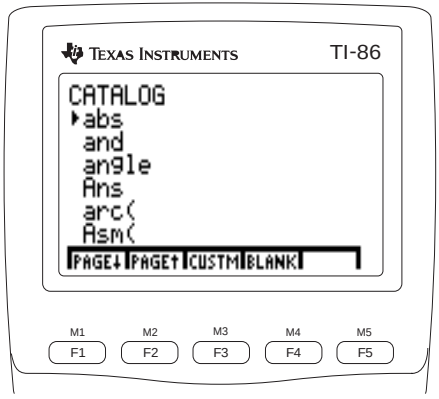
20 A to Z Function and Instruction Reference



Quick-Find Locator

This section lists the TI-86 functions and instructions in functional groups along with the page numbers where they are described in this chapter.

Graphing

Lists

Mathematics, Algebra, and Calculus

Matrices

Programming

Statistics

Strings

Vectors

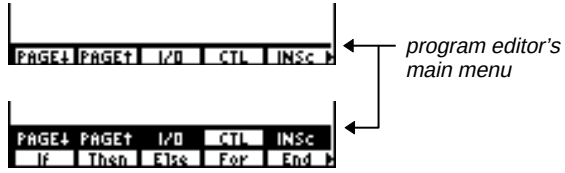
Alphabetical Listing of Operations

All the operations in this section are included in the CATALOG. Non-alphabetic operations (such as **+**, **!**, and **>**) are listed at the end of the CATALOG. In this A to Z Reference, however, these operations are listed under their alphabetic equivalent (such as addition, factorial, and greater than).

You always can use the CATALOG to select an operation and paste it to the home screen or to a command line in the program editor. You also can use the specific keystrokes, menus, or screens listed in this section.

† Indicates menus or screens that paste the operation's name only if you are in the program editor. In most cases, you can use these menus or screens from the home screen to perform the operation interactively, without pasting the name.

‡ Indicates menus or screens that are valid only from the program editor's main menu. From the home screen, you cannot use these menus or screens to select an operation.



The syntax for some operations uses brackets [ ] to indicate optional arguments. If you use an optional argument, do not enter the brackets.

abs

MATH NUM menu
CPLX menu
MATRX CPLX menu
VECTR CPLX menu

abs *realNumber* or **abs (***realExpression***)**

Returns the absolute value of *realNumber* or *realExpression*.

abs (*complexNumber***)**

Returns the magnitude (modulus) of *complexNumber*.

abs (*real***,***imaginary***)** returns $\sqrt{(real^2+imaginary^2)}$.
abs (*magnitude*∠*angle***)** returns *magnitude*.

abs *list*
abs *matrix*
abs *vector*

Returns a list, matrix, or vector in which each element is the absolute value of the corresponding real or complex element in the argument.

```
abs -256.4 ENTER                    256.4

abs -4*3+13 ENTER                    25
abs (-4*3+13) ENTER                   1

abs (3,4) ENTER                       5
abs (3∠4) ENTER                       3

abs {1.25,-5.67} ENTER
                            {1.25 5.67}

abs [(3,4),(3∠4)] ENTER
                                [5 3]
```

Addition: +

⊞

numberA **+** *numberB*

Returns the sum of two real or complex numbers.

number **+** *list*

Returns a list in which a real or complex *number* is added to each element of a real or complex *list*.

In **RectC** complex number mode:

```
(2,5)+(5,9) ENTER                 (7,14)

4+{1,2,3} ENTER                   {5 6 7}

3+{1,7,(2,1)} ENTER
                    {(4,0) (10,0) (5,1)}
```

listA **+** *listB*
matrixA **+** *matrixB*
vectorA **+** *vectorB*

Returns a list, matrix, or vector that is the sum of the corresponding real or complex elements in the arguments. The two arguments must have the same dimension.

For information about adding two strings, refer to **Concatenation** on page 274.

```
{1,2,3}+{4,5,6} ENTER        {5 7 9}

[[1,2,3][4,5,6]]+[[4,5,6][7,8,9]]
ENTER                    [[5  7  9 ]
                          [11 13 15]]

[1,2,3]+[4,5,6] ENTER        [5 7 9]
```

and

BASE BOOL menu

integerA **and** *integerB*

Compares two real integers bit by bit. Internally, both integers are converted to binary. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value is the sum of the bit results.

For example, 78 **and** 23 = 6.

78 = 1001110**b**
23 = 0010111**b**
0000110**b** = 6

You can enter real numbers instead of integers, but they are truncated automatically before the comparison.

In **Dec** number base mode:

```
78 and 23 ENTER                  6
```

In **Bin** number base mode:

```
1001110 and 10111 ENTER
                              110b
Ans▶Dec ENTER                  6d
```

angle

CPLX menu
MATRX CPLX menu
VECTR CPLX menu

angle (*complexNumber***)**

Returns the polar angle of *complexNumber*, adjusted by +π in the 2nd quadrant or -π in the 3rd quadrant. The polar angle of a real number is always 0.

angle (*real***,***imaginary***)** returns **tan^{-1}(***imaginary/real***)**.
angle (*magnitude*∠*angle***)** returns *angle*, -π < *angle* ≤ π.

angle *complexList*
angle *complexMatrix*
angle *complexVector*

Returns a list, matrix, or vector in which each element is the polar angle of the corresponding element in the argument.

If *complexVector* has only two real elements, the returned value is a real number, not a vector.

In **Radian** angle mode and **PolarC** complex number mode:

```
angle (3,4) ENTER                .927295218002
angle (3∠2) ENTER                            2
(6∠π/3)→A ENTER          (6∠1.0471975512)
angle A ENTER                    1.0471975512
angle {(3,4),(3∠2)} ENTER
                          {.927295218002 2}
```

Ans

2nd [ANS]

Ans

Returns the last answer.

```
1.7*4.2 ENTER                          7.14
147/Ans ENTER                20.5882352941
```

arc(

CALC menu

arc (*expression***,***variable***,***start***,***end***)**

Returns the length along *expression* with respect to *variable*, from *variable* = *start* to *variable* = *end*.

```
arc(x²,x,0,1) ENTER
                           1.47894285752
arc(cos x,x,0,π) ENTER
                           3.82019778904
```

Asm(

CATALOG

Asm(*assemblyProgramName***)**

Executes an assembly language program. For more information, refer to Chapter 16.

AsmComp(

CATALOG

AsmComp(*AsciiAssemblyPrgmName*,*HexAssemblyPrgmName***)**

Compiles an assembly language program written in ASCII and stores the hex version. The compiled hex version, which uses about half the storage space of the ASCII version, cannot be edited.

When you execute the ASCII version, the TI-86 compiles it each time. To speed up execution, use **AsmComp(** to compile the ASCII version once and then execute the hex version each time you want to run the program.

AsmPrgm

CATALOG

AsmPrgm

Must be used as the first line of an assembly language program.

Assignment: =

ALPHA [=]

equationVariable **=** *expression*

Stores *expression* to *equationVariable*, without evaluating *expression*. (If you use STO▸ to store an expression to a variable, the expression is evaluated and then the result is stored.)

```
y1=2 x²+6 x−5 ENTER                    Done
```

The built-in equation variables used for graphing are case-sensitive. Use **y1**, not **Y1**.

aug(

LIST OPS menu

MATRX OPS menu

aug(*listA*,*listB***)**

Returns a list consisting of *listB* appended (concatenated) to the end of *listA*. The lists can be real or complex.

```
aug({1,-3,2},{5,4}) ENTER
                          {1 -3 2 5 4}
```

aug(*matrixA*,*matrixB***)**

Returns a matrix consisting of *matrixB* appended as new columns to the end of *matrixA*. The matrices can be real or complex. Both must have the same number of rows.

aug(*matrix*,*vector***)**

Returns a matrix consisting of *vector* appended as a new column to the end of *matrix*. The arguments can be real or complex. The number of rows in *matrix* must equal the number of elements in *vector*.

```
[[1,2,3][4,5,6]]→MATA ENTER
                        [[1 2 3]
                         [4 5 6]]
[[7,8][9,10]]→MATB ENTER
                        [[7 8 ]
                         [9 10]]
aug(MATA,MATB) ENTER
                    [[1 2 3 7 8 ]
                     [4 5 6 9 10]]
```

Axes(

† GRAPH VARS menu

Axes(*xAxisVariable*,*yAxisVariable***)**

Specifies the variables plotted for the axes in **DifEq** graphing mode. The *xAxisVariable* or *yAxisVariable* can be **t**, **Q1** through **Q9**, or **Q′1** through **Q′9**.

```
Axes(Q1,Q2) ENTER                Done
```

AxesOff

† graph format screen

AxesOff

Turns off the graph axes.

AxesOn

† graph format screen

AxesOn

Turns on the graph axes.

b

BASE TYPE menu

*integer***b**

Designates a real *integer* as binary, regardless of the number base mode setting.

In **Dec** number base mode:

```
10b ENTER                          2
10b+10 ENTER                      12
```

Bin

† mode screen

Bin

Sets binary number base mode. Results are displayed with the **b** suffix. In any number base mode, you can designate an appropriate value as binary, decimal, hexadecimal, or octal by using the **b**, **d**, **h**, or **o** designator, respectively, from the BASE TYPE menu.

In **Bin** number base mode:

```
10+Fh+10o+10d ENTER          100011b
```

▸Bin

BASE CONV menu

number▸**Bin**
list▸**Bin**
matrix▸**Bin**
vector▸**Bin**

Returns the binary equivalent of the real or complex argument.

In **Dec** number base mode:

```
2*8 ENTER                        16
Ans▸Bin ENTER                10000b

{1,2,3,4}▸Bin ENTER
                  {1b 10b 11b 100b}
```

Box

† STAT DRAW menu

Box *xList*,*frequencyList*

Draws a box plot on the current graph, using the real data in *xList* and the frequencies in *frequencyList*.

Box *xList*

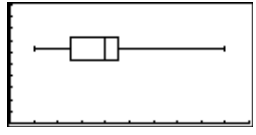
Uses frequencies of 1.

Box

Uses the data in built-in variables **xStat** and **fStat**. These variables must contain valid data of the same dimension; otherwise, an error occurs.

Starting with a **ZStd** graph screen:

```
{1,2,3,4,5,9}→XL ENTER
                      {1 2 3 4 5 9}
{1,1,1,4,1,1}→FL ENTER
                      {1 1 1 4 1 1}
0→xMin:0→yMin ENTER                0
Box XL,FL ENTER
```



Circl(

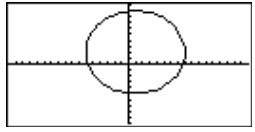
† GRAPH DRAW menu

Circl(*x*,*y*,*radius*)

Draws a circle with center (*x*,*y*) and *radius* on the current graph.

Starting with a **ZStd** graph screen:

```
ZSqr:Circl(1,2,7) ENTER
```



ClDrw

† GRAPH DRAW menu

† STAT DRAW menu

ClDrw

Clears all drawn elements from the current graph.

ClLCD

‡ program editor I/O menu

ClLCD

Clears the home screen (LCD).

ClrEnt

MEM menu

ClrEnt

Clears the contents of the Last Entry storage area.

ClTbl

‡ program editor I/O menu

ClTbl

Clears all values from the current table if **Indpnt: Ask** (**IAsk**, page 304) is set.

cnorm

MATRX MATH menu

cnorm *matrix*

Returns the column norm of a real or complex *matrix*. For each column, **cnorm** sums the absolute values (magnitudes of complex elements) of the elements in that column and returns the largest of those column sums.

```
[[1,-2,3][4,5,-6]]→MAT ENTER
                          [[1 -2 3 ]
                           [4 5  -6]]
cnorm MAT ENTER                    9
```

cnorm *vector*

Returns the sum of the absolute values of the real or complex elements in *vector*.

```
[-1,2,-3]→VEC ENTER          [-1 2 -3]
cnorm VEC ENTER                      6
```

Concatenation: **+**

⊞

stringA **+** *stringB*

Returns a string consisting of *stringB* appended (concatenated) to the end of *stringA*.

```
"your name:"→STR ENTER
                        your name:
"Enter "+STR ENTER
                   Enter your name:
```

---

cond

MATRX MATH menu

cond *squareMatrix*

Returns the condition number of a real or complex *squareMatrix*, which is calculated as:

cnorm *squareMatrix* ***** **cnorm** *squareMatrix*$^{-1}$

The condition number indicates how well-behaved *squareMatrix* is expected to be for certain matrix functions, particularly inverse. For a well-behaved matrix, the condition number is close to 1.

log(cond *squareMatrix***)** indicates the number of digits that may be lost due to round-off errors in computing the inverse.

For a matrix with no inverse, **cond** returns an error.

```
[[1,0,0][0,1,0][0,0,1]]→MAT1
ENTER                       [[1 0 0]
                             [0 1 0]
                             [0 0 1]]

cond MAT1 ENTER                     1
log (Ans) ENTER                     0

[[1,2,3][4,5,6][7,8,9]]→MAT2
ENTER                       [[1 2 3]
                             [4 5 6]
                             [7 8 9]]

cond MAT2 ENTER                1.8E14
log (Ans) ENTER        14.2552725051
```

conj

CPLX menu

MATRX CPLX menu

VECTR CPLX menu

conj (*complexNumber***)**

Returns the complex conjugate of *complexNumber*.

In **RectC** mode, **conj (***real***,***imaginary***)** returns **(***real*,-*imaginary***)**.

In **PolarC** mode, **conj (***magnitude*∠*angle***)** returns **(***magnitude*∠-*angle***)**, -π < *angle* ≤ π.

conj *complexList*
conj *complexMatrix*
conj *complexVector*

Returns a complex list, matrix, or vector in which each element is the complex conjugate of the original.

In **RectC** complex number mode:

```
conj (3,4) ENTER                    (3,-4)
conj (3∠2) ENTER
                  (-1.24844050964,-2.7…
```

In **PolarC** complex number mode:

```
conj (3∠2) ENTER
                                    (3∠-2)
conj (3,4) ENTER
                        (5∠-.927295218002)
conj {√-2,(3,4)} ENTER
                   {(1.41421356237∠-1.5…
```

CoordOff

† graph format screen

CoordOff

Turns off cursor coordinates so they are not displayed at the bottom of a graph.

CoordOn

† graph format screen

CoordOn

Displays cursor coordinates at the bottom of a graph.

cos

[COS]

cos *angle* or **cos (***expression***)**

Returns the cosine of *angle* or *expression*, which can be real or complex.

An angle is interpreted as degrees or radians according to the current angle mode. In any angle mode, you can designate an angle as degrees or radians by using the ° or ʳ designator, respectively, from the MATH ANGLE menu.

In **Radian** angle mode:

```
cos π/2 ENTER                    -.5
cos (π/2) ENTER                   0
cos 45° ENTER          .707106781187
```

In **Degree** angle mode:

```
cos 45 ENTER           .707106781187
cos (π/2)ʳ ENTER                  0
```

cos *list*

Returns a list in which each element is the cosine of the corresponding element in *list*.

In **Radian** angle mode:

```
cos {0,π/2,π} ENTER          {1 0 -1}
```

In **Degree** angle mode:

```
cos {0,60,90} ENTER          {1 .5 0}
```

cos *squareMatrix*

The squareMatrix cannot have repeated eigenvalues.

Returns a square matrix that is the matrix cosine of *squareMatrix*. The matrix cosine corresponds to the result calculated using power series or Cayley-Hamilton Theorem techniques. This is *not* the same as simply calculating the cosine of each element.

cos⁻¹

2nd [COS⁻¹]

cos⁻¹ *number* or **cos⁻¹ (***expression***)**

Returns the arccosine of *number* or *expression*, which can be real or complex.

In **Radian** angle mode:

```
cos⁻¹ .5 ENTER          1.0471975512
```

In **Degree** angle mode:

```
cos⁻¹ 1 ENTER                     0
```

cos⁻¹ *list*

Returns a list in which each element is the arccosine of the corresponding element in *list*.

In **Radian** angle mode:

```
cos⁻¹ {0,.5} ENTER
                {1.57079632679,1.047…
```

cosh

MATH HYP menu

cosh *number* or **cosh (***expression***)**

Returns the hyperbolic cosine of *number* or *expression*, which can be real or complex.

```
cosh 1.2 ENTER                1.81065556732
```

cosh *list*

Returns a list in which each element is the hyperbolic cosine of the corresponding element in *list*.

```
cosh {0,1.2} ENTER
                        {1 1.81065556732}
```

cosh^{-1}

MATH HYP menu

cosh^{-1} *number* or **cos^{-1} (***expression***)**

Returns the inverse hyperbolic cosine of *number* or *expression*, which can be real or complex.

```
cosh-1 1 ENTER                          0
```

cosh^{-1} *list*

Returns a list in which each element is the inverse hyperbolic cosine of the corresponding element in *list*.

```
cosh-1 {1,2.1,3} ENTER
                    {0 1.37285914424 1.7…
```

cross(

VECTR MATH menu

cross(*vectorA***,***vectorB***)**

Returns the cross product of two real or complex vectors, where:

cross([a,b,c],[d,e,f]) = [bf−ce cd−af ae−bd]

Both vectors must have the same dimension (either 2 or 3 elements). A 2-D vector is treated as a 3-D vector with 0 as the third element.

```
cross([1,2,3],[4,5,6]) ENTER
                                [-3 6 -3]
cross([1,2],[3,4]) ENTER
                                  [0 0 -2]
```

cSum(

LIST OPS menu

cSum(*list*)

Returns a list of the cumulative sums of the real or complex elements in *list*, starting with the first element.

```
cSum({1,2,3,4}) ENTER      {1 3 6 10}
{10,20,30}→L1 ENTER        {10 20 30}
cSum(L1) ENTER             {10 30 60}
```

▸Cyl

VECTR OPS menu

vector▸**Cyl**

Displays a 2- or 3-element real *vector* result in cylindrical form, [r∠θ z], even if the display mode is not set for cylindrical (**CylV**).

```
[-2,0]▸Cyl ENTER
                  [2∠3.14159265359 0]
[-2,0,1]▸Cyl ENTER
                  [2∠3.14159265359 1]
```

CylV

† mode screen

CylV

Sets cylindrical vector coordinate mode ([r∠θ z]).

In **CylV** vector coordinate mode and **Radian** angle mode:

```
[3,4,5] ENTER  [5∠.927295218002 5]
```

d

BASE TYPE menu

*number***d**

Designates a real *number* as decimal, regardless of the number base mode setting.

In **Bin** number base mode:

```
10d ENTER                  1010b
10d+10 ENTER               1100b
```

Dec

† mode screen

Dec

Sets decimal number base mode. In any number base mode, you can designate an appropriate value as binary, decimal, hexadecimal, or octal by using the **b**, **d**, **h**, or **o** designator, respectively, from the BASE TYPE menu.

In **Dec** number base mode:

```
10+10b+Fh+10o ENTER        35
```

▸Dec

BASE CONV menu

number▸**Dec**
list▸**Dec**
matrix▸**Dec**
vector▸**Dec**

Returns the decimal equivalent of the real or complex argument.

In **Hex** number base mode:

```
2*F ENTER                    1Eh
Ans▸Dec ENTER                30d

{A,B,C,D,E}▸Dec ENTER
                {10d 11d 12d 13d 14d}
```

Degree

† mode screen

Degree

Sets degree angle mode.

In **Degree** angle mode:

```
sin 90 ENTER                  1
sin (π/2) ENTER       .027412133592
```

Degree entry: °

MATH ANGLE menu

number° or **(***expression***)**°

Designates a real *number* or *expression* as degrees, regardless of the angle mode setting.

list°

Designates each element in *list* as degrees.

In **Radian** angle mode:

```
cos 90 ENTER          -.448073616129
cos 90° ENTER                  0

cos {45,90,180}° ENTER
               {.707106781187 0 -1}
```

Deltalst(

LIST OPS menu
(Deltal shows on menu)

Deltalst(*list***)**

Returns a list containing the differences between consecutive real or complex elements in *list*. This subtracts the first element in *list* from the second element, the second from the third, and so on. The resulting list is always one element shorter than *list*.

```
Deltalst({20,30,45,70}) ENTER
                        {10 15 25}
```

DelVar(

‡ program editor CTL menu (DelVa shows on menu)

DelVar(*variable***)**

Deletes the specified user-created *variable* from memory.

You cannot use **DelVar(** to delete a program variable or built-in variable.

```
2→A ENTER                          2
(A+2)² ENTER                      16
DelVar(A) ENTER                 Done
(A+2)² ENTER      ERROR 14 UNDEFINED
```

der1(

CALC menu

der1(*expression*,*variable*,*value***)**

Returns the first derivative of *expression* with respect to *variable* at the real or complex *value*.

```
der1(x^3,x,5) ENTER                75
```

der1(*expression*,*variable***)**

Uses the current value of *variable*.

```
3→x ENTER                           3
der1(x^3,x) ENTER                  27
```

der1(*expression*,*variable*,*list***)**

Returns a list containing the first derivatives at the values specified by the elements in *list*.

```
der1(x^3,x,{5,3}) ENTER       {75 27}
```

der2(

CALC menu

der2(*expression*,*variable*,*value***)**

Returns the second derivative of *expression* with respect to *variable* at the real or complex *value*.

```
der2(x^3,x,5) ENTER                30
```

der2(*expression*,*variable***)**

Uses the current value of *variable*.

```
3→x ENTER                           3
der2(x^3,x) ENTER                  18
```

der2(*expression*,*variable*,*list***)**

Returns a list containing the second derivatives at the values specified by the elements in *list*.

```
der2(x^3,x,{5,3}) ENTER       {30 18}
```

det

MATRX MATH menu

det *squareMatrix*

Returns the determinant of *squareMatrix*. The result is real for a real matrix, complex for a complex matrix.

```
[[1,2][3,4]]→MAT ENTER      [[1 2]
                             [3 4]]
det MAT ENTER                    -2
```

DifEq

† mode screen

DifEq

Sets differential equation graphing mode.

dim

MATRX OPS menu

VECTR OPS menu

dim *matrix*

Returns a list containing the dimensions (number of rows and columns) of a real or complex *matrix*.

dim *vector*

Returns the length (number of elements) of a real or complex *vector*.

```
[[2,7,1][-8,0,1]]→MAT ENTER
                            [[2  7 1]
                             [-8 0 1]]
dim MAT ENTER                   {2 3}
dim [-8,0,1] ENTER                 3
```

→dim

STO▸, then MATRX OPS menu

STO▸, then VECTR OPS menu

{*rows***,***columns***}→dim** *matrixName*

If *matrixName* does not exist, creates a new matrix with the specified dimensions and fills it with zeros.

If *matrixName* exists, redimensions that matrix to the specified dimensions. Existing elements within the new dimensions are not changed; elements outside the new dimensions are deleted. If additional elements are created, they are filled with zeros.

```
[[2,7][-8,0]]→MAT ENTER
                             [[2  7]
                              [-8 0]]
{3,3}→dim MAT ENTER             {3 3}
MAT ENTER                   [[2  7 0]
                             [-8 0 0]
                             [0  0 0]]
```

#ofElements→**dim** *vectorName*

If *vectorName* does not exist, creates a new vector with the specified *#ofElements* and fills it with zeros.

If *vectorName* exists, redimensions that vector to the specified *#ofElements*. Existing elements within the new dimension are not changed; elements outside the new dimension are deleted. If additional elements are created, they are filled with zeros.

```
DelVar(VEC) ENTER                Done
4→dim VEC ENTER                     4
VEC ENTER                  [0 0 0 0]

[1,2,3,4]→VEC ENTER        [1 2 3 4]
2→dim VEC ENTER                     2
VEC ENTER                      [1 2]
3→dim VEC ENTER                     3
VEC ENTER                   [1 2 0]
```

dimL

LIST OPS menu

dimL *list*

Returns the length (number of elements) of a real or complex *list*.

```
dimL {2,7,-8,0} ENTER               4
1/dimL {2,7,-8,0} ENTER           .25
```

→dimL

STO▸, then LIST OPS menu

#ofElements→**dimL** *listName*

If *listName* does not exist, creates a new list with the specified *#ofElements* and fills it with zeros.

If *listName* exists, redimensions that list to the specified *#ofElements*. Existing elements within the new dimension are not changed; elements outside the new dimension are deleted. If additional elements are created, they are filled with zeros.

```
3→dimL NEWLIST ENTER                3
NEWLIST ENTER                 {0 0 0}

{2,7,-8,1}→L1 ENTER        {2 7 -8 1}
5→dimL L1 ENTER                     5
L1 ENTER                  {2 7 -8 1 0}
2→dimL L1 ENTER                     2
L1 ENTER                       {2 7}
```

DirFld

† graph format screen (scroll down to second screen)

DirFld

In **DifEq** graphing mode, turns on direction fields. To turn off direction and slope fields, use **FldOff**.

Disp

‡ program editor I/O menu

Disp *valueA*,*valueB*,*valueC*, ...

Displays each value. The values can include strings and variable names.

Disp

Displays the home screen.

```
10→x ENTER                    10
Disp x^3+3 x-6 ENTER          1024
                              Done
"Hello"→STR ENTER
                        Hello
Disp STR+", Jan" ENTER
                        Hello, Jan
                              Done
```

DispG

† GRAPH menu

‡ program editor I/O menu

DispG

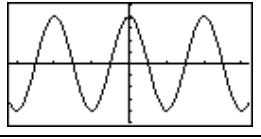
Displays the current graph.

*Function names are case-sensitive. Use **y1**, not **Y1**.*

To select from a list of window variable names, press 2nd [CATLG-VARS] MORE MORE F5.

Program segment in **Func** graphing mode:

```
  ⋮
:y1=4cos x
:-10→xMin:10→xMax
:-5→yMin:5→yMax
:DispG
  ⋮
```



DispT

‡ program editor
I/O menu

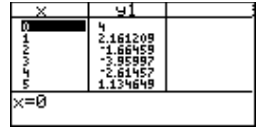
DispT

Displays the table.

*Function names are case-sensitive. Use **y1**, not **Y1**.*

Program segment in **Func** graphing mode:

```
  ⋮
:y1=4cos x
:DispT
  ⋮
```



Division: /

÷

numberA **/** *numberB* or **(***expressionA***) / (***expressionB***)**

Returns one argument divided by another. The arguments can be real or complex.

```
-98/4 ENTER                    -24.5
-98/(4*3) ENTER          -8.16666666667
```

number **/** *list* or **(***expression***) /** *list*

Returns a list in which each element is *number* or *expression* divided by the corresponding element in *list*.

```
100/{10,25,2} ENTER          {10 4 50}
```

list **/** *number* or *list* **/ (***expression***)**
vector **/** *number* or *vector* **/ (***expression***)**

Returns a list or vector in which each element of *list* or *vector* is divided by *number* or *expression*.

```
{120,92,8}/4 ENTER           {30 23 2}
```

In **RectC** complex number mode:

```
[8,1,(5,2)]/2 ENTER
                [(4,0) (.5,0) (2.5,1…
```

listA **/** *listB*

Returns a list in which each element of *listA* is divided by the corresponding element of *listB*. The lists must have the same dimension.

```
{1,2,3}/{4,5,6} ENTER
                          {.25 .4 .5}
```

DMS entry: '

MATH ANGLE menu

In a trig calculation, the result of a DMS entry is treated as degrees in the ***Degree*** *angle mode only. It is treated as radians in* ***Radian*** *angle mode.*

degrees'*minutes*'*seconds*'

Designates the entered angle is in DMS format. *degrees* (≤ 999,999), *minutes* (< 60), and *seconds* (< 60, may have decimal places) must be entered as real numbers, not as variable names or expressions.

Do not use ° and " symbols to specify *degrees* and *seconds*. For example, 5°59' is interpreted as implied multiplication of 5° * 59' according to the current angle mode setting.

```
54'32'30' ENTER                54.5416666667
```

In **Degree** angle mode:

```
cos 54'32'30' ENTER            .580110760699
```

In **Radian** angle mode:

```
cos 54'32'30' ENTER            -.422502666138
```

Do not use the following notation; in **Degree** angle mode:

```
5°59' ENTER                            295
```

▸DMS

MATH ANGLE menu

angle▸**DMS**

Displays *angle* in DMS format. The result is shown in *degrees*°*minutes*'*seconds*" format, even though you use *degrees*'*minutes*'*seconds*' to enter a DMS angle.

In **Degree** angle mode:

```
45.371▸DMS ENTER               45°22'15.6"
54'32'30'*2 ENTER              109.083333333
Ans▸DMS ENTER                  109°5'0"
```

dot(

VECTR MATH menu

dot(*vectorA*,*vectorB***)**

Returns the dot product of two real or complex vectors.

dot([a,b,c],[d,e,f]) returns **a*d+b*e+c*f**.

```
dot([1,2,3],[4,5,6]) ENTER             32
```

DrawDot

† graph format screen

DrawDot

Sets dot graphing format.

DrawF

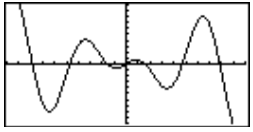
GRAPH DRAW menu

DrawF *expression*

Draws *expression* (in terms of **x**) on the current graph.

In **Func** graphing mode:

```
ZStd:DrawF 1.25 x cos x ENTER
```



DrawLine

† graph format screen

DrawLine

Sets connected line graphing format.

DrEqu(

† GRAPH menu

To enter the ' character for the Q' variables, use the CHAR MISC *menu.*

DrEqu(*xAxisVariable*,*yAxisVariable*,*xList*,*yList*,*tList***)**

In **DifEq** graphing mode, draws the solution to a set of differential equations stored in the **Q'** variables specified by *xAxisVariable* and *yAxisVariable*. If direction fields are off (**FldOff** is selected), the initial values must be stored also.

After the solution is drawn, **DrEqu(** waits for you to move the cursor to a new initial value and press ENTER to draw the new solution.

You then are prompted to press **Y** (to specify another initial value) or **N** (to stop).

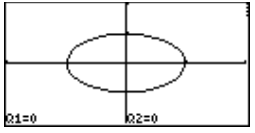
For the last-drawn solution, the **x**, **y**, and **t** values (beginning at their initial values) are stored to *xList*, *yList*, and *tList*, respectively.

DrEqu(*xAxisVariable*,*yAxisVariable***)**

Does not store **x**, **y**, and **t** values for the solution.

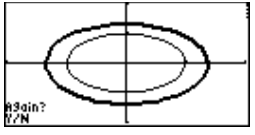
In **DifEq** graphing mode, starting with a **ZStd** graph screen:

```
Q'1=Q2:Q'2=-Q1 ENTER                Done
0→tMin:1→QI1:0→QI2 ENTER              0
DrEqu(Q1,Q2,XL,YL,TL) ENTER
```



Move the cursor to a new initial value.

ENTER



Press **N** to stop graphing. You can then examine **XL**, **YL**, and **TL**.

DrInv

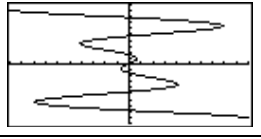
GRAPH DRAW menu

DrInv *expression*

Draws the inverse of *expression* by plotting **x** values on the y-axis and **y** values on the x-axis.

In **Func** graphing mode:

```
ZStd:DrInv 1.25 x cos x ENTER
```



DS<(

‡ program editor CTL menu

:DS<(*variable*,*value***)**
:*command-if-variable≥value*
:*commands*

Decrements *variable* by 1. If the result is < *value*, skips *command-if-variable≥value*.

If the result is ≥ *value*, then *command-if-variable≥value* is executed.

variable cannot be a built-in variable.

Program segment:

```
  ⋮
:9→A
:Lbl Start
:Disp A
:DS<(A,5)
:Goto Start
:Disp "A is now <5"
  ⋮
```

dxDer1

† mode screen

dxDer1

Sets **der1** as the current differentiation type. **der1** differentiates exactly and calculates the value for each function in an expression. It is more accurate than **dxNDer**, but more restrictive in that only certain functions are valid in the expression.

The current differentiation type is used by the **arc(** and **TanLn(** functions, as well as interactive graphing operations dy/dx, dr/dθ, dy/dt, dx/dt, ARC, TanLn, and INFLC.

dxNDer

† mode screen

dxNDer

Sets **nDer** as the current differentiation type. **nDer** differentiates numerically and calculates the value for an expression. It is less accurate than **dxDer1**, but less restrictive in the functions that are valid in the expression.

The current differentiation type is used by the **arc(** and **TanLn(** functions, as well as interactive graphing operations dy/dx, dr/dθ, dy/dt, dx/dt, ARC, TanLn, and INFLC.

e^

2nd [e^x]

e^*power* or **e^(***expression***)**

Returns **e** raised to *power* or *expression*. The argument can be real or complex.

```
e^0 ENTER                    1
```

e^*list*

Returns a list in which each element is **e** raised to the power specified by the corresponding element in *list*.

```
e^{1,0,.5} ENTER
              {2.71828182846 1 1.6…
```

e^*squareMatrix*

The squareMatrix cannot have repeated eigenvalues.

Returns a square matrix that is the matrix exponential of *squareMatrix*. The matrix exponential corresponds to the result calculated using power series or Cayley-Hamilton Theorem techniques. This is *not* the same as simply calculating the exponential of each element.

eigVc

MATRX MATH menu

eigVc *squareMatrix*

The squareMatrix cannot have repeated eigenvalues.

Returns a matrix containing the eigenvectors for a real or complex *squareMatrix*, where each column in the result corresponds to an eigenvalue. The eigenvectors of a real matrix may be complex. Note that an eigenvector is not unique; it may be scaled by any constant factor. TI-86 eigenvectors are normalized.

In **RectC** complex number mode:

```
[[-1,2,5][3,-6,9][2,-5,7]]→MAT
ENTER                    [[-1 2  5]
                          [3  -6 9]
                          [2  -5 7]]
eigVc MAT ENTER
              [[(.800906446592,0) …
               [(-.484028886343,0)…
               [(-.352512270699,0)…
```

eigVl

MATRX MATH menu

eigVl *squareMatrix*

Returns a list of the eigenvalues of a real or complex *squareMatrix*. The eigenvalues of a real matrix may be complex.

In **RectC** complex number mode:

```
[[-1,2,5][3,-6,9][2,-5,7]]→MAT
ENTER                    [[-1 2  5]
                          [3  -6 9]
                          [2  -5 7]]
eigVl MAT ENTER
              {(-4.40941084667,0) …
```

Else

‡ program editor
CTL menu

Refer to syntax information for **If**, beginning on page 305. See the **If:Then:Else:End** syntax.

End

‡ program editor
CTL menu

End

Identifies the end of a **While**, **For**, **Repeat**, or **If-Then-Else** loop.

Eng

† mode screen

Eng

Sets engineering notation mode, in which the power-of-10 exponent is a multiple of 3.

In **Eng** notation mode:

```
123456789 ENTER          123.456789E6
```

In **Normal** notation mode:

```
123456789 ENTER             123456789
```

Eq▶St(

STRNG menu

Eq▶St(*equationVariable*,*stringVariable*)

Converts the contents of *equationVariable* to a string and stores it to *stringVariable*. Be sure to specify an equation variable, not an equation.

To create an equation variable, use an equal sign (=) to define the variable. For example, enter **A=B*C**, not **B*C→A**.

```
A=B*C ENTER                    Done
5→B ENTER                         5
2→C ENTER                         2
A ENTER                          10
Eq▶St(A,STR) ENTER             Done
STR ENTER                   B*C
```

Equal: =

ALPHA [=]

Refer to syntax information for **Assignment** on page 270.

If you use **=** in an expression in which the first argument is not a variable name at the beginning of a line, the **=** is treated as −(.

Example of = treated as −(, where 4=6+1 is evaluated as 4−(6+1):

```
4=6+1 ENTER                      -3
```

For true/false comparison, use == instead:

```
4==6+1 ENTER                      0
```

Equal to: ==

TEST menu

The == operator is used to compare arguments, while = is used to assign a value or expression to a variable.

numberA **==** *numberB*
matrixA **==** *matrixB*
vectorA **==** *vectorB*
stringA **==** *stringB*

Tests whether the condition *argumentA* == *argumentB* is true or false. Numbers, matrices, and vectors can be real or complex. If complex, the magnitude (modulus) of each element is compared. Strings are case-sensitive.

- If true (*argumentA* = *argumentB*), returns **1**.
- If false (*argumentA* ≠ *argumentB*), returns **0**.

```
2+2==2+2 ENTER                    1
2+(2==2)+2 ENTER                  5
[1,2]==[3-2,-1+3] ENTER           1
"A"=="a" ENTER                    0
```

listA **==** *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is = the corresponding element in *listB*.

```
{1,5,9}=={1,-6,9} ENTER      {1 0 1}
```

Euler

† graph format screen (scroll down to second screen)

Euler

In **DifEq** graphing mode, uses an algorithm based on the Euler method to solve differential equations. Typically, **Euler** is less accurate than **RK** but finds the solutions much quicker.

eval

MATH MISC menu

eval *xValue*

Returns a list containing the **y** values of all defined and selected functions evaluated at a real *xValue*.

Remember that built-in equation variables **y1** and **y2** are case-sensitive:

```
y1=x^3+x+5 ENTER                  Done
y2=2 x ENTER                      Done
eval 5 ENTER                  {135 10}
```

evalF(

CALC menu

evalF(*expression*,*variable*,*value***)**

Returns the value of *expression* evaluated with respect to *variable* at a real or complex *value*.

```
evalF(x^3+x+5,x,5) ENTER          135
```

evalF(*expression*,*variable*,*list***)**

Returns a list containing the values of *expression* evaluated with respect to *variable* at each element in *list*.

```
evalF(x^3+x+5,x,{3,5}) ENTER
                              {35 135}
```

Exponent: E

EE

number **E** *power* or **(***expressionA***) E (***expressionB***)**

Returns a real or complex *number* raised to the *power* of 10, where *power* is a real integer such that $-999 < power < 999$. Any *expressions* must evaluate to appropriate values.

```
12.3456789E5 ENTER          1234567.89
(1.78/2.34)E2 ENTER
                          76.0683760684
```

list **E** *power* or *list* **E (***expression***)**

Returns a list in which each element is the corresponding element in *list* raised to the *power* of 10.

```
{6.34,854.6}E3 ENTER
                          {6340 854600}
```

ExpR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

ExpR *xList*,*yList*,*frequencyList*,*equationVariable*

Fits an exponential regression model ($y=ab^x$) to real data pairs in *xList* and *yList* (**y** values must be > 0) and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

ExpR *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

ExpR *xList*,*yList*,*frequencyList*

Stores the regression equation to **RegEq** only.

ExpR *xList*,*yList*

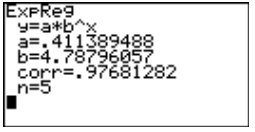
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

ExpR *equationVariable*

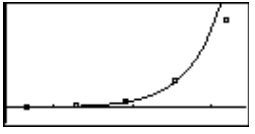
Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

In **Func** graphing mode:

```
{1,2,3,4,5}→L1 ENTER
                              {1 2 3 4 5}
{1,20,55,230,742}→L2 ENTER
                          {1 20 55 230 742}
ExpR L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



ExpR

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

Factorial: !

MATH PROB menu

*number***!** or **(***expression***)!**

Returns the factorial of a real integer or non-integer, where 0 ≤ integer ≤ 449 and 0 ≤ non-integer ≤ 449.9. For a non-integer, the Gamma function is used to find the factorial. An *expression* must evaluate to an appropriate value.

```
6! ENTER                    720
12.5! ENTER          1710542068.32
```

*list***!**

Returns a list in which each element is the factorial of the corresponding element in *list*.

```
{6,7,8}! ENTER    {720 5040 40320}
```

fcstx

† STAT menu

fcstx *yValue*

Based on the current regression equation (**ReqEq**), returns the forecasted **x** at a real *yValue*.

fcsty

† STAT menu

fcsty *xValue*

Based on the current regression equation (**ReqEq**), returns the forecasted **y** at a real *xValue*.

Fill(

LIST OPS menu

MATRX OPS menu

VECTR OPS menu

Fill(*number*,*listName*)
Fill(*number*,*matrixName*)
Fill(*number*,*vectorName*)

Replaces each element in an existing *listName*, *matrixName*, or *vectorName* with a real or complex *number*.

```
{3,4,5}→L1 ENTER                {3 4 5}
Fill(8,L1) ENTER                  Done
L1 ENTER                        {8 8 8}

Fill((3,4),L1) ENTER              Done
L1 ENTER          {(3,4) (3,4) (3,4)}
```

Fix

† mode screen

Fix *integer* or **Fix (***expression***)**

Sets fixed decimal mode for *integer* number of decimal places, where $0 \le integer \le 11$. An *expression* must evaluate to an appropriate integer.

```
Fix 3 ENTER                       Done
π/2 ENTER                        1.571
Float ENTER                       Done
π/2 ENTER                1.57079632679
```

FldOff

† graph format screen (scroll down to second screen)

FldOff

In **DifEq** graphing mode, turns off the slope and direction fields. To turn on slope fields, use **SlpFld**. To turn on direction fields, use **DirFld**.

Float

† mode screen

Float

Sets floating decimal mode.

In **Radian** angle mode:

```
Fix 11 ENTER                      Done
sin (π/6) ENTER        .50000000000
Float ENTER                       Done
sin (π/6) ENTER                   .5
```

fMax(

CALC menu

fMax(*expression*,*variable*,*lower*,*upper***)**

Returns the value at which a local maximum of *expression* with respect to *variable* occurs, between real *lower* and *upper* values for *variable*.

The tolerance is controlled by the built-in variable **tol**, whose default is 1E-5. To view or set **tol,** press 2nd [MEM] F4 to display the tolerance editor.

```
fMax(sin x,x,-π,π) ENTER
                1.57079632598
```

fMin(

CALC menu

fMin(*expression*,*variable*,*lower*,*upper***)**

Returns the value at which a local minimum of *expression* with respect to *variable* occurs, between real *lower* and *upper* bounds for *variable*.

The tolerance is controlled by the built-in variable **tol**, whose default is 1E-5. To view or set **tol,** press 2nd [MEM] F4 to display the tolerance editor.

```
fMin(sin x,x,-π,π) ENTER
               -1.57079632691
```

fnInt(

CALC menu

fnInt(*expression*,*variable*,*lower*,*upper***)**

Returns the numerical function integral of *expression* with respect to *variable*, between real *lower* and *upper* bounds for *variable*.

The tolerance is controlled by the built-in variable **tol**, whose default is 1E-5. To view or set **tol,** press 2nd [MEM] F4 to display the tolerance editor.

```
fnInt(x²,x,0,1) ENTER
                .333333333333
```

FnOff

† GRAPH VARS menu

FnOff *function#*,*function#*, ...

Deselects the specified equation function numbers.

```
FnOff 1,3 ENTER          Done
```

FnOff

```
FnOff ENTER
Done
```

Deselects all equation function numbers.

FnOn

† GRAPH VARS menu

FnOn *function#*,*function#*, ...

```
FnOn 1,3 ENTER
Done
```

Selects the specified equation function numbers, in addition to any others already selected.

FnOn

```
FnOn ENTER
Done
```

Selects all equation function numbers.

For(

‡ program editor CTL menu

:For(*variable*,*begin*,*end*,*step***)** or **:For(***variable*,*begin*,*end***)**
:loop *:loop*
:End **:End**
:commands *:commands*

Executes the commands in *loop* iteratively, where the number of repetitions is controlled by *variable*. The first time through the loop, *variable* = *begin*. At the **End** of the loop, *variable* is incremented by *step*. The loop is repeated until *variable* > *end*. If you do not specify *step*, the default is 1.

You can specify values such that *begin* > *end*. If so, be sure to specify a negative *step*.

Program segment:

```
  ⋮
For(A,0,8,2)
Disp A²
End
  ⋮
```

Displays 0, 4, 16, 36, and 64.

```
  ⋮
For(A,0,8)
Disp A²
End
  ⋮
```

Displays 0, 1, 4, 9, 16, 25, 36, 49, and 64.

Form(

LIST OPS menu

Form("*formula***",***listName***)**

Generates the contents of *listName* automatically, based on the attached *formula*. If you express *formula* in terms of a list, you can generate one list based on the contents of another.

The contents of *listName* are updated automatically if you edit *formula* or edit a list referenced in *formula*.

```
{1,2,3,4}→L1 ENTER          {1 2 3 4}
Form("10*L1",L2) ENTER          Done
L2 ENTER                 {10 20 30 40}

{5,10,15,20}→L1 ENTER
                            {5 10 15 20}
L2 ENTER               {50 100 150 200}

Form("L1/5",L2) ENTER           Done
L2 ENTER                    {1 2 3 4}
```

fPart

MATH NUM menu

fPart *number* or **fPart (***expression***)**

Returns the fractional part of a real or complex *number* or *expression*.

fPart *list*
fPart *matrix*
fPart *vector*

Returns a list, matrix, or vector in which each element is the fractional part of the corresponding element in the specified argument.

```
fPart 23.45 ENTER                .45

fPart (-17.26*8) ENTER           -.08

[[1,-23.45][-99.5,47.15]]→MAT
ENTER                  [[1     -23.45]
                        [-99.5 47.15 ]]

fPart MAT ENTER          [[0   -.45]
                          [-.5 .15 ]]
```

▸Frac

MATH MISC menu

number▸**Frac**

Displays a real or complex *number* as its rational equivalent, a fraction reduced to its simplest terms.

If *number* cannot be simplified or if the denominator is more than four digits, the decimal equivalent is returned.

```
1/3+2/7 ENTER            .619047619048
Ans▸Frac ENTER                  13/21
```

list▸**Frac**
matrix▸**Frac**
vector▸**Frac**

Returns a list, matrix, or vector in which each element is the rational equivalent of the corresponding element in the argument.

```
{1/2+1/3,1/6-3/8}→L1 ENTER
                {.833333333333 -.208…
Ans▸Frac ENTER          {5/6 -5/24}
```

Func

† mode screen

Func

Sets function graphing mode.

gcd(

MATH MISC menu

gcd(*integerA*,*integerB***)**

Returns the greatest common divisor of two nonnegative integers.

```
gcd(18,33) ENTER                  3
```

gcd(*listA*,*listB***)**

Returns a list in which each element is the gcd of the two corresponding elements in *listA* and *listB*.

```
gcd({12,14,16},{9,7,5}) ENTER
                          {3 7 1}
```

Get(

‡ program editor I/O menu

Get(*variable***)**

Gets data from a CBL or CBR System or another TI-86 and stores it to *variable*.

getKy

‡ program editor
I/O menu

getKy

Returns the key code for the last key pressed. If no key has been pressed, **getKy** returns **0**. Refer to the TI-86 key code diagram in Chapter 16.

Program:

```
PROGRAM:CODES
:Lbl TOP
:getKy→KEY
:While KEY==0
: getKy→KEY
:End
:Disp KEY
:Goto TOP
```

To break the program, press [ON] and then [F5].

Goto

‡ program editor
CTL menu

Goto *label*

Transfers (branches) program control to the *label* specified by an existing **Lbl** instruction.

Program segment:

```
  ⋮
:0→TEMP:1→J
:Lbl TOP
:TEMP+J→TEMP
:If J<10
:Then
: J+1→J
: Goto TOP
:End
:Disp TEMP
  ⋮
```

Greater than: >

TEST menu

numberA **>** *numberB* or **(***expressionA***) > (***expressionB***)**

Tests whether the condition is true or false. The arguments must be real numbers.

- If true (*numberA* > *numberB*), returns **1**.
- If false (*numberA* ≤ *numberB*), returns **0**.

```
2>0 [ENTER]                    1
88>123 [ENTER]                 0
-5>-5 [ENTER]                  0
(20*5/2)>(18*2) [ENTER]        1
```

number **>** *list*

Returns a list of **1**s and/or **0**s to indicate if *number* is > the corresponding element in *list*.

```
1>{1,-6,10} ENTER          {0 1 0}
```

listA **>** *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is > the corresponding element in *listB*.

```
{1,5,9}>{1,-6,10} ENTER    {0 1 0}
```

Greater than or equal to: ≥

TEST menu

numberA ≥ *numberB* or **(***expressionA***)** ≥ **(***expressionB***)**

Tests whether the condition is true or false. The arguments must be real numbers.

- If true (*numberA* ≥ *numberB*), returns **1**.
- If false (*numberA* < *numberB*), returns **0**.

```
2≥0 ENTER                  1
88≥123 ENTER               0
-5≥-5 ENTER                1
(20*5/2)≥(18*2) ENTER      1
```

number ≥ *list*

Returns a list of **1**s and/or **0**s to indicate if *number* is ≥ the corresponding element in *list*.

```
1≥{1,-6,10} ENTER          {1 1 0}
```

listA ≥ *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is ≥ the corresponding element in *listB*.

```
{1,5,9}≥{1,-6,10} ENTER    {1 1 0}
```

GridOff

† graph format screen

GridOff

Turns off grid format so that grid points are not displayed.

GridOn

† graph format screen

GridOn

Turns on grid format so that grid points are displayed in rows and columns corresponding to the tick marks on each axis.

GrStl(

CATALOG

GrStl(*function#*,*graphStyle#*)

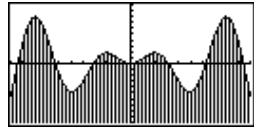
Sets the graph style for *function#*. For *graphStyle#*, specify an integer from **1** through **7**:

1 = ⟍ (line) **4** = ◣ (below) **7** = ⋱ (dot)
2 = ⟍ (thick) **5** = ⟍ (path)
3 = ⟍ (above) **6** = ⟍ (animate)

Depending on the graphing mode, some graph styles may not be available.

In **Func** graphing mode:

```
y1=x sin x ENTER                Done
GrStl(1,4) ENTER                Done
ZStd ENTER
```



h

BASE TYPE menu

*integer***h**

Designates a real *integer* as hexadecimal, regardless of the number base mode setting.

In **Dec** number base mode:

```
10h ENTER                       16
10h+10 ENTER                    26
```

Hex

† mode screen

Hex

Sets hexadecimal number base mode. Results are displayed with the **h** suffix. In any number base mode, you can designate an appropriate value as binary, decimal, hexadecimal, or octal by using the **b**, **d**, **h**, or **o** designator, respectively, from the BASE TYPE menu.

To enter hexadecimal numbers **A** through **F**, use the BASE A-F menu. Do not use ALPHA to type a letter.

In **Hex** number base mode:

```
F+10b+10o+10d ENTER             23h
```

▸Hex

BASE CONV menu

number▸**Hex**
list▸**Hex**
matrix▸**Hex**
vector▸**Hex**

Returns the hexadecimal equivalent of the real or complex argument.

In **Bin** number base mode:

```
1010*1110 ENTER                10001100b
Ans▸Hex ENTER                       8Ch

{100,101,110}▸Hex ENTER
                          {4h 5h 6h}
```

Hist

† STAT DRAW menu

Hist *xList*,*frequencyList*

Draws a histogram on the current graph, using the real data in *xList* and the frequencies in *frequencyList*.

Hist *xList*

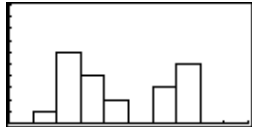
Uses frequencies of 1.

Hist

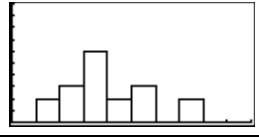
Uses the data in built-in variables **xStat** and **fStat**. These variables must contain valid data of the same dimension; otherwise, an error occurs.

Starting with a **ZStd** graph screen:

```
{1,2,3,4,6,7}→XL ENTER
                        {1 2 3 4 6 7}
{1,6,4,2,3,5}→FL ENTER
                      {1 6 4 2 3 5}
0→xMin:0→yMin ENTER                 0
Hist XL,FL ENTER
```



```
{1,1,2,2,2,3,3,3,3,3,3,4,4,5,5,5,
7,7}→XL ENTER
              {1 1 2 2 2 3 3 3 3 3 …
ClDrw:Hist XL ENTER
```



Horiz

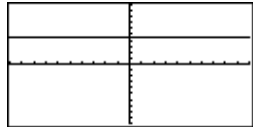
† GRAPH DRAW menu

Horiz *yValue*

Draws a horizontal line on the current graph at *yValue*.

In a **ZStd** graph screen:

```
Horiz 4.5 ENTER
```



IAsk

CATALOG

IAsk

Sets the table so that the user can enter individual values for the independent variable.

IAuto

CATALOG

IAuto

Sets the table so that the TI–86 generates the independent-variable values automatically, based on values entered for **TblStart** and **ΔTbl**.

ident

MATRX OPS menu

ident *dimension*

Returns the identity matrix of *dimension* rows × *dimension* columns.

```
ident 4 ENTER
```

```
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

If

‡ program editor CTL menu

:If *condition*
:*command-if-true*
:*commands*

If *condition* is true, executes *command-if-true*. Otherwise, skips *command-if-true*. The *condition* is true if it evaluates to any nonzero number, or false if it evaluates to zero.

To execute multiple commands if *condition* is true, use **If:Then:End** instead.

Program segment:

```
  ⋮
:If x<0
:Disp "x is negative"
  ⋮
```

:If *condition*
:Then
:*commands-if-true*
:End
:*commands*

If *condition* is true (nonzero), executes *commands-if-true* from **Then** to **End**. Otherwise, skips *commands-if-true* and continues with the next command following **End**.

Program segment:

```
  ⋮
:If x<0
:Then
: Disp "x is negative"
: abs(x)→x
:End
  ⋮
```

:If *condition*
:Then
:*commands-if-true*
:Else
:*commands-if-false*
:End
:*commands*

If *condition* is true (nonzero), executes *commands-if-true* from **Then** to **Else** and then continues with the next command following **End**.

If *condition* is false (zero), executes *commands-if-false* from **Else** to **End** and then continues with the next command following **End**.

Program segment:

```
  ⋮
:If x<0
:Then
: Disp "x is negative"
:Else
: Disp "x is positive or zero"
:End
  ⋮
```

imag

CPLX menu

imag (*complexNumber***)**

Returns the imaginary (nonreal) part of *complexNumber*. The imaginary part of a real number is always 0.

imag (*real*,*imaginary***)** returns *imaginary*.
imag (*magnitude*∠*angle***)** returns *magnitude* **sin** *angle*.

```
imag (3,4) ENTER                    4
imag (3∠4) ENTER      -2.27040748592
```

imag *complexList*
imag *complexMatrix*
imag *complexVector*

Returns a list, matrix, or vector in which each element is the imaginary part of the original argument.

```
imag {-2,(3,4),(3∠4)} ENTER
                {0 4 -2.27040748592}
```

InpSt

‡ program editor I/O menu

InpSt *promptString*,*variable*

Pauses a program, displays *promptString*, and waits for the user to enter a response. The response is stored to *variable* always as a string. When entering the response, the user should not enter quotation marks.

To prompt for a number or expression instead of a string, use **Input**.

InpSt *variable*

Displays **?** as the prompt.

Program segment:

```
    ⋮
:InpSt "Enter your name:",STR
    ⋮
```

Input

‡ program editor I/O menu

Input *promptString*,*variable*

Pauses a program, displays *promptString*, and waits for the user to enter a response. The response is stored to *variable* in the form in which the user enters it.

- A number or expression is stored as a number or expression.
- A list, vector, or matrix is stored as a list, vector, or matrix.
- An entry enclosed in " marks is stored as a string.

Input *variable*

Displays **?** as the prompt.

Program segment:

```
    ⋮
:Input "Enter test score:",SCR
    ⋮
```

Input

Pauses a program, displays the graph screen, and lets the user update **x** and **y** (or **r** and θ in **PolarGC** graph format) by moving the free-moving cursor. To resume the program, press ENTER.

Program segment in **RectGC** graph format:

```
   ⋮
:Input
:Disp x,y
   ⋮
```

Input "CBLGET",*variable*

Receives list data sent from a CBL or CBR System and stores it to *variable* on the TI-86. Use this **"CBLGET"** syntax for both CBL and CBR.

You can receive data also by using **Get(** as described on page 299.

```
Input "CBLGET",L1 ENTER        Done
```

int

MATH NUM menu

int *number* or **int (***expression***)**

Returns the largest integer ≤ *number* or *expression*. The argument can be real or complex.

For a negative non-integer, **int** returns the integer that is one less than the integer part of the number. To return the exact integer part, use **iPart** instead.

```
int 23.45 ENTER                23
int -23.45 ENTER               -24
```

int *list*
int *matrix*
int *vector*

Returns a list, matrix, or vector in which each element is the largest integer less than or equal to the corresponding element in the specified argument.

```
[[1.25,-23.45][-99,47.15]]→MAT
ENTER                [[1.25 -23.45]
                      [-99  47.15 ]]
int MAT ENTER          [[1   -24]
                       [-99 47 ]]
```

inter(

† MATH menu

inter(*x1*,*y1*,*x2*,*y2*,*xValue*)**

Calculates the line through points (*x1*,*y1*) and (*x2*,*y2*) and then interpolates or extrapolates a **y** value for the specified *xValue*.

Using points (3,5) and (4,4), find the y value at x=1:

```
inter(3,5,4,4,1) ENTER                    7
```

inter(*y1*,*x1*,*y2*,*x2*,*yValue*)**

Interpolates or extrapolates an **x** value for the specified *yValue*. Notice that points (*x1*,*y1*) and (*x2*,*y2*) must be entered as (*y1*,*x1*) and (*y2*,*x2*).

Using points (-4,-7) and (2,6), find the x value at y=10:

```
inter(-7,-4,6,2,10) ENTER
                          3.84615384615
```

Inverse: $^{-1}$

2nd [x^{-1}]

number^{-1} or (*expression*)$^{-1}$

Returns 1 divided by a real or complex *number*, where *number* ≠ 0.

```
5^-1 ENTER                                .2
(10*6)^-1 ENTER                  .016666666667
```

list^{-1}

Returns a list in which each element is 1 divided by the corresponding element in *list*.

```
{-.5,10,2/8}^-1 ENTER                {-2 .1 4}
```

squareMatrix^{-1}

Returns an inverted *squareMatrix*, where det ≠ 0.

```
[[1,2][3,4]]^-1 ENTER          [[-2  1  ]
                                [1.5 -.5]]
```

iPart

MATH NUM menu

iPart *number* or **iPart (***expression***)**

Returns the integer part of *number* or *expression*. The argument can be real or complex.

```
iPart 23.45 ENTER                         23
iPart -23.45 ENTER                       -23
```

iPart *list*
iPart *matrix*
iPart *vector*

Returns a list, matrix, or vector in which each element is the integer part of the corresponding element in the specified argument.

```
[[1.25,-23.45][-99.5,47.15]]→MAT
ENTER                [[1.25  -23.45]
                      [-99.5 47.15 ]]

iPart MAT ENTER           [[1   -23]
                           [-99 47 ]]
```

IS>(

‡ program editor CTL menu

:IS>(*variable*,*value*)
:*command-if-variable*≤*value*
:*commands*

Increments *variable* by 1. If the result is > *value*, skips *command-if-variable*≤*value*.

If the result is ≤ *value*, then *command-if-variable*≤*value* is executed.

variable cannot be a built-in variable.

Program segment:

```
  ⋮
:0→A
:Lbl Start
:Disp A
:IS>(A,5)
:Goto Start
:Disp "A is now >5"
  ⋮
```

LabelOff

† graph format screen

LabelOff

Turns off axes labels.

LabelOn

† graph format screen

LabelOn

Turns on axes labels.

Lbl

‡ program editor CTL menu

Lbl *label*

Creates a *label* of up to eight characters. A program can use a **Goto** instruction to transfer control (branch) to a specified label.

***InpSt** stores input as a string, so be sure to store a string to the **password** variable.*

Program segment, assuming a correct password has already been stored to the **password** variable:

```
  :
:Lbl Start
:InpSt "Enter password:",PSW
:If PSW≠password
:Goto Start
:Disp "Welcome"
  :
```

lcm(

MATH MISC menu

lcm(*integerA*,*integerB***)**

Returns the least common multiple of two nonnegative integers.

```
lcm(5,2) ENTER                10
lcm(6,9) ENTER                18
lcm(18,33) ENTER             198
```

LCust(

‡ program editor CTL menu

LCust(*item#*,**"***title***" [**,*item#*,**"***title***",** ...**])**

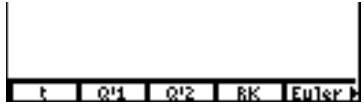
Loads (defines) the TI−86's custom menu, which is displayed when the user presses CUSTOM. The menu can have up to 15 items, shown in three groups of five items. For each *item#*/*title* pair:

- *item#* — integer from 1 through 15 that identifies the item's position in the menu. The item numbers must be specified in order, but you can skip numbers.
- **"***title***"** — string with up to 8 characters (not counting the quotes) that will be pasted to the current cursor location when the item is selected. This can be a variable name, expression, function name, program name, or any text string.

Program segment:

```
  :
:LCust(1,"t",2,"Q'1",3,"Q'2",4,"R
K",5,"Euler",6,"QI1",7,"QI2",8,"t
Min")
  :
```

After executed and when the user presses CUSTOM:



Less than: <

TEST menu

numberA < *numberB* or (*expressionA*) < (*expressionB*)

Tests whether the condition is true or false. The arguments must be real numbers.

- If true (*numberA* < *numberB*), returns **1**.
- If false (*numberA* ≥ *numberB*), returns **0**.

```
2<0 ENTER                                   0
88<123 ENTER                                1
-5<-5 ENTER                                 0
(20*5/2)<(18*3) ENTER                       1
```

number < *list*

Returns a list of **1**s and/or **0**s to indicate if *number* is < the corresponding element in *list*.

```
1<{1,-6,10} ENTER                           {0 0 1}
```

listA < *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is < the corresponding element in *listB*.

```
{1,5,9}<{1,-6,10} ENTER                     {0 0 1}
```

Less than or equal to: ≤

TEST menu

numberA ≤ *numberB* or (*expressionA*) ≤ (*expressionB*)

Tests whether the condition is true or false. The arguments must be real numbers.

- If true (*numberA* ≤ *numberB*), returns **1**.
- If false (*numberA* > *numberB*), returns **0**.

```
2≤0 ENTER                                   0
88≤123 ENTER                                1
-5≤-5 ENTER                                 1
(20*5/2)≤(18*3) ENTER                       1
```

number ≤ *list*

Returns a list of **1**s and/or **0**s to indicate if *number* is ≤ the corresponding element in *list*.

```
1≤{1,-6,10} ENTER                           {1 0 1}
```

listA ≤ *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is ≤ the corresponding element in *listB*.

```
{1,5,9}≤{1,-6,10} ENTER                     {1 0 1}
```

LgstR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

LgstR *returns a* **tolMet** *value that indicates if the result meets the TI-86's internal tolerance.*

- *If* **tolMet**=**1**, *the result is within the internal tolerance.*
- *If* **tolmet**=**0**, *the result is outside the internal tolerance, although it may be useful for general purposes.*

LgstR [*iterations*,]*xList*,*yList*,*frequencyList*,*equationVariable*

Fits a logistic regression model ($y=a/(1+be^{cx})+d$) to real data pairs in *xList* and *yList* and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**. The equation's coefficients always are stored as a list to built-in variable **PRegC**.

The number of *iterations* is optional. If omitted, 64 is the default. A large number of *iterations* may produce more accurate results but may require longer calculation times. A smaller number may produce less accurate results but with shorter calculation times.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

LgstR [*iterations*,]*xList*,*yList*,*equationVariable*

Uses frequencies of 1.

LgstR [*iterations*,]*xList*,*yList*,*frequencyList*

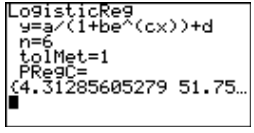
Stores the regression equation to **RegEq** only.

LgstR [*iterations*,]*xList*,*yList*

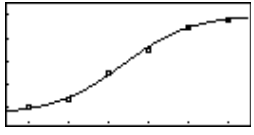
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                      {1 2 3 4 5 6}
{1,1.3,2.5,3.5,4.5,4.8}→L2 ENTER
              {1 1.3 2.5 3.5 4.5 4…
LgstR L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER          Done
ZData ENTER
```



LgstR [*iterations*,]*equationVariable*

Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

LgstR [*iterations*]

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

Line(

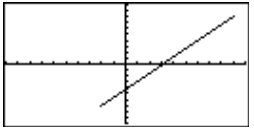
† GRAPH DRAW menu

Line(*x1*,*y1*,*x2*,*y2***)**

Draws a line from point (*x1*,*y1*) to (*x2*,*y2*).

In **Func** graphing mode and a **ZStd** graph screen:

```
Line(-2,-7,9,8) ENTER
```



LinR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

LinR *xList*,*yList*,*frequencyList*,*equationVariable*

Fits a linear regression model (y=a+bx) to real data pairs in *xList* and *yList* and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

LinR *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

LinR *xList*,*yList*,*frequencyList*

Stores the regression equation to **RegEq** only.

LinR *xList*,*yList*

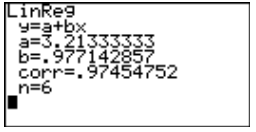
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

LinR *equationVariable*

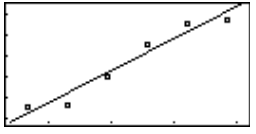
Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                        {1 2 3 4 5 6}
{4.5,4.6,6,7.5,8.5,8.7}→L2 ENTER
                  {4.5 4.6 6 7.5 8.5 8.7}
LinR L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



LinR

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

List entry: **{ }**

LIST menu

{*element1***,***element2***, ...}**

Defines a list in which each element is a real or complex number or variable.

```
{1,2,3}→L1 ENTER                {1 2 3}
```

In **RectC** complex number mode:

```
{3,(2,4),8*2}→L2 ENTER
                  {(3,0) (2,4) (16,0)}
```

---

li▸vc

LIST OPS menu

VECTR OPS menu

li▸vc *list*

Returns a vector converted from a real or complex *list*.

```
li▸vc {2,7,-8,0} ENTER
                          [2 7 -8 0]
```

---

ln

LN

ln *number* or **ln (***expression***)**

Returns the natural logarithm of a real or complex *number* or *expression*.

```
ln 2 ENTER                  .69314718056
ln (36.4/3) ENTER       2.49595648597
```

In **RectC** complex number mode:

```
ln -3 ENTER    (1.09861228867,3.141…
```

ln *list*

Returns a list in which each element is the natural logarithm of the corresponding element in *list*.

```
ln {2,3} ENTER
                {.69314718056 1.0986…
```

---

lngth

STRNG menu

lngth *string*

Returns the length (number of characters) of *string*. The character count includes spaces but not quotation marks.

```
lngth "The answer is:" ENTER      14
"The answer is:"→STR ENTER
                          The answer is:
lngth STR ENTER                   14
```

LnR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

LnR *xList*,*yList*,*frequencyList*,*equationVariable*

Fits a logarithmic regression model (y=a+b ln x) to the real data pairs in *xList* and *yList* (**x** values must be > 0) and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

LnR *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

LnR *xList*,*yList*,*frequencyList*

Stores the regression equation to **RegEq** only.

LnR *xList*,*yList*

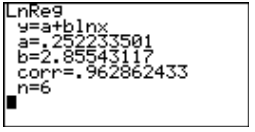
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

LnR *equationVariable*

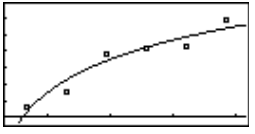
Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                        {1 2 3 4 5 6}
{.6,1.5,3.8,4.2,4.3,5.9}→L2 ENTER
                  {.6 1.5 3.8 4.2 4.3 5.9}
LnR L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



LnR

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

log

LOG

log *number* or **log (***expression***)**

Returns the logarithm of a real or complex *number* or *expression*, where:

$10^{logarithm} = number$

```
log 2 ENTER                .301029995664
log (36.4/3) ENTER         1.08398012893
```

In **RectC** complex number mode:

```
log (3,4) ENTER
                  (.698970004336,.4027…
```

log *list*

Returns a list in which each element is the logarithm of the corresponding element in *list*.

In **RectC** complex number mode:

```
log {-3,2} ENTER
                  {(.47712125472,1.364…
```

LU(

MATRX MATH menu

LU(*matrix***,***lMatrixName***, *uMatrixName***, *pMatrixName***)**

Calculates the Crout LU (lower-upper) decomposition of a real or complex *matrix*. The lower triangular matrix is stored in *lMatrixName*, the upper triangular matrix in *uMatrixName*, and the permutation matrix (which describes the row swaps done during the calculation) in *pMatrixName*.

lMatrixName * *uMatrixName* = *pMatrixName* * *matrix*

```
[[6,12,18][5,14,31][3,8,18]]
→MAT ENTER                 [[6 12 18]
                            [5 14 31]
                            [3 8  18]]

LU(MAT,L,U,P) ENTER                Done

L ENTER                     [[6 0 0]
                            [5 4 0]
                            [3 2 1]]

U ENTER                     [[1 2 3]
                            [0 1 4]
                            [0 0 1]]

P ENTER                     [[1 0 0]
                            [0 1 0]
                            [0 0 1]]
```

Matrix entry: [ ]

2nd [[ ] and 2nd [ ]]

[[*row1***][***row2***] ...]**

Defines a matrix entered row-by-row in which each element is a real or complex number or variable.

Enter each **[***row***]** as **[***element*,*element*, ... **]**.

```
[[1,2,3][4,5,6]]→MAT ENTER
                      [[1 2 3]
                       [4 5 6]]
```

max(

MATH NUM menu

max(*numberA*,*numberB***)**

Returns the larger of two real or complex numbers.

```
max(2.3,1.4) ENTER                    2.3
```

max(*list***)**

Returns the largest element in *list*.

```
max({1,9,π/2,e^2}) ENTER               9
```

max(*listA*,*listB***)**

Returns a list in which each element is the larger of the corresponding elements in *listA* and *listB*.

```
max({1,10},{2,9}) ENTER          {2 10}
```

MBox

† STAT DRAW menu

MBox *xList*,*frequencyList*

Draws a modified box plot on the current graph, using the real data in *xList* and the frequencies in *frequencyList*.

MBox *xList*

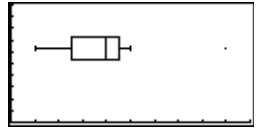
Uses frequencies of 1.

MBox

Uses the data in built-in variables **xStat** and **fStat**. These variables must contain valid data of the same dimension; otherwise, an error occurs.

Starting with a **ZStd** graph screen:

```
{1,2,3,4,5,9}→XL ENTER
                    {1 2 3 4 5 9}
{1,1,1,4,1,1}→FL ENTER
                    {1 1 1 4 1 1}
0→xMin:0→yMin ENTER                0
MBox XL,FL ENTER
```



Menu(

‡ program editor
CTL menu

Menu(*item#*,"*title1*",*label1*[, … ,*item#*,"*title15*",*label15*])

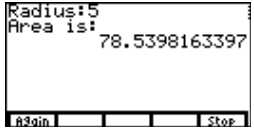
Generates a menu of up to 15 items during program execution. Menus are displayed as three groups of five items. For each item:

- *item#* — integer from 1 through 15 that identifies this item's position in the menu.
- "*title*" — text string that will be displayed for this item on the menu. Typically, use from 1 through 5 characters; additional characters may not be seen on the menu.
- *label* — valid label to which program execution will branch when the user selects this item.

Program segment:

```
  ⋮
:Lbl A
:Input "Radius:",RADIUS
:Disp "Area is:",π*RADIUS²
:Menu(1,"Again",A,5,"Stop",B)
:Lbl B
:Disp "The End"
```

Example when executed:



min(

MATH NUM menu

min(*numberA*,*numberB*)

Returns the smaller of two real or complex numbers.

```
min(3,-5) ENTER                    -5
min(-5.2,-5.3) ENTER               -5.3
min(5,2+2) ENTER                   4
```

min(*list*)

Returns the smallest element in *list*.

```
min({1,3,-5}) ENTER                -5
```

min(*listA*,*listB*)

Returns a list in which each element is the smaller of the corresponding elements in *listA* and *listB*.

```
min({1,2,3},{3,2,1}) ENTER
                                   {1 2 1}
```

mod(

MATH NUM menu

mod(*numberA*,*numberB*)

Returns *numberA* modulo *numberB*. The arguments must be real.

```
mod(7,0) ENTER                     7
mod(7,3) ENTER                     1
mod(-7,3) ENTER                    2
mod(7,-3) ENTER                    -2
mod(-7,-3) ENTER                   -1
```

mRAdd(

MATRX OPS menu

mRAdd(*number*,*matrix*,*rowA*,*rowB***)**

Returns the result of a "multiply and add row" matrix operation, where:

a. *rowA* of a real or complex *matrix* is multiplied by a real or complex *number*.

b. The results are added to (and then stored in) *rowB*.

```
[[5,3,1][2,0,4][3,-1,2]]→MAT
ENTER                    [[5 3  1]
                          [2 0  4]
                          [3 -1 2]]

mRAdd(5,MAT,2,3) ENTER
                       [[5  3  1 ]
                        [2  0  4 ]
                        [13 -1 22]]
```

Multiplication: *

×

numberA * *numberB*

Returns the product of two real or complex numbers.

number * *list* or *list* * *number*
number * *matrix* or *matrix* * *number*
number * *vector* or *vector* * *number*

Returns a list, matrix, or vector in which each element is *number* multiplied by the corresponding element in *list*, *matrix*, or *vector*.

listA * *listB*

Returns a list in which each element of *listA* is multiplied by the corresponding element of *listB*. The lists must have the same dimension.

matrix * *vector*

Returns a vector in which *matrix* is multiplied by *vector*. The number of columns in *matrix* must equal the number of elements in *vector*.

```
2*5 ENTER                          10

4*{10,9,8} ENTER          {40 36 32}
```

In **RectC** complex number mode:

```
[8,1,(5,2)]*3 ENTER
              [(24,0) (3,0) (15,6)]

{1,2,3}*{4,5,6} ENTER      {4 10 18}

[[1,2,3][4,5,6]]→MAT ENTER
                           [[1 2 3]
                            [4 5 6]]
MAT*[7,8,9] ENTER
                            [50 122]
```

matrixA ***** *matrixB*

Returns a matrix in which *matrixA* is multiplied by *matrixB*. The number of columns in *matrixA* must equal the number of rows in *matrixB*.

```
[[2,2][3,4]]→MATA ENTER      [[2 2]
                              [3 4]]
[[1,2,3][4,5,6]]→MATB ENTER
                           [[1 2 3]
                            [4 5 6]]
MATA*MATB ENTER          [[10 14 18]
                          [19 26 33]]
```

multR(

MATRX OPS menu

multR(*number*,*matrix*,*row***)**

Returns the result of a "row multiplication" matrix operation, where:

a. The specified *row* of a real or complex *matrix* is multiplied by a real or complex *number*.

b. The results are stored in the same *row*.

```
[[5,3,1][2,0,4][3,-1,2]]→MAT
ENTER                      [[5 3  1]
                            [2 0  4]
                            [3 -1 2]]
multR(5,MAT,2) ENTER
                         [[5  3  1 ]
                          [10 0  20]
                          [3  -1 2 ]]
```

nCr

MATH PROB menu

items **nCr** *number*

Returns the number of combinations of *items* (**n**) taken *number* (**r**) at a time. Both arguments must be real nonnegative integers.

```
5 nCr 2 ENTER                    10
```

nDer(

CALC menu

To view or set the value for δ, press 2nd [MEM] F4 *to display the tolerance screen.*

nDer(*expression*,*variable*,*value*)

Returns an approximate numerical derivative of *expression* with respect to *variable* evaluated at a real or complex *value*. The approximate numerical derivative is the slope of the secant line through the points:

(*value*−δ,f(*value*−δ)) and (*value*+δ,f(*value*+δ))

As the step value δ gets smaller, the approximation usually gets more accurate.

For δ=.001:

```
nDer(x^3,x,5) ENTER                    75.000001
```

For δ=1E-4:

```
nDer(x^3,x,5) ENTER                          75
```

nDer(*expression*,*variable*)

Uses the current value of *variable*.

```
5→x ENTER                                     5
nDer(x^3,x) ENTER                            75
```

Negation: -

(−)

-*number* or -**(***expression***)**
-*list*
-*matrix*
-*vector*

Returns the negative of the real or complex argument.

```
-2+5 ENTER                                    3
-(2+5) ENTER                                 -7
-{0,-5,5} ENTER                        {0 5 -5}
```

norm

MATRX MATH menu

VECTR MATH menu

norm *matrix*

Returns the Frobenius norm of a real or complex *matrix*, calculated as:

$$\sqrt{\Sigma(real^2+imaginary^2)}$$

where the sum is over all elements.

```
[[1,-2][-3,4]]→MAT ENTER
                                       [[1 -2]
                                       [-3 4 ]]
norm MAT ENTER                  5.47722557505
```

norm *vector*

Returns the length of a real or complex *vector*, where:

norm [a,b,c] returns $\sqrt{a^2+b^2+c^2}$.

```
norm [3,4,5] ENTER
                    7.07106781187
```

norm *number* or **norm (***expression***)**
norm *list*

Returns the absolute value of a real or complex *number* or *expression*, or of each element in *list*.

```
norm -25 ENTER                  25
```

In **Radian** angle mode:

```
norm {-25,cos -(π/3)} ENTER
                          {25 .5}
```

Normal

† mode screen

Normal

Sets normal notation mode.

In **Eng** notation mode:

```
123456789 ENTER        123.456789E6
```

In **Sci** notation mode:

```
123456789 ENTER        1.23456789E8
```

In **Normal** notation mode:

```
123456789 ENTER          123456789
```

not

BASE BOOL menu

not *integer*

Returns the one's complement of a real *integer*. Internally, *integer* is represented as a 16-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one's complement.

For example, **not** 78:

78 = 0000000001001110**b**
1111111110110001**b** (one's complement)
└— *Sign bit; 1 indicates a negative number*

To find the magnitude of a negative binary number, determine its two's complement (take the one's complement and then add 1). For example:

1111111110110001**b** = one's complement of 78
0000000001001110**b** (one's complement)
\+ 0000000000000001**b**
0000000001001111**b** = 79 (two's complement)

Therefore, **not** 78 = -79.

You can enter real numbers instead of integers, but they are truncated automatically before the comparison.

In **Dec** number base mode:

```
not 78 ENTER                    -79
```

In **Bin** number base mode:

```
not 1001110 ENTER
                  1111111110110001b
Ans▸Dec ENTER                  -79d
```

Not equal to: ≠

TEST menu

numberA ≠ *numberB*
matrixA ≠ *matrixB*
vectorA ≠ *vectorB*
stringA ≠ *stringB*

Tests whether the condition *argumentA* ≠ *argumentB* is true or false. Numbers, matrices, and vectors can be real or complex. If complex, the magnitude (modulus) of each element is compared. Strings are case-sensitive.

- If true (*argumentA* ≠ *argumentB*), returns **1**.
- If false (*argumentA* = *argumentB*), returns **0**.

```
2+2≠3+2 ENTER                    1
2+(2≠3)+2 ENTER                  5
[1,2]≠[3-2,-1+3] ENTER           0
"A"≠"a" ENTER                    1
```

listA ≠ *listB*

Returns a list of **1**s and/or **0**s to indicate if each element in *listA* is ≠ the corresponding element in *listB*.

```
{1,5,9}≠{1,-6,9} ENTER           {0 1 0}
```

nPr

MATH PROB menu

items **nPr** *number*

Returns the number of permutations of *items* (**n**) taken *number* (**r**) at a time. Both arguments must be real nonnegative integers.

```
5 nPr 2 ENTER                    20
```

o

BASE TYPE menu

*integer***o**

Designates a real *integer* as octal, regardless of the number base mode setting.

In **Dec** number base mode:

```
10o ENTER                        8
10o+10 ENTER                     18
```

Oct

† mode screen

Oct

Sets octal number base mode. Results are displayed with the **o** suffix. In any number base mode, you can designate an appropriate value as binary, decimal, hexadecimal, or octal by using the **b**, **d**, **h**, or **o** designator, respectively, from the BASE TYPE menu.

In **Oct** number base mode:

```
10+10b+Fh+10d ENTER                    43o
```

▸Oct

BASE CONV menu

number▸**Oct**
list▸**Oct**
matrix▸**Oct**
vector▸**Oct**

Returns the octal equivalent of the real or complex argument.

In **Dec** number base mode:

```
2*8 ENTER                              16
Ans▸Oct ENTER                          20o

{7,8,9,10}▸Oct ENTER
                        {7o 10o 11o 12o}
```

OneVar

STAT CALC menu
(OneVa shows on menu)

OneVar *xList*,*frequencyList*

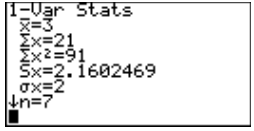
Performs one-variable statistical analysis using real data points in *xList* and frequencies in *frequencyList*.

The values used for *xList* and *frequencyList* are stored automatically to built-in variables **xStat** and **fStat**, respectively.

OneVar *xList*

Uses frequencies of 1.

```
{0,1,2,3,4,5,6}→XL ENTER
                        {0 1 2 3 4 5 6}
OneVar XL ENTER
```



Scroll down to see more results.

OneVar

Uses **xStat** and **fStat** for *xList* and *frequencyList*. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs.

---

or

BASE BOOL menu

integerA **or** *integerB*

Compares two real integers bit by bit. Internally, both integers are converted to binary. When corresponding bits are compared, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value is the sum of the bit results.

For example, 78 **or** 23 = 95.

78 = 1001110**b**
23 = 0010111**b**
1011111**b** = 95

You can enter real numbers instead of integers, but they are truncated automatically before the comparison.

In **Dec** number base mode:

```
78 or 23 ENTER                    95
```

In **Bin** number base mode:

```
1001110 or 10111 ENTER
                            1011111b
Ans▸Dec ENTER                     95d
```

Outpt(

‡ program editor I/O menu

Outpt(*row*,*column*,*string***)**

Displays *string* beginning at *row* and *column*, where $1 \le row \le 8$ and $1 \le column \le 21$.

Outpt(*row*,*column*,*value***)**

Displays *value* beginning at the specified *row* and *column*.

Outpt("CBLSEND",*listName***)**

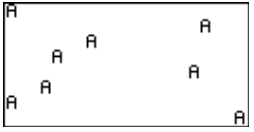
Sends the contents of *listName* to the CBL or CBR System.

You can send data also by using **Send(** as described on page 350.

Program segment:

```
  ⋮
:ClLCD
:For(i,1,8)
: Outpt(i,randInt(1,21),"A")
:End
  ⋮
```

Example result after execution:



P2Reg

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

P2Reg *xList*,*yList*,*frequencyList*,*equationVariable*

Performs a second order polynomial regression using real data pairs in *xList* and *yList* and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**. The equation's coefficients always are stored as a list to built-in variable **PRegC**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

P2Reg *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

P2Reg *xList*,*yList*,*frequencyList*

Stores the regression equation to **RegEq** only.

P2Reg *xList*,*yList*

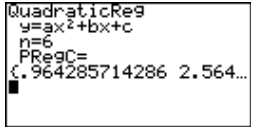
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

P2Reg *equationVariable*

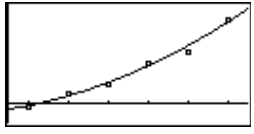
Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                        {1 2 3 4 5 6}
{-2,6,11,23,29,47}→L2 ENTER
                      {-2 6 11 23 29 47}
P2Reg L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



P2Reg

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

P3Reg

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

P3Reg *xList*,*yList*,*frequencyList*,*equationVariable*

Performs a third order polynomial regression using real data pairs in *xList* and *yList* and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**. The equation's coefficients always are stored as a list to built-in variable **PRegC**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

P3Reg *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

P3Reg *xList*,*yList*,*frequencyList*

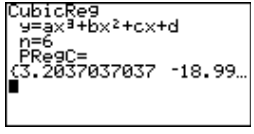
Stores the regression equation to **RegEq** only.

P3Reg *xList*,*yList*

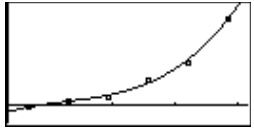
Uses frequencies of 1, and stores the regression equation to **RegEq** only.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                        {1 2 3 4 5 6}
{-6,15,27,88,145,294}→L2 ENTER
                   {-6 15 27 88 145 294}
P3Reg L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



P3Reg *equationVariable*

Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

P3Reg

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

P4Reg

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

P4Reg *xList*,*yList*,*frequencyList*,*equationVariable*

Performs a fourth order polynomial regression using real data pairs in *xList* and *yList* and frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**. The equation's coefficients always are stored as a list to built-in variable **PRegC**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

P4Reg *xList*,*yList*,*equationVariable*

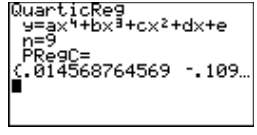
Uses frequencies of 1.

P4Reg *xList*,*yList*,*frequencyList*

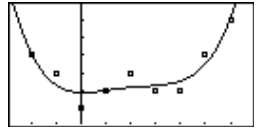
Stores the regression equation to **RegEq** only.

In **Func** graphing mode:

```
{-2,-1,0,1,2,3,4,5,6}→L1 ENTER
                {-2 -1 0 1 2 3 4 5 6}
{4,3,1,2,3,2,2,4,6}→L2 ENTER
                  {4 3 1 2 3 2 2 4 6}
P4Reg L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER          Done
ZData ENTER
```



P4Reg *xList*,*yList*

Uses frequencies of 1, and stores the regression equation to **RegEq** only.

P4Reg *equationVariable*

Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

P4Reg

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

Param

† mode screen

Param

Sets parametric graphing mode.

Pause

‡ program editor CTL menu

Pause *string*
Pause *value*
Pause *list*
Pause *matrix*
Pause *vector*

Displays the specified argument and then suspends program execution until the user presses ENTER.

Program segment:

```
  ⋮
:Input "Enter x:",x
:y1=x²−6
:Disp "y1 is:",y1
:Pause "Press ENTER to graph"
:ZStd
  ⋮
```

Pause

Suspends program execution until the user presses ENTER.

Percent: %

MATH MISC menu

*number***%** or **(***expression***)%**

Returns a real *number* or *expression* divided by 100.

```
5% ENTER                .05
5%*200 ENTER            10
(10+5)%*200 ENTER       30
```

---

pEval(

MATH MISC menu

pEval(*coefficientList***,***xValue***)**

Returns the value of a polynomial (whose coefficients are given in *coefficientList*) at *xValue*.

Evaluate $y=2x^2+2x+3$ at x=5:

```
pEval({2,2,3},5) ENTER  63
```

PlOff

STAT PLOT menu

PlOff [1,2,3]

Deselects the specified stat plot numbers.

```
PlOff 1,3 ENTER         Done
```

PlOff

Deselects all stat plot numbers.

```
PlOff ENTER             Done
```

PlOn

STAT PLOT menu

PlOn [1,2,3]

Selects the specified stat plot numbers, in addition to any plot numbers that are already selected.

```
PlOn 2,3 ENTER          Done
```

PlOn

Selects all stat plot numbers.

```
PlOn ENTER              Done
```

Plot1(
Plot2(
Plot3(

† STAT PLOT menu

The syntax and descriptions to the right refer to **Plot1(**, *but they apply as well to* **Plot2(** *and* **Plot3(**.

Scatter plot

Plot1(1,*xListName*,*yListName*,*mark***)**
Plot1(1,*xListName*,*yListName***)**

Defines and selects the plot using real data pairs in *xListName* and *yListName*.

The optional *mark* specifies the character used to plot the points. If you omit *mark*, a box is used.

mark: **1** = box (□) **2** = cross (+) **3** = dot (•)

xyLine plot

Plot1(2,*xListName*,*yListName*,*mark***)**
Plot1(2,*xListName*,*yListName***)**

Modified box plot

Plot1(3,*xListName*,**1** *or frequencyListName*,*mark***)**
Plot1(3,*xListName*,**1** *or frequencyListName***)**
Plot1(3,*xListName***)**

Defines and selects the plot using real data points in *xListName* with the specified frequencies. If you omit **1** *or frequencyListName*, frequencies of 1 are used.

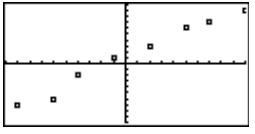
Histogram

Plot1(4,*xListName*,**1** *or frequencyListName***)**
Plot1(4,*xListName***)**

Box plot

Plot1(5,*xListName*,**1** *or frequencyListName***)**
Plot1(5,*xListName***)**

```
{-9,-6,-4,-1,2,5,7,10}→L1 ENTER
                {-9 -6 -4 -1 2 5 7 1…
{-7,-6,-2,1,3,6,7,9}→L2 ENTER
                 {-7 -6 -2 1 3 6 7 9}
Plot1(1,L1,L2) ENTER            Done
ZStd ENTER
```



Pol

† mode screen

Pol

Sets polar graphing mode.

▸Pol

CPLX menu

complexNumber▸**Pol**

Displays *complexNumber* in polar form (*magnitude*∠*angle*), regardless of the complex number mode.

In **RectC** complex number mode:

```
√-2 ENTER                (0,1.41421356237)
Ans▸Pol ENTER
                (1.41421356237∠1.570…
```

list▸**Pol**
matrix▸**Pol**
vector▸**Pol**

Returns a list, matrix, or vector in which each element of the argument is displayed in polar form.

```
{1,√-2} ENTER
                {(1,0) (0,1.141421356…
Ans▸Pol ENTER
                {(1∠0) (1.4142135623…
```

PolarC

† mode screen

PolarC

Sets polar complex number mode (*magnitude*∠*angle*).

In **PolarC** complex number mode:

```
√-2 ENTER      (1.41421356237∠1.570…
```

Polar complex: ∠

2nd [∠]

magnitude∠*angle*

Used to enter complex numbers in polar form. The *angle* is interpreted according to the current angle mode.

In **Radian** angle mode and **PolarC** complex number mode:

```
(1,2)+(3∠π/4) ENTER
                (5.16990542093∠.9226…
```

PolarGC

† graph format screen

PolarGC

Displays graph coordinates in polar form.

poly

† 2nd [POLY]

poly *coefficientList*

Returns a list containing the real and complex roots of a polynomial whose coefficients are given in *coefficientList*.

$a_n x^n + ... + a_2 x^2 + a_1 x^1 + a_0 x^0 = 0$

Find the roots of $2x^3-8x^2-14x+20=0$:

```
poly {2,-8,-14,20} ENTER
                    {5 -2 1}
```

Power: ^

^

number^*power* or (*expression*)^(*expression*)

Returns *number* raised to *power*. The arguments can be real or complex.

```
4^2 ENTER                16
2^-5 ENTER               .03125
```

listA^*listB*

Returns a list in which each element of *listA* is raised to the power specified by the corresponding element in *listB*.

```
{2,3,4}^{3,4,5} ENTER
                    {8 81 1024}
```

squareMatrix^*power*

Returns a matrix equivalent to *squareMatrix* multiplied by itself *power* number of times, where $0 \le power \le 255$. This is not the same as simply raising each element to *power*.

```
[[2,3][4,5]]^3 ENTER
                    [[116 153]
                     [204 269]]
```

Power of 10: 10^

2nd [10^x]

10^*power* or **10^(***expression***)**

Returns 10 raised to *power* or *expression*, which can be real or complex.

```
10^1.5 ENTER             31.6227766017
10^-2 ENTER              .01
```

10^*list*

Returns a list in which each element is 10 raised to the power specified by the corresponding element in *list*.

```
10^{1.5,-2} ENTER
                {31.6227766017 .01}
```

prod

LIST OPS menu

MATH MISC menu

prod *list*

Returns the product of all real or complex elements in *list*.

```
prod {1,2,4,8} ENTER          64
prod {2,7,-8} ENTER          -112
```

Prompt

‡ program editor
I/O menu
(Promp shows on menu)

Prompt *variableA*[,*variableB*, ...]

Prompts the user to enter a value for *variableA*, then *variableB*, and so on.

Program segment:

```
  ⋮
:Prompt A,B,C
  ⋮
```

PtChg(

† GRAPH DRAW menu

PtChg(*x*,*y***)**

Reverses the point at graph coordinates (*x*,*y*).

```
PtChg(-6,2)
```

PtOff(

† GRAPH DRAW menu

PtOff(*x*,*y***)**

Erases the point at graph coordinates (*x*,*y*).

```
PtOff(3,5)
```

PtOn(

† GRAPH DRAW menu

PtOn(*x*,*y***)**

Draws the point at graph coordinates (*x*,*y*).

```
PtOn(3,5)
```

PwrR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

PwrR *xList*,*yList*,*frequencyList*,*equationVariable*

Fits a power regression model ($y=ax^b$) to positive real data pairs in *xList* and *yList*, using frequencies in *frequencyList*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to built-in variables **xStat**, **yStat**, and **fStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

PwrR *xList*,*yList*,*equationVariable*

Uses frequencies of 1.

PwrR *xList*,*yList*,*frequencyList*

Stores the regression equation to **RegEq** only.

PwrR *xList*,*yList*

Uses frequencies of 1, and stores the regression equation to **RegEq** only.

PwrR *equationVariable*

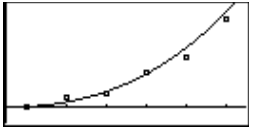
Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

In **Func** graphing mode:

```
{1,2,3,4,5,6}→L1 ENTER
                        {1 2 3 4 5 6}
{1,17,21,52,75,133}→L2 ENTER
                    {1 17 21 52 75 133}
PwrR L1,L2,y1 ENTER
```

```
PwrReg
 y=a*x^b
 a=1.43992723
 b=2.56096944
 corr=.977662979
 n=6
```

```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



PwrR

Uses **xStat**, **yStat**, and **fStat**, and stores the regression equation to **RegEq** only.

PxChg(

GRAPH DRAW menu

PxChg(*row*,*column*)

Reverses the pixel at (*row*, *column*), where $0 \le row \le 62$ and $0 \le column \le 126$.

```
PxChg(10,95)
```

PxOff(

GRAPH DRAW menu

PxOff(*row*,*column*)

Erases the pixel at (*row*, *column*), where $0 \le row \le 62$ and $0 \le column \le 126$.

```
PxOff(10,95)
```

PxOn(

GRAPH DRAW menu

PxOn(*row*,*column*)

Draws the pixel at (*row*, *column*), where $0 \le row \le 62$ and $0 \le column \le 126$.

```
PxOn(10,95)
```

PxTest(

GRAPH DRAW menu

PxTest(*row*,*column*)

Returns **1** if the pixel at (*row*, *column*) is on, **0** if it is off; $0 \le row \le 62$ and $0 \le column \le 126$.

Assuming the pixel at (10,95) is already on:

```
PxTest(10,95) ENTER                1
```

rAdd(

MATRX OPS menu

rAdd(*matrix*,*rowA*,*rowB*)

Returns a matrix in which *rowA* of a real or complex *matrix* is added to (and stored in) *rowB*.

```
[[5,3,1][2,0,4][3,-1,2]]→MAT
ENTER                      [[5 3  1]
                            [2 0  4]
                            [3 -1 2]]

rAdd(MAT,2,3) ENTER        [[5 3  1]
                            [2 0  4]
                            [5 -1 6]]
```

Radian

† 2nd [MODE]

Radian

Sets radian angle mode.

In **Radian** angle mode:

```
sin (π/2) ENTER                    1
sin 90 ENTER              .893996663601
```

Radian entry: ʳ

MATH ANGLE menu

*number*ʳ or **(***expression***)**ʳ

Designates a real *number* or *expression* as radians, regardless of the angle mode setting.

*list*ʳ

Designates each element in a real *list* as radians.

In **Degree** angle mode:

```
cos (π/2) ENTER            .999624216859
cos (π/2)ʳ ENTER                       0
cos {π/2,π}ʳ ENTER                {0 -1}
```

rand

MATH PROB menu

rand

Returns a random number between 0 and 1.

To control a random number sequence, first store an integer seed value to **rand** (such as **0→rand**).

You may have different results for the first two examples:

```
rand ENTER                 .943597402492
rand ENTER                 .146687829222

0→rand:rand ENTER          .943597402492
0→rand:rand ENTER          .943597402492
```

randBin(

MATH PROB menu
(randBi shows on menu)

randBin(*#ofTrials*,*probabilityOfSuccess*,*#ofSimulations***)**

Returns a list of random integers from a binomial distribution, where *#ofTrials* ≥ 1 and 0 ≤ *probabilityOfSuccess* ≤ 1. The *#ofSimulations* is an integer ≥ 1 that specifies the number of integers returned in the list.

A seed value stored to **rand** also affects **randBin(.**

```
1→rand:randBin(5,.2,3) ENTER
                                  {0 3 2}
```

randBin(*#ofTrials*,*probabilityOfSuccess***)**

Returns a single random integer.

```
0→rand:randBin(5,.2) ENTER             1
```

randInt(

MATH PROB menu
(randIn shows on menu)

randInt(*lower*,*upper*,*#ofTrials*)

Returns a list of random integers bound by the specified integers, *lower* ≤ integer ≤ *upper*. The *#ofTrials* is an integer ≥ 1 that specifies the number of integers returned in the list.

A seed value stored to **rand** also affects **randInt(**.

```
1→rand:randInt(1,10,3) ENTER
                        {8 9 3}
```

randInt(*lower*,*upper*)

Returns a single random integer.

```
0→rand:randInt(1,10) ENTER      10
```

randM(

MATRX OPS menu

randM(*rows*,*columns*)

Returns a *rows* × *columns* matrix filled with random one-digit integers (-9 to 9).

```
0→rand:randM(2,3) ENTER
                  [[4  -2 0]
                   [-7 8  8]]
```

randNorm(

MATH PROB menu
(randN shows on menu)

randNorm(*mean*,*stdDeviation*,*#ofTrials*)

Returns a list of random numbers from a normal distribution specified by *mean* and *stdDeviation*. The *#ofTrials* is an integer ≥ 1 that specifies how many numbers are returned. Each returned number could be any real number, but most will be within the interval:

[*mean*−3(*stdDeviation*), *mean*+3(*stdDeviation*)].

A seed value stored to **rand** also affects **randNorm(**.

```
1→rand:randNorm(0,1,3) ENTER
                {-.660585055265 -1.0…
```

randNorm(*mean*,*stdDeviation*)

Returns a single random number.

```
0→rand:randNorm(0,1) ENTER
                -1.58570962271
```

RcGDB

† GRAPH menu

RcGDB *graphDataBaseName*

Restores all settings stored in *graphDataBaseName*. For a list of settings, refer to **StGDB** on page 361.

RcPic

† GRAPH menu

RcPic *pictureName*

Displays the current graph and adds the picture stored in *pictureName*.

real

CPLX menu

real (*complexNumber***)**

Returns the real part of *complexNumber*.

real (*real*,*imaginary***)** returns *real*.
real (*magnitude*∠*angle***)** returns *magnitude****cos** (*angle*).

In **Radian** angle mode:

```
real (3,4) ENTER                    3
real (3∠4) ENTER      -1.96093086259
```

real *complexList*
real *complexMatrix*
real *complexVector*

Returns a list, matrix, or vector in which each element is the real part of the corresponding element in the argument.

In **Radian** angle mode:

```
real {-2,(3,4),(3∠4)} ENTER
                {-2 3 -1.96093086259}
```

▸Rec

CPLX menu

complexNumber▸**Rec**

Displays *complexNumber* in rectangular form (*real*,*imaginary*) regardless of the complex number mode.

In **PolarC** complex number mode:

```
√-2 ENTER       (1.41421356237∠1.570…
Ans▸Rec ENTER      (0,1.41421356237)
```

complexList▸**Rec**
complexMatrix▸**Rec**
complexVector▸**Rec**

Returns a list, matrix, or vector in which each element of the argument is displayed in rectangular form.

In **PolarC** complex number mode:

```
[(3∠π/6),√-2] ENTER
                [(3∠.523598775598) (…
Ans▸Rec ENTER
                [(2.59807621135,1.5)…
```

RectC

† mode screen

RectC

Sets rectangular complex number mode (*real*,*imaginary*).

In **RectC** complex number mode:

```
√-2 ENTER          (0,1.41421356237)
```

RectGC

† graph format screen

RectGC

Displays graph coordinates in rectangular form.

RectV

† mode screen

RectV

Sets rectangular vector coordinate mode **[x y z]**.

In **RectV** vector coordinate mode:

```
3*[4∠5] ENTER
                [3.40394622556 -11.5…
```

ref

MATRX OPS menu

ref *matrix*

Returns the row-echelon form of a real or complex *matrix*. The number of columns must be greater than or equal to the number of rows.

```
[[4,5,6][7,8,9]]→MAT ENTER
                              [[4 5 6]
                               [7 8 9]]
ref MAT ENTER
                [[1 1.14285714286 1.…
                 [0 1              2 …
```

Repeat

‡ program editor CTL menu (Repea shows on menu)

:Repeat *condition*
:*commands-to-repeat*
:End
:*commands*

Executes *commands-to-repeat* until *condition* is true.

Program segment:

```
  ⋮
:6→N
:1→Fact
:Repeat N<1
: Fact*N→Fact
: N-1→N
:End
:Disp "6!=",Fact
  ⋮
```

Return

‡ program editor CTL menu (Retur shows on menu)

Return

In a subroutine, exits the subroutine and returns to the calling program. In the main program, stops execution and returns to the home screen.

Program segment in the calling program:

```
  ⋮
:Input "Diameter:",DIAM
:Input "Height:",HT
:AREACIRC
:VOL=AREA*HT
:Disp "Volume =",VOL
  ⋮
```

AREACIRC subroutine program:

```
PROGRAM:AREACIRC
:RADIUS=DIAM/2
:AREA=π*RADIUS²
:Return
```

RK

† graph format screen (scroll down to second screen)

RK

In **DifEq** graphing mode, uses an algorithm based on the Runge-Kutta method to solve differential equations. Typically, **RK** is more accurate than **Euler** but takes longer to find the solutions.

rnorm

MATRX MATH menu

rnorm *matrix*

Returns the row norm of a real or complex *matrix*. For each row, **rnorm** sums the absolute values (magnitudes of complex elements) of all elements on that row. The returned value is the largest of the sums.

```
[[-5,6,-7][3,3,9][9,-9,-7]]
→MAT ENTER
                    [[-5 6  -7]
                     [3  3  9 ]
                     [9  -9 -7]]
rnorm MAT ENTER                25
```

rnorm *vector*

Returns the largest absolute value (or magnitude) in a real or complex *vector*.

```
rnorm [15,-18,7] ENTER         18
```

Root: $\sqrt[x]{}$

MATH MISC menu

$x^{th}root\ \sqrt[x]{}number$ or $x^{th}root\ \sqrt[x]{}(expression)$

Returns the $x^{th}root$ of *number* or *expression*. The arguments can be real or complex.

```
5^x√32 ENTER                   2
```

$x^{th}root\ \sqrt[x]{}list$

Returns a list in which each element is the $x^{th}root$ of the corresponding element in *list*.

```
5^x√{32,243} ENTER           {2 3}
```

$x^{th}rootList\ \sqrt[x]{}list$

Returns a list in which each element is the root specified by the corresponding elements in $x^{th}rootList$ and *list*.

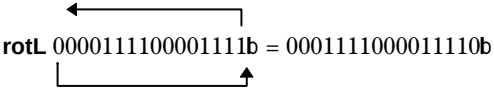
```
{5,2}^x√{32,25) ENTER        {2 5}
```

rotL

BASE BIT menu

rotL *integer*

Returns a real *integer* with bits rotated one to the left. Internally, *integer* is represented as a 16-bit binary number. When the bits are rotated left, the leftmost bit rotates to the rightmost bit.



rotL is not valid in **Dec** number base mode. To enter hexadecimal numbers **A** through **F**, use the BASE A-F menu. Do not use [ALPHA] to type a letter.

In **Bin** number base mode:

```
rotL 0000111100001111 [ENTER]
                  1111000011110b
```

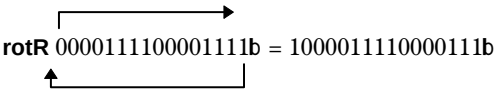
Leading zeros are not displayed.

rotR

BASE BIT menu

rotR *integer*

Returns a real *integer* with bits rotated one to the right. Internally, *integer* is represented as a 16-bit binary number. When the bits are rotated right, the rightmost bit rotates to the leftmost bit.



rotR is not valid in **Dec** number base mode. To enter hexadecimal numbers **A** through **F**, use the BASE A-F menu. Do not use [ALPHA] to type a letter.

In **Bin** number base mode:

```
rotR 0000111100001111 [ENTER]
                  1000011110000111b
```

round(

MATH NUM menu

round(*number*,*#ofDecimals*)
round(*number*)

Returns a real or complex *number* rounded to the specified *#ofDecimals* (0 to 11). If *#ofDecimals* is omitted, *number* is rounded to 12 decimal places.

```
round(π,4) ENTER                3.1416
round(π/4,4) ENTER               .7854
round(π/4) ENTER          .785398163397
```

round(*list*,*#ofDecimals*)
round(*matrix*,*#ofDecimals*)
round(*vector*,*#ofDecimals*)

Returns a list, matrix, or vector in which each element is the rounded value of the corresponding element in the argument. *#ofDecimals* is optional.

```
round({π,√2,ln 2},3) ENTER
                    {3.142 1.414 .693}
round([[ln 5,ln 3][π,e^1]],2)
ENTER                    [[1.61 1.1 ]
                          [3.14 2.72]]
```

rref

MATRX OPS menu

rref *matrix*

Returns the reduced row-echelon form of a real or complex *matrix*. The number of columns must be greater than or equal to the number of rows.

```
[[4,5,6][7,8,9]]→MAT ENTER
                              [[4 5 6]
                               [7 8 9]]
rref MAT ENTER
                 [[1 0 -.999999999999…
                  [0 1 2              …
```

rSwap(

MATRX OPS menu

rSwap(*matrix*,*rowA*,*rowB*)

Returns a matrix with *rowA* of a real or complex *matrix* swapped with *rowB*.

```
[[5,3,1][2,0,4][3,-1,2]]→MAT
ENTER                         [[5 3  1]
                               [2 0  4]
                               [3 -1 2]]
rSwap(MAT,2,3) ENTER
                              [[5 3  1]
                               [3 -1 2]
                               [2 0  4]]
```

Scatter

† STAT DRAW menu (Scatte shows on menu)

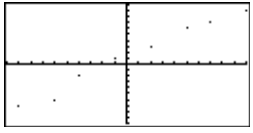
Scatter *xList*,*yList*

Draws a scatter plot on the current graph, using the real data pairs in *xList* and *yList*.

Scatter

Uses the data in built-in variables **xStat** and **yStat**. These variables must contain valid data of the same dimension; otherwise, an error occurs.

```
{-9,-6,-4,-1,2,5,7,10}→XL ENTER
                {-9 -6 -4 -1 2 5 7 1…
{-7,-6,-2,1,3,6,7,9}→YL ENTER
                  {-7 -6 -2 1 3 6 7 9}
ZStd:Scatter XL,YL ENTER
```



Sci

† mode screen

Sci

Sets scientific notation display mode.

In **Sci** notation mode:

```
123456789 ENTER          1.23456789E8
```

In **Normal** notation mode:

```
123456789 ENTER            123456789
```

Select(

LIST OPS menu

Select(*xListName*,*yListName*)

If a scatter plot or xyline plot is currently selected and plotted on the graph screen, you can select a subset (range) of those data points. The selected data points are stored to *xListName* and *yListName*.

Select(*xListName*,*yListName***)** displays the current graph screen and starts an interactive session during which you select a range of data points.

a. Move the cursor to the leftmost (left bound) point of the range you want to select and press ENTER.

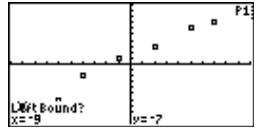
b. Then move the cursor to the rightmost (right bound) point of the range you want to select and press ENTER.

A new stat plot of *xListName* and *yListName* replaces the plot from which you selected the points.

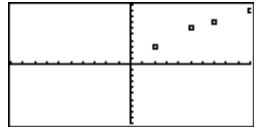
```
{-9,-6,-4,-1,2,5,7,10}→L1 ENTER
                {-9 -6 -4 -1 2 5 7 1…
{-7,-6,-2,1,3,6,7,9}→L2 ENTER
                 {-7 -6 -2 1 3 6 7 9}
Plot1(1,L1,L2):ZStd ENTER
```

After the graph is displayed:

```
Select(L10,L20) ENTER
```



Move the cursor to point (2,3) and press ENTER. Then move to (10,9) and press ENTER.



```
L10 ENTER                    {2 5 7 10}
L20 ENTER                    {3 6 7 9}
```

Send(

‡ program editor
I/O menu

Send(*listName***)**

Sends the contents of *listName* to the CBL or CBR System.

```
{1,2,3,4,5}→L1:Send(L1) ENTER
                              Done
```

seq(

MATH MISC menu

seq(*expression*,*variable*,*begin*,*end*,*step***)**

Returns a list containing a sequence of numbers created by evaluating *expression* from *variable* = *begin* to *variable* = *end* in increments of *step*.

```
seq(x²,x,1,8,2) ENTER
                {1 9 25 49}
```

seq(*expression*,*variable*,*begin*,*end***)**

Uses a *step* of 1.

```
seq(x²,x,1,8) ENTER
        {1 4 9 16 25 36 49 6…
```

SeqG

† graph format screen

SeqG

Sets sequential graphing format, in which selected functions are plotted one at a time.

SetLEdit

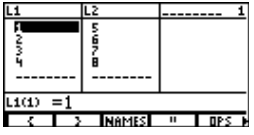
LIST OPS menu
(SetLE shows on menu)

SetLEdit *column1ListName*[**,** … **,***column20ListName*]

Removes all lists from the list editor and then stores one or more *ListNames* in the specified order, starting with column 1.

```
{1,2,3,4}→L1 ENTER      {1 2 3 4}
{5,6,7,8}→L2 ENTER      {5 6 7 8}
SetLEdit L1,L2 ENTER         Done
```

The list editor now contains:



SetLEdit

Removes all lists from the list editor and stores built-in lists **xStat**, **yStat**, and **fStat** in columns 1 through 3, respectively.

Shade(

GRAPH DRAW menu

Shade(*lowerFunc*,*upperFunc*,*xLeft*,*xRight*,*pattern*,*patternRes*)**

Draws *lowerFunc* and *upperFunc* in terms of **x** on the current graph and shades the area bounded by *lowerFunc*, *upperFunc*, *xLeft*, and *xRight*. The shading style is determined by *pattern* (**1** through **4**) and *patternRes* (**1** through **8**).

pattern:

1 = vertical (default)	**3** = negative-slope 45°
2 = horizontal	**4** = positive-slope 45°

patternRes (resolution):

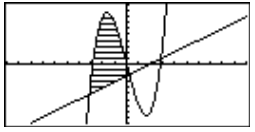
1 = every pixel (default)	5 = every 5th pixel
2 = every 2nd pixel	6 = every 6th pixel
3 = every 3rd pixel	7 = every 7th pixel
4 = every 4th pixel	8 = every 8th pixel

Shade(*lowerFunc*,*upperFunc*)**

Sets *xLeft* and *xRight* to **xMin** and **xMax**, respectively, and uses the defaults for *pattern* and *patternRes*.

In **Func** graphing mode:

```
Shade(x-2,x^3-8 x,-5,1,2,3) ENTER
```



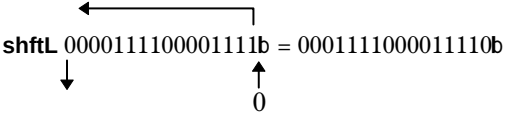
```
ClDrw:Shade(x^3-8 x,x-2) ENTER
```

shftL

BASE BIT menu

shftL *integer*

Returns a real *integer* with bits shifted one to the left. Internally, *integer* is represented as a 16-bit binary number. When the bits are shifted left, the leftmost bit is dropped and 0 is used as the rightmost bit.



shftL is not valid in **Dec** number base mode. To enter hexadecimal numbers **A** through **F**, use the BASE A-F menu. Do not use ALPHA to type a letter.

In **Bin** number base mode:

```
shftL 0000111100001111 ENTER
                  1111000011110b
```

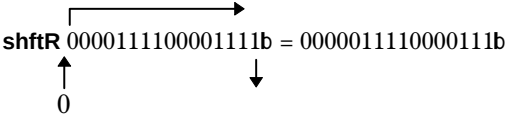
Leading zeros are not displayed.

shftR

BASE BIT menu

shftR *integer*

Returns a real *integer* with bits shifted one to the right. Internally, *integer* is represented as a 16-bit binary number. When the bits are shifted right, the rightmost bit is dropped and 0 is used as the leftmost bit.



shftR is not valid in **Dec** number base mode. To enter hexadecimal numbers **A** through **F**, use the BASE A-F menu. Do not use ALPHA to type a letter.

In **Bin** number base mode:

```
shftR 0000111100001111 ENTER
                   11110000111b
```

Leading zeros are not displayed.

ShwSt

CATALOG

ShwSt

Displays the results of the most recent stat calculation.

sign

MATH NUM menu

sign *number* or **sign (***expression***)**

Returns -**1** if the argument is < 0, **1** if it is > 0, or **0** if it is = 0. The argument must be real.

```
sign -3.2 ENTER                          -1
sign (6+2-8) ENTER                        0
```

sign *list*

Returns a list in which each element is -1, 1, or 0 to indicate the sign of the corresponding element in *list*.

```
sign {-3.2,16.8,6+2-8} ENTER
                                  {-1 1 0}
```

SimulG

† graph format screen

SimulG

Sets simultaneous graphing format, in which all selected functions are plotted at the same time.

simult(

† 2nd [SIMULT]

simult(*squareMatrix***,***vector***)**

Returns a vector containing the solutions to a system of simultaneous linear equations that have the form:

$a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + ... = b_1$
$a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + ... = b_2$
$a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + ... = b_3$

Each row in *squareMatrix* contains the **a** coefficients of an equation, and *vector* contains the **b** constants.

Solve the following for x and y:

```
3x - 4y = 7
 x + 6y = 6
```

```
[[3,-4][1,6]]→MAT ENTER     [[3 -4]
                             [1 6 ]]
[7,6]→VEC ENTER               [7 6]
simult(MAT,VEC) ENTER         [3 .5]
```

The solution is x=3 and y=.5.

sin

SIN

sin *angle* or **sin (***expression***)**

Returns the sine of *angle* or *expression*, which can be real or complex.

An angle is interpreted as degrees or radians according to the current angle mode. In any angle mode, you can designate an angle as degrees or radians by using the ° or ʳ designator, respectively, from the MATH ANGLE menu.

In **Radian** angle mode:

```
sin π/2 ENTER                    0
sin (π/2) ENTER                  1
sin 45° ENTER          .707106781187
```

In **Degree** angle mode:

```
sin 45 ENTER           .707106781187
sin (π/2)ʳ ENTER                 1
```

sin *list*

Returns a list in which each element is the sine of the corresponding element in *list*.

In **Radian** angle mode:

```
sin {0,π/2,π} ENTER          {0 1 0}
```

In **Degree** angle mode:

```
sin {0,30,90} ENTER          {0 .5 1}
```

sin *squareMatrix*

The squareMatrix cannot have repeated eigenvalues.

Returns a square matrix that is the matrix sine of *squareMatrix*. The matrix sine corresponds to the result calculated using power series or Cayley-Hamilton Theorem techniques. This is *not* the same as simply calculating the sine of each element.

sin⁻¹

2nd [SIN⁻¹]

sin⁻¹ *number* or **sin⁻¹ (***expression***)**

Returns the arcsine of *number* or *expression*, which can be real or complex.

In **Radian** angle mode:

```
sin⁻¹ .5 ENTER           .523598775598
sin⁻¹ {0,.5} ENTER
                {0 .523598775598}
```

sin⁻¹ *list*

Returns a list in which each element is the arcsine of the corresponding element in *list*.

In **Degree** angle mode:

```
sin⁻¹ 1 ENTER                    90
```

sinh

MATH HYP menu

sinh *number* or **sinh (***expression***)**

Returns the hyperbolic sine of *number* or *expression*, which can be real or complex.

```
sinh 1.2 ENTER                1.50946135541
```

sinh *list*

Returns a list in which each element is the hyperbolic sine of the corresponding element in *list*.

```
sinh {0,1.2} ENTER
                        {0 1.50946135541}
```

sinh^{-1}

MATH HYP menu

sinh^{-1} *number* or **sinh^{-1}(***expression***)**

Returns the inverse hyperbolic sine of *number* or *expression*, which can be real or complex.

```
sinh-1 1 ENTER                 .88137358702
```

sinh^{-1} *list*

Returns a list in which each element is the inverse hyperbolic sine of the corresponding element in *list*.

```
sinh-1 {1,2.1,3} ENTER
                 {.88137358702 1.4874…
```

SinR

STAT CALC menu

*Built-in equation variables such as **y1**, **r1**, and **xt1** are case-sensitive. Do not use **Y1**, **R1**, and **XT1**.*

SinR [*iterations*,]*xList*,*yList*[,*period*],*equationVariable*

Attempts to fit a sinusoidal regression model (y=a sin(bx+c)+d) to real data pairs in *xList* and *yList*, using an optional estimated *period*. The regression equation is stored to *equationVariable*, which must be a built-in equation variable such as **y1**, **r1**, and **xt1**. The equation's coefficients always are stored as a list to built-in variable **PRegC**.

iterations is optional; it specifies the maximum number of times (1 through 16) the TI-86 will attempt to find a solution. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

If you specify a period, the TI-86 may find a solution more quickly or it may find a solution when one would not have been found otherwise.

If you omit the optional *period*, the difference between values in *xList* should be equal and in sequential order. If you specify *period*, the differences between x values can be unequal.

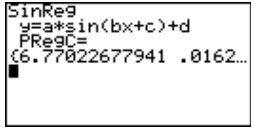
Values used for *xList* and *yList* are stored automatically to built-in variables **xStat** and **yStat**, respectively. The regression equation is stored also to built-in equation variable **RegEq**.

The output of **SinR** is always in radians, regardless of the angle mode setting.

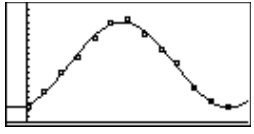
SinR [*iterations*,]*xList*,*yList*[,*period*]

Stores the regression equation to **RegEq** only.

```
seq(x,x,1,361,30)→L1 ENTER
            {1 31 61 91 121 151 …
{5.5,8,11,13.5,16.5,19,19.5,17,
14.5,12.5,8.5,6.5,5.5}→L2 ENTER
             {5.5 8 11 13.5 16.5…
SinR L1,L2,y1 ENTER
```




```
Plot1(1,L1,L2) ENTER                Done
ZData ENTER
```



SinR [*iterations*,] *equationVariable*

Uses **xStat** and **yStat** for *xList* and *yList*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs. The regression equation is stored to *equationVariable* and **RegEq**.

SinR [*iterations*]

Uses **xStat** and **yStat**, and stores the regression equation to **RegEq** only.

SlpFld

† graph format screen (scroll down to second screen)

SlpFld

In **DifEq** graphing mode, turns on slope fields. To turn off direction and slope fields, use **FldOff**.

Solver(

† 2nd [SOLVER]

Solver(*equation*,*variable*,*guess*,{*lower*,*upper*}**)**

Solves *equation* for *variable*, given an initial *guess* and *lower* and *upper* bounds within which the solution is sought. *equation* can be an expression, which is assumed to equal 0.

Solver(*equation*,*variable*,*guess***)**

Uses -1E99 and 1E99 for *upper* and *lower*, respectively.

Solver(*equation*,*variable*,{*guessLower*,*guessUpper*}**)**

Uses the secant line between *guessLower* and *guessUpper* to start the search. **Solver(** will still search for a solution outside of this range.

If y=5, solve $x^3+y^2=125$ for x. You guess the solution is approximately 4:

```
5→y ENTER                                5
Solver(x^3+y^2=125,x,4) ENTER    Done
x ENTER                    4.64158883361
```

sortA

LIST OPS menu

SortA *list*

Returns a list in which the real or complex elements of *list* are sorted in ascending order.

```
{5,8,-4,0,-6}→L1 ENTER
                      {5 8 -4 0 -6}
SortA L1 ENTER        {-6 -4 0 5 8}
```

sortD

LIST OPS menu

SortD *list*

Returns a list in which the real or complex elements of *list* are sorted in descending order.

```
{5,8,-4,0,-6}→L1 ENTER
                      {5 8 -4 0 -6}
SortD L1 ENTER        {8 5 0 -4 -6}
```

Sortx

LIST OPS menu

Sortx *xListName*,*yListName*,*frequencyListName*
Sortx *xListName*,*yListName*

In ascending order of **x** elements, sorts real or complex **x** and **y** data pairs and, optionally, their frequencies in *xListName*, *yListName*, and *frequencyListName*. The lists' contents are updated to reflect the changes.

Sortx

Uses built-in variables **xStat** and **yStat** for *xListName* and *yListName*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs.

```
{3,1,2}→XL ENTER      {3 1 2}
{0,8,-4}→YL ENTER     {0 8 -4}
Sortx XL,YL ENTER        Done
XL ENTER              {1 2 3}
YL ENTER              {8 -4 0}
```

Sorty

LIST OPS menu

Sorty *xListName*,*yListName*,*frequencyListName*
Sorty *xListName*,*yListName*

In ascending order of **y** elements, sorts real or complex **x** and **y** data pairs and, optionally, their frequencies in *xListName*, *yListName*, and *frequencyListName*. The lists' contents are updated to reflect the changes.

```
{3,1,2}→XL ENTER      {3 1 2}
{0,8,-4}→YL ENTER     {0 8 -4}
Sorty XL,YL ENTER        Done
YL ENTER              {-4 0 8}
XL ENTER              {2 3 1}
```

Sorty

Uses built-in variables **xStat** and **yStat** for *xListName* and *yListName*, respectively. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs.

▸Sph

VECTR OPS menu

vector▸**Sph**

Displays a 2- or 3-element *vector* as spherical coordinates in $[r \angle\theta \ \angle 0]$ or $[r \angle\theta \ \angle\phi]$ form, respectively, even if the display mode is not set for spherical (**SphereV**).

In **RectV** vector coordinate mode:

```
[0,-1]▸Sph ENTER
          [1∠-1.57079632679∠1.…
[0,0,-1]▸Sph ENTER
            [1∠0∠3.14159265359]
```

SphereV

† 2nd [MODE]

SphereV

Sets spherical vector coordinate mode $[r \angle\theta \ \angle\phi]$.

In **SphereV** vector coordinate mode:

```
[1,2] ENTER
          [2.2360679775∠1.1071…
```

Square: ²

x^2

*number*² or (*expression*)²
*list*²
*squareMatrix*²

Returns a real or complex argument multiplied by itself. To square a negative number, enclose it in parentheses.

A *squareMatrix* multiplied by itself is not the same as simply squaring each element.

```
25² ENTER                    625
(16+9)² ENTER                625

-2² ENTER                    -4
(-2)² ENTER                   4

{-2,4,25}² ENTER        {4 16 625}

[[2,3][4,5]]² ENTER      [[16 21]
                          [28 37]]
```

Square root: √

2nd [√]

√*number* or √(*expression*)

Returns the square root of *number* or *expression*, which can be real or complex.

```
√25 ENTER                    5
√(25+11) ENTER               6
```

√list

Returns a list in which element is the square root of the corresponding element in *list*.

In **RectC** complex number mode:

```
√{-2,25} ENTER
              {(0,1.41421356237) (…
```

St▸Eq(

STRNG menu

St▸Eq(*stringVariable*,*equationVariable*)

Converts *stringVariable* to a number, expression, or equation, and stores it in *equationVariable*.

To convert the string and retain the same variable name, you can set *equationVariable* equal to *stringVariable*.

*If you use **Input** instead of **InpSt** here, the entered expression is evaluated at the current value of x and the result (not the expression) is stored.*

```
"5"→x:6 x ENTER
                    ERROR 10 DATA TYPE
"5"→x:St▸Eq(x,x):6 x ENTER        30
```

Program segment:

```
   ⋮
:InpSt "Enter y1(x):",STR
:St▸Eq(STR,y1)
:Input "Enter x:",x
:Disp "Result is:",y1(x)
   ⋮
```

You cannot store a string directly to a built-in equation variable.

StGDB

† GRAPH menu

StGDB *graphDataBaseName*

Creates a graph database (GDB) variable that contains the current:

- Graphing mode, graph format settings, and range variables.
- Functions in the equation editor, whether they are selected, and their graph styles.

To restore the database and recreate the graph, use **RcGDB** (page 343).

Stop

‡ program editor
CTL menu

Stop

Ends program execution and returns to the home screen.

Use N==999, not N=999.

Program segment:

```
   ⋮
:Input N
:If N==999
:Stop
   ⋮
```

Store to variable: →

STO▸

number→*variable* or (*expression*)→*variable*
string→*variable*
list→*variable*
vector→*variable*
matrix→*variable*

Stores the specified argument to *variable*.

```
10→A:4*A ENTER                    40
"Hello"→STR ENTER          Hello
{1,2,3}→L1 ENTER                  {1 2 3}
[1,2,3]→VEC ENTER                 [1 2 3]
[[1,2,3][4,5,6]]→MAT ENTER
                                  [[1 2 3]
                                   [4 5 6]]
```

StPic

† GRAPH menu

StPic *pictureName*

Stores a picture of the current graph screen to *pictureName*.

StReg(

STAT CALC menu

StReg(*variable***)**

Stores the most recently calculated regression equation to *variable*. This lets you save a regression equation by storing it to any variable as opposed to a built-in equation variable.

2nd [RCL] EQ ENTER *recalls the equation. Then* ENTER *evaluates it at the current value of x.*

```
{1,2,3,4,5}→L1 ENTER
                              {1 2 3 4 5}
{1,20,55,230,742}→L2 ENTER
                        {1 20 55 230 742}
ExpR L1,L2:StReg(EQ) ENTER
                                    Done
8→x ENTER                             8
Rcl EQ ENTER
   .41138948780597*4.7879605684671^x
ENTER                      113620.765451
```

String entry: "

STRNG menu

‡ program editor I/O menu

"*string***"**

Defines a string. When you display a string, it is left-justified on the screen.

Strings are interpreted as text characters, not numbers. For example, you cannot perform a calculation with strings such as "4" or "A✱8". To convert between string variables and equation variables, use **Eq▸St(** and **St▸Eq(** as described on pages 290 and 361, respectively.

```
"Hello"→STR ENTER
                    Hello
Disp STR+", Jan" ENTER
                    Hello, Jan
                          Done
```

sub(

STRNG menu

sub(*string*,*begin*,*length***)**

Returns a new string that is a subset of *string*, starting at character number *begin* and continuing for the specified *length*.

```
"The answer is:"→STR ENTER
                    The answer is:
sub(STR,5,6) ENTER
                    answer
```

Subtraction: −

-

numberA – *numberB*

Returns the value of *numberB* subtracted from *numberA*. The arguments can be real or complex.

```
6-2 ENTER                          4
10--4.5 ENTER                   14.5
```

list – *number*

Returns a list in which *number* is subtracted from each element of *list*. The arguments can be real or complex.

```
{10,9,8}-4 ENTER              {6 5 4}
```

In **RectC** complex number mode:

```
{8,1,(5,2)}-3 ENTER
                  {(5,0) (-2,0) (2,2)}
```

listA – *listB*
matrixA – *matrixB*
vectorA – *vectorB*

Returns a list, matrix, or vector that is the result of each element in the second argument subtracted from the corresponding element in the first argument. The two real or complex arguments must have the same dimension.

```
{5,7,9}-{4,5,6} ENTER          {1 2 3}
[[5,7,9][11,13,15]]-[[4,5,6][7,8,
9]] ENTER                    [[1 2 3]
                              [4 5 6]]
[5,7,9]-[1,2,3] ENTER          [4 5 6]
```

sum

MATH MISC menu

LIST OPS menu

sum *list*

Returns the sum of all real or complex elements in *list*.

```
sum {1,2,4,8} ENTER                15
sum {2,7,-8,0} ENTER                1
```

tan

TAN

tan *angle* or **tan (***expression***)**

Returns the tangent of *angle* or *expression*, which can be real or complex.

An angle is interpreted as degrees or radians according to the current angle mode. In any angle mode, you can designate an angle as degrees or radians by using the ° or ʳ designator, respectively, from the MATH ANGLE menu.

In **Radian** angle mode:

```
tan π/4 ENTER                      0
tan (π/4) ENTER                    1
tan 45° ENTER                      1
```

In **Degree** angle mode:

```
tan 45 ENTER                       1
tan (π/4)ʳ ENTER                   1
```

tan *list*

Returns a list in which each element is the tangent of the corresponding element in *list*.

In **Degree** angle mode:

```
tan {0,45,60} ENTER
                  {0 1 1.73205080757}
```

tan⁻¹

2nd [TAN⁻¹]

tan⁻¹ *number* or **tan⁻¹ (***expression***)**

Returns the arctangent of *number* or *expression*, which can be real or complex.

In **Radian** angle mode:

```
tan⁻¹ .5 ENTER                .463647609001
```

In **Degree** angle mode:

```
tan⁻¹ 1 ENTER                            45
```

tan⁻¹ *list*

Returns a list in which each element is the arctangent of the corresponding element in *list*.

In **Radian** angle mode:

```
tan⁻¹ {0,.2,.5} ENTER
                {0 .19739555985 .463…
```

tanh

MATH HYP menu

tanh *number* or **tanh (***expression***)**

Returns the hyperbolic tangent of *number* or *expression*, which can be real or complex.

```
tanh 1.2 ENTER                .833654607012
```

tanh *list*

Returns a list in which each element is the hyperbolic tangent of the corresponding element in *list*.

```
tanh {0,1.2} ENTER
                      {0 .833654607012}
```

tanh⁻¹

MATH HYP menu

tanh⁻¹ *number* or **tanh⁻¹(***expression***)**

Returns the inverse hyperbolic tangent of *number* or *expression*, which can be real or complex.

```
tanh⁻¹ 0 ENTER                           0
```

tanh⁻¹ *list*

Returns a list in which each element is the inverse hyperbolic tangent of the corresponding element in *list*.

In **RectC** complex number mode:

```
tanh⁻¹ {0,2.1} ENTER
                {(0,0) (.51804596584…
```

TanLn(

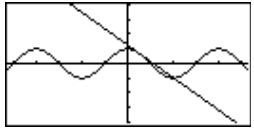
GRAPH DRAW menu

TanLn(*expression*,*xValue*)**

Draws *expression* on the current graph and then draws a tangent line at *xValue*.

In **Func** graphing mode and **Radian** angle mode:

```
ZTrig:TanLn(cos x,π/4) ENTER
```



Text(

† GRAPH DRAW menu

Text(*row*,*column*,*string*)**

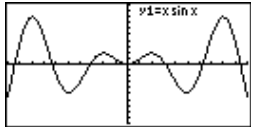
Writes a text *string* on the current graph beginning at pixel (*row*,*column*), where $0 \le row \le 57$ and $0 \le column \le 123$.

Text at the bottom of the graph may be covered by a displayed menu. To remove the menu, press CLEAR.

Program segment in **Func** graphing mode and a **ZStd** graph screen:

```
  ⋮
:y1=x sin x
:Text(0,70,"y1=x sin x")
  ⋮
```

When executed:



Then

‡ program editor CTL menu

Refer to syntax information for **If**, beginning on page 305. See the **If:Then:End** and **If:Then:Else:End** syntax.

Trace

† GRAPH menu

Trace

Displays the current graph and lets the user trace a function. From a program, press ENTER to stop tracing and continue with the program.

Transpose: T

MATRX MATH menu

*matrix*T

Returns a transposed real or complex matrix in which element *row*,*column* is swapped with element *column*,*row* of *matrix*. For example:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^T \text{ returns } \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

For complex matrices, the complex conjugate of each element is taken.

```
[[1,2][3,4]]→MATA ENTER
                          [[1 2]
                           [3 4]]
MATA^T ENTER              [[1 3]
                           [2 4]]
[[1,2,3][4,5,6][7,8,9]]→MATB
ENTER                    [[1 2 3]
                          [4 5 6]
                          [7 8 9]]
MATB^T ENTER             [[1 4 7]
                          [2 5 8]
                          [3 6 9]]
```

In **RectC** complex number mode:

```
[[(1,2),(1,1)][(3,2),(4,3)]]
→MATC ENTER
                  [[(1,2) (1,1)]
                   [(3,2) (4,3)]]
MATC^T ENTER     [[(1,-2) (3,-2)]
                  [(1,-1) (4,-3)]]
```

TwoVar

STAT CALC menu
(TwoVa shows on menu)

TwoVar *xList*,*yList*,*frequencyList*

Performs two-variable statistical analysis on the real data pairs in *xList* and *yList*, using the frequencies in *frequencyList*.

Values used for *xList*, *yList*, and *frequencyList* are stored automatically to the built-in variables **xStat**, **yStat**, and **fStat**, respectively.

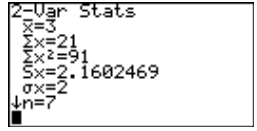
TwoVar *xList*,*yList*

Uses frequencies of 1.

TwoVar

Uses **xStat**, **yStat**, and **fStat** for *xList*, *yList*, and *frequencyList*. These built-in variables must contain valid data of the same dimension; otherwise, an error occurs.

```
{0,1,2,3,4,5,6}→L1 ENTER
                   {0 1 2 3 4 5 6}
{0,1,2,3,4,5,6}→L2 ENTER
                   {0 1 2 3 4 5 6}
TwoVar L1,L2 ENTER
```



Scroll down to see more results.

unitV

VECTR MATH menu

unitV *vector*

Returns a unit vector of a real or complex *vector*, where:

unitV [a,b,c] returns $\left[\frac{\mathbf{a}}{\mathbf{norm}}\ \frac{\mathbf{b}}{\mathbf{norm}}\ \frac{\mathbf{c}}{\mathbf{norm}}\right]$

and

norm is $\sqrt{(\mathbf{a}^2+\mathbf{b}^2+\mathbf{c}^2)}$.

In **RectV** vector coordinate mode:

```
unitV [1,2,1] ENTER
              [.408248290464 .8164…
```

vc▸li

LIST OPS menu
VECTR OPS menu

vc▸li *vector*

Returns a real or complex *vector* converted to a list.

```
vc▸li [2,7,-8,0] ENTER      {2 7 -8 0}
(vc▸li [2,7,-8,0])² ENTER
                          {4 49 64 0}
```

Vector entry: [ ]

2nd [{] and 2nd [}]

[*element1*,*element2*,…**]**

Defines a vector in which each element is a real or complex number or variable.

```
[4,5,6]→VEC ENTER          [4 5 6]
```

In **PolarC** complex number mode:

```
[5,(2∠π/4)]→VEC ENTER
                  [(5∠0) (2∠.785398163…
```

Vert

† GRAPH DRAW menu

Vert *xValue*

Draws a vertical line on the current graph at *xValue*.

In a **ZStd** graph screen:

```
Vert -4.5 ENTER
```

While

‡ program editor
CTL menu

:While *condition*
:*commands-while-true*
:End
:*command*

Executes *commands-while-true* as long as *condition* is true.

Program segment:

```
  ⋮
:1→J
:0→TEMP
:While J≤20
: TEMP+1/J→TEMP
: J+1→J
:End
:Disp "Reciprocal sums to
20",TEMP
  ⋮
```

xor

BASE BOOL menu

integerA **xor** *integerB*

Compares two real integers bit by bit. Internally, both integers are converted to binary. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value is the sum of the bit results.

For example, 78 **xor** 23 = 89.

78 = 1001110**b**
23 = 0010111**b**
1011001**b** = 89

You can enter real numbers instead of integers, but they are truncated automatically before the comparison.

In **Dec** number base mode:

```
78 xor 23 ENTER                    89
```

In **Bin** number base mode:

```
1001110 xor 10111 ENTER
                              1011001b
Ans▸Dec ENTER                    89d
```

xyline

† STAT DRAW menu

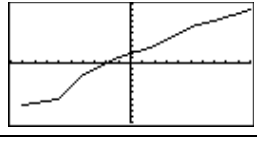
xyline *xList*,*yList*

Draws a line plot on the current graph, using the real data pairs in *xList* and *yList*.

xyline

Uses the data in built-in variables **xStat** and **yStat**. These variables must contain valid data of the same dimension; otherwise, an error occurs.

```
{-9,-6,-4,-1,2,5,7,10}→XL ENTER
                  {-9 -6 -4 -1 2 5 7 1…
{-7,-6,-2,1,3,6,7,9}→YL ENTER
                   {-7 -6 -2 1 3 6 7 9}
ZStd:xyline XL,YL ENTER
```



ZData

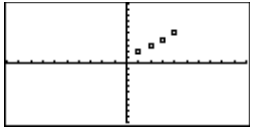
† GRAPH ZOOM menu

ZData

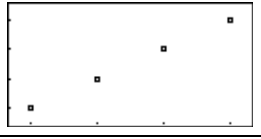
Adjusts the window variable values based on the currently defined statistical plots so that all stat data points will be plotted, and then updates the graph screen.

In **Func** graphing mode:

```
{1,2,3,4}→XL ENTER          {1 2 3 4}
{2,3,4,5}→YL ENTER          {2 3 4 5}
Plot1(1,XL,YL) ENTER             Done
ZStd ENTER
```



```
ZData ENTER
```



ZDecm

† GRAPH ZOOM menu

ZDecm

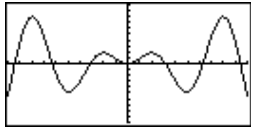
Sets the window variable values such that Δ**x**=Δ**y**=.**1**, and then updates the graph screen with the origin centered on the screen.

xMin=⁻6.3 yMin=⁻3.1
xMax=6.3 yMax=3.1
xScl=1 yScl=1

One of the benefits of **ZDecm** is that you can trace in .1 increments.

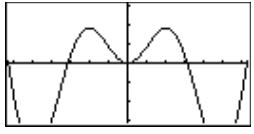
In **Func** graphing mode:

```
y1=x sin x ENTER                    Done
ZStd ENTER
```



If you trace the graph above, **x** values start at 0 and increment by .1587301587.

```
ZDecm ENTER
```



If you trace this graph, the **x** values increment by .1.

ZFit

† GRAPH ZOOM menu

ZFit

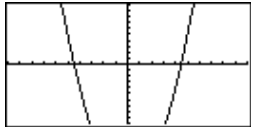
Recalculates **yMin** and **yMax** to include the minimum and maximum **y** values of the selected functions between the current **xMin** and **xMax**, and then updates the graph screen.

This does not affect **xMin** and **xMax**.

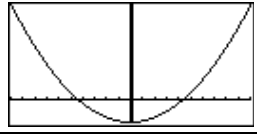
In **Func** graphing mode:

`y1=x²−20` ENTER
`ZStd` ENTER

Done



`ZFit` ENTER



ZIn

† GRAPH ZOOM menu

ZIn

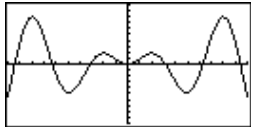
Zooms in on the part of the graph centered around the current cursor location.

Zoom factors are set by the values of built-in variables **xFact** and **yFact**; the default is 4 for both factors.

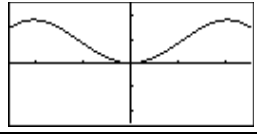
In **Func** graphing mode:

`y1=x sin x` ENTER
`ZStd` ENTER

Done



`ZIn` ENTER



ZInt

† GRAPH ZOOM menu

ZInt

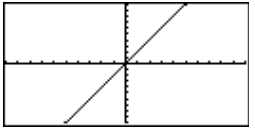
Sets the window variable values so that each pixel is an integer in all directions (Δ**x**=Δ**y**=**1**), sets **xScl**=**yScl**=**10**, and then updates the graph screen.

The current cursor location becomes the center of the new graph.

One of the benefits of **ZInt** is that you can trace in whole number increments.

In **Func** graphing mode:

`y1=der1(x²-20,x)` ENTER Done
`ZStd` ENTER



If you trace the graph above, **x** values start at 0 and increment by .1587301587.

`ZInt` ENTER

If you trace this graph, **x** values increment by 1.

ZOut

† GRAPH ZOOM menu

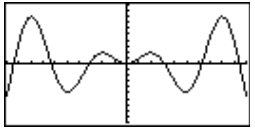
ZOut

Zooms out to display more of the graph, centered around the current cursor location.

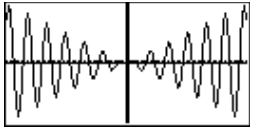
Zoom factors are set by the values of built-in variables **xFact** and **yFact**; the default is 4 for both factors.

In **Func** graphing mode:

```
y1=x sin x ENTER                    Done
ZStd ENTER
```



```
ZOut ENTER
```



ZPrev

† GRAPH ZOOM menu

ZPrev

Replots the graph using the window variable values of the graph that was displayed before you executed the previous **ZOOM** instruction.

ZRcl

† GRAPH ZOOM menu

ZRcl

Sets the window variables to values stored previously in the user-defined zoom-window variables, and then updates the graph screen.

To set user-defined zoom-window variables, either:

- Press GRAPH F3 MORE MORE MORE F1 (**ZSTO**) to store the current graph's window variables.
  – or –
- Store the applicable values to the zoom-window variables, whose names begin with **z** followed by the regular window variable name. For example, store a value for xMin to **zxMin**, yMin to **zyMin**, etc.

ZSqr

† GRAPH ZOOM menu

ZSqr

Sets the window variable values to produce "square" pixels where $\Delta x = \Delta y$, and then updates the graph screen.

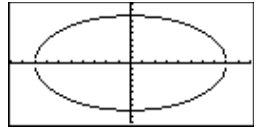
The center of the current graph (not necessarily the axes intersection) becomes the center of the new graph.

In other types of zooms, squares may look like rectangles and circles may look like ovals. Use **ZSqr** for a more accurate shape.

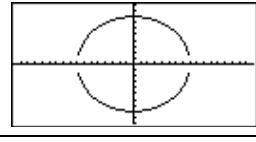
In **Func** graphing mode:

`y1=√(8²−x²):y2=⁻y1` ENTER Done

`ZStd` ENTER



`ZSqr` ENTER



ZStd

† GRAPH ZOOM menu

ZStd

Sets the window variables to the standard default values, and then updates the graph screen.

Func graphing mode:

xMin=–10	**yMin=–10**
xMax=10	**yMax=10**
xScl=1	**yScl=1**

Pol graphing mode:

θMin=0	**xMin=–10**	**yMin=–10**
θMax=6.28318530718 (2π)	**xMax=10**	**yMax=10**
θStep=.130899693899… (π/24)	**xScl=1**	**yScl=1**

Param graphing mode:

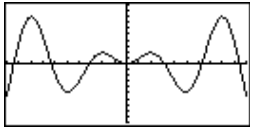
tMin=0	**xMin=–10**	**yMin=–10**
tMax=6.28318530718 (2π)	**xMax=10**	**yMax=10**
tStep=.130899693899… (π/24)	**xScl=1**	**yScl=1**

DifEq graphing mode:

tMin=0	**xMin=–10**	**yMin=–10**
tMax=6.28318530718 (2π)	**xMax=10**	**yMax=10**
tStep=.130899693899… (π/24)	**xScl=1**	**yScl=1**
tPlot=0		**difTol=.001**

In **Func** graphing mode:

```
y1=x sin x ENTER                Done
ZStd ENTER
```



ZTrig

† GRAPH ZOOM menu

ZTrig

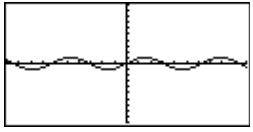
Sets the window variables to preset values appropriate for plotting trig functions in **Radian** angle mode (Δ**x**=π/24), and then updates the graph screen.

xMin=-8.24668071567	**yMin=-4**
xMax=8.24668071567	**yMax=4**
xScl=1.5707963267949 (π/2)	**yScl=1**

In **Func** graphing mode:

```
y1=sin x ENTER
ZStd ENTER
```

Done



```
ZTrig ENTER
```