Appendix D: Command Quick Reference Guide

The following quick reference guide is a condensation of the command reference in the *TI-89/TI-92 Plus Guidebook*. It consists of the following sections:

- Summary of commands
- Reserved system variable names
- EOS (Equation Operating System) Hierarchy

I have included this information in the tip list to save you the trouble of referring to the guidebook while using the tip list. It may also be useful in its own right, if printed out at reduced size to be carried with your calculator.

Note that the codes for *setMode()*, *getMode()*, *setGraph()* and *setTable()* are shown in tables in the respective function definitions. These tables include the more recent numeric string codes.

Quick reference guide to functions and commands

The function or command name is shown in **bold** text as the first line of the description. The lines immediately following show the arguments, if any. Arguments are shown in *italic* text. Optional arguments are shown in square brackets []. *expr* is an expression, *var* is a variable. If a function can accept more than one type of argument, they may be shown on one line separated by commas, for example,

(*expr*), (*list*), (*matrix*)

If a function may optionally take no arguments, that is indicated as (). Program and function structures may be shown with statements separated with the colon :, which is the symbol actually used for that purpose in programs. Functions and commands indicated with symbols are shown at the end of the reference guide.

abs()
(*expr*), (*list*), (*matrix*)
Return absolute value of real argument or modulus of complex argument.

and
expr1 and *expr2*
list1 and *list2*
matrix 1 and *matrix 2*
Return *true* or *false* or a simplifed form.

integer1 and *integer2*
Bit-by-bit 32-bit integer compare. Arguments larger than 32-bit signed values are reduced with symmetric modulo operation. Mode must be Auto or Exact.

AndPic *picVar [, row, column]*
Logical-AND of graph screen and *picvar* at (*row, column*); default is (0,0)

angle()
(expr), (list), (matrix)
Return angle of arguments interpreted as complex numbers. Undefined variables are treated as real variables.

ans()
(), (integer)
Return answer from home screen. *integer* can be 1 to 99; not an expression; default is zero.

approx()
(expr), (list), (matrix)
Evaluate argument as decimal value if possible.

Archive *var1[,var2] [,var3] ...*
Move variables to flash ROM.

arcLen()
(expr,var,start,end), *(list,var,start,end)*
Return arc length of *expression* (or each element of *list*) with respect to *var* from *start* to *end.*

augment()
(list1,list2)
Return list which is *list2* appended to *list1*

(*matrix1,matrix2*)
Append *matrix2* appended to *matrix1* as columns.

(*matrix1;matrix2*)
Append *matrix2* to *matrix1* as rows.

avgRC(*expr,var* [,*h*]**)**
Return forward-difference quotient (average rate of change) of *expr* with respect to *var*. *h* is the step value; if omitted, *h*=0.001

▸Bin
integer1▸Bin
Convert *integer1* to binary. Non-decimal integers must be preceeded with 0b or 0h. Arguments larger than 32-bit signed values are reduced with symmetric modulo operation.

BldData [*datavar]*
Create the data variable *datavar* based on the current graph settings. If *datavar* is omitted, the system variable *sysdata* is used. The variable is a table of function values evaluated at each plot point.

ceiling()
(*expr*), (*list*), (*matrix*)
Return the nearest integer that is greater than or equal to *expr* or each element of *list* or *matrix.* Argument may be real or complex.

cFactor()
(expr[,var])

(list[,var]), (matrix[,var])
Return complex factors of argument over a common denominator. If *var* is omitted, argument is factored with respect to all variables. If *var* is used, cfactor() tries to factor the argument toward factors which are linear in *var*.

char(*integer***)**
Return string of character indicated by *integer*. *Integer* must be in the range 0 to 255.

Circle *x*, *y,r [,drawmode]*
Draw a circle on the Graph with radius *r* and center (*x*, *y*). All arguments are in window coordinate units.
drawmode = 1: draw circle (default)
drawmode = 0: turn off the circle
drawmode = -1: invert circle pixels

ClrDraw
Clear the Graph screen; reset Smart Graph

ClrErr
Clear error status and internal error context variables. Sets *errnum* to zero.

ClrGraph
Clears graphed functions or expressions.

ClrHome
Clears the home screen, and sets arbitrary constant suffixes to 1.

ClrIO
Clears the program I/O screen.

ClrTable
Clears table settings which apply to the *ASK* setting in the *Table Dialog* setup box.

colDim(*matrix***)**
Return number of rows of *matrix*.

colNorm(*matrix***)**
Return maximum of sums of absolute values of *matrix* column elements.

comDenom()
(*expr [,var]*)
(*list [,var]*)
(*matrix [,var]*)
Return reduced ratio of expanded numerator over expanded denominator, with respect to *var* if used. Using *var* can save time, memory and screen space, and result in expressions on which further operations are less likely to result in *Memory* errors.

conj()
(*expr*), (*list*), (*matrix*)
Return the complex conjugate of the argument. Undefined variables are treated as real.

CopyVar
var1, var2
Copy the contents of *var1* to *var2*. Unlike the *store* operation (→), *CopyVar* does not simplify the source variable. *CopyVar* must be used with non-algebraic variable types such as Pic and GDB.

cos()
(*expr*), (*list*)
Return the cosine of the argument. You may override the current angle mode with ° or ʳ.

(*matrix*)
Return the matrix cosine of square diagonalizable *matrix*, which is not the cosine of each element. Symbolic elements must have assigned values. The matrix cosine is calculated with floating-point arithmetic (Approx mode).

cos^{-1}()
(*expr*), (*list*)
Return the angle whose cosine is *expression*.

(*matrix*)
Return the matrix inverse cosine of square diagonalizable *matrix*, which is not the inverse cosine of each element.Results are found with floating-point arithmetic.

cosh()
(*expr*), (*list*)
Return the hyperbolic cosine of the argument.

(*matrix*)
Return the hyperbolic cosine of square diagonalizable *matrix*, which is not cosh() of each element. Results are found with floating-point arithmetic.

cosh^{-1}()
(*expr*), (*list*)
Return the inverse hyperbolic cosine of the argument.

(*matrix*)
Return the inverse hyperbolic cosine of square diagonalizable *matrix*, which is not $\cosh^{-1}()$ of each element. Results are found with floating-point arithmetic.

crossP()
(*list1*,*list2*), (*matrix1*,*matrix2*)
Return the cross product of the arguments. Lists must have equal dimensions of 2 or 3. Matrices must both be either row or column vectors of equal dimensions 2 or 3.

cSolve(*equation*,*var***)**
Return real and complex solutions for *var*. Complex results are possible in Real mode. Fractional powers with odd denominators use the principle branch, not the real branch. Specify complex variables with the underscore '_' suffix.

(*equation1* and *equation2* [and ...]
,{*var1*,*var2*[,...]})
Return real and complex solutions for *var* to system of equations *equation1*, *equation2*, ... *var* may be a variable or a variable with a solution guess such as x=1+3i. Not all equation variables need be in the solution variable list. A close complex solution guess may be needed to get a solution.

CubicReg
xlist,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate cubic polynomial regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is y = a*x^3 + b*x^2 + c*x + d.

cumSum()
(*list*), (*matrix*)
Return list of cumulative sum of elements or list, or return matrix of cumulative sums of columns from top to bottom.

CustmOff
Remove a custom toolbar.

CustmOn
Activate a custom toolbar as set up by Custom ... EndCustom block.

Custom
Set up a custom toolbar. Similar to ToolBar except that Title and Item statements cannot have labels.

Cycle
Transfer program control immediately to the next iteration of the current For, While or Loop loop.

CyclePic
picString, *n* [, [*wait*], *cycles*], [*direction*]]
Display each of *n* picture variables specified by *picString* for *wait* seconds. *cycles* specifies the number of display cycles. *direction* is set to 1 (default) for a circular cycle or to -1 for a forward-backward cycle. To display three variables *pic1, pic2* and *pic3*, *picString* is "pic" and *n* is 3.

▸Cylind
vector ▸Cylind
Display row or column *vector* in cylindrical form [r∠θ, z]. *vector* must be a row or column vector with three elements.

cZeros(*expr*,*var***)**
Equivalent to exp▸list(cSolve(*expr*=0,*var*),*var*). Return list of real and complex solutions for *var* which make *expr*=0. Use the underscore '_' suffix to specify complex variables. *var* may be a variable name or a solution guess such as x = 1 + 3i.

({*expr1*,*expr2* [,...]},{*var1*,*var2* [,...]})
Return matrix of solutions for *var1, var2, ...* where the expressions are simultaneously zero. *var1, var2, ...* may be variable names or solution guesses such as x = 1 + 3i. Solution zeros may include real and complex solutions. Each solution matrix row represents a zero, with the components in the same order as the variable list. You need not include all the expression variables in the variable list. You may also include variables which are not in the expressions. Complex guesses are often necessary, and the guess may have to be close to the solution.

d*(expr*,*var* [,*order*]**)**
(*list*,*var* [,*order*])
(*matrix*,*var* [,*order*])
Return derivative of *order* of *expr*, *list* or *matrix* argument with respect to *var*. *order* must be an

integer. If *order* is less than zero, the anti-derivative is returned. *d*() does not fully simplify the *expression* and *var* before finding the derivative.

▸DD
number▸DD
list▸DD
matrix▸DD
Return decimal equivalent of the angle argument. The current Angle Mode sets the argument as degrees or radians. The argument may be radians.

▸Dec
number▸Dec
Convert *number* to a decimal number, regardless of the current Base mode.

Define
Define *functionName*(*arg1*,*arg2*, ...)=*expr*
Create function *functionName()* with arguments *arg1, arg2. fName()* evaluates *expression* with the supplied arguments and returns the result. Do not use *arg1* or *arg2* as arguments when calling *functionName()*. This form of Define is equivalent to *expr*→*functionName*(*arg1*,*arg2*).

Define *functionName*(*arg1*,*arg2*,...)=Func
block
EndFunc
Same as above except that *block* can include multiple expressions and use Return to return the result.

Define *programName*(*arg1*,*arg2*,...)=Prgm
block
EndPrgm
Same as above except that program *programName()* is defined. Programs cannot return results.

DelFold *folder1* [,*folder2*, *folder3*, ...]
Delete folders. The *Main* folder cannot be deleted. An error message is shown if any folder contains variables.

DelVar *var1 [var2, var3, ...]*
Delete variables from memory.

deSolve(*ode12*,*indVar*,*depVar***)**
Return general solution of 1st- or 2nd order ordinary differential equation *ode12* with independent variable *indVar* and dependent variable *depVar*. The prime symbol ' indicates the 1st derivative, two prime symbols ' ' indicate the second derivative.

(*ode1* and *inCond*,*indVar*,*depVar*)
Return particular solution of 1st-order differential equation *ode1* with initial condition *inCond*. *inCond* is in the form *depVar*(*inIndVar)* = *inDepVar;* for example, y(0) = 1. *inIndVar* and *inDepVar* can be variables with no stored values.

(*ode2* and *inCond1* and *inCond2*,*indVar*,*depVar*)
Return particular solution of 2nd-order differential equation *ode2* with initial conditions *inCond1* and *inCond2*. *inCond1* specifies the value of the independent variable at a point, in the form *depVar(inIndVal)* = *inDepVal*. *inCond2* specifies the value of the first derivative at a point in the form *depVar'*(*inIndVal*) = *in1stDeriv*.

(*ode2* and *BndCnd1* and *BndCnd2*,*indVar*, *depVar*)
Return particular solution of 2nd-order differential equation with boundary conditions *BncCnd1* and *BndCnd2*.

det(*matrix* [,*tol*]**)**
Return determinant of square *matrix*. Any element less than *tol* is considered zero. The default *tol* is 5E-14 * max(dim(*matrix*)) * rowNorm(*matrix*).

diag()
(*list*), (*rowMatrix*), (*colMatrix*)
Return matrix with main diagonal elements of argument.

(*matrix*)
Return row matrix whose elements are the main diagonal elements of square *matrix*.

Dialog
Dialog : *block* : EndDlog
Display a dialog box during program execution. *block* consists of Text, Request, DropDown and Title commands. Dialog box variables are displayed as default values. If [ENTER] is pressed, the variables are updated and the system variable *ok* is set to 1. If [ESC] is pressed, the variables are not updated and *ok* is set to zero.

dim(*list*)
Return dimension of list.

(*matrix*)
Return list of dimensions of *matrix* as {rows,columns}

(*string*)
Return dimension (number of characters) of *string*.

Disp [*expOrString1*], [*expOrString2*], ...
Display the program I/O screen. If any arguments (expressions or strings) are used, they are displayed on separate screen lines. Arguments are displayed in Pretty Print if Pretty Print mode is On.

DispG
Display the Graph Screen.

DispHome
Display the Home screen.

DispTbl
Display the Table screen. Use the cursor pad to scroll. Press [ENTER] or [ESC] to resume program operation.

▸DMS
expr ▸DMS, *list* ▸DMS, *matrix* ▸DMS
Display the argument as an angle in format DDDDDD°MM'SS.ss". ▸DMS converts from radians in radian mode.

dotP()
(*list1*,*list2*), (*vector1*,*vector2*)
Return the dot product of two lists or vectors. Vectors must both be row or column vectors.

DrawFunc *expr*
Draw *expr* as a function of *x* on the Graph screen.

DrawInv *expr*
Draw the inverse of *expr* on the Graph screen by plotting *x* values on the *y* axis and vice versa.

DrawParm *expr1, expr2* [,*tmin*] [,*tmax*] [,*tstep*]
Draw parametric expressions *expr1* and *expr2* with *t* as the independent variable, from *tmin* to *tmax* with *tstep* between each evaluated *t* value. The current Window variables are the defaults for *tmin, tmax* and *tstep*. Using the *t*-arguments does not change the Window settings. The *t*-arguments must be used if the current Graph mode is not Parametric.

DrawPol *expr,* [,θ*min*] [,θ*max*] [,θ*step*]
Draw *expr* as a polar graph with independent variable θ. Defaults for θ*min*, θ*max* and ,θ*step* are the current Window settings. Specifying θ-arguments does not change the Window settings. Theθ-arguments must be used if the Graph mode is not Polar.

DrawSlp *x1*, *y1*, *slope*
Display the Graph screen and draw the line y = *slope**(x-*x1*) + *y1*.

DropDown *title*, {*item1*, *item2*, ...},*var*
Display a drop-down menu with the name *title* in a Dialog box. The drop-down menu choices are 1:*item2*, 2:*item2* and so on. The number of the selected item is stored in *var*. If *var* exists and its value is in the range of items, the referenced item is the default menu selection.

DrwCtour
expr
list
Draw contours on the current 3D graph, in addition to the contours specfied by the *ncontour* system variable. *expr* or *list* specifies the z-values at which the contours are drawn. 3D graph mode must be set. *DrwCtour* sets the graph style to CONTOUR LEVELS.

E (enter exponent)
*mantissa*E*exponent*
Enter a number in scientific notation as *mantissa* * $10^{exponent}$. To avoid a decimal value result, use $10^{integer}$, instead.

***e*^()**
(*expr*)
Raise *e* to the *expr* power. *e* is the natural logarithm base 2.718..., not the character 'e'. In Radian angle mode, you may enter complex numbers in the polar format $re^{i\theta}$.

(*list*)
Return list of *e* raised to each element in *list*.

(*matrix*)
Return the matrix exponential of square *matrix*, which is not the same as *e* raised to the power of each element. See cos() for calculation

details. *matrix* must be diagonalizable. The result always contains floating-point numbers.

eigVc(*matrix***)**
Return matrix of eigenvectors of square *matrix*, whose elements may be real or complex. The eigenvectors are not unique, but are normalized such that if the eigenvector is

$$V = [x_1, x_2, \ldots, x_n]$$

then

$$\sqrt{x_1^2 + x_2^2 + \ldots x_n^2} = 1$$

eigVl(*matrix***)**
Return the eigenvalues of square diagnonalizable *matrix*, whose elements may be real or complex.

Else See If
ElseIf See If

EndCustm See Custom
EndDlog See Dialog
EndFor See For
EndFunc See Func
Endif See If
EndLoop See Loop
EndPrgm See Prgm
EndTBar See ToolBar
EndTry See Try
EndWhile See While

entry([*integer***])**
Return previous entry-line expression from the history area. If used, *integer* cannot be an expression and must be in the range 1 to 99. See also ans().

exact()
(*expr* [,*tol*]), (*list* [,*tol*]), (*matrix* [,*tol*])
Evaluate argument with Exact mode arithmetic regardless of the current Mode setting. *tol*, if used, specifies the conversion tolerance and is zero by default.

Exec *string* [,*expr1*] [,*expr2*] ...
Execute *string* interpreted as Motorola 68000 assembly language op-codes. *expr1*and *expr2* are optional input arguments. Misinformed use of *Exec* can lock up the calculator and cause data loss.

Exit
Exit the current For, While or Loop block.

exp▸list(*expr*,*var***)**
Convert *expr* to a list of the right-hand sides of equations separated by 'or' in *expr*. Used to extract individual solutions from results of solve(), cSolve(), fMin() and fMax().

expand()
(*expr*), (*list*), (*matrix*)
Expand argument with respect to all variables with transformation into a sum and/or difference of simple terms. See also Factor()

(*expr,var*), (*list*,*var*), (*matrix*,*var*)
Expand argument with respect to *var* by collecting similar powers of *var* and sorting the factors with *var* as the main variable. expand() also distributes logarithms and fractional powers regardless of *var*. Specifying *var* can save time, memory and result in a smaller expression. If the argument only contains one variable, specifying it as *var* may result in more complete partial factorization. propFrac() is faster but less extreme than expand(). See also comDenom() and tExpand().

expr(*string***)**
Return the evaluated expression in *string*.

ExpReg
xlist,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate exponential polynomial regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is y = a*b^x.

factor()
(*expr*), (*list*), (*matrix*)
Factor argument with respect to all variables over a common denominator. Argument is factored as much as possible toward linear rational factors without introducing new non-real sub-expressions.

(*expr*,*var*), (*list*,*var*), (*matrix*,*var*)
Factor argument with respect to *var*, as much as possible toward real factors linear in *var*, even if irrational constants or subexpressions are

introduced. Also, comDenom() can achieve partial factoring if factor() is too slow or exhausts memory. See also cFactor()

(*ratNum*)
Return factors of rational number *ratNum*. Use isPrime(), instead, to more quickly determine if *ratNum* is prime.

Fill
expr,matVar
expr,listVar
Replace each element in *matVar* or *listVar* with *expr. matVar* and *listVar* are variable names, and the variables must exist.

floor()
(*expr*), (*list*), (*matrix*)
Return greatest integer which is less than or equal to the argument. The argument may be real or complex. floor() is identical to int(). See also ceiling().

fMax(*expr*,*var***)**
Return Boolean expression specifying possible values of *var* which maximize *expr* with respect to *var*, or locate the least upper bound. Use the "with" operator "|" to limit the solution range or add constraints. In Approx mode, fMax() finds one approximate local maximum.

fMin(*expr*,*var***)**
Return Boolean expression specifying possible values of *var* which minimize *expr* with respect to *var*, or locate the greatest lower bound. Use the "with" operator "|" to limit the solution range or add constraints. In Approx mode, fMin() finds one approximate local minimum.

FnOff [1] [,2] ... [,99]
Deselects all Y= functions, with no arguments. Optional arguments specify Y= functions to deselect.

FnOn [1] [,2] ... [,99]
Selects all Y= functions, with no arguments. Optional arguments specify Y= functions to select. In 3D mode only, selecting any function deselects all other functions.

For
For *var*,*low*,*high* [,*step*] : *block* : EndFor
Execute each statement in *block* iteratively for each value of *var*. *var* ranges from *low* to *high* and increments by *step. step* may be positive or negative and the default is 1. *var* cannot be a system variable.

format(*expr*,*format***)**
Return expression *expression* as a string formatted with the numeric format string *format*:

"F[*n*]" Fixed format; *n* is number of digits after the decimal point.

"S[*n*]" Scientific format; *n* is the number of digits after the decimal point.

"E[*n*]" Engineering format; *n* is the number of digits after the first significant digit. Mantissa radix point is adjusted so that exponent is a power of three.

"G[*n*][*c*]" General format; samed as fixed, but digits to the left of the radix are separated in groups of three, separated by the *c* character. By default *c* is a comma; if *c* is a period, the radix point is a comma.

The suffix [R*c*] may be added to any of the *format* codes. *c* is a single character which specifies the radix point character.

fpart()
(*expr*), (*list*), (*matrix*)
Return the fractional part of the argument, which may be real or complex.

Func
Func : *block* : EndFunc
Func is required as the first statement of a multi-statement function definition.

gcd()
(*number1*,*number2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the greatest common denominator of the arguments. The GCD of two fractions is the GCD of the numerators divided by the least common multiple of the denominators. The GCD of fractional floating-point numbers is 1.0 in Auto or Approx mode. If the arguments are lists or matricec, the GCD of each corresponding element is returned.

Get *var*
Retrieve a CBL or CBR value from the link port and store it in *var*.

GetCalc *var*
Retrieve a value from the link port from another calculator and store it in *var*.

GetConfg
Return a list of calculator attribute pairs. The first pair element is a string which specifies the attribute, and the second pair element is the attribute. The attribute names are the same for the TI-89 and the TI-92+, but the attributes may differ. The "Cert. Rev. #" attribute pair appears in the list only if a certificate has been installed.

{
"Product Name",*productName*,
"Version","*versionString*",
"Product ID",*idString*,
"ID#",*id#String*,
"Cert. Rev. #",*certRev#*,
"Screen Width",*screenWidth*,
"Screen Height",*screenHeight*,
"Window Width",*windowWidth*,
"Window Height",*windowHeight*,
"RAM size",*ramSize*,
"Free RAM",*freeRAM*,
"Archive Size",*archiveSize*,
"Free Archive",*freeArchive*
}

getDenom(*expr***)**
Return the reduced common denominator of *expression*.

getFold()
Return a string which is the current folder name.

getKey()
Return key code of a pressed key as an integer, or return zero if no key is pressed.

getMode(*modeString***)**
Return a string which is the current setting for *modeString*.

("ALL")
Return a list of string pairs of all Mode settings. The first element of the string pair is the mode name string; the second element is the mode setting string. See SetMode() for possible settings.

{
"Graph", *graphType*,
"Display Digits", *digitsFormat*,
"Angle", *angleUnits*,
"Exponential Format", *expFormat*,
"Complex Format", *complexFormat*,
"Vector Format", *vectorFormat*,
"Pretty Print", *prettyPrintStatus*,
"Split Screen", *splitScreenMode*,
"Split 1 App", *app1Name*,
"Split 2 App", *app2Name*,
"Number of Graphs", *numGraphs*,
"Graph 2", *graphType*,
"Split Screen Ratio", *ratio*,
"Exact/Approx", *exactApproxMode*,
"Base", *numberBase*
}

getNum(*expr***)**
Return numerator of *expression* reduced to a common denominator.

getType(*varName***)**
Return string indicating the data type of variable *varName*.

"ASM" assembly-language program
"DATA" Data type
"EXPR" expression; includes complex, arbitrary, undefined, ∞, -∞, TRUE, FALSE, π, e
"FUNC" Function
"GDB" Graph data base
"LIST" List
"MAT" Matrix
"NONE" Variable does not exist
"NUM" Real number
"OTHER" Reserved for future use
"PIC" Picture
"PRGM" Program
"STR" String
"TEXT" Text type
"VAR" Name of another variable

getUnits()
Return a list of strings which specify the default units. Constants, temperature, amount of a substance, luminous intensity and acceleration are not included. The list has the form {"*system*","*cat1*","*unit1*","*cat2*","*unit2*",...} where *system* is the unit system: SI, ENG/US or CUSTOM. The *cat* strings specify the category, and the *unit* strings specify the corresponding default units.

Goto *label*
Transfer program control to *label*.

Graph
expr [,*var*] (function graph)
xExpr,yExpr [,*var*] (parametric graph)
expr [,θvar] (polar graph)
expr [,*xvar*] [,*yvar*] (3D graph)
Graph the *expr* argument with the current Graph mode. Expressions created with Graph (or Table) are assigned increasing function numbers starting with 1. Modify or delete them with the [F4] Header function in the table display. The currently selected Y= functions are not graphed. The independent variable of the current graph mode is used if a *var* argument is omitted. Use ClrGraph to clear the functions, or start the Y= editor to enable the system variables.

▸Hex
integer1▸Hex
Convert *integer1* to hexadecimal. Non-decimal integers must be preceeded with 0b or 0h. Arguments larger than 32-bit signed values are reduced with symmetric modulo operation.

identity(*expr***)**
Return identitiy matrix with dimension of *expr*.

If
If *BooleanExpr* : *statement*
Execute single *statement* if *BooleanExpr* evaluates to TRUE.

If *BooleanExpr* then : *block* :EndIf
Execute *block* if *BooleanExpr* evaluates to TRUE.

If *BooleanExpr* then : *block1*
Else : *block2* : EndIf
If *BooleanExpr* evaluates to TRUE, execute *block1* but not *block2*; otherwise execute *block2* but not *block1.*

If *BooleanExpr1* Then : *block1*
ElseIf *BooleanExpr2* Then : *block2*
...
ElseIf *BooleanExprN* Then : *blockN*
EndIf
Execute *block1* only if *BooleanExpr1* evaluates to TRUE, execute *block2* only if *BooleanExpr2* evaluates to TRUE, etc.

imag()
(*expr*), (*list*), (*matrix*)
Return the imaginary part of the argument. All undefined variables are treated as real variables.

Input [[*promptString*,]*var*]
If no arguments, pause program execution, display the Graph screen and update *xc* and *yc* (or *rc* and θ*c* in Polar mode) by positioning the cursor.

If argument *var* is used, the program pauses, displays *promptString* on the Program I/O screen and waits for the entry of an expression. The expression is stored in *var*. If *promptString* is omitted, "?" is displayed for the prompt.

InputStr [*promptString*,]*var*
Pause program execution, display *promptString* on the Program I/O screen and wait for the entry of an expression, which is stored as a string in *var*. "?" is displayed as a prompt if *promptStrin* is omitted.

inString(*sourceString*,*targetString*[,*start*]**)**
Return position at which *targetString* starts in *sourceString*. The search for *targetString* begins at character number *start*, if *start* is used. *start* is 1 by default.

int()
(*expr*), (*list*), (*matrix*)
Return the greatest integer that is less than or equal to the argument, which may be real or complex. int() is identical to floor().

intDiv()
(*number1*,*number2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the signed integer part of the first argument divided by the second argument.

integrate
See ∫().

iPart()
(*number*), (*list*), (*matrix*)
Return the integer part of the argument, which may be real or complex.

isPrime(*number***)**
Return True or False to indicate if *number* is prime. Display error message if *number* has more than about 306 digits and no factors less than or equal to 1021.

Item
itemNameString
itemNameString,label
Set up a drop-down menu within Custom ... EndCustm, or ToolBar ... EndTBar blocks. In a Custom block, specifies the text that is pasted. In a ToolBar block, specifies the branch label.

Lbl *labelName*
Define a label with name *labelName* in a program. Goto *labelName* transfers control to the instruction following *labelName*. *labelName* must meet same requirements as variable names.

lcm()
(*number1,number2*)
(*list1,list2*)
(*matrix1*,*matrix2*)
Return the least common multiple of the two arguments. The LCM of two fractions is the LCM of the numerators divided by the GCM of the denominators. The LCM of fractional floating point numbers is their product.

left()
(*sourceString*[,*num*])
Return the leftmost *num* characters from *sourceString*.

(*list*[,*num*])
Return the leftmost *num* elements of *list*.

(*comparison*)
Return the left-hand side of the equality or inequality of *comparison*.

limit()
(*expr*,*var*,*point*[,*direction*])
(*list*,*var*,*point*[,*direction*])
(*matrix*,*var*,*point*[,*direction*])
Return the limit of the argument *expr, list*, or *matrix* with respect to *var* at *point*. The limit is taken from the left if *direction* is negative, from the right if *direction* is positive, or from both directions otherwise. The default *direction* is both.

Line *xStart*,*yStart*,*xEnd*,*yEnd*[,*mode*]
Display Graph screen and draw a line, including the endpoints, from (*xStart*,*yStart*) to (*yStart*, *yEnd*) according to *mode*:
mode=1: draw the line (default)
mode=0: turn off the line
mode=-1: invert the line

LineHoriz *y*[,*mode*]
Display Graph screen and draw a horizontal line at window coordinate *y*, according to *mode*:
mode=1: draw the line (default)
mode=0: turn off the line
mode=-1: invert the line

LineTan *expr1*,*expr2*
Display Graph screen and draw line tangent to *expr1* at point x = *expr2*. The independent variable for *expr1* is x.

LineVert *x*[,*mode*]
Display Graph screen and draw a vertical line at window coordinate *x*, according to *mode*:
mode=1: draw the line (default)
mode=0: turn off the line
mode=-1: invert the line

LinReg *xlist*,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate linear regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is y = a*x + b.

list▸mat(*list*[,*rowElements*]**)**
Create a matrix whose elements are filled row-by-row from. *rowElements* specifies the number of elements in each matrix row, if used. The default is one matrix row. If *list* does not fill the matrix, zeros are added.

Δlist(*list***)**
Return a list which is the difference between successive elements of *list*. Each element of *list* is subtracted from the next element, so the returned list is always one element shorter than *list*.

ln()
(*expr*), (*list*)
Return the natural logarithm of the argument.

(*matrix*)
Return the matrix natural logarithm of square diagonalizable *matrix*. This is not the same as finding the logarithm of each matrix element.

Floating-point results are used. See cos() for calculation details.

LnReg *xlist*,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate logarithmic regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is y = a + b*ln(x).

Local *var1* [,*var2*, *var3*, ...]
Declare the arguments as local variables in a program or function. Local variables exist only during program or function execution. Local variables must be used for loop index variables and for storage in multi-line functions, since global variables are not allowed in functions.

Lock *var1* [,*var2*, *var3*, ...]
Lock the argument variables. Locked variables cannot be changed or deleted unless Unlock() is used.

log()
(*expr*), (*list*)
Return the base-10 logarithm of the argument.

(*matrix*)
Return the matrix base-10 logarithm of square diagonalizable *matrix*. This is not the same as finding the logarithm of each *matrix* element. Floating-point arithmetic is used. Refer to cos() for calculation details.

Logistic *xlist*,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate logistic regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is $y = \frac{a}{1+b \cdot e^{c \cdot x}} + d$ where *e* is the natural logarithm base.

Loop
Loop : *block* : EndLoop
Repeatedly execute *block* until a Goto or Exit instruction is executed.

LU *matrix*, *lMat*, *uMat*, *pMat* [,*tol*]
Calculator the Doolittle LU (lower-upper) matrx decomposition or real or complex *matrix*. *matrix* must be a matrix name. The results are
lMat = lower triangular matrix
uMat = upper triangular matrix
pMat = permutation matrix
such that *lMat* * *uMat* = *pMat* * *matrix*
Any matrix element is set to zero if its value is less than *tol* and *matrix* contains no floating-point or symbolic elements. The default value for *tol* is 5E-14 * max(dim(*matrix*)) * rowNorm(*matrix*). Computations are done with floating-point arithmetic in Approx mode.

mat▸list(*matrix***)**
Return a list whose elements are those of *matrix*, row by row.

max()
(*expr1*,*expr2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the maximum of the arguments. List and matrix arguments are compared element by element.

(*list*)
Return the maximum element of *list*.

(*matrix*)
Return a row vector whose elements are the maximum elements of the columns of *matrix*.

mean(*list*[,*freqList*]**)**
Return the mean of the elements of *list*. *freqList* indicates the number of occurrences of the corresponding *list* element.

(*matrix*[,*freqMatrix*])
Return a row vector whose elements are the means of the *matrix* columns. *freqMatrix* indicates the number of occurrences of the corresponding *matrix* element.

median()
(*list*)
Return the median of the elements of *list*. All *list* elements must simplify to numbers.

(*matrix*)
Return a row vector whose elements are the medians of the columns of *matrix*. All *matrix* elements must simplify to numbers.

MedMed *xlist*,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate median-median regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is y = a*x + b.

mid()
For both variations of mid(): if *count* is omitted or greater than the dimension of the argument, the complete argument is returned. *count* must be greater than or equal to zero. If *count* = 0, an empty string or list is returned.

(*string*,*start*[,*count*])
Return *count* characters from *string* beginning at character position *start.*

(*list*,*start*[,*count*])
Return *count* elements of *string* beginning at element *start.*

min()
(*expr1*,*expr2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the minimum of the arguments. List and matrix arguments are compared element by element.

(*list*)
Return the minimum element of *list*.

(*matrix*)
Return a row vector whose elements are the minimum elements of the columns of *matrix*.

mod()
(*expr1*,*expr2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the first argument modulo the second argument where
mod(x,0) = x
mod(x,y) = x - y*floor(x/y)

The result is periodic in the second argument when that argument is non-zero. The result is either zero or has the same sign as the second argument.

MoveVar *var*,*curFolder*,*newFolder*
Move *var* from *curFolder* to *newFolder.* *newFolder* is created if it does not exist.

mRow(*expr*,*matrix*,*index***)**
Return *matrix* with row *index* of *matrix* multiplied by *expr*.

mRowAdd(*expr*,*matrix*,*index1*,*index2***)**
Return *matrix* with row *index2* replaced with *expr* * row *index1* + row *index2*

nCr(*expr1*,*expr2***)**
Return number of combinations of *expr1* items taken *expr2* at a time, for *expr1* ≥ *expr2* ≥ 0, and integer *expr1* and *expr2.* This is the binomial coefficient.

(*expr*,*n*)
If *n* is zero or a negative integer, return 0.

If *n* is a positive integer or non-integer, return

$$\frac{expr!}{n!(expr-n)!}$$

(*list1*,*list2*)
(*matrix1*,*matrix2*)
Return list of nCr() of corresponding element pairs of each argument. Both arguments must have the same dimension.

nDeriv()
(*expr*,*var*[,*h*])
Return an estimate of the numerical derivative of *expr* with respect to *var* using the central difference quotient formula. *h* is the step size and defaults to 0.001.

(*expr*,*var*,*list*)
Return list of derivative estimates for each step size in *list*.

(*list*,*var*[,*h*])
(*matrix*,*var*[,*h*])
Return list or matrix of derivative estimates for each element of *list* or *matrix*.

NewData *var*,*list1*[,*list2*] [,*list3*] ...
Create data variable *var* whose columns are the *lists* in order. The *list*s can be lists, expressions or list variable names. Make the new variable current in the Data/Matix editor.

var,*matrix*
Create data variable *var* based on *matrix*.

sysdata,*matrix*
Load *matrix* in the *sysdata* system data variable.

NewFold *folderName*
Create a folder with the name *folderName*, and make that folder current.

newList(*expr***)**
Return a list of zeros with dimension *expr*.

newMat(*exprRow*,*exprCol***)**
Return a matrix of zeros with *exprRow* rows and *exprCol* columns.

NewPic *matrix*,*var* [,*maxRow*] [,*maxCol*]
Create a pic variable *var* based on *matrix*. *matrix* has two columns, and each row is the coordinates of a pixel which is turned 'on' in the picture. *var* is replaced if it exists. The default size of *var* is the minimum boundary area required by the *matrix* coordinates. *maxRow* and *maxCol* specify optional maximum boundary limits for *var*.

NewPlot *n*,*type*,*xlist*[,[*ylist*],[*frqList*],[*catList*], [*incList*], [*mark*],[*bucketSize*]]
Create a new data plot definition for plot number *n*, where *n* is 1 to 9. *xlist* and *ylist* are the data to plot. *frqList* is the frequency list, *catList* is the category list and *incList* is the category include list. *type* specifies the plot type:
1 = scatter plot
2 = xyline plot
3 = box plot
4 = histogram
5 = modified box plot

mark specifies the plot symbol:
1 = box
2 = cross
3 = plus
4 = square
5 = dot

bucketSize must be > 0 and specifies the historgram bucket width and varies with *xmin* and *xmax*. *bucketSize* defaults to 1. *incList* need not be a variable name and cannot be c1 - c99. The other list arguments must be variable names or c1 - c99.

NewProb
Clear all unlocked and unarchived single-character variable names in the current folder. Turn off all function and stat plots in the current graph mode. Perform ClrDraw, ClrErr, ClrGraph, SlrHome, ClrIO and ClrTable.

nInt(*expr*,*var*,*lower*,*upper***)**
Return approximate integral of *expr* with respect to *var* with integration limits *lower* and *upper*. *var* must be the only variable in *expr*. *lower* and *upper* must be constants, +∞ or -∞. The adaptive algorithm uses weighted sample values of the integrand in the interval *lower*<*var*<*upper*. nInt() attempts an accuracy of six significant digits and displays the warning message "Questionable accuracy" if the accuracy may be less. Nested nInt() calls can be used for multiple integration, and the integration limits can depend on integration variables outside them.

norm(*matrix***)**
Return the Forbenius norm of *matrix*.

not
not *BooleanExp*
not *integer*
not *list*
not *matrix*
For Boolean arguments, return true, false or a simplified Boolean expression. For integer arguments, return the 1's complement of *integer*. Results are displayed according to the Base mode. Use the prefixes 0b or 0h for binary and hexadecimal integers. If the integer argument is larger than the maximum signed 32-bit binary number, a symmetric modulo operation brings the argument into range. Auto or Exact mode must be used with integer arguments.

nPr(*expr1*,*expr2***)**
For integer *expr1* and *expr2*, and *expr1* ≥ *expr2* ≥ 0, return the number of permutations of *expr1* items taken *expr2* at a time.

(*m*,*r*)
If *r* = 0, return 1. Otherwise return m!/(m-r)!

nSolve(*eqn*,*var***)**
Solve for eqn for one approximate solution of *var*. *var* may be a variable name in *eqn*, or a variable name with a solution guess in the form *var* = *guess.* Use the "|" operator to constrain the solution range. nSolve() will return the string "no solution found" if it cannot find a plausible solution.

OneVar *xList*[[,*freqList*][,*catList*][,*incList*]]
Calculate one-variable statistics and update system statistics variables. All lists must have equal dimension except *incList*. The first three arguments must be variable names of c1 - c99. *incList* need not be a variable name and cannot be c1 - c99.

or
exp1 or *exp2*
list1 or *list2*
matrix 1 or *matrix 2*
Return *true* or *false* or a simplifed form.

integer1 and *integer2*
Bit-by-bit 32-bit integer logical *or*. Arguments larger than 32-bit signed values are reduced with symmetric modulo operation. Mode must be Auto or Exact.

ord()
(*string*), (*list*)
Return numeric code of the first character in *string*. Return list of numeric codes of first characters of strings in *list*.

Output *row*,*column*,*exprOrString*
Display expression or string *exprOrString* at text coordinates *row*, *column*. *exprOrString* can include conversion operations such as ▸DD. *exprOrString* is pretty-printed if Pretty Print is on.

P▸Rx()
(*rExpr*,*θExpr*), (*rList*,*θList*), (*rMatrix*,*θMatrix*)
Return the x-coordinate of the (r,*θ*) pair*θ* is interpreted in degrees or radians according to the current angle mode. Use ° or ʳ to over-ride the current mode setting.

P▸Ry()
(*rExpr*,*θExpr*), (*rList*,*θList*), (*rMatrix*,*θMatrix*)
Return the y-coordinate of the (r,*θ*) pair*θ* is interpreted in degrees or radians according to the current angle mode. Use ° or ʳ to over-ride the current mode setting.

part(*expr*[,*n*]**)**
Extract subexpressions of expression *expr*. *n* is a non-negative integer.

(*expr*)
Simplify *expr* and return the number of top-level arguments or operands. Return 0 if *expr* is a number, variable or symbolic constant.

(*expr*,0)
Simplify *expr* and return a string which contains the top-level function name or operator. Return string(*expr*) if *expr* is a number, variable or symbolic constant.

(*expr*,*n*)
Simplify *expr* and return the *n*th argument or operand, where *n* is greater than zero and less than or equal to the number of operands returned by part(*expr*). Otherwise return an error.

PassErr
Pass an error to the next level. Typically used in an Else clause (in place of ClrErr), when error handling is not yet determined.

Pause [*expr*]
Suspend program execution until [ENTER] is pressed. If included, *expr* is displayed on the Program I/O screen. If *expr* does not fit on the screen, the cursor pad scrolls the display. *expr* can include conversion operation suffixes such as ▸DD.

PlotsOff [1] [,2] [,3] ... [,9]
Turn all plots off with no arguments, or turn the specified plots off. Only affects the current graph in 2-graph mode.

PlotsOn [1] [,2] [,3] ... [,9]
Turn all plots on with no arguments, or turn the specified plots on. Only affects the current graph in 2-graph mode.

▸Polar
vector ▸Polar
Display vector in polar form [r∠θ]. *vector* must be a row or column vector of dimension 2. Can only be used at the end of an entry line, and does not update *ans*.

complexValue ▸Polar
Display *complexValue* in polar form: (r∠θ) in Degree mode, or $re^{i\theta}$ in Radian mode. *complexValue* can have any complex form, but $re^{i\theta}$ causes an error in Degree mode.

polyEval()
(*list*,*expr*), (*list*,*list2*)
Interpret *list* as the coefficients of a polynomial in descending degree, and return the polynomial evaluated for *expr* or each element of *list2*.

PopUp *itemList*,*var*
Display a pop-up menu with the character strings in *itemList*, wait for the press of a number key, and store the number in *var*. If *var* exists and contains a valid item number, that item is the default choice. *itemList* must have at least one character string.

PowerReg
xlist,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate power regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is $y = ax^b$.

Prgm
Prgm : *block* : EndPrgm
Required instruction that identifies the first line of a TI-Basic program.

product(*list* [,*start*[,*end*]]**)**
Return the product of the elements of *list*, from *start* to *end*.

(*matrix* ,[*start*[,*end*]])
Return row vector whose elements are the product of the column elements of *matrix*. *start* and *end* specify a row range.

Prompt *var1*[,*var2*][,*var3*]...
Display one prompt for each argument on the Program I/O screen. The prompt is the variable name suffixed with "?". The expression entered for each prompt is stored in the variable.

propFrac(*rationalNumber***)**
Return *rationalNumber* as an integer and fraction with the same sign, where the fraction denominator is greater than the numerator.

(*rationalExpr*,*var*)
Return *rationalExpr* as a sum of proper ratios and polynomial in *var*. The degree of *var* in each ratio denominator is greater than the degree in the numerator. Similar powers of *var* are collected, and the terms and powers are sorted with respect to *var*.

(*rationalExpr*)
Return a proper fraction expansion with respect to the most main variable. The polynomial coefficients are then made proper with respect to their most main variable, and so on.

PtChg *x*,*y*
xList,*yList*
Display Graph screen and reverse pixel nearest to window coordinates (*x*,*y*)

PtOff *x*,*y*
xList,*yList*
Display Graph screen and turn off pixel nearest window coordinates (*x*,*y*).

PtOn *x*,*y*
xList,*yList*
Display Graph screen and turn on pixel nearest window coordinate (*x*,*y*).

PtTest()
(*x*,*y*), (*xList*,*yList*)
Return True if the pixel nearest window coordinates (*x*,*y*) is on, otherwise return False.

PtText *string*,*x*,*y*
Display Graph screen and place *string* with the upper-left corner of the first character at the pixel nearest window coordinates (*x*,*y*)

PxlChg *row*, *col*
rowList,*colList*
Display Graph screen and reverse the pixel at pixel coordinates (*row*,*col*) or (*rowList*,*colList*)

PxlCrcl *row*,*col*,*r* [,*drawMode*]
Display Graph screen and draw circle with radius *r* pixels and center at pixel coordinates (*row*,*col*).
If *drawMode* = 1, draw the circle (default).
If *drawMode* = 0, turn off the circle.
If *drawMode* = -1, invert pixels along the circle .

PxlHorz *row*[,*drawMode*]
Display Graph Screen and draw horizontal line at pixel coordinate *row*.
If *drawMode* = 1, draw the line (default).
If *drawMode* = 0, turn the line pixels off.
If *drawMode* = -1, invert the line pixels.

PxlLine
rowStart,*colStart*,*rowEnd*,*colEnd*[,*drawMode*]
Display Graph screen and draw line from pixel coordinates (*rowStart*,*colStart*) to (*rowEnd*, *colEnd*), including both endpoints.
If *drawMode* = 1, draw the line (default).
If *drawMode* = 0, turn the line pixels off.
If *drawMode* = -1, invert the line pixels.

PxlOff
row,*col*
rowList,*colList*
Display the Graph screen and turn off the pixel at pixel coordinates (*row*,*col*).

PxlOn
row,*col*
rowList,*colList*
Display Graph screen and turn on the pixel at pixel coordinates (*row*,*col*).

PxlTest()
(*row*,*col*), (*rowList*,*colList*)
Return True if the pixel at pixel coordinates (*row*,*col*) is on, otherwise return False.

PxlText *string*,*row*,*col*
Display Graph screen and place *string* with the upper-left corner of the first character at the pixel coordinates (*row*,*col*)

PxlVert *col*[,*drawMode*]
Display Graph Screen and draw vertical line at pixel coordinate *col*.
If *drawMode* = 1, draw the line (default).
If *drawMode* = 0, turn the line pixels off.
If *drawMode* = -1, invert the line pixels.

QR *matrix*,*qMatName*,*rMatName*[,*tol*]
Calculate Householder QR factorization of real or complex *matrix*. The Q and R result matrices are stored to the *matName* variables: Q is unitary, R is upper triangular. If the matrix has floating-point elements and no symbolic variables, then any matrix element less than *tol* is set to zero. The default value for *tol* is 5E-14 * max(dim(*matrix*)) * *rowNorm*(*matrix*). Computations are done with floating-point arithmetic if the mode is Approximate.

QuadReg
xlist,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate quadratic polynomial regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is $y = ax^2 + bx + c$.

QuartReg
xlist,*ylist*[,[*freqList*],[*catlLst*],[*catincList*]]
Calculate quartic polynomial regression and update all statistics variables. First four arguments must be variable names or c1-c99. Last argument need not be a variable name and cannot be c1-c99. All lists except *catincList* must have equal dimension. The regression equation is $y = ax^4 + bx^3 + cx^2 + dx + e$.

R▸Pθ()
(*xExpr*,*yExpr*), (*xList*,*yList*)
(*xMatrix*,*yMatrix*)
Return the θ-coordinate of the (*x*,*y*) arguments, as either a radian or degree angle, depending on the current Angle mode.

R▸Pr()
(*xExpr*,*yExpr*), (*xList*,*yList*), (*xMatrix*,*yMatrix*)
Return the r-coordinate of the (*x*,*y*) arguments.

rand([*n*]**)**
n is a non-zero integer. With no argument, return the next random number between 0 and 1 in the sequence. When *n*>0, return random integer in the interval [1,*n*]. When *n*<0, return random integer in the interval [-*n*,-1].

randMat(*numRows*,*numColumns***)**
Return matrix with dimensions (*numRows*, *numColumns*) whose elements are random integers between -9 and 9.

randNorm(*mean*,*sdev***)**
Return floating-point number from the standard distribution with *mean* and standard deviation *sdev*.

randPoly(*var*,*order***)**
Return a polynomial of *order* in *var* whose coefficients are random integers from -9 to 9, where the leading coefficient is not zero.

RandSeed *n*
Reseed the random number generator. If *n* = 0, reseed with the factory defaults. Otherwise, *n* is

used to generate the two system variables *seed1* and *seed2*.

RclGDB *GDBvar*
Restore all Graph database settings from the variable *GDBvar*. See *StoGDB* for the settings.

RclPic *picVar*[,*row*,*column*]
Display Graph screen and add the picture *picVar* at the upper-left corner coordinates of (*row*,*column*), by logically OR-ing *picVar* with the Graph screen. The default coordinates are (0,0).

real()
(*expr*), (*list*), (*matrix*)
Return the real part of *expr*, or the real parts of the elements of *list* or *matrix*. Undefined variables are treated as real. See also *imag()*.

▸Rect
vector ▸Rect
Display *vector* in rectangular form [x,y,z]. *vector* must be a row or column vector with dimension 2 or 3. Rect can only be used at the end of an entry line and does not update *ans*.

complexValue ▸Rect
Display *complexValue* in rectangular for a + b*i*. *complexValue* may have any form, but an $re^{i\theta}$ entry causes an error in Degree mode. For polar entries the form (r∠θ) must be used.

ref(*matrix*[,*tol*]**)**
Return the row echelon form of *matrix*. Treat any element as zero if it is less than *tol*, and the matrix contains floating-point elements and no symbolic elements. The default *tol* is 5E-14 * max(dim(*matrix*)) * rowNorm(*matrix*). Floating-point arithmetic is used in Approx mode.

remain()
(*expr1*,*expr2*), (*list1*,*list2*), (*matrix1*,*matrix2*)
Return the remainder of the first argument with respect to the second argument as
remain(x,0) = x
remain(x,y) = x-y*iPart(x/y)
Note that remain(-x,y) = -remain(x,y).

Rename *oldVarName*,*newVarName*
Rename variable *oldVarName* as *newVarName*.

Request *promptString*,*var*
If used inside a Dialog...EndDlog structure, create a user input box in the dialog box, otherwise create a dialog box with the input box. *promptString* must be less than 21 characters. If *var* contains a string, it is displayed as the default.

Return [*expression*]
Exit a program or function if no argument used, otherwise return *expression* from a function.

right()
(*list*[,*num*])
Return the rightmost *num* elements of *list*. Return *list* if *num* not used.

(*string*,[*num*])
Return the rightmost *num* characters of *string.* Return *string* if *num* not used.

(*comparison*)
Return the right side of equation or inequality.

rotate()
(*integer*,[*n*])
Rotate signed 32-bit *integer n* bits. Rotate to the left if *n* is positive, to the right if *n* is negative. Default *n* is -1.

(*list*,[*n*])
Return *list* with elements rotated by *n* elements. Rotate to the left for positive *n*, to the right for negative *n*. Default *n* is -1.

(*string*,[*n*])
Return *string* with characters rotated by *n* characters. Rotate to the left for for positive *n*, to the left for negative *n*. Default *n* is -1.

round()
(*expr*[,*d*]), (*list*[,*d*]), (*matrix*[,*d*])
Return argument elements rounded to *d* digits after the decimal point. *d* must be an integer in the range 0-12. Default *d* is 12.

rowAdd(*matrix*,*r1*,*r2***)**
Return *matrix* with row *r1* replaced by the sum of rows *r1* and *r2*.

rowDim(*matrix***)**
Return number of rows of *matrix*.

rowNorm(*matrix***)**
Return the maximum of the sums of the row elements of *matrix*. All elements must simplify to numbers.

rowSwap(*matrix*,*r1*,*r2***)**
Return *matrix* with rows *r1* and *r2* swapped.

RplcPic *picVar*[,*row*][,*column*]
Place picture *picVar* in the Graph screen with its upper left corner at pixel coordinates *row, column*. The area of the Graph screen affected by *picVar* is cleared. *row*, *column* default to 0,0.

rref(*matrix*[,*tol*]**)**
Return the reduced row echelon format of matrix. Treat any element as zero if it is less than *tol*, and the matrix contains floating-point elements and no symbolic elements. The default *tol* is 5E-14 * max(dim(*matrix*)) * rowNorm(*matrix*). Floating-point arithmetic is used in Approx mode.

Send [*list*]
Send *list* to the link port; used with CBL and CBR.

SendCalc *var*
Send *var* to link port to be received by another calculator. Receiving calculator must be on Home screen or execute GetCalc from a program. Use SendChat, instead, to send from a TI-89/92+ to a TI-92.

SendChat *var*
Alternative to SendCalc which works with either a TI-92 or TI-92+. Will only send variables which are compatible with the TI-92. Will not send archived variables or graph data base, etc.

seq(*expr*,*var*,*low*,*high*[,*step*])**
Return list whose elements are *expr* evaluated at *var,* for each *var* from *low* to *high* incremented by *step*. The default for *step* is 1.

setFold(*folderName***)**
Return the name of the current folder as a string, and set the current folder to *folderName*. The folder *folderName* must exist.

setGraph(*modeNameString,settingString***)**
Set the Graph mode *modeName* to *setting* and return the previous setting as a string. *modeNameString* and *settingString* may be descriptive strings, or numeric code strings. Numeric codes are preferred because the descriptive strings depend on the language localization setting. This table shows both the descriptive string (in English) followed by the equivalent numeric code.

modeNameString	*settingString*
"Coordinates" "1"	"Rect", "1" "Polar", "2" "Off", "3"
"Graph Order" "2"	"Seq", "1" "Simul" "2" [a]
"Grid" "3"	"Off" "1" "On" "2" [b]
"Axes" "4"	Not 3D mode: "Off" "1" "On" "2" 3D mode: "Off" "1" "Axes" "2" "Box" "3"
"Leading cursor" "5"	"Off" "1" "On" "2"
"Labels" "6"	"Off" "1" "On" "2"
"Seq Axes" "7"	"Time" "1" "Web" "2" "U1-vsU2" "3" [d]
"Solution method" "8"	"RK" "1" "Euler" "2" [e]
"Fields" "9"	"SlpFld" "1" "DirFld" "2" "FldOff" "3" [e]
"DE Axes" "10"	"Time" "1" "t-vs-y'" "2" "y-vs-t" "3" "y1-vs-y2" "4" "y1-vs-y2'" "5" "y1'-vs'y2'" "6" [e]
"Style" "11"	"Wire Frame" "1" "Hidden Surface" "2" "Contour Levels" "3" "Wire and Contour" "4" "Implicit Plot" "5" [c]

[a] Not available in Sequence, 3D or Diff Equations graph mode
[b] Not available in 3D graph mode
[c] Applies only to 3D graph mode
[d] Applies only to Sequence graph mode
[e] Applies only to Diff Equations graph mode

setMode()
(*modeString*,*settingString*), (*list*)
Set mode *modeString* to *settingString*, and return the current setting. *list* contains pairs of *modeString* and *settingString* pairs. *modeString* and *settingString* may be descriptive strings, or numeric code strings. Numeric codes are preferred because the descriptive strings depend on the language localization setting. Use the *list* argument to set several modes at once. This table shows both the descriptive string (in English) followed by the equivalent numeric code.

modeString	*settingString*
"ALL" "0"	Only applies to getMode(), not setMode()
"Graph" "1"	"Function" "1" "Parametric" "2" "Polar" "3" "Sequence" "4" "3D" "5" "Diff Equations" "6"
"Display digits" "2"	"Fix 0", ..., "Fix 12" "Fix n" is "n+1" "Float" "14" "Float 1", ..., "Float 12" "Float n" is "n+14"
"Angle" "3"	"Radian" "1" "Degree" "2"
"Exponential Format" "4"	"Normal" "1" "Scientific" "2" "Engineering" "3"
"Complex Format" "5"	"Real" "1" "Rectangular" "2" "Polar" "3"
"Vector Format" "6"	"Rectangular" "1" "Cylindrical" "2" "Spherical" "3"
"Pretty Print" "7"	"Off" "1" "On" "2"
"Split Screen" "8"	"Full" "1" "Top-Bottom" "2" "Left-Right" "3"
"Split 1 App" "9"	(no number codes) "Home" "Y= Editor" "Window Editor" "Graph" "Table" "Data/Matrix Editor" "Program Editor" "Text Editor" "Numeric Solver" "*Flash App"*
"Split 2 App"	Same as "Split 1 App"
"10"	
"Number of Graphs" "11"	"1" "1" "2" "2"
"Graph2" "12"	Same as "Graph"
"Split Screen Ratio" "13"	"1:1" "1" "1:2" "2" "2:1" "3" (TI-92+ only)
"Exact/Approx" "14"	"Auto" "1" "Exact" "2" "Approximate" "3"
"Base" "15"	"Dec" "1" "Hex" "2" "Bin" "3"
"Language" (no number code)	"English", "*Alternate Language"*

setTable(*modeNameString*,*settingString***)**
Set table parameter *modeNameString* to *settingString*, and return the previous setting. *modeNameString* and *settingString* may be descriptive strings, or numeric code strings. Numeric codes are preferred because the descriptive strings depend on the language localization setting. This table shows both the descriptive string (in English) followed by the equivalent numeric code.

modeNameString	*settingString*
"Graph<->Table" "1"	"Off" "1" "On" "2"
"Independent" "2"	"Auto" "1" "Ask" "2"

setUnits(*list***)**
Set default units to values specified by *list* and return the previous defaults. Specify the built-in SI (metric) units with {"SI"}, or the English/US units with {"ENG/US"}.

Specify a custom set of default units with {"CUSTOM","*cat1*","*unit1*"[,"*cat2*","*unit2*",...]} where "*cat*" specifcies a unit category and "*unit*" specifies the default unit. Any unspecified category uses the previous custom unit.

Specify the previous custom default units with {"CUSTOM"}

Shade
expr1,*expr2*[,*xlow*][,*xhigh*][,*pattern*][,*patRes*]
Display Graph screen, graphs *expr1* and *expr2*, and shade areas where *expr1* < *expr2. expr1*

and *expr2* use *x* as the independent variable. *xlow* and *xhigh* specify left and right shading boundaries, *xlow* and *xhigh* are bounded by, and default to *xmin* and *xmax*.

pattern sets the shading pattern:
1 = vertical (default)
2 = horizontal
3 = negative-slope 45°
4 = positive-slope 45°

patRes specifies the shading pixel spacing resolution:
1 = solid
2 = 1 pixel spacing (default)
3 = 2 pixel spacing
...
10 = 9 pixel spacing

shift()
(*integer*[,*n*])
Shift bits in binary *integer*, *n* times. Shift to left if *n* is positive; to right if *n* is negative. Default *n* is -1. In a right shift, the rightmost bit is dropped, and 0 or 1 is inserted at the left to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

(*list*[,*n*])
Return *list* shifted left or right by *n* elements. Shift to left if *n* is positive, to right if *n* is negative. Default *n* is -1. Elements introduced by the shift are set to the symbol *undef*.

(*string*[,*n*])
Return *string* shifted left or right by *n* characters. Shift to left if *n* is positive, to right if *n* is negative. Default *n* is -1. Elements introduced by the shift are set to a space.

ShowStat
Show dialog box with the last computed statistics results, if the results are still valid. Results are cleared (not valid) if the data to compute them has changed.

sign()
(*expr*), (*list*), (*matrix*)
For real and complex arguments, return *expr*/abs(*expr*) when *expr* is not equal to zero. Return 1 if *expr* is positive, or -1 if *expr* is negative. sign(0) returns ±1 in the REAL complex mode, otherwise it returns itself.

simult(*coefMatrix*,*constVector*[,*tol*]**)**
Return column vector with solutions to the system of linear equations where *coefMatrix* is a square matrix of the equation coefficients and *constVector* is a column vector with the same number of rows as *coefMatrix*. If the matrix has floating point elements and no symbolic variables, then any element is treated as zero if its value is less than *tol*. The default for *tol* is 5E-14 * max(dim(*coefMatrix*)) * rowNorm(*coefMatrix*). Computations are done with floating-point arithmetic in Approx mode.

(*coefMatrix*,*constMatrix*[,*tol*])
Solve multiple systems of linear equations, where each system has the same equation coefficients in *coefMatrix*. Each column of *constMatrix* contains the constants for a different system of equations. Each column of the returned matrix is the solution for the corresponding *constMatrix* column.

sin()
(*expr*), (*list*)
Return the sine of the argument. The argument is interpreted in degrees or radians according to the current mode setting. Use ° or ʳ to override the mode setting.

(*matrix*)
Return the matrix sine of square *matrix*, which is not the sine of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

sin^{-1}()
(*expr*), (*list*)
Return the angle whose sine is the argument. The result is returned as degrees or radians depending on the current Angle mode setting.

(*matrix*)
Return the matrix inverse sine of square *matrix*, which is not the same as the inverse sine of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

sinh()
(*expr*), (*list*)
Return the hyperbolic sine of the argument. The argument is interpreted in degrees or radians according to the current mode setting. Use ° or ʳ to override the mode setting.

(*matrix*)
Return the matrix hyperbolic sine of square *matrix*, which is not the hyperbolic sine of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

sinh^{-1}()
(*expr*), (*list*)
Return the angle whose hyperbolic sine is the argument. The result is returned as degrees or radians depending on the current Angle mode setting.

(*matrix*)
Return the matrix inverse hyperbolic sine of square *matrix*, which is not the same as the inverse hyperbolic sine of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

SinReg
xlist,*ylist*[,[*iterations*][,*period*][,*catList*,*incList*]]
Calculate the sinusoidal regression and update the system statistics variables. *xlist* and *ylist* are the x- and y-data points. *iterations* is the the maximum number of solution interations; the range is 1 to 16 and the default is 8. Larger values may result in better accuracy but longer execution time. *period* specifies the estimated period. If not used, the elements of *xlist* should be in sequential order and equally spaced. *xlist*, *ylist* and *catList* must be variable names or c1 - c99. *incList* need not be a variable name and cannot be c1 - c99. All lists must have equal dimensions except *incList*.

solve()
(*eqn*,*var*), (*inequality*,*var*)
Return candidate real solutions (as Boolean expressions) for *var* of equation *eqn* or *inequality*. Attempts to return all solutions, but for some arguments there are infinite solutions. In the Auto mode setting, concise exact solutions are attempted, supplemented by approximate solutions. Solutions may exist only in the limit from one or both sides due to default cancellation of the greatest common divisor from ratio numerators and denominators. *false* is returned if no real solutions can be found. *true* is returned if solve() determines that any finite real value is a solution. Solutions may include unique arbitrary integers of the form @n*j*, where *j* is an integer from 1 to 255. Use the "|" operator to constrain the solution interval or other variables.

In Real mode, fractional powers with odd denominators denote only the real branch. Otherwise, multiple branched expressions (fractional powers, logarithms, inverse trigonometric functions) denote the principle branch. solve() produces solutions only for that one real or principle branch.

Explicit solutions to inequalities are unlikely unless the inequality is linear and only includes *var*. In the Exact mode setting, portions which cannot be solved are returned as implicit equations or inequalities.

(*eqn1* and *eqn2* [and ...],{*var1*,*var2*[,...]})
Return candidate real solutions (as Boolean expressions) to the simultaneous equations. *var* arguments may be variable names, or variable names with a solution guess in the form *var* = *guess*. If all equations are polynomials and you supply no guesses, solve() uses the lexical Gröbner/Buchberger elimination to attempt to find all solutions. Simultaneous polynomial equations can have extra variables with no values. Solution variables of no interest may be omitted. Solutions may include arbitrary constants of the form @*k*, where *k* is an integer from 1 to 255. Computation time or memory exhaustion may depend on the order of the *var*s in the equations or variables list.

solve() attempts to find all real solutions with Gaussian elimination if you include no guesses, any equation is in non-polynomial in any variable, but all equations are linear in the solution variables.

solve() attempts to find one real solution (with an interative approximate method) if the system is neither polynomial in all its variables nor linear in its solution variables. The number of solution variables must equal the number of equations and all other variables must simplify to numbers. Each solution variable starts at its guess value, or 0.0 if a guess is not used. Guesses may need to be close to the solution for convergence.

SortA
listName1[,*listName2*][,*listName3*],...

vectorName1[,*vectorName2*][,*vectorName3*],...
Sort the elements of the first argument in ascending order. Sort elements of optional arguments so that the element positions match the new positions of the elements in the first argument. All arguments must be names of lists or vectors, and have equal dimensions.

SortD
listName1[,*listName2*][,*listName3*],...
vectorName1[,*vectorName2*][,*vectorName3*],...
Same as SortA, but sort the elements in descending order.

▸Sphere
vector ▸Sphere
Display 3-element row or column *vector* in spherical form [ρ∠θ∠ϕ]. θ is the angle from the x-axis to the projection of the vector in the xy plane, and ϕ is the angle from the z-axis to the vector.

stdDev(*list*[,*freqList*]**)**
Return the sample (not population) standard deviation of the elements of *list*. *freqList* elements specify the frequency of the corresponding elements of *list*. Both lists must have at least two elements, and the same number of elements.

(*matrix*[,*freqMatrix*])
Return row vector of the sample standard deviations of the column elements of *matrix*. *freqMatrix* elements specify the frequency of corresponding *matrix* elements. *matrix* must have at least two rows.

StoGDB *GDBvar*
Create a Graph database variable (GDB) with the current settings for:
Graphing mode; Y= functions; Window variables; Graph format settings (1- or 2-Graph setting; split-screen and ratio settings if 2-Graph mode); Angle mode; Real/Complex mode; Initial conditions (Sequence or Diff Equations mode); Table flags; tblStart, Δtbl, tblInput
Use RclGDB *GDBvar* to restore the graph environment.

Stop
Stop program execution.

StoPic *picVar*[,*pxlRow*,*pxlCol*][,*width*,*height*]
Display the Graph screen and copy a rectangular area of the display to *picVar*. *pxlRow* and *pxlCol* specify the upper left corner of the copied area, the default is 0,0. *width* and *height* specify the pixel dimensions of the area and default to the width and height of the current graph screen.

Store
See →

String(*expr***)**
Simplify *expr* and return the result as a string.

Style *eqNum*,*styleString*
Set graph function *eqNum* to use the graphing property *styleString*. *eqNum* must be 1 - 99 and the function must exist. *styleString* must be one of: "Line", "Dot", "Square", "Thick", "Animate", "Path", "Above" or "Below". Only some styles are valid for particular graph modes:

Function: all styles
Parametric/Polar: line, dot, square, thick, animate, path
Sequence: line, dot, square, thick
3D: none
Diff Equations: line, dot, square, thick, animate, path.

subMat()
(*matrix*[,*startRow*][,*startCol*][,*endRow*][,*endCol*])
Return specified submatix of *matrix*. Defaults are *startRow* = 1, *startCol* = 1, *endRow* = last row, *endCol* = last column

sum(*list*[,*start*[,*end*]]**)**
Return sum of *list* elements from *start* to *end*.

(*matrix*[,*start*[,*end*]])
Return row vector whose elements are the sums of the columns of *matrix*. *start* and *end* specify start and end rows.

switch([*n*]**)**
Return the number of the active window, and switch the active window based on *n*. Window 1 is top or left, window 2 is right or bottom. With no argument, switch the current window and return the previous active window number. *n* is ignored if a split screen is not displayed. Otherwise, for
n = 0: return the current window

n = 1: activate window 1, return previous active window number.
n = 2: activate window 2, return previous active window number.

ᵀ (transpose)
*matrix*ᵀ
Return complex conjugate transpose of *matrix*.

Table *expr1*[,*expr2*][,*var*]
Build a table of the argument expression(s) or functions. The current Y= Editor functions are temporarily ignored. Expressions entered with Table or Graph are assigned increasing function numbers beginning with 1. To clear these functions execute ClrGraph or display the Y= Editor. If *var* is omitted, the current graph mode independent variable is used. Table cannot be used with 3D, sequence or differential equations graphing. Valid variations are:
Function graph: Table *expr*,*x*
Parametric graph: Table *xExpr*,*yExpr*,*t*
Polar graph: Table *expr*, θ
See also BldData.

tan()
(*expr*), (*list*)
Return the tangent of the argument. The argument is interpreted in degrees or radians according to the current mode setting. Use ° or ʳ to override the mode setting.

(*matrix*)
Return the matrix tangent of square *matrix*, which is not the tangent of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

tan^{-1}()
(*expr*), (*list*)
Return the angle whose tangent is the argument. The result is returned as degrees or radians depending on the current Angle mode setting.

(*matrix*)
Return the matrix inverse tangent of square *matrix*, which is not the same as the inverse tangent of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

tanh()
(*expr*), (*list*)
Return the hyperbolic tangent of the argument. The argument is interpreted in degrees or radians according to the current mode setting. Use ° or ʳ to override the mode setting.

(*matrix*)
Return the matrix hyperbolic tangent of square *matrix*, which is not the hyperbolic tangent of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers

tanh^{-1}()
(*expr*), (*list*)
Return the angle whose hyperbolic tangent is the argument. The result is returned as degrees or radians depending on the current Angle mode setting.

(*matrix*)
Return the matrix inverse hyperbolic tangent of square *matrix*, which is not the same as the inverse hyperbolic tangent of each element. Refer to cos() for details. *matrix* must be diagonalizable and the result always constains floating-point numbers.

taylor(*expr*,*var*,*order*[,*point*]**)**
Return the Taylor polynomial for *expr* in *var*. *point* is the expansion point and defaults to zero. The polynomial includes non-zero terms of integer degree from 0 through *order* in (*var* - *point*). taylor() returns itself if there is not truncated power series of this *order*, or negative or fractional exponents are required. Use substitution or temporary multiplication by a power of (*var* - *point*) for a more general series.

tCollect(*expr***)**
Return expression in which products and integer powers of sines and cosines are converted to a linear combination of sines and cosines of multiple arguments, angle sums and angle differences. Trigonometric polynomials are converted to linear combinations of harmonics. tCollect() tends to reverse tExpand(). Applying tExpand() to a tCollect() result, or vice versa, in two steps may simplify an expression.

tExpand(*expr***)**
Return expression in which sines and cosines of integer-multiple angles, angle sums and angle

differences are expanded. Best used in Radian mode, because degree-mode scaling interferes with expansion. tExpand() tends to reverse tCollect(). Applying tCollect() to a tExpand() result, or vice versa, in two steps may simplify an expression.

Text *string*
Display *string* in a dialog box. If used outside of a Dialog...EndDlog block, a dialog box is created, otherwise *string* is shown in the defined dialog box.

Then
See If.

Title *string*[,*label*]
Create the title of a menu or dialog box when used in a ToolBar or Custom structure, or a Dialog ... EndDlog block. *label* is only valid in the ToolBar structure. When used, it allows the menu choice to branch to *label*.

tmpCnv(*expr_°unit1,_°unit2***)**
Return temperature value *expr* converted from *unit1* to *unit2*. Valid units are _°C, _°F, _°K and _°R. To convert a temperature range instead of a value, use ΔtmpCnv().

ΔtmpCnv(*expr_°unit1,_°unit2***)**
Return temperature range *expr* converted from *unit1* to *unit2*. Valid units are _°C, _°F, _°K and _°R. To convert a temperature value instead of a range, use tmpCnv().

Toolbar
ToolBar : *block* : EndTBar
Create a toolbar menu. *block* statements can be either Title or Item. Item statements must have labels. Title statements must have labels if there are no Item statements.

Trace
Draw a SmartGraph and place the trace cursor on the first Y= function, at the previous cursor position, or at the reset position if regraphing was necessary. Allows operation of the cursor keys and most coordinate editing keys. Press [ENTER] to resume operation.

Try
Try : *block1* : Else : *block2* : EndTry
Execute *block1* until an error occurs, then transfer execution to *block2*. *errornum* contains the error number. See also ClrErr and PassErr.

TwoVar *xList,yList*[[,*freqList*][,*catList*][,*incList*]]
Calculate two-variable statistics and update system statistics variables. All lists must have equal dimension except *incList*. The first four arguments must be variable names of c1 - c99. *incList* need not be a variable name and cannot be c1 - c99.

Unarchiv *var1*[,*var2*][,*var3*]...
Move argument variables from user data archive memory (flash) to RAM. Archived variables can be accessed, but not deleted, renamed or stored to, since the variable is locked. Use Archive to archive variables.

unitV(*vector***)**
Return row- or column-unit vector, depending on the form of *vector*, which must be a single-row or single-column matrix.

Unlock *var1*[,*var2*][,*var3*]...
Unlock the argument variables. Use Lock to lock variables.

variance(*list*[,*freqList*]**)**
Return the population variance of the elements of *list*. *freqList* elements specify the frequency of the corresponding elements of *list*. Both lists must have at least two elements, and the same number of elements.

(*matrix*[,*freqMatrix*])
Return row vector of the population variance of the column elements of *matrix*. *freqMatrix* elements specify the frequency of corresponding *matrix* elements. *matrix* must have at least two rows.

when()
(*condition*,*trueRes*[,*falseRes*][,*unknownRes*])
Return *trueRes*, *falseRes*, or *unknownRes*, when *condition* is true, false or unknown. Returns the arguments when there too few arguments to determine the result. Omit both *falseRes* and *unknownRes* to define an expression only when *condition* is true. *undef* can be used as a result.

While
While *condition* : *block* : EndWhile

Execute *block* as long as *condition* is true.

"With" See |

xor
exp1 xor *exp2*
list1 xor *list2*
matrix 1 xor *matrix 2*
Return *true*, *false* or a simplifed form.

integer1 and *integer2*
Bit-by-bit 32-bit integer logical exclusive-or. Arguments larger than 32-bit signed values are reduced with symmetric modulo operation. Mode must be Auto or Exact.

XorPic *picVar*[,*row*][,*column*]
Display Graph screen and logically exclusive-or Graph screen pixels with those of *picVar* picture variable. Only pixels exclusive to the screen or *picVar* are turned on. *row* and *column* specify the pixel coordinates for the upper left corner of *picVar*; defaults are 0,0.

zeros()
(*expr*,*var*)
Return list of candidate values for *var* which make *expr* = 0. Equivalent to
exp▸list(solve(*expr*=0,*var*))
zeros() cannot express implicit solutions, solutions which require inequalities or solutions which do not involve *var*.

({*expr1*,*expr2*[,*expr3*...]},{*var1*,*var2*[,*var3*...]})
Return matrix of candidate solutions of simultaneous algebraic expressions *expr* in *var*. *var* may be a variable or a solution guess in the form *var* = *guess*. Each matrix row represents an alternate solution, with the variables ordered as in the {*var*} list.

If all equations are polynomials and you supply no guesses, zeros() uses the lexical Gröbner/Buchberger elimination to attempt to find all solutions. Simultaneous polynomial equations can have extra variables with no values. Solution variables of no interest may be omitted. Solutions may include arbitrary constants of the form @*k*, where *k* is an integer from 1 to 255. Computation time or memory exhaustion may depend on the order of the *var*s in the equations or variables list.

zeros() attempts to find all real solutions with Gaussian elimination if you include no guesses, any equation is in non-polynomial in any variable, but all equations are linear in the solution variables.

zeros() attempts to find one real solution (with an interative approximate method) if the system is neither polynomial in all its variables nor linear in its solution variables. The number of solution variables must equal the number of equations and all other variables must simplify to numbers. Each solution variable starts at its guess value, or 0.0 if a guess is not used. Guesses may need to be close to the solution for convergence.

ZoomBox
Display the Graph screen, pauses so that a box can be drawn to define the new view window, then updates the window.

ZoomData
Adjust the window settings based on the current data plots and function graphs so that all data points are sampled, then displays the Graph Screen. Does not adjust *ymin* and *ymax* for histograms.

ZoomDec
Set Δx and Δy = 0.1 and displays the Graph screen with the origin centered.

ZoomFit
Display the Graph screen and set the dependent variable Window dimensions so that all the picture is displayed for the current independent variable settings.

ZoomIn
Display the Graph screen, pauses so that a center point can be set to zoom in, then updates the viewing window. The zoom magnitude depends on the Zoom factors *xFact* and *yFact* in 2D graph modes, and on *xFact*, *yFact* and *zFact* in 3D mode.

ZoomInt
Display the Graph screen, pauses so that a zoom center point can be set, adjusts the Window settings such that each pixel is an integer in all directions, then updates the viewing window.

ZoomOut
Display the Graph screen, pauses so that a zoom center point can be set, then updates the viewing window. The zoom magnitude depends on the Zoom factors *xFact* and *yFact* in 2D graph modes, and on *xFact*, *yFact* and *zFact* in 3D mode.

ZoomPrev
Display the Graph screen and update the viewing window with the settings before the previous zoom.

ZoomRcl
Display the Graph screen and update the viewing window with the settings stored with ZoomSto.

ZoomSqr
Display the Graph screen and adjust the x or y window settings so that each pixel represents an equal width and height in the coordinate system, then update the viewing window. In 3D mode, lengthen the shortest two axes to equal the longest axis.

ZoomStd
Set the Window variables to the following standard values, then update the viewing window.
Function graphing:
x:[-10,10,1], y:[-10,10,1], *xres* = 2
Parametric graphing:
t:[0,2π,π/24], x:[-10,10,1], y:[-10,10,1]
Polar graphing:
θ:[0,2π,π/24], x:[-10,10,1], y:[-10,10,1]
Sequence graphing:
x:[-10,10,1], y:[-10,10,1]
nmin=1, *nmax*=10, *plotStrt*=1, *plotStep*=1
3D graphing:
x:[-10,10,14], y:[-10,10,14], z[-10,10]
eyeθ°=20, eyeϕ°=70, eyeψ°=0, *ncontour*=5
Differential equations graphing:
t:[0,10,.1,0], x:[-1,10,1], y:[-10,10,1]
ncurves=0, *Estep*=1, *diftol*=.001, *fldres*=14, *dtime*=0

ZoomSto
Store the current Window settings in Zoom memory. Use ZoomRcl to restore the settings.

ZoomTrig
Display the Graph screen, set x = π/24, xscl = π/2, center the origin, set the y settings to [-4,4,.5] and update the viewing window.

+ (add)
expr1 + *expr2*
Return the sum of *expr1* and *expr2*

list1 + *list2*
matrix1 + *matrix2*
Return list or matrix whose elements are the sum of the corresponding elements. Argument dimensions must be equal.

expr + *list*
list + *expr*
Return list whos elements are the sum of the list elements and *expr*.

expr + *matrix*
matrix + *expr*
Return matrix with *expr* added to each diagonal element of square *matrix*. Use .+ (dot plus) to add an expression to each element.

- (subtract)
expr1 - *expr2*
Return *expr1* - *expr2*

list1 - *list2*
matrix1 - *matrix2*
Return list or matrix whose elements are the elements of *list2* (or *matrix2*) subtracted from the corresponding elements of *list1* (or *matrix1*). Argument dimensions must be equal.

expr - *list*
list - *expr*
Return list where each element is each *list* element subtracted from *expr, or expr* subtracted from each list element.

expr - *matrix*
Return matrix: (*expr* * *identityMatrix*) - *matrix*
matrix must be square.

matrix - *expr*
Return matrix: *matrix* - (*expr* * *identityMatrix*)
matrix must be square.

(Note: use .- (dot minus) to subtract an expression from each element.)

*** (multiply)**
expr1 * *expr2*
Return the product of *expr1* and *expr2*

list1 * *list2*
Return list whose elements are the products of the corresponding elements of *list1* and *list2*. List dimensions must be equal.

matrix1 * *matrix2*
Return the matrix product of *matrix1* and *matrix2*. The number of rows of *matrix1* must equal the number of columns of *matrix2*.

expr * *list*
list * *expr*
Return list where each element is each *list* element multiplied by *expr*.

expr * *matrix*
matrix * *expr*
Return matrix whose elements are the product of *expr* and each element of *matrix*. This is the same as .* (dot multiply).

/ (divide)
expr1 / *expr2*
Return the quotient of *expr1* divided by *expr2*

list1 / *list2*
Return list whose elements are the quotients of the corresponding elements of *list1* and *list2*. List dimensions must be equal.

expr / *list*
list / *expr*
Return list where each element is the quotient of *expr* divided by each *list* element, or each *list* element divided by *expr*.

matrix / *expr*
Return matrix whose elements are the quotient of each element of *matrix* divided by *expr*. Use ./ (dot divide) to divide an expression by each matrix element.

- (negate)
-*expr*
-*list*
-*matrix*
Return the negation of the argument. If the argument is a binary or hexadecimal integer, return the two's-complement.

% (percent)
expr%
list%
matrix%
Return (argument/100)

= (equal)
expr1 = *expr2*
list1 = *list2*
matrix1 = *matrix2*
Return True if first argument can be determined to be equal to second argument. Return False if first argument cannot be determined to be equal to second argument. Otherwise, return a simplified form of the equations. For list and matrix arguments, return element-by-element comparisons.

≠ (not equal)
expr1 ≠ *expr2*
list1 ≠ *list2*
matrix1 ≠ *matrix2*
Return True if first argument can be determined to be not equal to second argument. Return False if first argument can be determined to be equal to second argument. Otherwise, return a simplified form of the inequality. For list and matrix arguments, return element-by-element comparisons.

< (less than)
expr1 < *expr2*
list1 < *list2*
matrix1 < *matrix2*
Return True if first argument can be determined to be less than second argument. Return False if first argument can be determined to be greater than or equal to the second argument. Otherwise, return a simplified form of the inequality. For list and matrix arguments, return element-by-element comparisons.

≤ (less than or equal to)
expr1 ≤ *expr2*
list1 ≤ *list2*
matrix1 ≤ *matrix2*
Return True if first argument can be determined to be less or equal to the second argument. Return False if first argument can be determined to be greater than the second argument. Otherwise, return a simplified form of the comparison. For list and matrix arguments, return element-by-element comparisons.

> (greater than)
expr1 > *expr2*
list1 > *list2*
matrix1 > *matrix2*
Return True if first argument can be determined to be greater than the second argument. Return False if first argument can be determined to be less than or equal to the second argument. Otherwise, return a simplified form of the comparison. For list and matrix arguments, return element-by-element comparisons.

≥ (greater than or equal to)
expr1 ≥ *expr2*
list1 ≥ *list2*
matrix1 ≥ *matrix2*
Return True if first argument can be determined to be greater than or equal to the second argument. Return False if first argument can be determined to be less than the second argument. Otherwise, return a simplified form of the comparison. For list and matrix arguments, return element-by-element comparisons.

.+ (dot add)
matrix1 .+ *matrix2*
Return matrix whose elements are the sums of the corresponding elements of *matrix1* and *matrix2*.

expr .+ *matrix*
matrix .+ *expr*
Return matrix whose elements are the sums of *expr* and *matrix.*

.- (dot subtract)
matrix1 .- *matrix2*
Return matrix whose elements are the corresponding elements of *matrix2* subtracted from *matrix1*.

expr .- *matrix*
Return matrix whose elements are the elements of *matrix* subtracted from *expr*.

matrix .- *expr*
Return matrix whose elements are *expr* subtracted from the elements of *matrix.*

.* (dot multiply)
matrix1 .* *matrix2*
Return matrix whose elements are the products of each corresponding elements of *matrix1* and *matrix2*.

expr .* *matrix*
matrix .* *expr*
Return matrix whose elements are the product of *expr* and the elements of *matrix*.

./ (dot divide)
matrix1 .* *matrix2*
Return matrix whose elements are the quotients of the corresponding elements of *matrix1* divided by *matrix2*.

expr .* *matrix*
matrix .* *expr*
Return matrix whose elements are the quotients of *expr* divided by the elements of *matrix*, or the elements of *matrix* divided by *expr*.

.^ (dot power)
matrix1 .^ *matrix2*
Return matrix whose elements are the elements of *matrix1* raised to the power of the corresponding *matrix2* elements.

expr .^ *matrix*
matrix .^ *expr*
Return matrix whose elements are *expr* raised to the power of each *matrix* element, or the elements of *matrix* raised to the *expr* power.

! (factorial)
expr!
list!
matrix!
Return the factorial of the argument. Only returns a numeric value for non-negative integer arguments.

& (append)
string1 & *string2*
Return string which is *string2* appended to *string1*.

∫() (integrate)
(*expr*,*var*[,*lower*][,*upper*])
(*list*,*var*[,*lower*][,*upper*])
(*matrix*,*var*[,*lower*][,*upper*])
Return the integral of the first argument with respect to *var*, from *lower* to *upper* integration limits. Return anti-derivative if *lower* and *upper* are omitted; constant of integration is omitted, but *lower* will be added as a constant of integration if *upper* is not used. Anti-derivatives may differ by a numeric constant. Piece-wise

constants may be added so that the anti-derivative is valid over a larger interval.

∫() can be nested for multiple integrals, and the integration limits can depend on integration variables outside the limits.

When both *lower* and *upper* are present, ∫() attempts to subdivide the integral at discontinuities. Numerical integration is used in Auto mode when a symbolic anti-derivative cannot be determined. Numerical integration is tried first in Approx mode, but anti-derivatives are sought if numerical differentiation is inapplicable or fails.

∫() returns itself for pieces of *expr* it cannot determine.

√() (square root)
√(*expr*), √(*list*)
Return the square root of the argument.

Π(*expr*,*var*,*low*,*high***) (product)**
Evaluate *expr* for each value of *var* from *low* to *high*; return the product of the results. Return 1 if *high* = *low* -1. If *high* < *low* -1, return 1 / Π(*expr*,*var*,*high*+1,*low*-1)

Σ(*expr*,*var*,*low*,*high***) (sum)**
Evaluate *expr* for each value of *var* from *low* to *high* and return the sum of the results. Return zero if *high* = *low* -1. If *high* < *low* -1, return -Σ(*expr*,*var*,*high*+1,*low*-1)

^ (power)
expr1^expr2
Return *expr1* raised to the power of *expr2*.

list1^list2
Return list whose elements are the elements of *list1* raised to the powers of the corresponding elements of *list2*.

expr^list
Return list whose elements are *expr* raised to the powers of the elements of *list*.

list^expr
Return list whose elements are the elements of *list* raised to the *expr* power.

matrix^integer
Return *matrix* raised to the *integer* power. *matrix* must be square. If *integer* = -1, return the matrix inverse. If *integer* < -1, return the inverse matrix raised to the -*integer* power.

 # (indirection)
 # varNameString
Refer to the variable whose name is *varNameString*.

ʳ (radian)
*expr*ʳ, *list*ʳ, *matrix*ʳ
In Degree mode, multiply the argument by 180/π. Return argument unchanged in Radian mode. Use ʳ to force radians regardless of the current Angle mode.

° (degree)
expr°, *list*°, *matrix*°
In Radian mode, multiply the argument by π/180. Return argument unchanged in Degree mode. Use ° to force radians regardless of the current Angle mode.

∠ (angle)
[*r*,∠θ_*angle*] (polar input)
[*r*,∠θ_*angle*,*z*] (cylindrical input)
[*r*,∠θ_*angle*,ϕ_*angle*] (spherical input)
Return argument coordinates as a vector depending on the current Vector format setting: rectangular, cylindrical or spherical

(*magnitude*∠*angle*) (polar input)
Enter a complex value in (r∠θ) format. *angle* is interpreted according to the current Angle mode setting.

°, ', " (degree, minute, second)
dd°*mm*'*ss.ss*"
Return *dd* + (*mm*/60) + (*ss.ss*/3600), where *dd* is may be positive or negative, *mm* and *ss.ss* are non-negative numbers. Enter a number in base-60 format, which allows entering angle in degrees, minutes and seconds, regardless of the current Angle mode. Also allows entry of times as hours, minutes and seconds.

' (prime)
var'
var''
Enter prime symbol in a differential equation (DE). A single prime denotes a first-order DE, two prime symbols denote a second-order DE.

_ (underscore)
expr_unit
Designate the *unit*s for *expr*. All unit names begin with the underscore.
*var_*
Treat *var* as complex if it has no value. By default, variables without the underscore are treated as real. Variables with values are treated as the value type; real or complex. Complex numbers can be stored in variables without the underscore, but complex operations work better if underscores are used.

▸ (convert)
expr_unit1▸*_unit2*
Convert *expr* with units *unit1* to units *unit2*, excluding temperature conversions. The units must be in the same category. Use tmpCnv() and ΔtmpCnv() for temperature conversions.

10^()
(*expr*), (*list*)
Return 10 raised to the power of the argument.

(*matrix*)
Return 10 raised to the power of square *matrix*. This is not the same as 10 raised to the power of each element. Refer to cos() for calculation details. *matrix* must be diagonalizable, and the result always contains floating-point numbers.

x^{-1} (^-1)
expr^-1
list^-1
Return the reciprocal of the argument.

matrix^-1
Return the inverse of square *matrix*, which must be non-singular.

| ("with")
expr|*Boolean1* [and *Boolean2*]...[and *BooleanN*]
Evaluate *expr* subject to the *Boolean* contraints. This allows substitutions, interval constraints and exclusions. If *Boolean* is an equality such as *var* = *value*, then each occurence of *var* in *expr* is substituted with *value*. A constraint is formed with two or more *Boolean*s joined with 'and', where each *Boolean* is an inequality. An exclusion is a *Boolean* of the form *var* ≠ *value*, which is primarily used to exclude exact solutions of cSolve(), cZeros(), solve(), etc.

→ (store)
expr→*var*
list→*var*
matrix→*var*
Create *var* if it does not exist and initialize it to *expr*, *list* or *matrix*. If *var* exists and is not locked or protected, replace its contents with *expr*, *list* or *matrix*.

expr→*functionName*([*parameter1*,...])
Create *functionName* if it does not exist and initialize it to *expr*. If *functionName* exists and is not locked or protected, replace its contents with *expr*. *parameter*s are optional function arguments.

list→*functionName*([*parameter1*,...])
matrix→*functionName*([*parameter1*,...])
Create *functionName* if it does not exist and initialize it to *list* or *matrix*. If *functionName* exists and is not locked or protected, replace its contents with *list* or *matrix*. *parameter*s are optional function arguments.

© (comment)
[*text*]
Process *text* as a comment line in a program or function. © may be at the beginning of a line or anywhere within the line. All *text* to the right of © to the end of the line is the comment.

0b, 0h
0b*binaryNumber*
0h*hexadecimalNumber*
Denote a binary or hexadecimal integer. 0b or 0h must be entered regardless of the Base mode setting, otherwise the number is treated as decimal (base 10).

Reserved system variable names

Δtbl	nmax	yfact
Δx	nmin	ygrid
Δy	nStat	yi1-yi99
Σx	ok	ymax
Σx^2	plotStep	ymin
Σxy	plotStrt	yscl
Σy	q1	yt1()-yt99()
Σy^2	q3	zθmax
σx	R^2	zθmin
σy	r1()-r99()	zθstep
θc	rc	z1()-z99()
θmax	regCoef	zc
θmin	regEq(x)	zeyeθ
θstep	seed1	zeyeϕ
c1-c99	seed2	zeyeψ
corr	Sx	zfact
diftol	Sy	zmax
dtime	sysData	zmin
eqn	sysMath	znmax
errornum	tØ	znmin
estep	tblInput	zplstep
exp	tblStart	zplstrt
eyeθ	tc	zscl
eyeϕ	tmax	ztØde
eyeψ	tmin	ztmax
fldpic	tplot	ztmaxde
fldres	tstep	ztmin
main	u1()-u99()	ztplotde
maxX	ui1-ui99	ztstep
maxY	$\bar{x}$	ztstepde
medStat	xc	zxgrid
medx1	xfact	zxmax
medx2	xgrid	zxmin
medx3	xmax	zxres
medy1	xmin	zxscl
medy2	xres	zygrid
medy3	xscl	zymax
minX	xt1()-xt99()	zymin
minY	$\bar{y}$	zyscl
nc	y1'()-y99'()	zzmax
ncontour	y1()-y99()	zzmin
ncurves	yc	zzscl

EOS (Equation Operating System) Hierarchy

Exponentiation (^) and element-by-element exponentiation (.^) are evaluated from right to left, for example, 2^3^4 = 2^(3^4).

Post operations and exponentiation are performed before negation, for example, -9^2 = -(9^2).

Other operations are performed according the these priorities:

1 Parentheses (), brackets [] and braces {}
2 Indirection (#)
3 Function calls
4 Post operators: degrees-minutes-seconds (°,',"), factorial (!), percentage (%), radian (ʳ), transpose (ᵀ)
5 Exponentiation, power operator (^)
6 Negation (-)
7 String concatenation (&)
8 Multiplication (*), division (/)
9 Addition (+), subtraction (-)
10 Equality relations: =, ≠, <, ≤, >, ≥
11 Logical 'not'
12 Logical 'and'
13 Logical 'or', 'xor'
14 Constraint "with" operator (|)
15 Store (→)