**Then**			
	Else			
End	**End**	**End**	**End**	**End**

And this is why the $\frac{1}{4}$ CTL menu is arranged the way it is:



7: End comes after the first six CTL menu items because it 'ends' each of those programming control structures.

End statements can appear many times in a program, and the computer knows how the Ends are connected to their control structures.

Unit 4 Application: Program "Warning Notices"

Most schools send out regular progress reports based on a student's current average. An average of 60% or higher is considered passing, but if the average is below 70% then the student is considered 'in danger' or 'marginal'.

Let's write a program that lets the user enter some test grades, determines the average, then Outputs the number of grades entered, the average of the grades, and an appropriate warning message to the user. The warning messages can be: "Passing", "Marginal", and "Failing".

We can use two methods to enter an unknown number of grades:

- Method 1: ask for the total number of grades first and use a **For** loop to enter the scores.
- Method 2: ask for scores but use a 'flag' value such as -999 to indicate that there are no more grades. This method will use a **While** loop or a **Repeat** loop.

In both methods we will have to keep a running total of the grades. In Method 2 we also have to count the grades so that we can divide the total by that count.,

Your program should display the number of grades entered, the average of the grades and the appropriate warning message:

If the average is below 60: "Failing"

… 60 to 70: "Marginal"

… above 70: "Passing"

©2015 Texas Instruments Incorporated

10 Minutes of Code

Teacher Tip: The section below discusses two related programming ideas: a counter and an accumulator variable. Emphasize that the variable on the left side of the store operator is the same variable as the one on the right but that they represent different values due to the processing involved.

Counters and Accumulators

A statement such as **C+1→C** is called a *counter* because it adds 1 to the variable **C** each time it is executed.

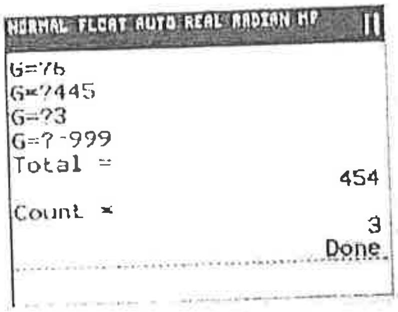
A statement such as **T+N→T** is called an *accumulator* because it keeps a running total of the values of the variable **N**. The value of **N** is added to the variable **T** and then that sum is stored back into the variable **T**. At the end of a loop **T** will contain the total of the **N** values.

Here's an example that uses a Counter and an Accumulator and a 'flag' value to keep track of the **G**'s in a program:

```
0→C
0→G
0→T
Prompt G
While G≠-999
C+1→C
T+G→T
Prompt G
End
Disp "Total =",T
Disp "Count =",C
```

Notes

initialize variables;
G is for Grade
C is for Count
T is for Total
get first Grade
as long as it is not -999
add 1 to the Count
add the Grade to the Total
ask for another Grade



```
G=?76
G=?445
G=?3
G=?-999
Total =
                  454
Count =
                    3
                 Done
```

The **While** loop above continues counting and accumulating the G's as long as -999 is not entered. When -999 is entered the loop stops and the results are displayed.

Teacher Tip: Point out the two Prompt statements in the code above. The first one gets the first grade and the last one gets the rest of the grades. The last one is at the *bottom* of the loop to prepare for going back to the While statement to test the value of G again.

Extension

As part of your input routine check to make sure that the value entered is a legitimate grade (between 0 and 100) and take appropriate action when the entered value is not legitimate.

Teacher Tip: The extension requires another loop around the input statements to make sure the value entered is legitimate. The sample code below simply stops the program when an invalid value is entered. Students can do better!

44

©2015 Texas Instruments Incorporated

Sample Answer (indenting is for instructional purposes only):

```
prgmWARNINGZ
ClrHome
0→A
0→C
0→T
Input "ENTER A GRADE:",A

While A≠999
  If A<0 or A>100
  Then
    Disp "OUT OF RANGE!"
    Stop
  Else
    C+1→C
    T+A→T
  End
  Input "ANOTHER? (OR -999):",A
End
T/C→M

ClrHome
Output(1,1,"NUMBER OF GRADES:")
Output(2,1," TOTAL OF GRADES:")
Output(4,1,"AVERAGE:")
Output(1,19,C)
Output(2,19,T)
Output(4,10,M)
If M<60
Then
  Output(5,9,"FAILING")
Else
  If M<70
  Then
    Output(5,9,"MARGINAL")
  Else
    Output(5,9,"PASSING")
  End
End
Pause
ClrHome
```

When testing a program try extreme cases such as entering -999 as the first score and entering negative values anywhere. If the program generates an error message then the programmer has not taken all scenarios into account. The program listing above will fail when -999 is entered as the first value because it will attempt to divide 0 by 0 which is undefined. The calculation of the average, M, should be wrapped in a test to make sure at least one grade has been entered. Below is a possible fix:

```
If C>0
Then
 T/C→M
Else
 Disp "NO GRADES ENTERED!"
 Stop
End
```

The **Stop** statement is used in this program to terminate a program *immediately*. This can be added anywhere in a program so that execution is interrupted and the home screen 'Done' message appears.

Unit 5: Graphics — Skill Builder 1: Drawing

… this first lesson for Unit 5 you will learn about some of Drawing commands that draw shapes on the Graph screen.

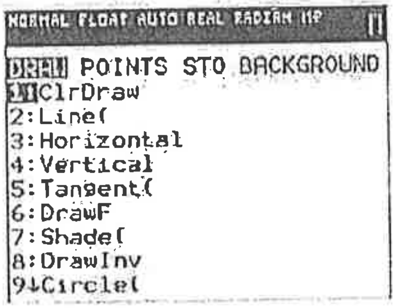
Objectives:

- Use the [draw] menu to get a drawing command.
- Learn the syntax of some of the drawing commands.
- Learn the difference between commands that use point coordinates and pixel coordinates.

The [DRAW] Menu

1. From the HOME screen press [DRAW] ([2nd] [PRGM])
2. Select **Line(**
3. Complete the command with **0,0,3,4)** so that the complete command states: **Line(0,0,3,4)**
4. Press [ENTER] to see a line drawn from the origin to the point (3, 4) on the GRAPH screen.

Most drawing commands such as **Line**, **Circle**, and **Pt-On** use the WINDOW coordinates as the frame of reference.

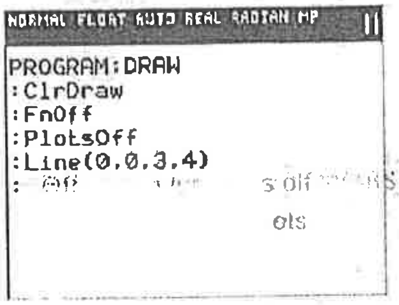


Drawing in Programs

There are many TI-Basic programming tools that affect the appearance of the GRAPH screen. Here we examine a few of them:

- **ClrDraw** to clear any drawn objects [DRAW]
- **FnOff** to turn functions off [VARS] **YVARS On/Off**
- **PlotsOff** to turn stat plots off

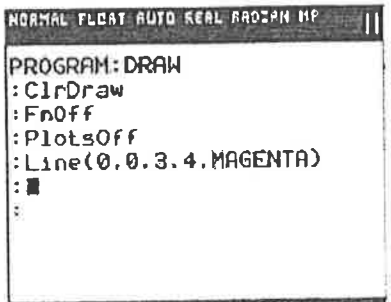
Use the [DRAW] menu to select an object to draw. See the example to the right.



```
PROGRAM:DRAW
:ClrDraw
:FnOff
:PlotsOff
:Line(0,0,3,4)
:
```

Color Options (TI-84 C and TI-84 CE only)

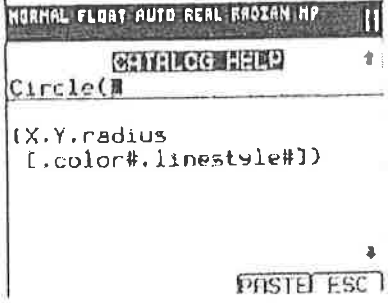
The **Line(** command has an optional fifth argument which determines the color to be used. To select a color press [PRGM] **COLOR** or [VARS] **COLOR** and select your color. The *name* of the color is inserted into your program but simply represents a number (BLUE=10, RED=11, BLACK=12, etc.). See the example to the right. Many drawing commands have a color option. On the TI-84 Plus the fourth argument can be a 1 or a 0: 1 to draw the line in black, 0 to draw it in white.



```
PROGRAM:DRAW
:ClrDraw
:FnOff
:PlotsOff
:Line(0,0,3,4,MAGENTA)
:
:
```

Help!

Help with any command in the calculator is available by pressing the [+] key while highlighting the command on any menu. To the right is the help screen for the **Circle(** command. It shows the number and order of the entries. X,Y are the coordinates of the center of the circle, then the radius. The [*optional*] entries are the color name or number and the line style (1 to 4). You can complete the command right on this screen and then press the [TRACE] key to 'PASTE' the command into your program.



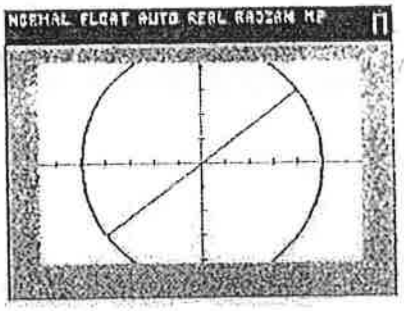
```
Circle(
(X,Y,radius
 [,color#,linestyle#])
```

Can you draw this?

Can you duplicate the drawing at the right in a program?

Hint 1: it's only two commands but the window and the numbers are important!

Hint 2: (X, Y) is the center of the circle and **radius** is the distance from the center to the circle. (color# and linestyle# are optional)

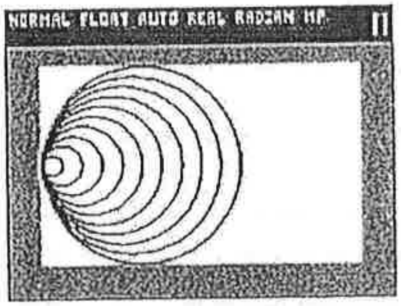


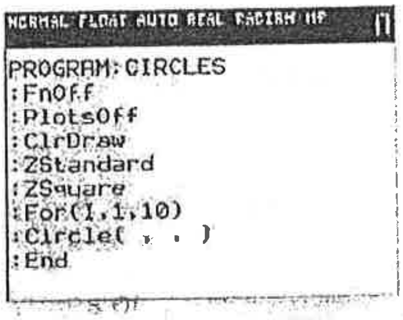
Tip: you can set the GRAPH window in a program. While editing a program press [ZOOM] and choose a setting or assign values for each window edge by assigning values to the variables found on [VARS] Window… such as:

-20→Xmin

Circle Art

Complete the Circle statement in the program below to produce the picture to the right.



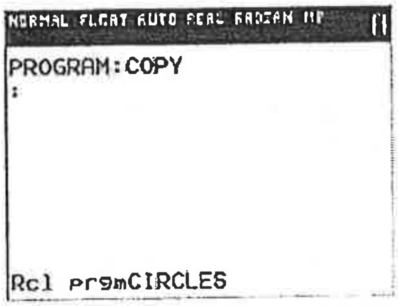


```
PROGRAM:CIRCLES
:FnOff
:PlotsOff
:ClrDraw
:ZStandard
:ZSquare
:For(I,1,10)
:Circle( , , )
:End
```

Note: **ZStandard** *and* **ZSquare** *are found on the* [ZOOM] *menu.*

Copying one program to another:

1. *Start a new program.*
2. *In the Editor press [RCL] ([2nd] [STO▸]).*
3. *Press [PRGM] and arrow to EXEC.*
4. *Select the program you want to copy. See the screen to the right where we are in the process of copying prgmCIRCLES into prgmCOPY.*
5. *Press [ENTER] to paste the code into the new program.*



```
PROGRAM:COPY
:



Rcl prgmCIRCLES
```

©2015 Texas Instruments Incorporated

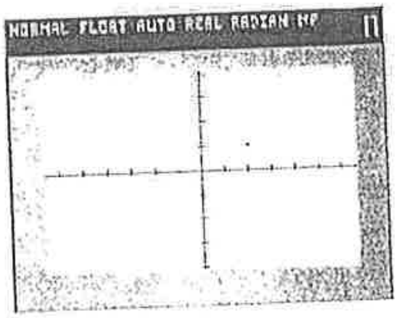
10 Minutes of Code

Skill Builder 2: Points and Pixels

In this second lesson for Unit 5 you will learn about plotting points and the difference between the Pt-On and Pxl-On commands.

Objectives:

- Use the point and pixel plotting statements.
- Develop formulas to utilize graphics in programs.



Point Command

See the [DRAW] POINTS menu.

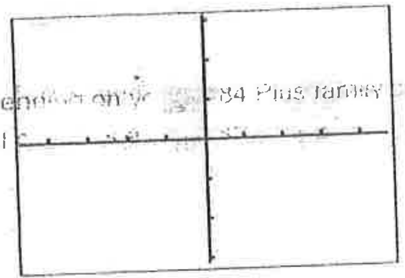
Pt-On(x,y) plots a point according to the graph WINDOW settings. It means 'turn the point on'. **Pt-On(2,1)** plots the point in the first quadrant in the screen to the right. **Pt-On(x,y,style,color)** has optional arguments: style (1 to 4) and color. See the syntax help by pressing [+] while on the command in the [DRAW] menu. **Pt-On(100,100)** will plot a point even if it's out of the current viewing window.

Pixel Command

Pxl-On(uses the screen's pixels and ignores the WINDOW settings. **Pxl-On(2,3)** plots the tiny pixel in the upper left corner of the same screen at row 2, column 3. It is hard to see! The coordinates are not in the usual (x,y) order: they're 'backwards' because they refer to (row#, column#). **Pxl-On(x,y,color)** has only an optional color argument. There are also corresponding **–Off(**, **–Change(**, and **-Test** commands that we'll not deal with here.

Pixels

Depending on your TI 84 Plus family calculator, your screen has a set number of pixel columns and rows: TI-84 Plus: 96 columns x 64 rows and TI-84 Plus C/CE: 265 columns x 165 rows. Rows are horizontal (across) and columns are vertical (up and down). Split screen settings affect the number of rows and/or columns depending on the setting. The **TI-84 Plus** graph screen does not use rightmost column or bottom row for *graphing* so that there are an odd number of points on the graph area. This ensures that there is a 'center' (origin) point. The upper left pixel is (0,0). **Pxl-On(0,0)** turns on the pixel at column 0, row 0. The rows and columns are numbered starting with zero, not one.



This TI-84 Plus graph screen has larger pixels so they are easier to see. This screen shows the result of Pxl-On(15,30). That's row 15, column 30 of the screen. The bottom right pixel is (63,95).

Teacher Tip: Pixel-related statements can be confusing because the order of the coordinates is reversed; the *y*-value (row number) is first and the *x*-value (column number) is second. Also, the *y*-direction is upside down; row 0 is at the top and row 165 (or 63 on a TI-84 Plus) is at the bottom of the screen. This is similar to the orientation of the HOME screen grid when using the **Output(** statement (line#, column#, with line 1 at the top). The **Text(** statement that draws text on the graph screen (in a later lesson) also uses pixels for positioning the text and not points.

The **–Test** statements are used as conditions in If statements or loops to see if a point or pixel is 'on'. Pt-Test(0,0) will be true when the point (0,0) is on. Note that a point can be 'on' and white so it does not appear on the screen!

48

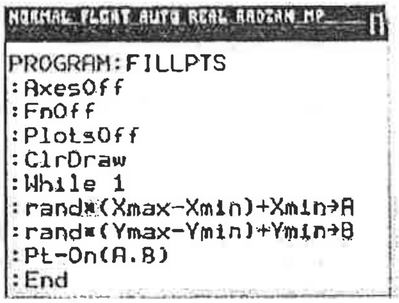
On a TI-84 Plus pixels and points are the same size. On the TI-84 Plus C/CE points are

©2015 Texas Instruments Incorporated

larger than pixels.

Programming with Points

Let's write a program that randomly fills the GRAPH screen with POINTS. This program will have an infinite loop so press [ON] to 'break' out of the program.



```
PROGRAM:FILLPTS
:AxesOff
:FnOff
:PlotsOff
:ClrDraw
:While 1
:rand*(Xmax-Xmin)+Xmin→A
:rand*(Ymax-Ymin)+Ymin→B
:Pt-On(A,B)
:End
```

This program works on all TI-84s.

We'll need an ***algorithm*** (formula) to get a random point on the current GRAPH screen within the window ranges:

rand is found on the [MATH] PROB menu and generates a random number between 0 and 1. **rand*(Xmax-Xmin)** generates a random number between 0 and Xmax-Xmin so we'll add **Xmin**.

rand*(Xmax-Xmin)+Xmin generates a random number between **Xmin** and **Xmax**.

...and we write a similar formula for the Y-coordinate.

And this is why **mathematics** is so important to programming!

Note:

AxesOff *is on the [FORMAT] screen.*

FnOff *is on the [VARS] Y-VARS On/Off... menu.*

PlotsOff *is on the [STAT PLOT] menu.*

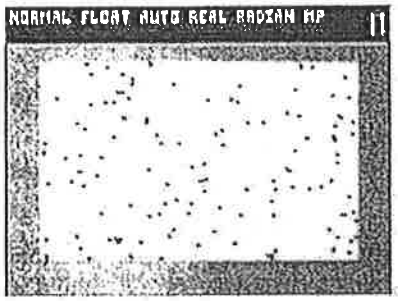
ClDraw *is on the [DRAW] menu.*

While 1 *creates an infinite loop since True is represented as* **1** *in the calculator.*

The random values are stored in **A** *and* **B** *for plotting.*

Remember to press [ON] to 'break' the program.

This program works the same on <u>any</u> TI-84 Plus Family device!



Enhancing prgmFILLPTS with Color

On the **TI-84 C/CE** you can add random COLOR to the **Pt-On(** statement. The lowest color number is 10 and the highest is 24.

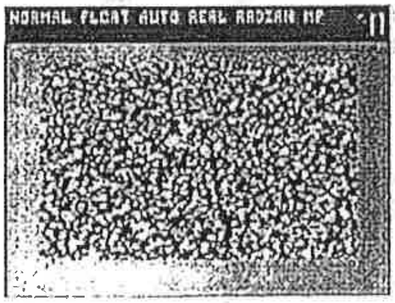
Write a statement that generates a random integer from 10 to 24 using **randInt( )** and add that as the *third* argument to the **Pt-On(** statement.

Add the following statement before **Pt-On(A,B)**. Change **Pt-On(A,B)** to **Pt-On(A,B,C)**.

<your color generator> → C

Note: There are two optional arguments to **Pt-On***: style and color. Style can be from 1 to 4 and color can be from 10 to 24. So if the third argument is a value from 1 to 9 then it's a style. If it's between 10 and 24 it's a color. Other values cause an error.*

Answer: randInt(10,24) →C



Multi-colored points

Unit 5: Graphics

Skill Builder 3: Lines, Text and Colors

... this third lesson for Unit 5 you will learn about drawing lines, text and color enhancements.

Objectives:

- Use the line, function, and text drawing statements.
- Use colors in graphics statements.
- Develop formulas to utilize graphics in programs.

Drawing Lines and Curves

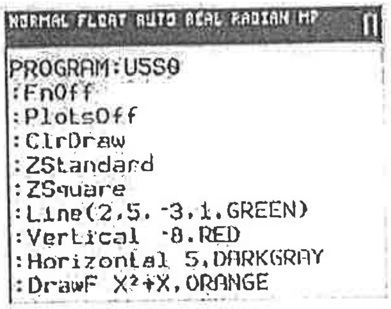
Line(X,Y,W,Z) draws a *segment* between points (X,Y) and (W,Z). See CATALOG HELP for the optional features.

Vertical A draws the vertical line X=A.

Horizontal B draws the horizontal line Y=B.

DrawF X^2+X draws the function. This is different than graphing the function.

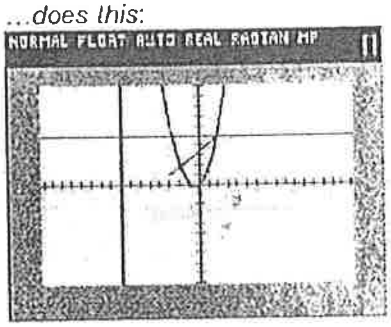
See the examples to the right. Note the optional colors found on the [PRGM] COLOR menu. Color is not available on the TI-84 Plus.



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:U5S0
:FnOff
:PlotsOff
:ClrDraw
:ZStandard
:ZSquare
:Line(2,5.-3,1.GREEN)
:Vertical -8.RED
:Horizontal 5,DARKGRAY
:DrawF X²+X,ORANGE
```

This program...

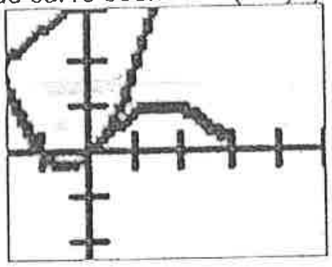
...does this:



Tip: to draw part of a function divide the function by the interval desired:

```
DrawF sin(X)/(X≥0 and X≤π)
```

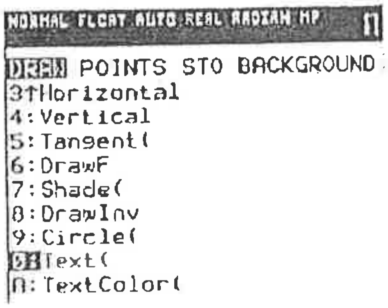
draws the blue curve seen here (Why?):



Teacher Tip: The DrawF feature is not the same as graphing a function using the [Y=] and [GRAPH] method. A function that is drawn cannot be traced or 'calculated' ([CALC]). The DrawF command only draws the pixels that the function determines. A repaint of the screen will erase any drawn function. It will not be redrawn unless the command is issued again. Notice that the Line( ) statement only draws a line segment. But Vertical and Horizontal draw complete lines from edge to edge of the screen. This is a teachable moment! A project follows that has students program a complete line rather than a segment.

Text Drawing

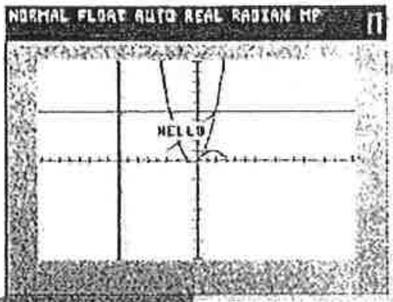
The **Text(** drawing statement is unique because it uses *pixel* values rather than window (point) values for positioning the text. There is also a separate **TextColor(** statement that sets the color of the next text drawn.



```
NORMAL FLOAT AUTO REAL RADIAN MP
DRAW POINTS STO BACKGROUND
3↑Horizontal
4:Vertical
5:Tangent(
6:DrawF
7:Shade(
8:DrawInv
9:Circle(
0:Text(
A:TextColor(
```

㊿

Text(50,100,"HELLO") displays HELLO in the same spot on the screen regardless of the WINDOW settings. Row 50, column 100 of the pixels represents *the upper left corner* of the text to be drawn.



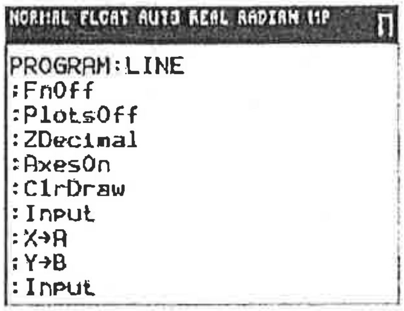
Note: Remember your screen's pixel dimensions: TI-84 Plus: 96 columns x 64 rows and TI-84 Plus C/CE: 265 columns x 165 rows.

Teacher Tip: This is an awkward statement because it is the only one of the Draw commands that utilizes *pixel* coordinates rather than *WINDOW* coordinates. An interesting project is discussed below.

Programming with Line(and Algebra

This programming activity 'enhances' the **Line(** statement.

The **Line(** statement only draws a *segment* between two points. We'd like to see a *line through* the two points and extend all the way across the screen regardless of which two points are selected. This activity makes use of Algebra concepts, so be prepared!



```
PROGRAM:LINE
:FnOff
:PlotsOff
:ZDecimal
:AxesOn
:ClrDraw
:Input
:X→A
:Y→B
:Input
```

1. Start a new program. We will call it **LINE**.
2. Add the usual graph setup statements to the beginning of the program.
3. Use two **Input** statements *without variables* to get the coordinates of two points on the screen. Input determines the values of X and Y so we need to store the first two values in other variables, **A** and **B**, so that we can get the second set of coordinates into **X** and **Y**.
4. Calculate and store the *slope* of the line.

```
:(Y-B)/(X-A)→M
```

5. Now we need the two points at the left and right sides of the screen for the line statement. The *x*-coordinates of these points are **Xmin** and **Xmax**.
6. We need to compute the *y*-coordinates.
7. The equation of the line is *y* = **M*(*x* - A)+B** (*point-slope form*).

Your Task...

1. Substitute **Xmin** and **Xmax** (the *names* not the values!) into the equation for *x* and store the results in the two variables Q and R representing the *y*-coordinates.

   Answer: M*(Xmin-A)+B → Q
   M*(Xmax-A)+B → R

2. Use the **Line(** statement to draw a line between the left and right side of the screen.

```
:Line(Xmin,Q,Xmax,R)
```

Extensions

1. Add a loop in this program to allow you to draw many lines without having to re-run the program which erases the screen.
2. This program fails when the line is vertical. Why? Incorporate an **If...** structure to handle this special case.

   Teacher Tip: Because the slope of the line is undefined. The program tries to divide by zero, causing an error.

©2015 Texas Instruments Incorporated

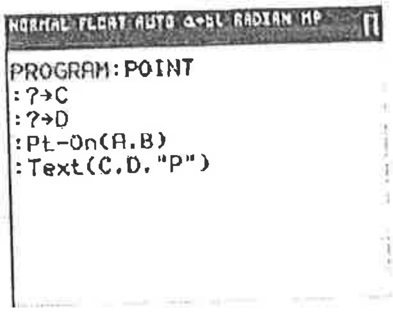
gramming Points to Pixels

Imagine this: you use the **Pt-On(** statement to draw a point (A,B) on the graph screen. You now would like to *label* the point with some text. Where do you draw the text?

Write two formulas (one for **C** and one for **D**) that convert WINDOW coordinates to pixel coordinates for the **Text(** statement. This table (TI-84 C/CE values) may help:

WINDOW	pixel
Xmin	0
Xmax	264
A	?
Ymax	0
Ymin	164
B	?

Complete the ? so that the point is labelled P:


NORMAL FLOAT AUTO a+bi RADIAN MP

```
PROGRAM:POINT
:?→C
:?→D
:Pt-On(A,B)
:Text(C,D,"P")
```

Note: remember that in the ***Text(*** *command the first argument is the ROW number which corresponds to the y-coordinate or the point!.*

Teacher Tip: This is another example of using a linear relationship. Think of (Xmin, 0) and (Xmax, 264) as two points on a line.

Then the slope of the line is

(264-0)/(Xmax-Xmin)

so the linear transformation for A is

264/(Xmax-Xmin)*(A-Xmin) → D *(equation of line)*

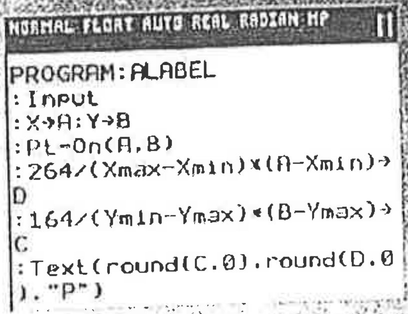
Similarly, for B we get a slope of

((164-0)/Ymin-Ymax)

so B's transformation is

164/(Ymin-Ymax)*(B-Ymax) → C *(equation of line?)*

remember that Pixel-oriented commands use the format (column#, row#, text). This is why A goes to D and B goes to C.


NORMAL FLOAT AUTO REAL RADIAN MP

```
PROGRAM:ALABEL
:Input
:X→A:Y→B
:Pt-On(A,B)
:264/(Xmax-Xmin)*(A-Xmin)→
D
:164/(Ymin-Ymax)*(B-Ymax)→
C
:Text(round(C,0),round(D,0
),"P")
```

round() is necessary because the command only allows whole numbers in the proper range. Let students discover this!

For a TI-84 Plus use 95 instead of 264 and 63 instead of 164.

Unit 5: Graphics

Application: Making a Point Bounce

In this application for Unit 5 we'll build graphics-based programs.

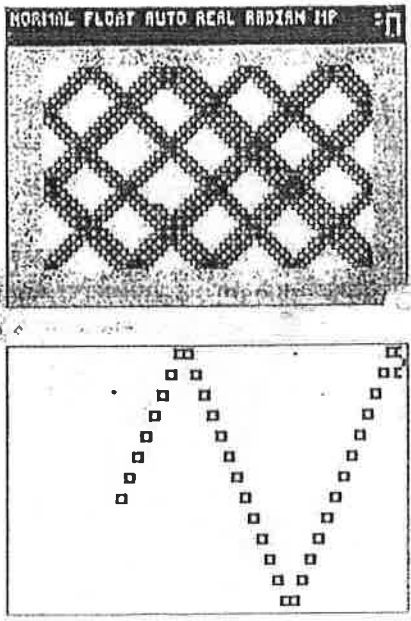
Objectives:

- Learn how to detect which device the program is running on.
- Making a point bounce around the screen.

Teacher Tip: This is a rather complex project that has some interesting physics hidden inside. The 'particle' moves by adding *delta-x* and *delta-y* values to its coordinates during each iteration of the loop. These are the horizontal and vertical components of velocity. When the particle hits an edge of the screen, the appropriate component is 'reversed' (negated) so that the particle appears to 'bounce' off the edge of the screen.

Program "Pong"

In the original video game 'Pong' a pixel bounced around on the screen. The players controlled 'paddles' as in ping-pong to keep the ball in play. This program will produce the 'bouncing ball'. A point will move in an oblique line across the screen and when it hits a boundary it will change direction to look like it's bouncing off the edge.



The same program running on two different calculators.

The first problem to consider is the screen size, since the TI-84 Plus has a different screen size than the TI-84 C/CE. The following code segment will detect the screen size:

```
0→Xmin
1→ΔX                  also on the [VARS] Window... menu
If Xmax>95 then
    <it's a C or CE>
Else
    <it's an 84 Plus>
End
```

At this point we now know the device and we set up variables in the **Then** and the **Else** blocks to use in the rest of the program: **M** for the width and **N** for the height.

For the C or CE: **264→M** and **165→N**

For the 84 Plus : **94→M** and **63→N**

Initialize Variables

We also set up the *y*-range to be nice values, too:

```
0→Ymin
1→ ΔY
```

We'll start the point **(A,B)** at a random location, but not too close to the edge of the screen:

```
randInt(10,M-10)→A
randInt(10,N-10)→B
```

We also set up two variables to represent the movement. These will be added to the point's coordinates to move the

©2015 Texas Instruments Incorporated

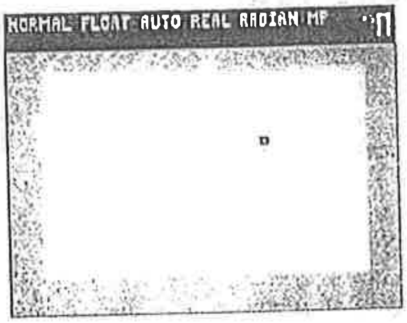
point to a new position:

```
randInt(2,5)→D        change in A
randInt(2,5)→E        change in B
```

Teacher Tip: These are the delta-x and delta-y values.

Getting Started

We're ready to start the action. We'll use an infinite loop…



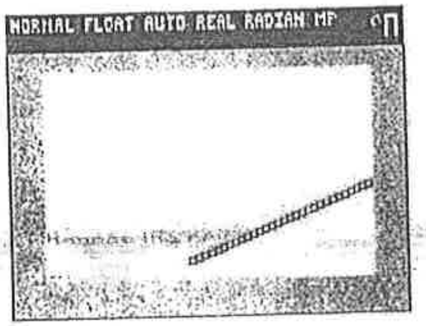
```
While 1
 Pt-On(A,B,2)          style 2 is a large square dot

   <The rest of the program>

End
```

Tip: It's a good idea to put the **End** *in at the same time to keep track of them.*

Making it Move

In the loop and after **Pt-On(** add the 'change' variables to the point's coordinates:

```
A+D→A
B+E→B
```

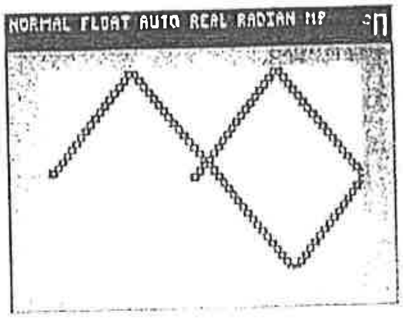


This changes the point's coordinates.
If you run the program now you will see the point go off in some direction and quickly go right off the screen as in the image to the right.

Bouncing

To get the point to detect the edges of the screen we add **If** statements…

```
If A>M or A<0
Then

<this happens when the point is off the right or left side of the screen>

End
```



Do you get this?

Two things need to happen inside the **Then**:

- + Move the point back on the screen `A-E→A`
- + Change the direction to the opposite direction `-E→E`

Write a similar If statement to handle the y-coordinate B and it's 'change' variable E.

Extensions

Don't leave a trail…

The **Pt-Off** statement will hide a point. Add a **Pt-Off** statement to your program so that the trail of points is not displayed, only the moving point. Caution: don't turn off the same point you turned on. Turn off the *previous* point (you'll need two more variables). The new code needs to go in the right place in the program.

Lose the infinite loop…

getKey ([PRGM] I/O menu) will get a value representing a key *without* pausing the program like **Prompt** and **Input**. [ENTER] has the value 105 (row 10, column 5 on the keypad).

Here's a skeleton of what needs to be done:

```
0→K
While K≠105

   <Your program here>

getKey→K
End
```

Your task is to decide where in the program these statements belong so that when you press [ENTER] the program ends.

How About This…

When you press [CLEAR] (what **getKey** value is that?) the program starts over with a clear screen and new random values and when you press [ENTER] the program ends.

©2015 Texas Instruments Incorporated

mple Answer:

```
gmPONG
FnOff
PlotsOff
AxesOff
ClrDraw
0→Xmin
1→ΔX
If Xmax>95
Then
  264→M
  165→N
Else
  94→M
  63→N
End
0→Ymin
1→ΔY

randInt(10,M-10)→A
randInt(10,N-10)→B
randInt(2,5)→D
  dInt(2,5)→E

While 1
  Pt-On(A,B,2)
  A+D→A
  B+E→B
  If A>M or A<0
  Then
    A-D→A
    -D→D
  End
  If B>N or B<0
  Then
    B-E→B
    -E→E
  End
End
```