# Programming Your Calculator

## Casio fx-7400G PLUS

Barry Kissane

*Programming Your Calculator:*
*Casio fx-7400G PLUS*

Published by Shriro Australia Pty Limited
72-74 Gibbes Street, Chatswood NSW 2067, Australia

A.C.N. 002 386 129

Telephone: 02 9370 9277
Facsimile: 02 9417 8957
Email: casio.edusupport@shriro.com.au
Internet: http://www.school.casio.com.au

ISBN 0 9758302 0 3

# INTRODUCTION

The Casio fx-7400G PLUS does not have all the capabilities of larger, more sophisticated (and considerably more expensive) graphics calculators. This booklet contains examples of programs that you can use to extend its capabilities in various ways.

Some of the programs allow the calculator to undertake tasks usually available only on more sophisticated calculators (such as solving equations). Some programs allow you to explore mathematical ideas in new ways, such as those involving generating random data. Some others provide mathematical functions not usually available on calculators (such as finding factors of a number).

A calculator program is a set of commands to be carried out by the calculator in a particular sequence. There are special programming commands available to direct the calculator, although most of the commands are those with which you are already familiar.

It is not necessary to know how to *write* a program in order to use a program. All that you need to do is to faithfully enter into your calculator a program written by someone else, and to know how and when to use the program.

The *User's Guide* provides detailed information on programming, as well as some other programs, if you would like to know more about this.

## Entering a Program

The programs in this book can be entered into your calculator in at least three different ways.

Provided you have the right calculator accessories and Internet access, you can download all the programs from the Australian Casio Education Site (ACES), with the URL http://www.school.casio.com.au and then transfer them directly to your calculator through the communications port. You will need the FA-122 or FA-123 kit for your computer to do this. Refer to Chapter 9 of the *User's Guide*.

A second way is to transfer the programs from another calculator that already has them entered. To do this, you will need the special SB-62 cable, or an equivalent cable. In this case, you use the communications port of your calculator, and use Link Mode for each calculator. Refer to Chapter 9 of the *User's Guide*.

A third way is to enter a program by typing in the commands by hand. First press MENU 6 to activate the program mode of the calculator. Press NEW to start. Enter a name (maximum eight letters) and press EXE when you're finished. It is best to use the names of the programs used in this book to avoid confusion.

Enter the program commands, exactly as they are written in this book, pressing EXE (shown as ↵) at the end of each line. Most of the commands you will use in a program are the same ones you use for other calculator operations. There are also some special commands that are used almost exclusively for programming. These are accessed by first accessing PRGM (SHIFT VARS). The inside back cover of the *User's Guide* shows where all programming commands are, while Chapter 8 describes in detail how each command works. When you have finished entering the program into the calculator, press QUIT to return to the programming list menu.

Test the program to make sure there are no typing errors. If possible, you should check that your program works by using some data for which you already know the answer, before you place any confidence in the program.

## Running a program

To *run* (i.e. to operate) a program, press MENU 6 to enter Program mode, use the ▲ and ▼ keys to highlight the program name in the list, and then press EXE (F1) or just the blue EXE key.

After each input number, you need to press EXE to continue running the program.

You also need to press EXE after a display pause on the screen (shown with the word 'Disp')

If a program doesn't work, check your typing in Edit mode. (See the next section.)

To stop a programfrom running, while it is running, press the AC/ON key. The word 'Break' will appear on the screen . Press AC/ON again to clear the screen and return to RUN mode. If you press EXE instead of AC/ON, the program will start again.

You can't stop a program while it is waiting for an input (showing a ? on the screen). Input a number and then press AC/ON.

## Editing a program

To *edit* a program means to correct errors that you have noticed or to change the program in some other way. Go to the program list, select the program name and then press EDIT (F2).

You can move around with all four cursor keys (◀▶▲▼) and move straight to the beginning or end of a program with TOP (F1) and BTM (F2) respectively.

Both the DEL and INS (SHIFT DEL) keys will be useful for editing too.

When you have finished making changes, press QUIT to return to the program list.

## Deleting programs

Programs remain in your calculator until you delete them. They are not affected by turning the calculator off. To permanently delete a program, go to the program list with MENU 6, and press the continuation key. Use ▲ and ▼ to select the program name from the list, and then press DEL (F1). To delete *all* the programs in a calculator, choose DEL.A (F2) instead of DEL (F1).

In case you have pressed any keys by mistake, you then have a second chance to confirm that you do actually want to erase the program. Press YES (F1) if you do and any other key or NO (F4) if you don't.

Programs can also be deleted from the Memory mode, accessed by MENU 9. In Memory mode, you have a choice of finding out how much calculator memory is being used for various things (*Memory Usage*) or of resetting the calculator (*Reset*).

There is also a DEL (F1) key on the memory usage screen, which you can use to delete all the programs at once, if you wish.

Resetting the calculator manually (with the emergency reset on the back of the calculator) will also delete all the programs. See Appendix A of the *User's Guide* for details and warnings about this.

If you press the Reset key in error (either in Memory mode or using the reset on the back of the calculator), you still have one last chance to avoid deleting everything: Press NO (F4) instead of YES (F1).

## And, finally ...

Using programs like those in this book will allow you to do many powerful things with your calculator.

However, the most powerful use of programming comes when you start to write programs for yourself. The programs in this book are all fairly short, partly to make it easier for you to see how they work. Once you have a good idea of how these programs work, you may like to experiment with writing other programs for yourself.

Some aspects of mathematics are fairly routine, and so lend themselves to programming. An example of this is the quadratic formula, used in program *QuadEqtn*. Other examples are the formulae used in the *Calendar* and *Heron* programs. There are many other formulas in mathematics for which programs can easily be written. These include working out simple interest, compound interest, the area of a circle and the cosine law.

Another aspect of mathematics for which programming is useful involves the use of repetition. You might like to try writing programs to deal with sequences and series, which are of this kind.

Many of the programs in this book make use of the random number generator to simulate probabilities. You might like to try writing other programs that use this calculator feature.

I express my gratitude to Debbie Taylor, who has been encouraging during this project and especially helpful in checking the text and the programs in this book, although I remain responsible for any lingering errors that you may find.

I hope that you enjoy using these programs and wish you well in starting the adventure of writing some others for yourself or your friends.

*Barry Kissane*

*The Australian Institute of Education*
*Murdoch University*
*Western Australia 6150*

http://wwwstaff.murdoch.edu.au/~kissane
kissane@murdoch.edu.au

# INDEX OF PROGRAMS

# CALENDAR

```
"DAY"?→D↲
"MONTH"?→M↲
"YEAR"?→Y↲
1+D+2M+Int(3(M+1)÷5)+Y+Int(Y÷4)−Int(Y÷100)+Int(Y÷400)→W↲
7×Frac (W÷7)→W↲
"WEEKDAY IS"↲
W
```

## Purpose

Gives the week day for any date, according to the Gregorian calendar, which is in everyday use today in western countries. The program evaluates a formula that takes leap years appropriately into account.

## Operation

Enter the day, month number and year (all four digits), pressing `EXE` after each. For example, 31 August 1949 is entered as `31 EXE 8 EXE 1949 EXE`.

**Important Note**: **January and February must be entered as the 13th and 14th months of the previous year.** So February 6, 1950 must be entered as 6, 14, 1949 instead of 6, 2, 1950.

The program uses the following codes to report the week day:

| | |
|---|---|
| 0 | Sunday |
| 1 | Monday |
| 2 | Tuesday |
| 3 | Wednesday |
| 4 | Thursday |
| 5 | Friday |
| 6 | Saturday |

Check the program by using today's date.

On what day of the week were you born? Check with your parents.

## HERON

```
"SIDE 1"?→A↲
"SIDE 2"?→B↲
"SIDE 3"?→C↲
(A+B+C)÷2→S↲
"AREA IS"↲
√(S(S-A)(S-B)(S-C))
```

### Purpose
Uses the lengths of the three sides of a triangle to calculate the area of any triangle, using Heron's formula, known since ancient times.

### Operation
Enter the length of each side, followed by EXE. Both numbers and expressions (such as √2) are permitted.

The program displays the area of the triangle.

Press EXE to enter the side lengths of another triangle.

Test the program by entering sides of a familiar triangle, such as a 3-4-5 triangle, and checking that the calculated area is correct.

Notice what happens when the program is run with side lengths that do not form a triangle (such as 3-4-11).

There is a discussion of the use of this sort of program in the text:

Kissane, Barry & Harradine, Anthony (2000) *Mathematical Interactions: Measurement*, Chatswood, NSW: Shriro, pp 19-20.

# ZERO

```
While Abs (Y1)>0.00000001↵
X-(Y1÷d/dx(Y1,X))→X↵
WhileEnd↵
{X,Y1}
```

## Purpose

The program finds approximately a zero of a function, i.e. a value of the variable for which the function has zero value.

## Operation

The function must be entered as `Y1` in the function list and then graphed.

Trace to a point near a zero you want to find and then activate the program. The coordinates of a point close to the zero are given as a list.

The first element is the zero itself (the *x*-value).
The second element (the *y*-value) gives an indication of how close it is to an exact zero.

Test by using the function $y = 2 - x^2$ and check that there is a zero near $x = 1.4142$ and near $x = -1.4142$. (You will need to use the program twice to get both zeroes.)

In each case, the *y*-value is about -0.00000000008, so the value given is very close to a zero.

# 2 X 2 EQ

```
"A11X+A12Y=B1"↵
"A21X+A22Y=B2"↵
"A-11 "?→A↵
"A-12 "?→B↵
"B-1 "?→C↵
"A-21 "?→D↵
"A-22 "?→E↵
"B-2 "?→F↵
AE−BD→G↵
If G≠0↵
Then Goto 1↵
Else "NO SOLUTION":Stop↵
Lbl 1↵
{(EC−BF)⌟G,(AF−DC)⌟G}
```

## Purpose

Solving a set of two simultaneous linear equations, by entering only the coefficients of each.

## Operation

The program reminds you of the form for the equations:

$a_{11}x + a_{12}y = b_1$
$a_{21}x + a_{22}y = b_2$

You may have to rewrite your equations into this form before entering them. Input the six coefficients in the correct order, pressing EXE after each.

If the coefficients are all integers, the result will be given as a fraction.

Test the program by finding the solutions to the system:

$2x + 3y = 3$
$12y - 2x = 7$

The solution is $x = 1/2$ and $y = 2/3$.

# 3 X 3 EQ

```
"A-11 "?→A↵
"A-12 "?→B↵
"A-13 "?→C↵
"B-1 "?→D↵
"A-21 "?→E↵
"A-22 "?→F↵
"A-23 "?→G↵
"B-2 "?→H↵
"A-31 "?→I↵
"A-32 "?→J↵
"A-33 "?→K↵
"B-3 "?→L↵
A(FK-GJ)-B(EK-IG)+C(EJ-IF)→W↵
If W≠0↵
Then Goto 1↵
Else "NO SOLUTION":Stop↵
Lbl 1↵
D(FK-GJ)-B(HK-LG)+C(HJ-FL)→X↵
A(HK-GL)-D(EK-GI)+C(EL-HI)→Y↵
A(FL-HJ)-B(EL-HI)+D(EJ-FI)→Z↵
{X⌟W,Y⌟W,Z⌟W}
```

## Purpose

Solving a set of three simultaneous linear equations, by entering only the coefficients of each.

## Operation

The program reminds you of the form for the equations:

$a_{11}x + a_{12}y + a_{13}z = b_1$
$a_{21}x + a_{22}y + a_{23}z = b_2$
$a_{31}x + a_{32}y + a_{33}z = b_3$

You may have to rewrite your equations into this form before entering them. Input the twelve coefficients in the correct order, pressing EXE after each.

If the coefficients are all integers, the result will be given as a fraction.

Test the program by finding the solutions to the system:

$2x + 3y + z = 3$
$12y - 2x + 2z = 7$
$x - z + 4y = 11$

The solution is $x = 21/160$, $y = 1/160$ and $z = 87/32$.

## QUADEQTN

```
"AX²+BX+C=0"↵
"A"?→A↵
"B"?→B↵
"C"?→C↵
B²−4AC→D↵
If D<0↵
Then "NO REAL SOLUTION":Stop↵
Else {(-B+√D)÷2A,(-B−√D)÷2A}↵
IfEnd
```

## Purpose

Finding the roots of a quadratic equation, using the quadratic formula.

## Operation

The program reminds you of the form of the equation: $ax^2 + bx + c = 0$. You may have to rewrite your equations into this form before entering them.

Input the three coefficients in the correct order, pressing EXE after each.

Test the program by finding the solutions to the equation:

$x^2 = 10 – x$

In this case, first rewrite the equation to the appropriate form, $x^2 + x – 10 = 0$ to see that the coefficients are A = 1, B = 1 and C = -10.

The two approximate solutions are $x = 2.702$ and $x = -3.702$.

# INTRSECT

```
"USE Y1 AND Y2"↵
Fix 3↵
(Xmax-Xmin)÷78→W↵
10→D↵
Lbl 1↵
X+W→X↵
D→P↵
Abs (Y2-Y1)→D↵
If D<0.0001↵
Then Goto 2↵
IfEnd↵
If D<P↵
Then Goto 1↵
Else -(W÷10)→W↵
Goto 1↵
Lbl 2↵
{X,Y1}◢
Norm
```

## Purpose

To find the coordinates of a point of intersection of the graphs of two functions. This can be used to solve a pair of simultaneous equations.

## Operation

Start by graphing the two functions, using Y1 and Y2.

Trace to a point close to the point of intersection. Then activate the program.
The result will be given correct to three decimal places.

The program reminds you that the two functions must be in Y1 and Y2.

Test the program by finding the points of intersection of the graphs of

$y = 2 - x^2$ and $y = 1 + x$.

One of the points is (0.618,1.618), while the other is (-1.618,-0.618).

## MINIMUM

```
(Xmax-Xmin)÷78→H↵
Lbl 1↵
Y1→A↵
X+H→X↵
Y1→B↵
If B<A↵
Then Goto 1↵
Else -H÷10→H↵
IfEnd↵
If Abs H<0.00001↵
Then Goto 2↵
Else Goto 1↵
IfEnd↵
Lbl 2↵
{X,Y1}
```

### Purpose
Find the coordinates of a relative minimum of a function.

### Operation
The function must be in Y1 in the list.

Draw a graph of the function and trace to a point near a minimum point.

Activate the program, which will give the coordinates of the minimum point, correct to three decimal places.

Test the program by finding the relative minimum of the function

$y = x^2 – x – 1$.

The result is (0.500,-1.250).

# MAXIMUM

```
(Xmax-Xmin)÷78→H↵
Lbl 1↵
Y1→A↵
X+H→X↵
Y1→B↵
If B>A↵
Then Goto 1↵
Else -H÷10→H↵
IfEnd↵
If Abs H<0.00001↵
Then Goto 2↵
Else Goto 1↵
IfEnd↵
Lbl 2↵
{X,Y}
```

## Purpose

Find the coordinates of a relative maximum of a function.

## Operation

The function must be in Y1 in the list.

Draw a graph of the function and trace to a point near a maximum point.

Activate the program, which will give the coordinates of the maximum point, correct to three decimal places.

Test the program by finding the relative maximum of the function

$y = x^3 - x - 1$.

The result is (-0.577,-0.664).

## SEARCH

```
For 10→X To 100↵
Int (X÷10)→A↵
10×Frac (X÷10)→B↵
If AB+A+B=X↵
Then X◢
IfEnd↵
Next
```

### Purpose

To search systematically through a range of values in order to solve a problem.

### Operation

In its present form, the program checks each of the numbers from 10 to 100 in turn to see whether any of them satisfy the following condition:

*The product of the tens and units digit, added to the sum of the tens and units digit, gives the original number.*

When the program is run, any numbers satisfying the condition are printed. Press EXE to continue after each answer.

Not all problems have solutions, of course.

This program can be modified, in order to answer different questions, by changing the first four lines.

# INTEGRAL

```
"FUNCTION Y1"↵
"LOWER "?→A↵
"UPPER "?→B↵
100→N↵
A→X:Y1→S↵
(B−A)÷N→W↵
For 1→K To N−1↵
X+W→X↵
2Y1+S→S↵
Next↵
X+W→X↵
Y1+S→S↵
WS÷2→S↵
S
```

## Purpose

To find the area under a curve and above the *x*-axis between two points, using the integral of a function.

## Operation

The program reminds you that the function must be in Y1.

Enter the function in Graph mode and then start the program in Program mode.

Enter the *x*-value for the lower limit and the upper limit, pressing EXE after each.

The area is then printed on the screen.

To test the program, check that the area under the function $y = \sqrt{1 - x^2}$ between $x = 0$ and $x = 1$ is 0.785. (Notice that this is an approximation to the area of a quadrant of the unit circle, the exact area of which is $\pi/4$.

The program uses 100 intervals to approximate the integral. This is set by the fourth line of the program. A more accurate (but slower!) result can be obtained by increasing this number. For example, the following new version of line 4 will change the number of intervals to 150:

```
150→N↵
```

A less accurate (but faster) result can be obtained by decreasing this number.

# DECIMAL

```
"NUMERATOR"?→N↵
"DENOMINATOR"?→X↵
Lbl 1↵
Int (N÷X)→D↵
10(N−DX)→N↵
D◢
Goto 1
```

## Purpose

To give the decimal expansion of a fraction to as many decimal places as desired.

## Operation

Enter the numerator and the denominator, both whole numbers, pressing EXE after each.

Each time you press EXE, the next digit in the decimal expansion will be given. Record these on paper if you wish. Note that the decimal point is not displayed by the calculator.

Press AC/ON when you wish to stop.

To test the program, check that the decimal expansion of 3/13 is 0.023076923..., which is a recurring decimal.

# FACTORS

```
"NUMBER"?→N↵
N→X↵
While Frac (X÷2)=0↵
2◢
X÷2→X↵
WhileEnd↵
For 3→D To √N Step 2↵
Lbl 1↵
If Frac (X÷D)=0↵
Then D◢
X÷D→X↵
Goto 1↵
IfEnd↵
Next↵
X
```

## Purpose
To find all the prime factors of a positive integer.

## Operation
Enter an integer at the prompt and press EXE to start.

Each time you press EXE, another prime factor will be displayed.

To test the program, enter 234 and check that the prime factors are 2 x 3 x 3 x 13.

## DICE

```
"NO OF ROLLS"?→N↵
Seq(0,X,1,N,1)→List 1↵
For 1→A To N↵
Int (6Ran#+1)→List 1[A]↵
Next↵
"MEAN SCORE"↵
Mean(List 1)
```

### Purpose
To simulate rolling a standard six-sided die, for which each of the six possible scores are equally likely.

### Operation
Enter the number of rolls required. (The maximum is 255.) Press EXE to start.

The program will display the mean score of the simulated dice rolls.

The actual rolls are stored in List 1. Any data already in List 1 will be lost.

Press Menu 2 to see the data, and to analyse them if you wish. You may like to use program *FREQDIST* in order to convert the data into a frequency distribution.

Each time the program is run, a different set of dice rolls is generated.

To simulate a dice with a different number of sides, edit the fourth line of the program. For example, if you change the six to an eight, the program will simulate rolling an 8-sided die, each side of which is equally likely. Here is the relevant line after editing:

```
Int (8Ran#+1)→List 1[A]↵
```

## TWODICE

```
"NO OF ROLLS"?→N↵
Seq(0,X,1,N,1)→List 1↵
For 1→A To N↵
Int (6Ran#+1)+Int (6Ran#+1)→List 1[A]↵
Next↵
"MEAN SCORE"↵
Mean(List 1)
```

### Purpose

To simulate rolling two standard six-sided die, for which each of the six possible scores are equally likely on each die. The two die rolls are added each time to produce a score.

### Operation

Enter the number of rolls required. (The maximum is 255.)

The program will display the mean score of the simulated dice rolls.

The actual totals are stored in List 1. Any data already in List 1 will be lost.

Press Menu 2 to see the data, and to analyse them if you wish. You may like to use program *FREQDIST* in order to convert the data into a frequency distribution.

Each time the program is run, a different set of dice rolls is generated.

To simulate dice with different numbers of sides, edit the fourth line of the program. For example, if you change the first six to a ten, the program will simulate rolling a ten-sided die followed by a six-sided die and then adding the scores. In each case, all sides of dice are equally likely. Here is the relevant line after editing:

```
Int (10Ran#+1)+Int (6Ran#+1)→List 1[A]↵
```

## COIN

```
"NO OF TOSSES"?→N↵
0→X↵
For 1→K To N↵
Int (Ran#+0.5)+X→X↵
Next↵
"NO OF HEADS"↵
X
```

### Purpose
Simulate tossing a coin and counting how often a head is obtained.

### Operation
Enter the (whole) number of tosses. There is no limit on the number, except your time and batteries. With fresh batteries, the calculator will need about 30 seconds for each thousand tosses.

The program will display how many of the tosses were 'heads'.

Do this a few times to see how much it varies each time. Compare your results with someone else's.

If you wish to simulate tosses of a biased coin, you will need to change the fourth line of the program. For example, a change to

```
Int (Ran#+0.45)+X→X↵
```

will simulate tossing a coin with only a 45% chance of a head each time.

## FREQDIST

```
"DATA IN LIST1"↵
"DATA INTEGERS"◢
Min(List 1)→A↵
Max(List 1)→B↵
B-A+1→N↵
Seq(X+A,X,0,N-1,1)→List 2↵
Seq(0,X,0,N-1,1)→List 3↵
Dim List 1→K↵
For 1→J To K↵
List 1[J]→X:X-A+1→X↵
List 3[X]+1→List 3[X]↵
Next↵
"FREQ DISTRIB"↵
"IN LISTS 2,3"↵
```

### Purpose

To generate a frequency distribution from raw data.

### Operation

The first two lines of the program remind you that the data must be in List 1 and must all be integers.

Press EXE to continue the program after the messages. The frequency distribution will be placed into Lists 2 and 3. Any data already in List 2 and List 3 will be lost.

When finished, the calculator displays a message reminding you that List 2 shows the possible scores and List 3 shows the frequencies of each score. Press Menu 2 or Menu 3 to see these.

## DIFFLIST

```
"DATA ARE"↵
"IN LIST 1"↵
Dim List 1→N↵
Seq(0,X,1,N-1,1)→List 2↵
Seq(0,X,1,N-2,1)→List 3↵
For 1→K To N-1↵
List 1[K+1]-List 1[K]→List 2[K]↵
Next↵
For 1→K To N-2↵
List 2[K+1]-List 2[K]→List 3[K]↵
Next↵
"PRESS MENU 2"↵
"FOR RESULTS"↵
```

### Purpose

Find the first and second differences for a sequence.

### Operation

Delete the contents of List 1 and then enter your sequence into List 1.

The program will store the first differences between elements of the sequence into List 2 and the second differences into List 3. Any data already in List 2 and List 3 will be lost.

Press `MENU 2` (or `MENU 3`) to see the results of finding the differences.

To test the program, put the squares of the first ten integers (1, 4, 9, … , 100) into List 1. When the program is activated, List 2 will show odd numbers and List 3 will show a constant value of 2 (the difference between successive odd numbers).

When the second differences are constant, you can deduce that the original sequence is quadratic.

Experienced programmers will be able to see how to modify the program so that third differences are also calculated.

# LONGRUN

```
Cls↵
-0.1→Ymin:1→Ymax:0.1→Yscl:0→Xmin:78→Xmax:10→Xscl↵
"PROBABILITY"?→P↵
Horizontal P↵
0→S:0→K:0→X↵
While X<78↵
Int (Ran#+P)→T↵
If T≠1↵
Then Goto 1↵
Else S+1→S↵
IfEnd↵
Lbl 1↵
K+1→K↵
PlotOn X,S÷K↵
X+1→X↵
WhileEnd↵
"REL FREQ ="↵
S÷K
```

## Purpose

To illustrate what happens in the long run when events with a certain probability are generated. For example, to see what happens when a fair coin is tossed repeatedly, by plotting the proportion of the heads after various numbers of tosses.

## Operation

Enter the probability concerned and press EXE to continue.

The program will display the 'ideal' theoretical probability as  a horizontal line. The vertical axis is marked in steps of 0.1 or 10%.

After each simulated event, the proportion of 'successful' times is plotted. This is likely to be eventually (in the long run) close to the horizontal line.

After a screen full of simulations, the relative frequency is displayed. This will not usually be the same as the probability.

Press the G–T key to switch between the graph and the numerical results.

Press the EXE key to repeat the simulation.

Do this a few times to see how much it varies each time. Compare your results with someone else's.

There is a discussion of the use of this sort of program in the text,

Kissane, Barry, Harradine, Anthony & Boys, Anthony (1999) *Mathematical Interactions: Data Analysis and Probability*, Chatswood, NSW: Shriro, pp 21-22.

## SRANSAMP

```
"SAMPLE SIZE"?→N↵
Seq(0,X,1,N,1)→List 6↵
Dim List 1→M↵
For 1→K To N↵
List 1[Int (MRan#+1)]→List 6[K]↵
Next↵
"SAMPLE MEAN"↵
Mean(List 6)↵
```

### Purpose

Take a simple random sample of a certain size from a population.

A simple random sample is one for which each element of the population has the same chance of being chosen. Notice that it is possible for the same elements to be chosen more than once. (That is, sampling *with replacement* is involved.)

### Operation

First delete the contents of List 1 and then enter the population of interest into List 1. (The population cannot exceed 255 elements.)

Start the program, enter the sample size and press `EXE`.

The calculator will select a simple random sample of the chosen size and display the mean of the sample. The sample will be stored into List 6. Any data already in List 6 will be lost.

The actual sample chosen can be seen in List 6 by first pressing `MENU 2`.

Compare several samples of the same size; you should find that, while they differ from each other, they have similar means.

Compare samples of different sizes; you should find that larger samples give more reliable information about your population.

There is a discussion of the use of this sort of program in the text,

Kissane, Barry, Harradine, Anthony & Boys, Anthony (1999) *Mathematical Interactions: Data Analysis and Probability*, Chatswood, NSW: Shriro, pp 23-26.

# NORMAL

```
"MEAN"?→M↵
"STD DEV"?→S↵
"(1÷(S√(2π)))e(-(X-M)²÷2S²)"→Y1↵
100→N↵
Lbl 1↵
"LOWER VALUE"?→L↵
"UPPER VALUE"?→U↵
L→X:Y1→T↵
(U-L)÷N→W↵
For 1→K To N-1↵
X+W→X↵
2Y1+T→T↵
Next↵
X+W→X↵
Y1+T→T↵
Fix 4↵
WT÷2→T◢
Norm↵
Goto 1
```

## Purpose

To generate values for areas between two values of a normal probability distribution with mean *M* and standard deviation *S*.

## Operation

Enter the mean and standard deviation, each followed by EXE. (For the standard normal (*z*) distribution, enter *M* = 0 and *S* = 1.)

Then enter the lower value and the upper value, each followed by EXE. The probability that the normal random variable will have a value between these two is then printed, correct to four decimal places.

Press EXE to enter another pair of lower and upper values for the same distribution.

To find probabilities associated with a *different* normal distribution, press AC/$^{ON}$ and then start the program again.

Test the program by inputting values of -1 and 2 to find Pr ($-1 < z < 2$) = 0.8186.

The program can be made faster (but less accurate) by changing the fourth line. For example, changing it to

```
50→N↵
```

will make it faster, but a little less accurate. Increasing *N* will, of course, make it more accurate but slower.

## BINOMIAL

```
"PROBABILITY"?→P↵
"NO. TRIALS"?→N↵
"HOW MANY"?→K↵
Seq(0,X,1,K,1)→List 1↵
For 1→I To K↵
0→X↵
For 1→J To N↵
Int (Ran#+P)+X→X↵
Next↵
X→List 1[I]↵
Next↵
```

### Purpose

To simulate a binomial random variable, consisting of a number of trials each of which has the same probability of success. For example, tossing a set of 12 coins, each of which has a probability of 0.5 of landing 'heads', and counting how many of the twelve land heads each time.

### Operation

Enter the probability of success each time, followed by EXE.

Enter the number of trials involved, followed by EXE.

Enter the number of separate simulations required, followed by EXE. (A maximum of 255 is possible.)

For example, to simulate 50 separate sets of 12 tosses of a fair coin, enter:

```
0.5 EXE 12 EXE 50 EXE.
```

The program will store the numbers of heads in each set of trials into List 1. Any data already in List 1 will be lost.

You may like to use program *FREQDIST* to analyse the resulting data in List 1.

Notice that, in the above case, you usually *don't* get exactly six 'heads' out of twelve tosses.

## BINPROB

```
"NUMBER"?→N↵
"PROBABILITY"?→P↵
"PROB X=0 IS"↵
(1-P)^N→F◢
Seq(0,X,1,N,1)→List 1↵
List 1→List 2↵
For 1→X To N↵
(NCX)(P^X)(((1-P)^(N-X))→List 1[X]↵
Next↵
List 1[1]+F→List 2[1]↵
For 2→X To N↵
List 1[X]+List 2[X-1]→List 2[X]↵
Next↵
"PRESS MENU 2"↵
"LIST1:PROB"↵
"LIST2:CUMPROB"↵
```

### Purpose

To generate values for the binomial and cumulative binomial probability distributions.

### Operation

Enter the number of trials (*N*) and the probability of success for each (*P*), followed by EXE in each case.

The probability of zero successes is displayed on the screen. *Write this result down on paper.*

Then press MENU 2 to see the probabilities of getting various numbers of successes, together with the associated cumulative probabilities. These are stored in List 1 and List 2 respectively. Any data already in these lists will be lost.

To test the program, consider the case of twelve tosses of a coin with probability 0.5 of getting 'heads'. Check the following results:

The probability of obtaining no heads in the twelve tosses is about 0.00024.
The probability of obtaining six heads out of twelve tosses is about 0.22559.
The probability of getting six heads or less out of twelve tosses is about 0.61279.

Notice that the final value in List 2 is always 1. Why is this?

## CMPNDINT

```
"PRINCIPAL"?→P↵
"AMOUNT"?→A↵
"RATE P.A."?→R↵
"# PERIODS"?→N↵
"TIMES/YEAR"?→C↵
Fix 2↵
If R≥1:Then R÷100→R:IfEnd↵
If P>0:Then Goto 1:Else
"PRINCIPAL":A÷((1+R÷C)^(N))→P:Stop↵
Lbl 1↵
If A>0:Then Goto 2:Else "AMOUNT":P(1+R÷C)^(N)→A:Stop↵
Lbl 2↵
If R>0:Then Goto 3:Else "INT RATE P.A.":100C((A÷P)^(1÷N)-
1)→R:Stop↵
Lbl 3↵
If C>0:Then Goto 4:Else "TIMES/YEAR":R÷((A÷P)^(1÷N)-
1)→C:Stop↵
Lbl 4↵
"# PERIODS":(ln (A÷P)÷(ln (1+R÷C))→N:Stop
```

## Purpose

To explore various compound interest situations involving a Principal, an Amount, an interest rate, a number of years of investment and a number of compounding periods per year. If all except one of these variables is known, the other one is calculated.

## Operation

Enter the variable values when requested, pressing EXE after each.

Enter a negative value, such as -1, for the *one* value you wish to calculate.

*RATE P.A.* refers to the annual interest rate. Enter it either as a decimal or as a percentage. (E.g., 5.4% per annum can be entered *either* as 5.4 or as 0.054.)

The variable *# PERIODS* refers to the *total* number of compounding periods. For example, if the principal is invested for 25 years with interest compounded monthly, then enter 300, as there are 12 x 25 = 300 compounding periods. (If you wish, you can enter 12 x 25, and let the calculator do the multiplication.)

The variable *TIMES/YEAR* refers to the number of compounding periods per year. For example, if interest is added monthly, the value is 12, while for quarterly interest, the value is 4.

Test the program by checking that $2800 will accumulate to $3871.89 if it is deposited into an account in which interest of 6.5% is compounded monthly for five years.