“Calculator Programming for the Algebra Classroom”

The POWER in the Palm of Your Hand

2006

John Hanna

jhanna@ti.com

http://www.johnhanna.us

PRGM

Entering Programs

All TI-** programming keywords can be selected from *menus*. Some functions are right on the keyboard. On the 8*, the [PRGM] key itself takes on a new meaning while you edit a program. The *program control* (CTL) and *I/O* statements can be found in the [PRGM] menus. When you see a programming statement like **'Disp'** or **'DispGraph'** ***DO NOT TYPE THE STATEMENT CHARACTER BY CHARACTER.*** The lower-case letters in the command are a *clue:* the TI-8* ***do not*** have lower-case letters. Most programming statements are found on the [PRGM] key. On the TI-89 and Voyage 200, you *can* type the commands, but you *have to* match the spelling exactly, sometimes including case.

When reading programs and you see an upper-case letter all by itself or in an algebraic expression (it's a variable), type the letter (press the [ALPHA] key and then the letter). If it is a literal string like "FERMI" use alpha-lock ([2nd][ALPHA]) and type all the letters, quotation marks and spaces ([ALPHA] **0**). For lower-case letters on the TI-85/6, press [ALPHA][ALPHA].

Depending on who *typed* the program, the [STO▸] (store) function may appear typed literally or indicated by '->' or '→'. If it is a direct printout from an ASCII file from TI-Graph-Link, special characters are enclosed in backslashes, like '\->\' for the [STO▸] key. It is *always* followed by a variable. When you see one of these symbols, press the [STO▸] key. You will see an arrow (→)on the display.

Finding a function, statement, or special system variable among the myriad menus and keys on the computer can seem time-consuming and, at times, frustrating. Be patient. With practice you will become familiar with the most common features. Keep the manual handy for the more obscure ones. The **[CATALOG]** key can be useful, too.

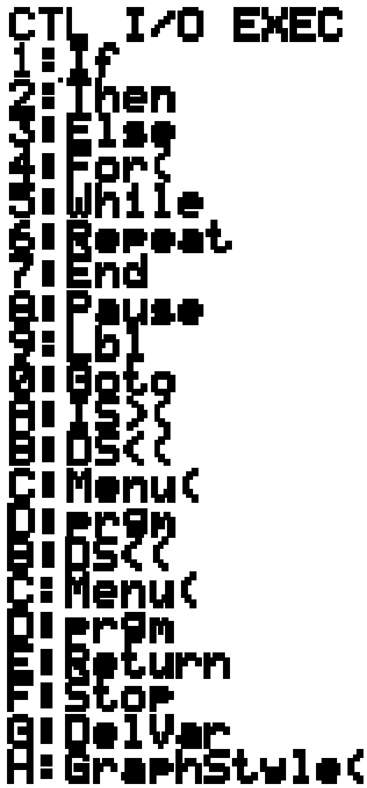
Not a complete program...

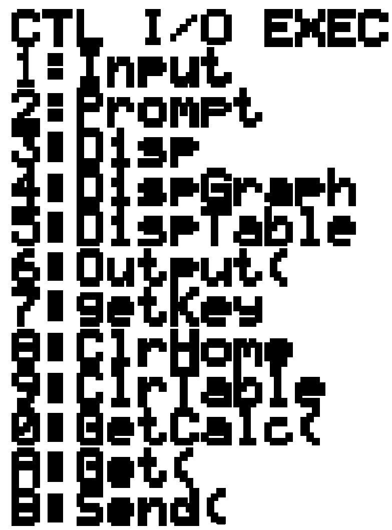
```
PROGRAM:BISECT
:If AB<0
:Then
:(B+A)/2→M
:While abs (B-A)
>.001 and Y1(M)≠
0█
:If (Y1(B)Y1(M)>
```

What can you find on the keyboard? What letters were typed? What items were selected from menus? Where is the cursor? What is '**AB'?**

PROGRAM COMMANDS: Control and **Input/Output**

These are the **TI-83/4+** programming **CONTROL** and **I/O** menus.





When editing a program, press the PRGM key. The **CTL** menu appears**.** Press ▸ to see the **I/O** menu. In addition to the 'program only' statements above, any calculator function that can be performed 'by hand' can also be entered in a program.

Variables

There are 27 'real/complex' variables on the TI-73/82/83/83+/84+. The TI-89 and Voyage 200 have an unlimited number of variables because they can be *multi-character*. On the TI-82/3/4, the letters A..Z and Θ are the real variable names. On the 89/v200, memory management makes it possible to create the same variable in different folders and to use variables that *exist only for the run of the program* (Local variables).

Other system variables that are built in to the calculators are available to programs. These variables include the window vars (xmin, xmax, etc.), matrices, lists, pictures, graph databases, y-vars, statistical variables, and *other programs*. It is up to the programmer to use a variable in its proper context. For example, you can only add two matrices or lists if they have the same dimension.

Assigning a value to a variable

Assign a value to a variable by using the Input or Prompt statement or compute a value with an expression and **store** the result in a variable:

Input "What is A?",A	Prompt A	B^2-4AC→A
display the message and wait	*display 'A=?'and wait*	*no display, no wait*

In response to an Input or Prompt statement, enter a number *or* expression.

Note: In the statement* B^2-4AC→A *the* 'A' *on the left has a different meaning than the* 'A' *on the right, although they represent the same variable.

Programming Concepts

'Control structure' is a computer programming term meaning 'how does my program flow?' <u>*All*</u> computer programs depend on *only* three control structures to make them work:

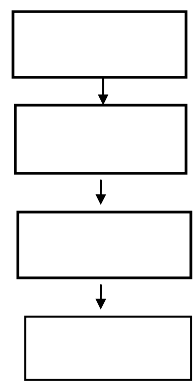
- **SEQUENCE -** The computer executes one instruction after another, one at a time, in the order that they appear in the program from top to bottom.
- **BRANCH -** The computer takes one of two or more possible routes depending on the truth value of a *condition*. (an **If..Then** statement) (Also known as 'SELECTION')
- **LOOP -** The computer repeats a certain section of statements over and over. (Also known as 'ITERATION')

These three control structures are present in all programming languages, and are implemented in several ways. Old programmers use the '**Goto..**', and '**Lbl**' statements to control flow. Structured programmers use '**For**, '**Repeat**', and '**While**' loops and the multi-line, or block, '**If..Then..Else**' statement.

The three prerequisites for computer programming are:
Know the **statements** in the language you re using, including syntax.
Know the type of **data structures** supported and the **operations** available.
Know how to design encodable **algorithms** for the problem you are trying to solve.

The SEQUENCE Structure

The PYTHAGOREAN program below asks for values for 'A' and 'B', the legs of a right triangle, then computes and displays the length of the hypotenuse.When you run this program, each statement is executed in the order that they appear in the program, top to bottom. A **SEQUENCE** structure does not contain any **Goto** statements, **If** statements or loops..

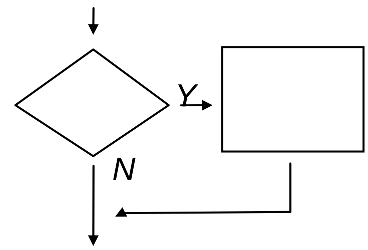


PROGRAM:PYTHAG *{for the TI-83}*
:Prompt A
:Prompt B
:$\sqrt{}(A^2+B^2) \rightarrow C$
:Disp "C=",C

Prompt and Disp are found in the PRGM I/O menu. The → is the 'STO' key. Other symbols in this program are on the keyboard.

The BRANCH Structure: IF...

There are several ways to implement the **BRANCH** structure using the '**If…**' statements, but all involve the evaluation of at least one *boolean condition*. The DISCRIMINANT program below is an example of a branch in a program. The '**If...**' statement is used to test a *condition;* if the *condition* is ***TRUE*** then the **Then** ***section is executed, otherwise the*** **Else** ***section is executed.*** Generally, **If..Then** looks like this:



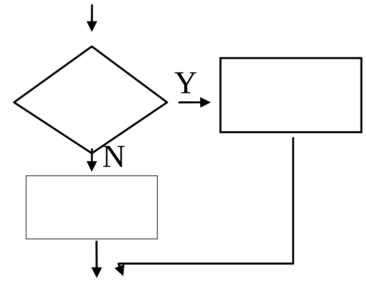

```
: {82/3/4 example}
:If (this is true)
:Then
  : (do this, otherwise, skip this)
:End
```

```
PROGRAM:DISCRIM                          {TI-82/3/4}
:ClrHome
:Disp "ENTER A"
:Input A
:Disp "ENTER B"
:Input B
:Disp "ENTER C"
:Input C
:B²-4AC → D
:If D<0                                  {the start of the 'If...' }
  :Then
    :Disp "ROOTS ARE COMPLEX"            {the 'true' action}
  :Else
    :Disp "ROOTS ARE REAL"     {the 'false' action}
:End                                     {the end of the 'If' structure}
```

TI-8* programs usually do not have descriptive instructions. There is not enough memory! The user is always the programmer herself, so she is supposed to know how to use the program!

In the TI-8*, there are three distinct 'If...' structures:

81/2/3/4	82/3/4	82/3/4
:If (*condition*) :(*then*) *{the 81 had only <u>this</u> If statement}*	**:If** (*condition*) **:Then** (*place as many statements as you like in here*) **:End** *	**:If** (*condition*) **:Then** (*true block*) :**Else** (*false block*) **:End** *



The calculators allow multiple statements on a line, so an '**If..**' *could* look like this:

```
:If D<0:Disp "D IS NEGATIVE"
```

You must *type* the colon to separate the two statements from each other.

* The **:End** statement does not mean the *END* of the program, but rather the *END* of the :**Then** block of statements and the *END* of the :**Else** block of statements.

The LOOP Structures: While, Repeat, and For(

A *loop* is a block of statements that is executed more than once. Infinite loops are allowed. In the following example, FIBONACCI numbers are displayed two at a time. Press **[Enter]** to see the next two numbers, press **ON** to terminate the program.

```
PROGRAM:FIBONACC    { displays the Fibonacci sequence}
:1 → A
:1 → B
:While B>0          {the beginning of an 'infinite' loop}
  :Disp A
  :Disp B
  :Pause            {press Enter to display the next two}
  :A+B → A
  :A+B → B
:End                {end of the while loop, not the program}
```

The following example uses a **'For('** loop. The **'For('** statement takes four arguments: the *loop control variable*, the *initial value*, the *terminating value*, and the *loop increment*. Program **ZFITY1** simulates the TI-85 Zoom FIT feature on the TI-82 (for Y1 only). The loop scans all pixel values from **Xmin** to **Xmax** to determine the largest and smallest values of Y1 in that domain and sets **Ymax** to the largest and **Ymin** to the smallest. This way, the function will 'fit' on the screen. Use this program *after* setting **Xmin** and **Xmax**. The variable ΔX is on the VARS/Window menu.

```
PROGRAM:ZOOMFIT              {a 'ZOOM-FIT' for the TI-82}
:Xmin → X
:Y1 → Ymin
:Y1 → Ymax
:For(X,Xmin,Xmax,ΔX)         {begin a FOR.. Loop}
  :If Y1<Ymin : Y1→Ymin      {the ':' is on the keyboard}
  :If Y1>Ymax : Y1→Ymax
:End                         { End of the FOR.. Loop}
:DispGraph
```

The '**:End**' statement above indicates the end of the **':For'** block of statements, not the end of the program.

LOOPS (continued)

Two confusing statements that began on the TI-81 (that are 'obsolete', but are included on all other models for compatibility) are used to create incrementing or decrementing loops. **IS>(** and **DS<(** look intimidating but helped to save a little space in the TI-81. These two statements can be avoided now by using the **For(** loop instead. These three programs do the same thing...

```
PrgmX:COUNT1
:1 → A
:Lbl 1
  :Disp A
  :A + 1 → A
  :If A<10
 :Goto 1
 : {more code}
```

```
PrgmY:COUNT2
:1 → A
:Lbl 1
 :Disp A
:IS>(A,10)
:Goto 1
: {more code}
```

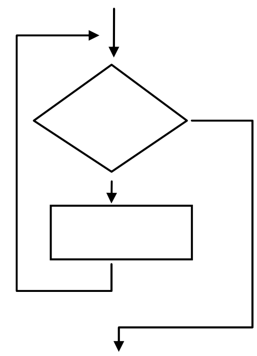
```
Program:COUNT3 (all calcs)
:For(A,1,10,1)
  :Disp A
:End        {of the For..}
: {more code}
```

IS>(A,10) means "***I***ncrement A and ***S***kip the next statement if A > 10".
DS<(   ) means "***D***ecrement   and ***S***kip   ... if ... ***is less than*** ...
As you can see from the three examples above, using '**IS>(**' combines two steps into one. When you only have 2400 bytes of memory, every little saving helps. The third example, PrgmCOUNT3 (82 & 85), shows the power of the **'For('** structure, combining 7 steps into 3. The **IS>(** and **DS<(** statements were leftovers from the TI-81. ***AVOID THEM!***

'**Repeat**' and '**While**' loops are similar to one another. They differ in that their continued execution depends upon *opposite* boolean values. I prefer '**While**'.

:Repeat (*until this is true*)
(loop body)
:End

:While (*this is true*)
(loop body)
:End



Examples:

```
:0→J
:Repeat {until} J > 0
  :Disp "POSITIVE NUMBER"
  :Input J
:End
```

```
:0→J
:While J ≤ 0
  :Disp "POSITIVE NUMBER"
  :Input J
:End
```

The **'Repeat'** loop ends when the condition **J>0** becomes *true*. The *{until}* is not typed into the program.

The **'While'** loop ends when the condition **J≤0** becomes *false*.

I prefer **While** because you have control over whether the loop gets executed at all, as opposed to the **Repeat**, which is executed at least once.

GRAPHING IN PROGRAMS

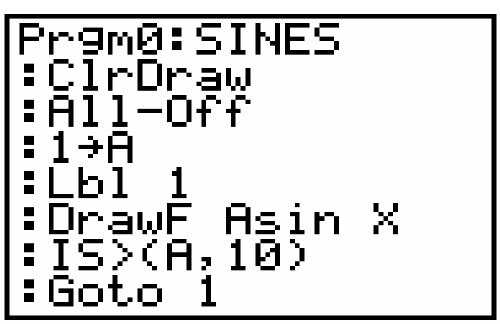
You can have a program enter the Y= functions and graph them. Put the function in quotation marks and use STO to store it in a **Y-VAR**iable. The function is stored literally as it appears inside the quotation marks in the program. The program does *not* replace the variables A, B, and C with their values.

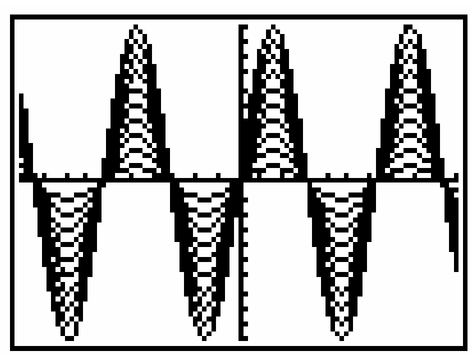
:PROGRAM:PARABOLA **for 82/3/4**
:"Ax^2 + Bx + C" → Y_1 {or **Prompt Y1 ,** see below}
:ClrHome
:Prompt A
:Prompt B
:Prompt C
:FnOff
:FnOn 1
:ClrDraw {erases the graph screen}
:Trace {press the TRACE key for this command}

You can set all the MODEs and WINDOW FORMATs in a program.
You can **DrawF**unctions in a program, which means that you can sketch many more than four, ten, or ninety-nine functions (but cannot TRACE them). You can ***Trace*** on a graph *within* a program to get *x* and *y* values and then press to continue with the program.

On the 83/84 you can prompt the user for a string, then store the function in one of the y-variables using the automatic 'string to equation' feature:

:Input " Y_1=",Str1
:Str1→ Y_1



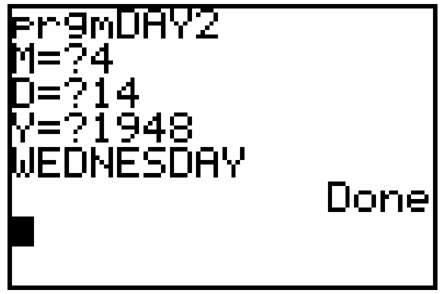


This *is* a complete TI-81 program. Can you do better?

Some 73/82/83/84 Programs

{determine the day of the week for a given date}

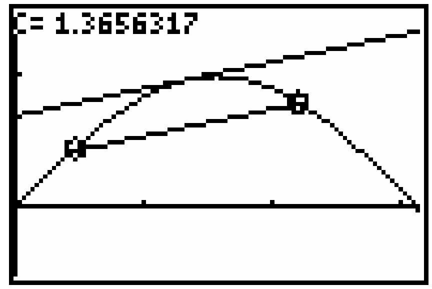
```
:Prompt M,D,Y
:Y/4→Q
:If M=1 and fPart Q=Ø:5→N
:If (M=1 and fPart Q≠Ø) or M=1Ø:6→N
:If M=2 and fPart Q=Ø:1→N
:If (M=2 and fPart Q≠Ø) or M=3 or M=11:2→N
:If M=4 or M=7:5→N
:If M=5:7→N
:If M=6:3→N
:If M=8:1→N
:If M=9 or M=12:4→N
:(1.25Y+N+D)/7→N
:7.1fPart N→N
:iPart N→N
:If N=Ø:Disp "SATURDAY"
:If N=1:Disp "SUNDAY"
:If N=2:Disp "MONDAY"
:If N=3:Disp "TUESDAY"
:If N=4:Disp "WEDNESDAY"
:If N=5:Disp "THURSDAY"
:If N=6:Disp "FRIDAY"
```



{Generate a *list* of prime numbers}

```
PROGRAM:PRIMES
:1→dim L6
:2→L6(1)
:Disp 2
:For(N,3,211,2)
  :1→P
  :1→I
  :While P and (I≤dim L6)
    :1-(N/L6(I)=int (N/L6(I))→P
    :I+1→I
  :End
  :If P
  :Then
    :Disp N
    :N→L6(dim L6+1)
  :End
:End
```

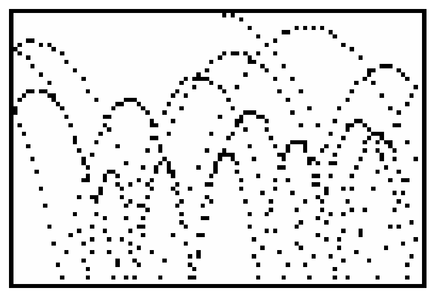
{the Mean Value Theorem}



Y1=sin(x) in [0,pi]x[0.1.5] ProgramMVT shows the chord joining two selected points, a tangent parallel to the chord, and a value of 'C' that satisfies the MVT.

```
PROGRAM:MVT
:ClrDraw
:Trace
:X→A
:Y1(A)→C
:Circle(A,C,2ΔX)
:Trace
:X→B
:Y1(B)→D
:Circle(B,D,2ΔX)
:Line(A,C,B,D)
:(D-C)/(B-A)→M
:solve(nDeriv(Y1,X,X)-M,X,(A+B)/2)→C
:Circle(C,Y1(C),2ΔX)
:DrawF M(X-C)+Y1(C)
:Text(Ø,Ø,"C=")
:Text(Ø,1Ø,C)
```

{ simulate a bouncing ball}



E= elasticity
V= horizontal velocity
A= acceleration constant
U= vertical velocity
Press [Enter] (key 11) to restart

```
PROGRAM:BALLFALL
:FnOff
:AxesOff
:Ø→K
:While K≠11
:ClrDraw
:47→X
:Ø→Y
:6rand-3→V
:rand→E
:rand→A
:.1→U
:getKey→K
:Ø→K
:While K≠1Ø5 and abs (V)>.1
:X+V→X
:U+A→U
:Y+U→Y
:If X>94
:Then
```

{continued next column}

```
:94→X
:⁻V→V
:End
:If X<Ø
:Then
:Ø→X
:⁻V→V
:End
:If Y≥62
:Then
:62→Y
:⁻UE→U
:.9V→V
:End
:Pxl-On(int Y,int X)
:getKey→K
:End
:ClrHome
:End
```

A game

BAGELS is an exercise in introductory computer science courses.

The play...

Computer picks a random three digit number, no two of which are the same. The leading digit could be zero. Human then has to guess the number. If one of the digits in human's guess is correct but in the wrong position (place), then computer displays the name 'PICO'. It could display up to three 'PICO's, depending on how many of the digits are correct. If one of human's digits is correct *and* in the correct position (place) then computer displays the name 'FERMI'. If three 'FERMI's are displayed then human has guessed the number. The computer displays the number of guesses it took to get the number and then pauses. The human can then choose (from a MENU) to either play again or quit.

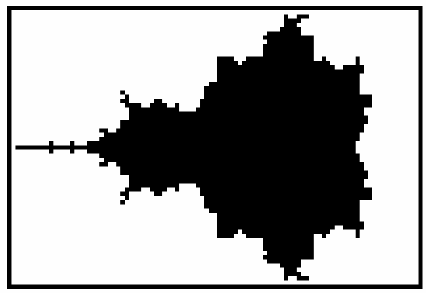
The color game of 'Mastermind' is based on 'Bagels'!

```
PrgmBAGELS                    {for the TI-82/3/4}
:Lbl 1                        {start of each game; see MENU( at the bottom}
:ClrHome                            {on the TI-85 use 'ClLCD' on the I/O menu}
:0→A:0→B:0→C
:Repeat (A≠B)(A≠C)(B≠C)       {repeat loop until all three digits are different}
  :int(10*rand)→A
  :int(10*rand)→B
  :int(10*rand)→C
:End                                {end of the repeat loop}
:100A + 10B + C → N       {N is computer's number}
:0→I                                {I is human's guess}
:0→F                                {F is the guess-counter variable}
:Repeat N=I                   {repeat until guessed (N=I)}
  :1 + F → F
  :Input I
  :int(I/100) → J         {Human hundred's digit}
  :10*fpart(I/10) → L         {One's digit}
  :(I-100J-L)/10 → K          {Ten's digit}
  :If A=J : Disp "FERMI"
  :If B=K : Disp "FERMI"
  :If C=L : Disp "FERMI"
  :If A=K : Disp "PICO"
  :If A=L : Disp "PICO"
  :If B=J : Disp "PICO"
  :If B=L : Disp "PICO
  :If C=J : Disp "PICO"
  :If C=K : Disp "PICO"
:End                                {end of the main game loop}
:Disp "YOU GOT IT IN",F
:Pause                        {press ENTER to continue after it pauses}
:Menu("BAGELS","ONE MORE TIME",1,"THATS ALL FOR NOW",2)
:Lbl 2
```

{The MANDELBROT SET}

```
:FnOff
:AxesOff
:-1.4 → Xmin
:.7 → Xmax
:Ø →Xscl
:-1→Ymin
:1→Ymax
:Ø→Yscl
:For(X,Xmin,Xmax,Δx)
  :For(Y,Ymin,Ymax,Δy)
    :Ø→A
    :Ø→B
    :Ø→D
    :1→I
    :While (I≤5Ø) and (D≤4)
      :A^2-B^2+X→T
      :2AB+Y→B
      :T→A
      :A²+B²→D
      :1+I→I
    :End
    :If D≤4
    :Pt-On(X,Y)
  :End
:End
:StorePic Pic4
```

This program approximates the Mandelbrot set.
For greater flexibility, remove the WINDOW settings in the program and set them by hand using the WINDOW key on the calculator. In this way, you can ZOOM in on any section of the set. You can even use the ZOOM BOX feature to define a new viewing window, and then re-run this program.
To speed up the program, change the value '50' in the WHILE statement to a smaller number. This reduces resolution, but speeds up the program

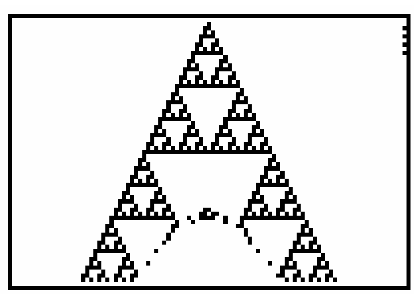


MANDELBROT [-2,1]x[-1,1]

Pascal's Triangle mod K...

```
:ClrHome
:Disp "PASCALS TRIANGLE"
:Disp "MOD K..."
:Prompt K
:AxesOff
:FnOff
:ClrDraw
:For(R,Ø,62,1)
  :For(C,Ø,R/2,1)
    :If fPart ((R nCr C)/K)≠Ø
      :Then
        :Pxl-On(R,C)
        :Pxl-On(R,R-C)
      :End
  :End
:End
```

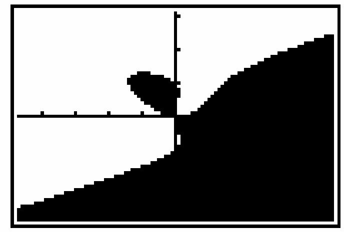
This is an interesting generation of 'Sierpinski-like' patterns. When K=2, triangles with perfect-number-of-pixels appear in the picture! Distortion at the bottom is due to overflow errors.



Pascal's Triangle MOD 2

GRAPHING IMPLICIT RELATIONS and INEQUALITIES

To graph an implicit relation, like '$y^5 + 2xy - x^3 - y^3 = 0$' change it to '...$< 0$' and look at the edge of the shaded region.



Enter *any* relation between x and y into Y1...

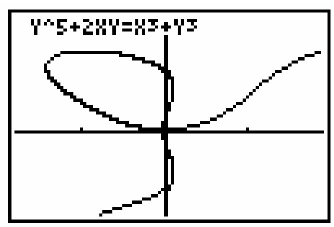
Y1= y^5 + 2xy - x^3 - y^3 < 0

Set the viewing window by hand.
Now run the following short program...

```
FnOff                      ; so not to graph the function
DispGraph                  ; watch the graph screen anyway
For(X,Xmin,Xmax,Δx)        ; for each...
  For(Y,Ymin,Ymax,Δy)      ;          ...pixel on the screen
    If Y1:Pt-On(X,Y)       ; if it satisfies the inequality, plot it
  End                      ; of For(y...
End                        ; of For(x...
```

OR...

enter this into Y1= Y^5 +2xy - x^3 - y^3
then run *this* program, which graphs only the *edge* of the shaded region above.:




```
FnOff
DispGraph
For(X,Xmin,Xmax,ΔX)
   0 → T
   For (Y,Ymin,Ymax,ΔY)
     If TY1 < 0: Pt-On(x,y)
     Y1 → T
   End
End
For(Y,Ymin,Ymax,ΔY)        ;two passes are necessary so that 'vertical' parts of the
   0 → T                   ; graph will be filled in.
   For (X,Xmin,Xmax,ΔX)
     If TY1 < 0: Pt-On(X,Y)
     Y1 → T
   End
End
```

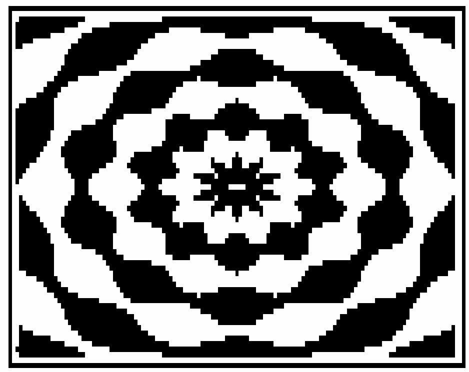
Xsin(2Y)=Ycos(2x) in [0,10]x[0,10]

Graphing “Polar Inequalities”

Inspired by the ‘graphing calculator’ on that other computer, ‘Polar Inequalities’ are expressions involving *any* relationship between R and θ, the polar variables. Polar inqualities generate surprisingly complex pictures. Feel free to explore your own concoctions!

enter $Y_1 = \sin(6R + 10\theta) < -0.3$

then run this program:



```
Program:POLINEQ
    For(X,Xmin,Xmax,Δx)
     For(Y,Ymin,Ymax,Δy)
       √(X^2+Y^2) → R
       π/2 → θ
       If X≠0:tan⁻¹ (Y/X) → θ
       If Y₁: Pt-On(X,Y)
      End
     End
```

Here are a few more interesting polar pictures:

$\cos(6\theta) < \sin(R)$

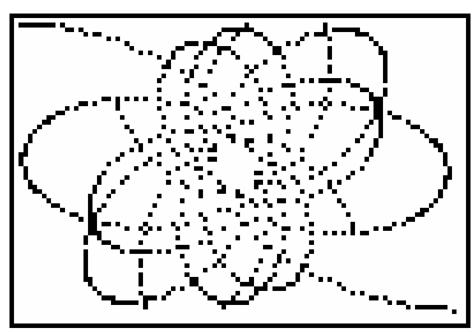
$\cos(10\theta) < \tan(\sin(2R))$

$\cos(10\theta) < \tan(R\sin(2R))$

Universal Gravitation

Simulates the paths of two ‘points’ affected only by the gravitational attraction between them. An interesting physics demonstration.

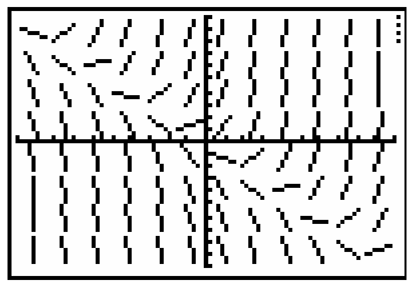
```
Program:GRAVITY
FnOff
AxesOff
ClrDraw
Input "MASS 1:",A
Input "MASS 2:",B
Input "X1:",D
Input "Y1:",C
Input "X2:",F
Input "Y2:",E
Input "DX1:",G
Input "DY1:",H
Input "DX2:",I
Input "DY2:",J
While getKey=Ø
  If C≥Ø and D≥Ø and C≤62 and D≤94:Pxl-On(int(C), int(D))
  If E≥Ø and F≥Ø and E≤62 and F≤94:Pxl-On(int(E), int(F))
  (E-C)²+(F-D)²→R
  B/R→K
  A/R→L
  G+K(F-D)/√(R→G
  H+K(E-C)/√(R→H
  I+L(D-F)/√(R→I
  J+L(C-E)/√(R→J
  D+G→D
  C+H→C
  F+I→F
  E+J→E
End
```



A Calculus Pair…

The “Slope Field” Program

An important topic in Calculus, the slope field illustrates the direction of a function at points in the plane by drawing representative tangents at various places. Use this program with “FNINTY1” below to generate graphical solutions to differential equations. Enter a differential equation (in terms of x and y) into Y1 and run SLOPEFLD.
(example, for *dy/dx* = *x*+*y*, enter Y1 = X + Y)

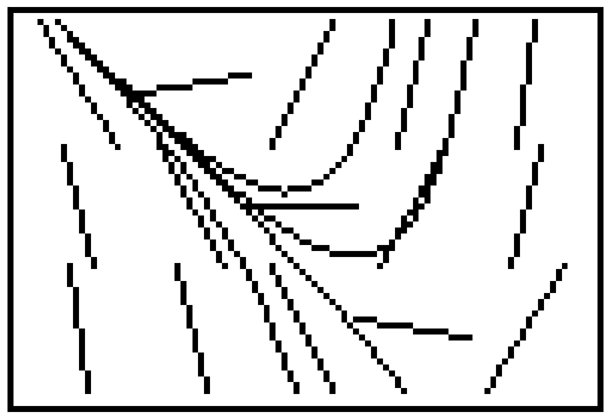


```
Program:SLOPEFLD
Func:ClrHome
Disp "BE SURE TO"
Disp "ENTER DY/DX="
Disp "F(X,Y) IN"
Disp "TERMS OF X"
Disp " AND Y IN "
Disp " THE Y= EDITOR"
Disp " PRESS ENTER"
Disp "TO BEGIN"
Pause
Lbl 1
ClrHome
Input "NO. X-VALUES ",W
Input "NO. Y-VALUES ",L
(Ymax-Ymin)/L→V
(Xmax-Xmin)/W→H
ClrDraw
FnOff
Ymin+V/2→Y
For(R,1,L)
  Xmin+H/2+.ØØØØØ1→X
  For(C,1,W)
    Y1→M
    ⁻M*H/2+Y→S
    M*H/2+Y→T
    If abs((T-S)>V
    Then
      Y+V/2→T
      Y-V/2→S
      (T-Y)/M+X→Q
      (S-Y)/M+X→P
    Else
      X-H/2→P
      X+H/2→Q
    End
    Y→F
    Line(P,S,Q,T)
    F→Y
    X+H→X
  End
  Y+V→Y
End
```

…and FNINTY1...

Enter your integrand (a dif-eq in *x* and *y, same as in SLOPEFLD above*) in Y1, set an appropriate viewing window, then run this program that uses *Euler's Method* to draw an antiderivative. From the menu, choose a 'speed factor': 1, 2, or 3 (higher is faster). After the function is graphed, Trace is activated to choose a starting point for integration. Select a point and press [ENTER]. An indefinite integral is plotted, then back to Trace to select another starting point. Press [ON] to break out of the "infinite loop" caused by the statement *While 1*.

```
Program:FNINTY1
      :MENU ("CHOOSE ΔX INC.","1ΔX",1,"2ΔX",2,"3ΔX",3)
      :LbL 1: 1→D: Goto 4
      :Lbl 2: 2→D: Goto 4
      :Lbl 3: 3→D
      :Lbl 4
      :ClrDraw
      :While 1
        :Trace
        :X→A:Y→B
        :X→J:Y→K
        :For(X,A,Xmin,⁻DΔX)
          :Line(J,K,X,Y)
          :X→J:Y→K
          :Y-DY₁ΔX→Y
        :End
        :Ø→Y:A→J:B→Y:Y→K
        :For(X,A,Xmax,DΔX)
          :Line(J,K,X,Y)
          :X→J:Y→K
          :DY₁ΔX+Y→Y
        :End
      :End
```

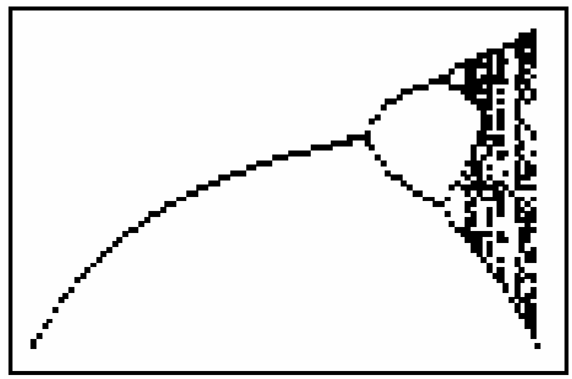


SLOPEFLD and FNINTY1 at work

Another Fractal

This 82/83/84 program produces the **Feigenbaum diagram** illustrating the transition from order to chaos and the intermittent bursts of order on the way! It will work on the TI-73, 82, 83, and 84 and the same structure will work on all the other calculators. First, set your viewing window to [1,4]x[0,1].

```
PROGRAM:FEIGEN
FnOff
PlotsOff
ClrDraw
For(a,xmin,xmax,Δx)
   Ø.5→x
   For(i,1,4Ø,1)
      a*x*(1-x)→x
   EndFor
   For(i,1,8Ø,1)
      a*x*(1-x)→x
      PtOn(a,x)
   EndFor
EndFor
```



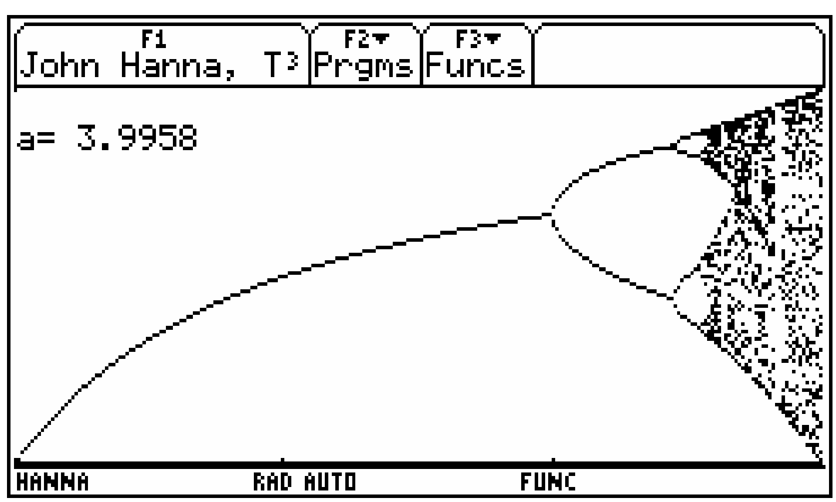
Notes on the program:

- The **For(a** loop goes across the screen and uses these as values for *a* in the expression *ax(1-x)*. Values outside [1,4] cause errors. Why?
- 0.5 is the initial value of the sequence. Does it matter?
- The first **For(i** loop computes the first 40 terms of the sequence U(n) = A * U(n-1)(1-U(n-1)) with U(0) = 0.5.
- The second **For(i** loop computes the next 80 terms *and* plots their values on the screen.

For more information on Period-Doubling and Chaos, see Peitgen, Jurgens, Saupe, **Fractals for the Classroom**, vol. 2, ch. 11, Springer-Verlag, 1992, available from NCTM.

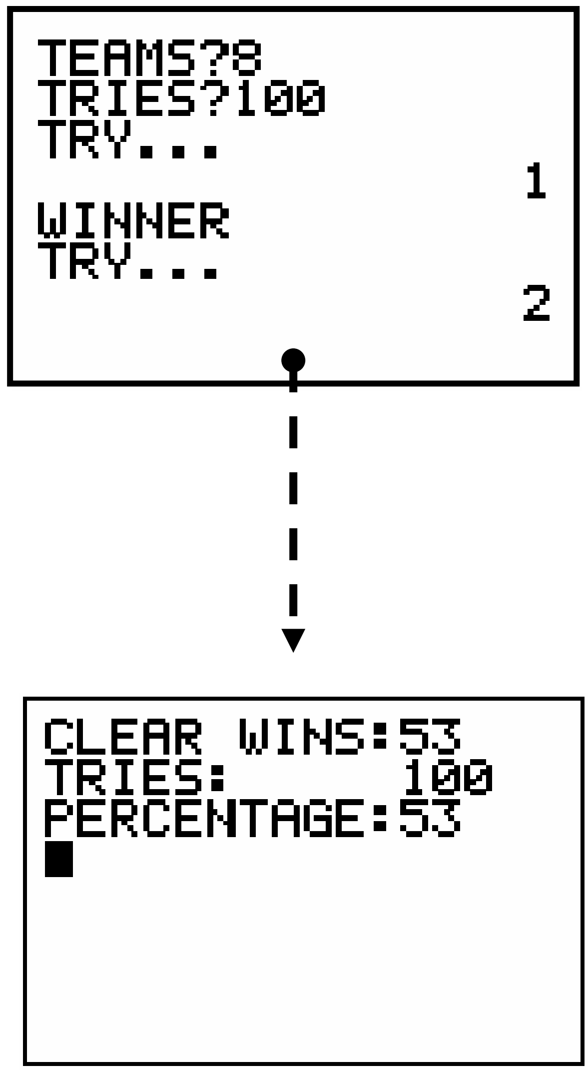
The TI-89/92/V200 feigen() program: note the similarities!

```
()
Prgm
1→Xmin:4→Xmax
Ø→Ymin:1→Ymax
FnOff :PlotsOff
For A,1,4,Δx
  .5→X
  For I,1,4Ø
    A*X*(1-X)→X
  EndFor
  For I,1,1Ø
    A*X*(1-X)→X
    PtOn A,X
  EndFor
EndFor
EndPrgm
```



A Round Robin Tournament

In a round robin tournament, each team plays every other team exactly once. If there are 8 teams in the tournament, then each team plays seven games. At the end of the tournament, the team with the most wins is the tournament champion. It’s easy to see that there is often no clear winner: two or more teams can tie with the most number of wins. Ann wanted a program that would simulate a large number (T in the program) of 8-team tournaments and calculate the percentage of tournaments in which there was a clear winner at the end. This program makes extensive use of a list and illustrates the nesting of control structures.




```
ClrHome
     Input "TEAMS?",N
     ClrList L1
     N→dim(L1)
     Ø→W
     Input "TRIES?",T
     For(K,1,T,1)
       Disp "TRY...",K
       ClrList L1
       N→dim(L1)
       For(I,1,N-1,1)
         For(J,I+1,N,1)
           If randInt(1,2)=1
           Then
             L1(I)+1→L1(I)
           Else
             L1(J)+1→L1(J)
           End
         End
       End
       Ø→C
       For(I,1,N,1)
         If L1(I)=max(L1)
         Then
           C+1→C
         End
       End
       If C=1
       Then
         W+1→W
         Disp "WINNER"
       End
     End
     ClrHome
     Disp "CLEAR WINS:","TRIES:","PERCENTAGE:"
     Output(1,12,W)
     Output(2,12,T)
     Output(3,12,W/T*1ØØ)
```

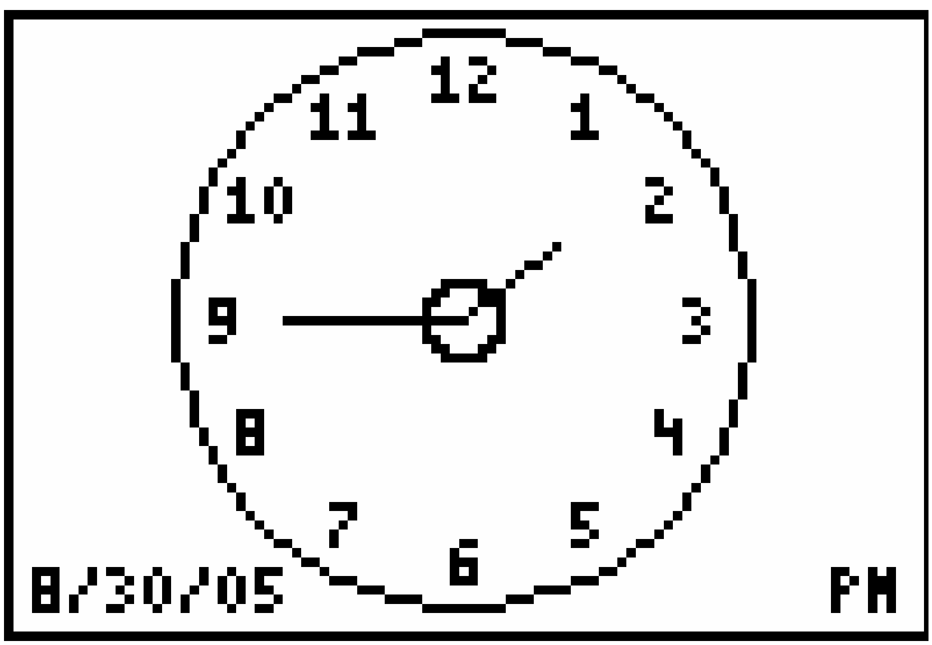
New for 2006

A simple timer for the TI-84+

The Ti-84+ has a clock chip onboard and several clock functions available to the programmer. We can use the built-in `checkTmr()` function to make a simple stopwatch…

```
PROGRAM: TIMERØØ1
:ClrHome
:Disp “PRESS ENTER TO START”
:PAUSE
:checkTmr(Ø)→A
:Disp “PRESS ENTER TO STOP”
:PAUSE
:checkTmr(A)→A
:Disp A
```

But, of course, this is just the beginning. We can do much better….



By Juan Gonzalez and John Hanna, 2005