

TI-Nspire™ CX CAS Reference Guide

Important Information

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

© 2021 Texas Instruments Incorporated

Actual products may vary slightly from provided images.

Contents

Expression Templates	1
Alphabetical Listing	8
A	8
B	17
C	20
D	44
E	57
F	67
G	76
I	86
L	95
M	111
N	119
O	128
P	130
Q	139
R	142
S	157
T	182
U	197
V	198
W	199
X	201
Z	202
Symbols	210
TI-Nspire™ CX II - Draw Commands	236
Graphics Programming	236
Graphics Screen	236
Default View and Settings	237
Graphics Screen Errors Messages	238
Invalid Commands While in Graphics Mode	238
C	239
D	240
F	243
G	245
P	246
S	248
U	250

Empty (Void) Elements	251
Shortcuts for Entering Math Expressions	253
EOS™ (Equation Operating System) Hierarchy	255
TI-Nspire CX II - TI-Basic Programming Features	257
Auto-indentation in Programming Editor	257
Improved Error Messages for TI-Basic	257
Constants and Values	260
Error Codes and Messages	261
Warning Codes and Messages	269
General Information	271
Online Help	271
Contact TI Support	271
Service and Warranty Information	271
Index	272

Expression Templates

Expression templates give you an easy way to enter math expressions in standard mathematical notation. When you insert a template, it appears on the entry line with small blocks at positions where you can enter elements. A cursor shows which element you can enter.

Position the cursor on each element, and type a value or expression for the element.

Fraction template

  keys



Note: See also / (divide), page 212.



Example:

$$\frac{12}{8 \cdot 2} \qquad \frac{3}{4}$$

Exponent template

 key





Note: Type the first value, press , and then type the exponent. To return the cursor to the baseline, press right arrow ().

Note: See also ^ (power), page 213.

Example:

$$2^3 \qquad 8$$

Square root template

  keys



Note: See also $\sqrt{()}$ (square root), page 223.

Example:

$$\begin{array}{l} \sqrt{4} \qquad 2 \\ \sqrt{\{9,a,4\}} \qquad \{3,\sqrt{a},2\} \\ \sqrt{4} \qquad 2 \\ \sqrt{\{9,16,4\}} \qquad \{3,4,2\} \end{array}$$

Nth root template

  keys



Note: See also root(), page 154.

Example:

Nth root template

ctrl \wedge keys

$$\sqrt[3]{8} \quad 2$$

$$\sqrt[3]{\{8, 27, b\}} \quad \left\{ 2, 3, b^{\frac{1}{3}} \right\}$$

e exponent template

e^x keys

e^{\square}

Natural exponential e raised to a power

Note: See also $e^{\square}()$, page 57.

Example:

$$e^1 \quad e$$

$$e^1. \quad 2.71828182846$$

Log template

ctrl 10^x key

$\log_{\square}(\square)$

Calculates log to a specified base. For a default of base 10, omit the base.

Note: See also $\log()$, page 107.

Example:

$$\log_{\frac{1}{4}}(2.) \quad 0.5$$

Piecewise template (2-piece)

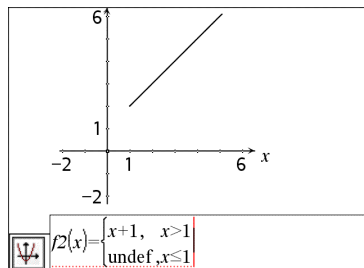
Catalog > $\left\{ \begin{array}{l} \square \\ \square \end{array} \right\}$

$\left\{ \begin{array}{l} \square \\ \square \end{array} \right\}$

Lets you create expressions and conditions for a two-piece piecewise function. To add a piece, click in the template and repeat the template.

Note: See also $\text{piecewise}()$, page 132.

Example:



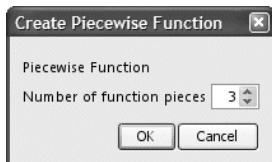
Piecewise template (N-piece)

Catalog > 

Lets you create expressions and conditions for an N -piece piecewise function. Prompts for N .

Example:

See the example for Piecewise template (2-piece).



Note: See also `piecewise()`, page 132.

System of 2 equations template

Catalog > 



Creates a system of two equations. To add a row to an existing system, click in the template and repeat the template.

Example:

$$\text{solve} \left(\begin{cases} x+y=0 \\ x-y=5 \end{cases}, x, y \right) \quad x = \frac{5}{2} \text{ and } y = -\frac{5}{2}$$

$$\text{solve} \left(\begin{cases} y=x^2-2 \\ x+2 \cdot y=-1 \end{cases}, x, y \right) \\ x = -\frac{3}{2} \text{ and } y = \frac{1}{4} \text{ or } x=1 \text{ and } y=-1$$

Note: See also `system()`, page 181.

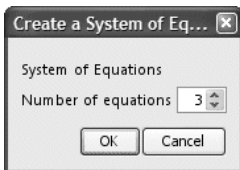
System of N equations template

Catalog > 

Lets you create a system of N equations. Prompts for N .

Example:

See the example for System of equations template (2-equation).



Note: See also `system()`, page 181.

Absolute value template

Catalog > 



Note: See also `abs()`, page 8.

Example:

Absolute value template

Catalog > 

$$\left\{ 2, -3, 4, -4^3 \right\} \quad \left\{ 2, 3, 4, 64 \right\}$$

dd°mm'ss.ss" template

Catalog > 

"

Example:

$$30^{\circ}15'10'' \quad \frac{10891 \cdot \pi}{64800}$$

Lets you enter angles in **dd°mm'ss.ss"** format, where **dd** is the number of decimal degrees, **mm** is the number of minutes, and **ss.ss** is the number of seconds.

Matrix template (2 x 2)

Catalog > 

]

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot a \quad \begin{bmatrix} a & 2 \cdot a \\ 3 \cdot a & 4 \cdot a \end{bmatrix}$$

Creates a 2 x 2 matrix.

Matrix template (1 x 2)

Catalog > 

Example:

$$\text{crossP}(\left[\begin{bmatrix} 1 & 2 \end{bmatrix}, \begin{bmatrix} 3 & 4 \end{bmatrix} \right]) \quad \left[\begin{bmatrix} 0 & 0 & -2 \end{bmatrix} \right]$$

Matrix template (2 x 1)

Catalog > 

]

Example:

$$\begin{bmatrix} 5 \\ 8 \end{bmatrix} \cdot 0.01 \quad \begin{bmatrix} 0.05 \\ 0.08 \end{bmatrix}$$

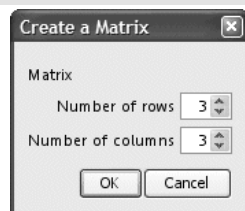
Matrix template (m x n)

Catalog > 

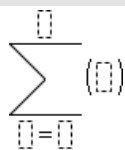
The template appears after you are prompted to specify the number of rows and columns.

Example:

$$\text{diag} \left(\begin{bmatrix} 4 & 2 & 6 \\ 1 & 2 & 3 \\ 5 & 7 & 9 \end{bmatrix} \right) \quad \left[\begin{bmatrix} 4 & 2 & 9 \end{bmatrix} \right]$$



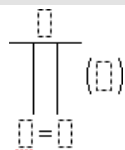
Note: If you create a matrix with a large number of rows and columns, it may take a few moments to appear.

Sum template (Σ)

Example:

$$\sum_{n=3}^7 (n) = 25$$

Note: See also $\Sigma()$ (**sumSeq**), page 224.

Product template (Π)

Example:

$$\prod_{n=1}^5 \left(\frac{1}{n}\right) = \frac{1}{120}$$

Note: See also $\Pi()$ (**prodSeq**), page 223.

First derivative template

$$\frac{d}{d\Box}(\Box)$$

Example:

$$\frac{d}{dx}(x^3) = 3 \cdot x^2$$

$$\frac{d}{dx}(x^3)|_{x=3} = 27$$

The first derivative template can also be used to calculate first derivative at a point.

Note: See also **d()** (**derivative**), page 221.

Second derivative template

Catalog > 

$$\frac{d^2}{dx^2}(\square)$$

The second derivative template can also be used to calculate second derivative at a point.

Note: See also **d()** (derivative), page 221.

Example:

$$\frac{d^2}{dx^2}(x^3) \quad 6 \cdot x$$

$$\frac{d^2}{dx^2}(x^3)|_{x=3} \quad 18$$

Nth derivative template

Catalog > 

$$\frac{d^n}{dx^n}(\square)$$

The n th derivative template can be used to calculate the n th derivative.

Note: See also **d()** (derivative), page 221.

Example:

$$\frac{d^3}{dx^3}(x^3)|_{x=3} \quad 6$$

Definite integral template

Catalog > 

$$\int_a^b \square dx$$

Note: See also **∫()** **integral()**, page 221.

Example:

$$\int_a^b x^2 dx \quad \frac{b^3}{3} - \frac{a^3}{3}$$

Indefinite integral template

Catalog > 

$$\int \square dx$$

Note: See also **∫()** **integral()**, page 221.

Example:

$$\int x^2 dx \quad \frac{x^3}{3}$$

Limit template

Catalog > 

$$\lim_{\square \rightarrow \square} \square$$

Example:

$$\lim_{x \rightarrow 5} (2 \cdot x + 3) \quad 13$$

Use $-$ or $(-)$ for left hand limit. Use $+$ for right hand limit.

Note: See also `limit()`, page 6.

Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, page 210. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

abs()

Catalog > 

abs(*Expr1*) ⇒ *expression*

$$\left\{ \begin{array}{c} \frac{\pi}{2}, \frac{\pi}{3} \end{array} \right\}$$

abs(*List1*) ⇒ *list*

abs(*Matrix1*) ⇒ *matrix*

$$2-3 \cdot i \qquad \sqrt{13}$$

$$|z| \qquad |z|$$

Returns the absolute value of the argument.

$$|x+y \cdot i| \qquad \sqrt{x^2+y^2}$$

Note: See also **Absolute value template**, page 3.

If the argument is a complex number, returns the number's modulus.

Note: All undefined variables are treated as real variables.

amortTbl()

Catalog > 

amortTbl(*NPmt*,*N*,*I*,*PV*, [*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [*roundValue*]) ⇒ *matrix*

amortTbl(12,60,10,5000,,,12,12)

Amortization function that returns a matrix as an amortization table for a set of TVM arguments.

NPmt is the number of payments to be included in the table. The table starts with the first payment.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY*, and *PmtAt* are described in the table of TVM arguments, page 195.

0	0.	0.	5000.
1	-41.67	-64.57	4935.43
2	-41.13	-65.11	4870.32
3	-40.59	-65.65	4804.67
4	-40.04	-66.2	4738.47
5	-39.49	-66.75	4671.72
6	-38.93	-67.31	4604.41
7	-38.37	-67.87	4536.54
8	-37.8	-68.44	4468.1
9	-37.23	-69.01	4399.09
10	-36.66	-69.58	4329.51
11	-36.08	-70.16	4259.35
12	-35.49	-70.75	4188.6

- If you omit *Pmt*, it defaults to $Pmt = \text{tvmPmt}(N, I, PV, FV, PpY, CpY, PmtAt)$.
- If you omit *FV*, it defaults to $FV = 0$.
- The defaults for *PpY*, *CpY*, and *PmtAt* are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.

The columns in the result matrix are in this order: Payment number, amount paid to interest, amount paid to principal, and balance.

The balance displayed in row *n* is the balance after payment *n*.

You can use the output matrix as input for the other amortization functions $\Sigma\text{Int}()$ and $\Sigma\text{Prn}()$, page 225, and **bal()**, page 17.

and

BooleanExpr1 and BooleanExpr2 \Rightarrow
Boolean expression

$x \geq 3$ and $x \geq 4$	$x \geq 4$
$\{x \geq 3, x \leq 0\}$ and $\{x \geq 4, x \leq -2\}$	$\{x \geq 4, x \leq -2\}$

BooleanList1 and BooleanList2 \Rightarrow
Boolean list

BooleanMatrix1 and BooleanMatrix2 \Rightarrow
Boolean matrix

Returns true or false or a simplified form of the original entry.

Integer1 and Integer2 \Rightarrow *integer*

In Hex base mode:

0h7AC36 and 0h3D5F	0h2C16
--------------------	--------

Compares two real integers bit-by-bit using an **and** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

Important: Zero, not the letter O.

In Bin base mode:

0b100101 and 0b100	0b100
--------------------	-------

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

In Dec base mode:

37 and 0b100	4
--------------	---

Note: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

angle()Catalog > **angle**(*Expr1*) ⇒ *expression*

Returns the angle of the argument, interpreting the argument as a complex number.

Note: All undefined variables are treated as real variables.

In Degree angle mode:

$\text{angle}(0+2\cdot i)$	90
----------------------------	----

In Gradian angle mode:

$\text{angle}(0+3\cdot i)$	100
----------------------------	-----

In Radian angle mode:

$\text{angle}(1+i)$	$\frac{\pi}{4}$
---------------------	-----------------

$\text{angle}(z)$	$-\pi \cdot \frac{(\text{sign}(z)-1)}{2}$
-------------------	---

$\text{angle}(x+i\cdot y)$	$\frac{\pi \cdot \text{sign}(y)}{2} - \tan^{-1}\left(\frac{x}{y}\right)$
----------------------------	--

$\text{angle}(\{1+2\cdot i, 3+0\cdot i, 0-4\cdot i\})$	$\left\{\frac{\pi}{2} - \tan^{-1}\left(\frac{1}{2}\right), 0, -\frac{\pi}{2}\right\}$
--	---

angle(*List1*) ⇒ *list***angle**(*Matrix1*) ⇒ *matrix*Returns a list or matrix of angles of the elements in *List1* or *Matrix1*, interpreting each element as a complex number that represents a two-dimensional rectangular coordinate point.**ANOVA**Catalog > **ANOVA** *List1, List2[, List3, ..., List20][, Flag]*Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the *stat.results* variable. (page 176)*Flag*=0 for Data, *Flag*=1 for Stats

Output variable	Description
stat.F	Value of the F statistic
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected

Output variable	Description
stat.df	Degrees of freedom of the groups
stat.SS	Sum of squares of the groups
stat.MS	Mean squares for the groups
stat.dfError	Degrees of freedom of the errors
stat.SSError	Sum of squares of the errors
stat.MSError	Mean square for the errors
stat.sp	Pooled standard deviation
stat.xbarlist	Mean of the input of the lists
stat.CLowerList	95% confidence intervals for the mean of each input list
stat.CUpperList	95% confidence intervals for the mean of each input list

ANOVA2way

Catalog > 

ANOVA2way *List1,List2[,List3,...,List10]*
[,LevRow]

Computes a two-way analysis of variance for comparing the means of two to 10 populations. A summary of results is stored in the *stat.results* variable. (See page 176.)

LevRow=0 for Block

LevRow=2,3,...,*Len*-1, for Two Factor,
 where $Len = \text{length}(List1) = \text{length}(List2) = \dots$
 $= \text{length}(List10)$ and $Len / LevRow \hat{=} \{2,3,\dots\}$

Outputs: Block Design

Output variable	Description
stat.F	F statistic of the column factor
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom of the column factor
stat.SS	Sum of squares of the column factor
stat.MS	Mean squares for column factor
stat.FBlock	F statistic for factor

Output variable	Description
stat.PValBlock	Least probability at which the null hypothesis can be rejected
stat.dfBlock	Degrees of freedom for factor
stat.SSBlock	Sum of squares for factor
stat.MSBlock	Mean squares for factor
stat.dfError	Degrees of freedom of the errors
stat.SSError	Sum of squares of the errors
stat.MSError	Mean squares for the errors
stat.s	Standard deviation of the error

COLUMN FACTOR Outputs

Output variable	Description
stat.Fcol	F statistic of the column factor
stat.PValCol	Probability value of the column factor
stat.dfCol	Degrees of freedom of the column factor
stat.SSCol	Sum of squares of the column factor
stat.MSCol	Mean squares for column factor

ROW FACTOR Outputs

Output variable	Description
stat.FRow	F statistic of the row factor
stat.PValRow	Probability value of the row factor
stat.dfRow	Degrees of freedom of the row factor
stat.SSRow	Sum of squares of the row factor
stat.MSRow	Mean squares for row factor

INTERACTION Outputs



Output variable	Description
stat.FInteract	F statistic of the interaction
stat.PValInteract	Probability value of the interaction
stat.dfInteract	Degrees of freedom of the interaction

Output variable	Description
stat.SSIInteract	Sum of squares of the interaction
stat.MSIInteract	Mean squares for interaction

ERROR Outputs

Output variable	Description
stat.dfError	Degrees of freedom of the errors
stat.SSError	Sum of squares of the errors
stat.MSError	Mean squares for the errors
s	Standard deviation of the error

Ans

  keys

Ans \Rightarrow *value*

Returns the result of the most recently evaluated expression.



56	56
56+4	60
60+4	64

approx()

Catalog > 

approx(*Expr1*) \Rightarrow *expression*

Returns the evaluation of the argument as an expression containing decimal values, when possible, regardless of the current **Auto** or **Approximate** mode.

This is equivalent to entering the argument and pressing  .

approx(*List1*) \Rightarrow *list*

approx(*Matrix1*) \Rightarrow *matrix*

Returns a list or *matrix* where each element has been evaluated to a decimal value, when possible.

$\text{approx}\left(\frac{1}{3}\right)$	0.333333
$\text{approx}\left(\left\{\frac{1}{3}, \frac{1}{9}\right\}\right)$	{0.333333, 0.111111}
$\text{approx}\left(\{\sin(\pi), \cos(\pi)\}\right)$	{0., -1.}
$\text{approx}\left(\left[\sqrt{2}, \sqrt{3}\right]\right)$	[1.41421 1.73205]
$\text{approx}\left(\left[\frac{1}{3}, \frac{1}{9}\right]\right)$	[0.333333 0.111111]
$\text{approx}\left(\{\sin(\pi), \cos(\pi)\}\right)$	{0., -1.}
$\text{approx}\left(\left[\sqrt{2}, \sqrt{3}\right]\right)$	[1.41421 1.73205]

► approxFraction()

Catalog >

Expr ► **approxFraction**([*Tol*]) ⇒ *expression*

$$\frac{1}{2} + \frac{1}{3} + \tan(\pi) \qquad 0.833333$$

List ► **approxFraction**([*Tol*]) ⇒ *list*

$$0.8333333333333333 \text{ ► } \text{approxFraction}(5.E-14)$$

Matrix ► **approxFraction**([*Tol*]) ⇒ *matrix*

$$\frac{5}{6}$$

Returns the input as a fraction, using a tolerance of *Tol*. If *Tol* is omitted, a tolerance of 5.E-14 is used.

$$\{\pi, 1.5\} \text{ ► } \text{approxFraction}(5.E-14)$$

$$\left\{ \frac{5419351}{1725033}, \frac{3}{2} \right\}$$

Note: You can insert this function from the computer keyboard by typing @>**approxFraction**(...).

approxRational()

Catalog >

approxRational(*Expr*[, *Tol*]) ⇒ *expression*

$$\text{approxRational}(0.333, 5 \cdot 10^{-5}) \qquad \frac{333}{1000}$$

approxRational(*List*[, *Tol*]) ⇒ *list*

$$\text{approxRational}(\{0.2, 0.33, 4.125\}, 5.E-14)$$

approxRational(*Matrix*[, *Tol*]) ⇒ *matrix*

$$\left\{ \frac{1}{5}, \frac{33}{100}, \frac{33}{8} \right\}$$

Returns the argument as a fraction using a tolerance of *Tol*. If *Tol* is omitted, a tolerance of 5.E-14 is used.

arccos()

See cos-1(), page 31.

arccosh()

See cosh-1(), page 32.

arccot()

See cot-1(), page 33.

arccoth()

See coth-1(), page 34.

arccsc()See $\text{csc}^{-1}()$, page 37.**arccsch()**See $\text{csch}^{-1}()$, page 37.**arcLen()**Catalog > **arcLen**(*Expr1*, *Var*, *Start*, *End*) \Rightarrow
*expression*Returns the arc length of *Expr1* from *Start* to *End* with respect to variable *Var*.

Arc length is calculated as an integral assuming a function mode definition.

arcLen(*List1*, *Var*, *Start*, *End*) \Rightarrow *list*Returns a list of the arc lengths of each element of *List1* from *Start* to *End* with respect to *Var*.

$$\frac{\text{arcLen}(\cos(x), x, 0, \pi)}{\text{arcLen}(f(x), x, a, b)} \quad 3.8202$$

$$\int_a^b \sqrt{\left(\frac{d}{dx}(f(x))\right)^2 + 1} dx$$

$$\frac{\text{arcLen}(\{\sin(x), \cos(x)\}, x, 0, \pi)}{\{3.8202, 3.8202\}}$$

arcsec()See $\text{sec}^{-1}()$, page 157.**arcsech()**See $\text{sech}^{-1}()$, page 158.**arcsin()**See $\text{sin}^{-1}()$, page 167.**arcsinh()**See $\text{sinh}^{-1}()$, page 168.**arctan()**See $\text{tan}^{-1}()$, page 183.

augment()Catalog > **augment(List1, List2)** ⇒ list

augment({1,-3,2},{5,4}) {1,-3,2,5,4}

Returns a new list that is *List2* appended to the end of *List1*.

augment(Matrix1, Matrix2) ⇒ matrix

1 2	→ m1	1 2
3 4		3 4
5	→ m2	5
6		6
augment(m1,m2)		1 2 5
		3 4 6

Returns a new matrix that is *Matrix2* appended to *Matrix1*. When the “,” character is used, the matrices must have equal row dimensions, and *Matrix2* is appended to *Matrix1* as new columns. Does not alter *Matrix1* or *Matrix2*.

avgRC()Catalog > **avgRC(Expr1, Var [=Value] [, Step])** ⇒ expressionavgRC(f(x),x,h) $\frac{f(x+h)-f(x)}{h}$ **avgRC(Expr1, Var [=Value] [, List1])** ⇒ listavgRC(sin(x),x,h)|x=2 $\frac{\sin(h+2)-\sin(2)}{h}$ **avgRC(List1, Var [=Value] [, Step])** ⇒ listavgRC(x²-x+2,x) 2·(x-0.4995)avgRC(x²-x+2,x,0.1) 2·(x-0.45)**avgRC(Matrix1, Var [=Value] [, Step])** ⇒ matrixavgRC(x²-x+2,x,3) 2·(x+1)

Returns the forward-difference quotient (average rate of change).

Expr1 can be a user-defined function name (see **Func**).

When *Value* is specified, it overrides any prior variable assignment or any current “|” substitution for the variable.

Step is the step value. If *Step* is omitted, it defaults to 0.001.

Note that the similar function **centralDiff()** uses the central-difference quotient.

bal()

Catalog >

bal(*NPmt*,*N*,*I*,*PV*, [*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [*roundValue*]) ⇒ *value*

bal(*NPmt*,*amortTable*) ⇒ *value*

Amortization function that calculates schedule balance after a specified payment.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY*, and *PmtAt* are described in the table of TVM arguments, page 195.

NPmt specifies the payment number after which you want the data calculated.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY*, and *PmtAt* are described in the table of TVM arguments, page 195.

- If you omit *Pmt*, it defaults to *Pmt*=**tvmpmt**(*N*,*I*,*PV*,*FV*,*PpY*,*CpY*,*PmtAt*).
- If you omit *FV*, it defaults to *FV*=0.
- The defaults for *PpY*, *CpY*, and *PmtAt* are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.

bal(*NPmt*,*amortTable*) calculates the balance after payment number *NPmt*, based on amortization table *amortTable*. The *amortTable* argument must be a matrix in the form described under **amortTbl()**, page 8.

Note: See also **ΣInt()** and **ΣPrn()**, page 225.

bal(5,6,5.75,5000,,12,12) 833.11

tbl:=**amortTbl**(6,6,5.75,5000,,12,12)

0	0.	0.	5000.
1	-23.35	-825.63	4174.37
2	-19.49	-829.49	3344.88
3	-15.62	-833.36	2511.52
4	-11.73	-837.25	1674.27
5	-7.82	-841.16	833.11
6	-3.89	-845.09	-11.98

bal(4,*tbl*) 1674.27

► **Base2**

Catalog >

Integer1 ► **Base2** ⇒ *integer*

256►**Base2** 0b10000000

Note: You can insert this operator from the computer keyboard by typing **@>Base2**.

0h1F►**Base2** 0b11111

Converts *Integer1* to a binary number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively. Use a zero, not the letter O, followed by b or h.

0b *binaryNumber*

0h *hexadecimalNumber*

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal (base 10). The result is displayed in binary, regardless of the Base mode.

Negative numbers are displayed in “two's complement” form. For example,

-1 is displayed as

0hFFFFFFFFFFFFFFFF in Hex base mode

0b111...111 (64 1's) in Binary base mode

-2⁶³ is displayed as

0h8000000000000000 in Hex base mode

0b100...000 (63 zeros) in Binary base mode

If you enter a decimal integer that is outside the range of a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. Consider the following examples of values outside the range.

2⁶³ becomes -2⁶³ and is displayed as

0h8000000000000000 in Hex base mode

0b100...000 (63 zeros) in Binary base mode

2⁶⁴ becomes 0 and is displayed as

0h0 in Hex base mode

0b0 in Binary base mode

-2⁶³ - 1 becomes 2⁶³ - 1 and is displayed as

0h7FFFFFFFFFFFFFFFF in Hex base mode

0b111...111 (64 1's) in Binary base mode

Integer1 ► Base10 ⇒ *integer*

0b10011 ► Base10

19

0h1F ► Base10

31

Note: You can insert this operator from the computer keyboard by typing @>Base10.

Converts *Integer1* to a decimal (base 10) number. A binary or hexadecimal entry must always have a 0b or 0h prefix, respectively.

0b *binaryNumber*

0h *hexadecimalNumber*

Zero, not the letter O, followed by b or h.

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal. The result is displayed in decimal, regardless of the Base mode.

Integer1 ►Base16 ⇒ *integer*

256►Base16

0h100

Note: You can insert this operator from the computer keyboard by typing @>Base16.

0b111100001111►Base16

0hFOF

Converts *Integer1* to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

0b *binaryNumber*

0h *hexadecimalNumber*

Zero, not the letter O, followed by b or h.

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal (base 10). The result is displayed in hexadecimal, regardless of the Base mode.

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ►Base2, page 17.

binomCdf(n,p) \Rightarrow *list*

binomCdf($n,p,lowBound,upBound$) \Rightarrow *number* if $lowBound$ and $upBound$ are numbers, *list* if $lowBound$ and $upBound$ are lists

binomCdf($n,p,upBound$) for $P(0 \leq X \leq upBound)$ \Rightarrow *number* if $upBound$ is a number, *list* if $upBound$ is a list

Computes a cumulative probability for the discrete binomial distribution with n number of trials and probability p of success on each trial.

For $P(X \leq upBound)$, set $lowBound=0$

binomPdf(n,p) \Rightarrow *list*

binomPdf($n,p,XVal$) \Rightarrow *number* if $XVal$ is a number, *list* if $XVal$ is a list

Computes a probability for the discrete binomial distribution with n number of trials and probability p of success on each trial.

C

ceiling($ExprI$) \Rightarrow *integer*

$ceiling(.456)$

1.

Returns the nearest integer that is \geq the argument.

The argument can be a real or a complex number.

Note: See also **floor()**.

ceiling($ListI$) \Rightarrow *list*

ceiling($MatrixI$) \Rightarrow *matrix*

$ceiling(\{-3.1, 1, 2.5\})$	$\{-3., 1, 3.\}$
$ceiling\left(\begin{pmatrix} 0 & -3.2 \cdot i \\ 1.3 & 4 \end{pmatrix}\right)$	$\begin{pmatrix} 0 & -3. \cdot i \\ 2. & 4 \end{pmatrix}$

Returns a list or matrix of the ceiling of each element.

centralDiff()

Catalog >

centralDiff(*Expr1*, *Var* [= *Value*][, *Step*]) ⇒ *expression***centralDiff**(*Expr1*, *Var* [, *Step*]) | *Var* = *Value* ⇒ *expression***centralDiff**(*Expr1*, *Var* [= *Value*][, *List*]) ⇒ *list***centralDiff**(*List1*, *Var* [= *Value*][, *Step*]) ⇒ *list***centralDiff**(*Matrix1*, *Var* [= *Value*][, *Step*]) ⇒ *matrix*

Returns the numerical derivative using the central difference quotient formula.

When *Value* is specified, it overrides any prior variable assignment or any current “|” substitution for the variable.*Step* is the step value. If *Step* is omitted, it defaults to 0.001.When using *List1* or *Matrix1*, the operation gets mapped across the values in the list or across the matrix elements.**Note:** See also **avgRC()** and **d()**.

$\text{centralDiff}(\cos(x), x, h)$	$\frac{-\cos(x-h) - \cos(x+h)}{2 \cdot h}$
$\lim_{h \rightarrow 0} (\text{centralDiff}(\cos(x), x, h))$	$-\sin(x)$
$\text{centralDiff}(x^3, x, 0.01)$	$3 \cdot (x^2 + 0.000033)$
$\text{centralDiff}(\cos(x), x) x = \frac{\pi}{2}$	$-1.$
$\text{centralDiff}(x^2, x, \{0.01, 0.1\})$	$\{2 \cdot x, 2 \cdot x\}$

cFactor()

Catalog >

cFactor(*Expr1*[, *Var*]) ⇒ *expression***cFactor**(*List1*[, *Var*]) ⇒ *list***cFactor**(*Matrix1*[, *Var*]) ⇒ *matrix***cFactor**(*Expr1*) returns *Expr1* factored with respect to all of its variables over a common denominator.*Expr1* is factored as much as possible toward linear rational factors even if this introduces new non-real numbers. This alternative is appropriate if you want factorization with respect to more than one variable.

$\text{cFactor}(a^3 \cdot x^2 + a \cdot x^2 + a^3 + a, x)$	$a \cdot (a^2 + 1) \cdot (x - i) \cdot (x + i)$
$\text{cFactor}\left(x^2 + \frac{4}{9}\right)$	$\frac{(3 \cdot x - 2 \cdot i) \cdot (3 \cdot x + 2 \cdot i)}{9}$
$\text{cFactor}(x^2 + 3)$	$x^2 + 3$
$\text{cFactor}(x^2 + a)$	$x^2 + a$

cFactor()

Catalog >

cFactor(*Expr1*, *Var*) returns *Expr1* factored with respect to variable *Var*.

Expr1 is factored as much as possible toward factors that are linear in *Var*, with perhaps non-real constants, even if it introduces irrational constants or subexpressions that are irrational in other variables.

The factors and their terms are sorted with *Var* as the main variable. Similar powers of *Var* are collected in each factor. Include *Var* if factorization is needed with respect to only that variable and you are willing to accept irrational expressions in any other variables to increase factorization with respect to *Var*. There might be some incidental factoring with respect to other variables.

For the Auto setting of the **Auto** or **Approximate** mode, including *Var* also permits approximation with floating-point coefficients where irrational coefficients cannot be explicitly expressed concisely in terms of the built-in functions. Even when there is only one variable, including *Var* might yield more complete factorization.

Note: See also **factor()**.

$\text{cFactor}(a^3 \cdot x^2 + a \cdot x^2 + a^3 + a \cdot x)$	$a \cdot (a^2 + 1) \cdot (x - i) \cdot (x + i)$
$\text{cFactor}(x^2 + 3x)$	$(x + \sqrt{3} \cdot i) \cdot (x - \sqrt{3} \cdot i)$
$\text{cFactor}(x^2 + a \cdot x)$	$(x + \sqrt{a \cdot i}) \cdot (x + \sqrt{a \cdot i})$

$\text{cFactor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3)$	$x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3$
$\text{cFactor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3, x)$	$(x - 0.964673) \cdot (x + 0.611649) \cdot (x + 2.12543) \cdot (x + 0.964673)$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

char()

Catalog >

char(*Integer*) \Rightarrow *character*

Returns a character string containing the character numbered *Integer* from the handheld character set. The valid range for *Integer* is 0–65535.

$\text{char}(38)$	"&"
$\text{char}(65)$	"A"

charPoly()Catalog > **charPoly**(*squareMatrix*, *Var*) ⇒
*polynomial expression***charPoly**(*squareMatrix*, *Expr*) ⇒
*polynomial expression***charPoly**(*squareMatrix1*, *Matrix2*) ⇒
polynomial expression

Returns the characteristic polynomial of *squareMatrix*. The characteristic polynomial of $n \times n$ matrix A , denoted by $p_A(\lambda)$, is the polynomial defined by

$$p_A(\lambda) = \det(\lambda \cdot \mathbf{I} - A)$$

where \mathbf{I} denotes the $n \times n$ identity matrix.

squareMatrix1 and *squareMatrix2* must have the equal dimensions.

$m := \begin{bmatrix} 1 & 3 & 0 \\ 2 & -1 & 0 \\ -2 & 2 & 5 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & 0 \\ 2 & -1 & 0 \\ -2 & 2 & 5 \end{bmatrix}$
$\text{charPoly}(m, x)$	$-x^3 + 5 \cdot x^2 + 7 \cdot x - 35$
$\text{charPoly}(m, x^2 + 1)$	$-x^6 + 2 \cdot x^4 + 14 \cdot x^2 - 24$
$\text{charPoly}(m, m)$	0

 χ^2 wayCatalog >  **χ^2 way** *obsMatrix***chi22way** *obsMatrix*

Computes a χ^2 test for association on the two-way table of counts in the observed matrix *obsMatrix*. A summary of results is stored in the *stat.results* variable. (page 176)

For information on the effect of empty elements in a matrix, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat. χ^2	Chi square stat: $\text{sum}(\text{observed} - \text{expected})^2 / \text{expected}$
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom for the chi square statistics
stat.ExpMat	Matrix of expected elemental count table, assuming null hypothesis
stat.CompMat	Matrix of elemental chi square statistic contributions

$\chi^2\text{Cdf}(\text{lowBound}, \text{upBound}, \text{df}) \Rightarrow$ number if *lowBound* and *upBound* are numbers, list if *lowBound* and *upBound* are lists

$\text{chi2Cdf}(\text{lowBound}, \text{upBound}, \text{df}) \Rightarrow$ number if *lowBound* and *upBound* are numbers, list if *lowBound* and *upBound* are lists

Computes the χ^2 distribution probability between *lowBound* and *upBound* for the specified degrees of freedom *df*.

For $P(X \leq \text{upBound})$, set *lowBound* = 0.

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

$\chi^2\text{GOF } \text{obsList}, \text{expList}, \text{df}$

$\text{chi2GOF } \text{obsList}, \text{expList}, \text{df}$

Performs a test to confirm that sample data is from a population that conforms to a specified distribution. *obsList* is a list of counts and must contain integers. A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

Output variable	Description
stat. χ^2	Chi square stat: $\text{sum}((\text{observed} - \text{expected})^2 / \text{expected})$
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom for the chi square statistics
stat.CompList	Elemental chi square statistic contributions

$\chi^2\text{Pdf}(XVal, \text{df}) \Rightarrow$ number if *XVal* is a

number, *list* if *XVal* is a list

chi2Pdf(*XVal*,*df*) \Rightarrow number if *XVal* is a number, *list* if *XVal* is a list

Computes the probability density function (pdf) for the χ^2 distribution at a specified *XVal* value for the specified degrees of freedom *df*.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

ClearAZ**ClearAZ**

Clears all single-character variables in the current problem space.

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See **unLock**, page 197.

$5 \rightarrow b$	5
<i>b</i>	5
ClearAZ	Done
<i>b</i>	<i>b</i>

ClrErr**ClrErr**

Clears the error status and sets system variable *errCode* to zero.

The **Else** clause of the **Try...Else...EndTry** block should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. If there are no more pending **Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.

Note: See also **PassErr**, page 131, and **Try**, page 191.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

For an example of **ClrErr**, See Example 2 under the **Try** command, page 191.

colAugment()

Catalog >

colAugment(*Matrix1*, *Matrix2*) ⇒ *matrix*

Returns a new matrix that is *Matrix2* appended to *Matrix1*. The matrices must have equal column dimensions, and *Matrix2* is appended to *Matrix1* as new rows. Does not alter *Matrix1* or *Matrix2*.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
$\begin{bmatrix} 5 & 6 \end{bmatrix} \rightarrow m2$	$\begin{bmatrix} 5 & 6 \end{bmatrix}$
colAugment (<i>m1</i> , <i>m2</i>)	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

colDim()

Catalog >

colDim(*Matrix*) ⇒ *expression*

Returns the number of columns contained in *Matrix*.

colDim ($\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$)	3
--	---

Note: See also **rowDim()**.**colNorm()**

Catalog >

colNorm(*Matrix*) ⇒ *expression*

Returns the maximum of the sums of the absolute values of the elements in the columns in *Matrix*.

$\begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix} \rightarrow mat$	$\begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$
colNorm (<i>mat</i>)	9

Note: Undefined matrix elements are not allowed. See also **rowNorm()**.**comDenom()**

Catalog >

comDenom(*Expr1*[,*Var*]) ⇒ *expression***comDenom**(*List1*[,*Var*]) ⇒ *list***comDenom**(*Matrix1*[,*Var*]) ⇒ *matrix*

comDenom(*Expr1*) returns a reduced ratio of a fully expanded numerator over a fully expanded denominator.

comDenom ($\frac{y^2+y}{(x+1)^2} + y^2+y$)	
$\frac{x^2 \cdot y^2 + x^2 \cdot y + 2 \cdot x \cdot y^2 + 2 \cdot x \cdot y + 2 \cdot y^2 + 2 \cdot y}{x^2 + 2 \cdot x + 1}$	

comDenom(*Expr1*,*Var*) returns a reduced ratio of numerator and denominator expanded with respect to *Var*. The terms and their factors are sorted with *Var* as the main variable. Similar powers of *Var* are collected. There might be some incidental factoring of the collected coefficients. Compared to omitting *Var*, this often saves time, memory, and screen space, while making the expression more comprehensible. It also makes subsequent operations on the result faster and less likely to exhaust memory.

If *Var* does not occur in *Expr1*, **comDenom**(*Expr1*,*Var*) returns a reduced ratio of an unexpanded numerator over an unexpanded denominator. Such results usually save even more time, memory, and screen space. Such partially factored results also make subsequent operations on the result much faster and much less likely to exhaust memory.

Even when there is no denominator, the **comden** function is often a fast way to achieve partial factorization if **factor()** is too slow or if it exhausts memory.

Hint: Enter this **comden()** function definition and routinely try it as an alternative to **comDenom()** and **factor()**.

$$\text{comDenom}\left(\frac{y^2+y}{(x+1)^2}+y^2+y,x\right) = \frac{x^2 \cdot y \cdot (y+1) + 2 \cdot x \cdot y \cdot (y+1) + 2 \cdot y \cdot (y+1)}{x^2 + 2 \cdot x + 1}$$

$$\text{comDenom}\left(\frac{y^2+y}{(x+1)^2}+y^2+y,y\right) = \frac{y^2 \cdot (x^2 + 2 \cdot x + 2) + y \cdot (x^2 + 2 \cdot x + 2)}{x^2 + 2 \cdot x + 1}$$

Define *comden*(*exprn*)=**comDenom**(*exprn*,*abc*)
Done

$$\text{comden}\left(\frac{y^2+y}{(x+1)^2}+y^2+y\right) = \frac{(x^2+2 \cdot x+2) \cdot y \cdot (y+1)}{(x+1)^2}$$

$$\text{comden}(1234 \cdot x^2 \cdot (y^3-y) + 2468 \cdot x \cdot (y^2-1)) = \frac{1234 \cdot x \cdot (x \cdot y + 2) \cdot (y^2-1)}{1}$$

completeSquare ()

completeSquare(*ExprOrEqn*,*Var*) ⇒
expression or equation

completeSquare(*ExprOrEqn*,*Var*^{*Power*}) ⇒
expression or equation

completeSquare(*ExprOrEqn*,*Var1*,*Var2*
[...]) ⇒ *expression or equation*

completeSquare(*ExprOrEqn*,{*Var1*,*Var2*
[...]}) ⇒ *expression or equation*

Converts a quadratic polynomial expression of the form $a \cdot x^2 + b \cdot x + c$ into the form $a \cdot (x-h)^2 + k$

$$\text{completeSquare}(x^2+2 \cdot x+3,x) = (x+1)^2+2$$

$$\text{completeSquare}(x^2+2 \cdot x=3,x) = (x+1)^2=4$$

$$\text{completeSquare}(x^6+2 \cdot x^3+3,x^3) = (x^3+1)^2+2$$

$$\text{completeSquare}(x^2+4 \cdot x+y^2+6 \cdot y+3=0,x,y) = (x+2)^2+(y+3)^2=10$$

completeSquare ()

Catalog > 

- or -

Converts a quadratic equation of the form $a \cdot x^2 + b \cdot x + c = d$ into the form $a \cdot (x-h)^2 = k$

$$\text{completeSquare}\left(3 \cdot x^2 + 2 \cdot y + 7 \cdot y^2 + 4 \cdot x = 3, \{x, y\}\right)$$

$$3 \cdot \left(x + \frac{2}{3}\right)^2 + 7 \cdot \left(y + \frac{1}{7}\right)^2 = \frac{94}{21}$$

The first argument must be a quadratic expression or equation in standard form with respect to the second argument.

$$\text{completeSquare}\left(x^2 + 2 \cdot x \cdot y, x, y\right) \quad (x+y)^2 - y^2$$

The Second argument must be a single univariate term or a single univariate term raised to a rational power, for example x , y^2 , or $z^{1/3}$.

The third and fourth syntax attempt to complete the square with respect to variables $Var1$, $Var2$ [...]).

conj()

Catalog > 

$\text{conj}(Expr1) \Rightarrow expression$

$\text{conj}(List1) \Rightarrow list$

$\text{conj}(Matrix1) \Rightarrow matrix$

Returns the complex conjugate of the argument.

$$\text{conj}(1+2 \cdot i) \quad 1-2 \cdot i$$

$$\text{conj}\left(\begin{bmatrix} 2 & 1-3 \cdot i \\ -i & -7 \end{bmatrix}\right) \quad \begin{bmatrix} 2 & 1+3 \cdot i \\ i & -7 \end{bmatrix}$$

$$\text{conj}(z) \quad z$$

$$\text{conj}(x+i \cdot y) \quad x-y \cdot i$$

Note: All undefined variables are treated as real variables.

constructMat()

Catalog > 

$\text{constructMat}(Expr, Var1, Var2, numRows, numCols) \Rightarrow matrix$

$$\text{constructMat}\left(\frac{1}{i+j}, i, j, 3, 4\right) \quad \begin{bmatrix} \frac{1}{1} & \frac{1}{1} & \frac{1}{1} & \frac{1}{1} \\ 2 & 3 & 4 & 5 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Returns a matrix based on the arguments.

Expr is an expression in variables *Var1* and *Var2*. Elements in the resulting matrix are formed by evaluating *Expr* for each incremented value of *Var1* and *Var2*.

Var1 is automatically incremented from **1** through *numRows*. Within each row, *Var2* is incremented from **1** through *numCols*.

CopyVar

Catalog >

CopyVar *Var1*, *Var2*Define $a(x) = \frac{1}{x}$ Done**CopyVar** *Var1*., *Var2*.Define $b(x) = x^2$ Done**CopyVar** *Var1*, *Var2* copies the value of variable *Var1* to variable *Var2*, creating *Var2* if necessary. Variable *Var1* must have a value.CopyVar *a,c*: $c(4)$ $\frac{1}{4}$ CopyVar *b,c*: $c(4)$ 16

If *Var1* is the name of an existing user-defined function, copies the definition of that function to function *Var2*. Function *Var1* must be defined.

Var1 must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.

CopyVar *Var1*., *Var2*. copies all members of the *Var1*. variable group to the *Var2*. group, creating *Var2*. if necessary.

aa.a:=45 45*aa.b*:=6.78 6.78CopyVar *aa*.,*bb*. Done

Var1. must be the name of an existing variable group, such as the statistics *stat.nn* results, or variables created using the **LibShortcut()** function. If *Var2*. already exists, this command replaces all members that are common to both groups and adds the members that do not already exist. If one or more members of *Var2*. are locked, all members of *Var2*. are left unchanged.

getVarInfo()

<i>aa.a</i>	"NUM"	"	0
<i>aa.b</i>	"NUM"	"	0
<i>bb.a</i>	"NUM"	"	0
<i>bb.b</i>	"NUM"	"	0

corrMat()

Catalog >

corrMat(*List1*,*List2*[,...[,*List20*]])Computes the correlation matrix for the augmented matrix [*List1*, *List2*, ..., *List20*].**► cos**

Catalog >

Expr ► **cos**

Note: You can insert this operator from the computer keyboard by typing @>**cos**.

 $(\sin(x))^2$ ► **cos** $1 - (\cos(x))^2$

Represents *Expr* in terms of cosine. This is a display conversion operator. It can be used only at the end of the entry line.

► **cos** reduces all powers of $\sin(\dots)$ modulo $1 - \cos(\dots)^2$

so that any remaining powers of $\cos(\dots)$ have exponents in the range $(0, 2)$. Thus, the result will be free of $\sin(\dots)$ if and only if $\sin(\dots)$ occurs in the given expression only to even powers.

Note: This conversion operator is not supported in Degree or Gradian Angle modes. Before using it, make sure that the Angle mode is set to Radians and that *Expr* does not contain explicit references to degree or gradian angles.

cos() key

cos(*Expr1*) \Rightarrow *expression*

In Degree angle mode:

$$\cos\left(\frac{\pi}{4}\right) \quad \frac{\sqrt{2}}{2}$$

$$\cos(45) \quad \frac{\sqrt{2}}{2}$$

cos(*List1*) \Rightarrow *list*

cos(*Expr1*) returns the cosine of the argument as an expression.

cos(*List1*) returns a list of the cosines of all elements in *List1*.

$$\cos(\{0, 60, 90\}) \quad \left\{1, \frac{1}{2}, 0\right\}$$

Note: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode temporarily.

In Gradian angle mode:

$$\cos(\{0, 50, 100\}) \quad \left\{1, \frac{\sqrt{2}}{2}, 0\right\}$$

In Radian angle mode:

$$\cos\left(\frac{\pi}{4}\right) \quad \frac{\sqrt{2}}{2}$$

$$\cos(45^{\circ}) \quad \frac{\sqrt{2}}{2}$$

cos(*squareMatrix1*) \Rightarrow *squareMatrix*

In Radian angle mode:

Returns the matrix cosine of *squareMatrix1*. This is not the same as calculating the cosine of each element.

cos()

When a scalar function $f(A)$ operates on *squareMatrix1* (A), the result is calculated by the algorithm:

$$\cos \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$$

Compute the eigenvalues (λ_i) and eigenvectors (V_i) of A .

$$\begin{bmatrix} 0.212493 & 0.205064 & 0.121389 \\ 0.160871 & 0.259042 & 0.037126 \\ 0.248079 & -0.090153 & 0.218972 \end{bmatrix}$$

squareMatrix1 must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

Form the matrices:

$$B = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \text{ and } X = [V_1, V_2, \dots, V_n]$$

Then $A = X B X^{-1}$ and $f(A) = X f(B) X^{-1}$. For example, $\cos(A) = X \cos(B) X^{-1}$ where:

$\cos(B) =$

$$\begin{bmatrix} \cos(\lambda_1) & 0 & \dots & 0 \\ 0 & \cos(\lambda_2) & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \cos(\lambda_n) \end{bmatrix}$$

All computations are performed using floating-point arithmetic.

cos-1()

$\cos^{-1}(Expr1) \Rightarrow expression$

In Degree angle mode:

$\cos^{-1}(List1) \Rightarrow list$

$$\cos^{-1}(1) \quad 0$$

$\cos^{-1}(Expr1)$ returns the angle whose cosine is $Expr1$ as an expression.

In Gradian angle mode:

$\cos^{-1}(List1)$ returns a list of the inverse cosines of each element of $List1$.

$$\cos^{-1}(0) \quad 100$$

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

In Radian angle mode:

Note: You can insert this function from the keyboard by typing `arccos (...)`.

$$\cos^{-1}(\{0,0.2,0.5\}) \quad \left\{ \frac{\pi}{2}, 1.36944, 1.0472 \right\}$$

cos-1()



cos-1(squareMatrix1) ⇒ *squareMatrix*

Returns the matrix inverse cosine of *squareMatrix1*. This is not the same as calculating the inverse cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular Complex Format:

$$\cos^{-1}\begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$$

$$\begin{bmatrix} 1.73485+0.064606\cdot i & -1.49086+2.10514 \\ -0.725533+1.51594\cdot i & 0.623491+0.77836\phi \\ -2.08316+2.63205\cdot i & 1.79018-1.27182\cdot \end{bmatrix}$$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

cosh()

Catalog >

cosh(Expr1) ⇒ *expression*

cosh(List1) ⇒ *list*

cosh(Expr1) returns the hyperbolic cosine of the argument as an expression.

cosh(List1) returns a list of the hyperbolic cosines of each element of *List1*.

cosh(squareMatrix1) ⇒ *squareMatrix*

Returns the matrix hyperbolic cosine of *squareMatrix1*. This is not the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Degree angle mode:

$$\cosh\left(\left(\frac{\pi}{4}\right)r\right) \qquad \cosh(45)$$

In Radian angle mode:

$$\cosh\begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$$

$$\begin{bmatrix} 421.255 & 253.909 & 216.905 \\ 327.635 & 255.301 & 202.958 \\ 226.297 & 216.623 & 167.628 \end{bmatrix}$$

cosh-1()

Catalog >

cosh-1(Expr1) ⇒ *expression*

cosh-1(List1) ⇒ *list*

cosh-1(Expr1) returns the inverse hyperbolic cosine of the argument as an expression.

new screenshots format (see Z_WriterNotes)

$$\cosh^{-1}(1) \qquad 0$$

$$\cosh^{-1}(\{1,2,1,3\}) \qquad \{0,1.37286,\cosh^{-1}(3)\}$$

cosh-1(List1) returns a list of the inverse hyperbolic cosines of each element of *List1*.

Note: You can insert this function from the keyboard by typing **arccosh (...)**.

cosh-1(squareMatrix1) ⇒ squareMatrix

Returns the matrix inverse hyperbolic cosine of *squareMatrix1*. This is not the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and In Rectangular Complex Format:

$$\cosh^{-1}\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$$

$$\begin{bmatrix} 2.52503+1.73485\cdot i & -0.009241-1.49086\cdot i \\ 0.486969-0.725533\cdot i & 1.66262+0.623491\cdot i \\ -0.322354-2.08316\cdot i & 1.26707+1.79018\cdot i \end{bmatrix}$$

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

cot() **key**

cot(Expr1) ⇒ expression

In Degree angle mode:

$$\cot(45) \quad 1$$

cot(List1) ⇒ list

Returns the cotangent of *Expr1* or returns a list of the cotangents of all elements in *List1*.

In Gradian angle mode:

$$\cot(50) \quad 1$$

Note: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use **°**, **G**, or **r** to override the angle mode temporarily.

In Radian angle mode:

$$\cot(\{1,2,1,3\}) \quad \left\{ \frac{1}{\tan(1)}, 0.584848, \frac{1}{\tan(3)} \right\}$$

cot⁻¹() **key**

cot⁻¹(Expr1) ⇒ expression

In Degree angle mode:

$$\cot^{-1}(1) \quad 45.$$

cot⁻¹(List1) ⇒ list

Returns the angle whose cotangent is *Expr1* or returns a list containing the inverse cotangents of each element of *List1*.

In Gradian angle mode:

$$\cot^{-1}(1) \quad 50.$$

cot⁻¹()

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

In Radian angle mode:

$\cot^{-1}(1)$	$\frac{\pi}{4}$
----------------	-----------------

Note: You can insert this function from the keyboard by typing **arccot (...)** .

coth()

coth(*Expr1*) \Rightarrow *expression*

$\coth(1.2)$	1.19954
--------------	---------

coth(*List1*) \Rightarrow *list*

$\coth(\{1,3.2\})$	$\left\{ \frac{1}{\tanh(1)}, 1.00333 \right\}$
--------------------	--

Returns the hyperbolic cotangent of *Expr1* or returns a list of the hyperbolic cotangents of all elements of *List1*.

coth-1()

coth-1(*Expr1*) \Rightarrow *expression*

$\coth^{-1}(3.5)$	0.293893
-------------------	----------

coth-1(*List1*) \Rightarrow *list*

$\coth^{-1}(\{-2.2,1.6\})$	$\left\{ \frac{-\ln(3)}{2}, 0.518046, \frac{\ln\left(\frac{7}{5}\right)}{2} \right\}$
----------------------------	---

Returns the inverse hyperbolic cotangent of *Expr1* or returns a list containing the inverse hyperbolic cotangents of each element of *List1*.

Note: You can insert this function from the keyboard by typing **arccoth (...)** .

count()

count(*Value1orList1* [, *Value2orList2* [, ...]]) \Rightarrow *value*

$\text{count}(2,4,6)$	3
-----------------------	---

Returns the accumulated count of all elements in the arguments that evaluate to numeric values.

$\text{count}(\{2,4,6\})$	3
---------------------------	---

Each argument can be an expression, value, list, or matrix. You can mix data types and use arguments of various dimensions.

$\text{count}\left(2, \{4,6\}, \begin{bmatrix} 8 & 10 \\ 12 & 14 \end{bmatrix}\right)$	7
--	---

For a list, matrix, or range of cells, each element is evaluated to determine if it should be included in the count.

$\text{count}\left(\frac{1}{2}, 3+4\cdot i, \text{undef}, \text{"hello"}, x+5, \text{sign}(0)\right)$	2
---	---

In the last example, only $1/2$ and $3+4\cdot i$ are counted. The remaining arguments, assuming *x* is undefined, do not evaluate to numeric values.

Within the Lists & Spreadsheet application, you can use a range of cells in place of any argument.

Empty (void) elements are ignored. For more information on empty elements, see page 251.

countif()

countif(List, Criteria) ⇒ value

Returns the accumulated count of all elements in *List* that meet the specified *Criteria*.

Criteria can be:

- A value, expression, or string. For example, **3** counts only those elements in *List* that simplify to the value 3.
- A Boolean expression containing the symbol ? as a placeholder for each element. For example, **?<5** counts only those elements in *List* that are less than 5.

Within the Lists & Spreadsheet application, you can use a range of cells in place of *List*.

Empty (void) elements in the list are ignored. For more information on empty elements, see page 251.

Note: See also **sumif()**, page 180, and **frequency()**, page 74.

countIf({1,3,"abc",undef,3,1},3) 2

Counts the number of elements equal to 3.

countIf({"abc","def","abc",3},"def") 1

Counts the number of elements equal to "def."

countIf({x^-2,x^-1,1,x,x^2},x) 1

Counts the number of elements equal to x ; this example assumes the variable x is undefined.

countIf({1,3,5,7,9},?<5) 2

Counts 1 and 3.

countIf({1,3,5,7,9},2<?<8) 3

Counts 3, 5, and 7.

countIf({1,3,5,7,9},?<4 or ?>6) 4

Counts 1, 3, 7, and 9.

cPolyRoots()

Catalog >

cPolyRoots(*Poly*,*Var*) ⇒ *list*

$$\frac{\text{polyRoots}(y^3+1,y)}{\quad} \quad \{-1\}$$

cPolyRoots(*ListOfCoeffs*) ⇒ *list*

$$\frac{\text{cPolyRoots}(y^3+1,y)}{\quad} \quad \left\{-1, \frac{1}{2} - \frac{\sqrt{3}}{2}i, \frac{1}{2} + \frac{\sqrt{3}}{2}i\right\}$$

The first syntax, **cPolyRoots**(*Poly*,*Var*), returns a list of complex roots of polynomial *Poly* with respect to variable *Var*.

$$\frac{\text{polyRoots}(x^2+2\cdot x+1,x)}{\quad} \quad \{-1,-1\}$$

Poly must be a polynomial in one variable.

$$\frac{\text{cPolyRoots}(\{1,2,1\})}{\quad} \quad \{-1,-1\}$$

The second syntax, **cPolyRoots**(*ListOfCoeffs*), returns a list of complex roots for the coefficients in *ListOfCoeffs*.

Note: See also **polyRoots()**, page 136.

crossP()

Catalog >

crossP(*List1*,*List2*) ⇒ *list*

$$\frac{\text{crossP}(\{a1,b1\},\{a2,b2\})}{\quad} \quad \{0,0,a1\cdot b2-a2\cdot b1\}$$

Returns the cross product of *List1* and *List2* as a list.

$$\frac{\text{crossP}(\{0.1,2.2,-5\},\{1,-0.5,0\})}{\quad} \quad \{-2.5,-5,-2.25\}$$

List1 and *List2* must have equal dimension, and the dimension must be either 2 or 3.

crossP(*Vector1*,*Vector2*) ⇒ *vector*

$$\frac{\text{crossP}(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} -3 & 6 & -3 \\ 0 & 0 & -2 \end{bmatrix})}{\quad} \quad \begin{bmatrix} -3 & 6 & -3 \\ 0 & 0 & -2 \end{bmatrix}$$

Returns a row or column vector (depending on the arguments) that is the cross product of *Vector1* and *Vector2*.

Both *Vector1* and *Vector2* must be row vectors, or both must be column vectors. Both vectors must have equal dimension, and the dimension must be either 2 or 3.

csc() **key****csc**(*Expr1*) ⇒ *expression*

In Degree angle mode:

$$\frac{\text{csc}(45)}{\quad} \quad \sqrt{2}$$

csc(*List1*) ⇒ *list*

In Gradian angle mode:

Returns the cosecant of *Expr1* or returns a list containing the cosecants of all elements in *List1*.

$$\frac{\text{csc}(50)}{\quad} \quad \sqrt{2}$$

csc() **key**

In Radian angle mode:

$$\text{csc}\left(\left\{1, \frac{\pi}{2}, \frac{\pi}{3}\right\}\right) \quad \left\{\frac{1}{\sin(1)}, 1, \frac{2\sqrt{3}}{3}\right\}$$

csc-1() **key****csc-1**(*Expr1*) \Rightarrow *expression*

In Degree angle mode:

$$\text{csc}^{-1}(1) \quad 90.$$

csc-1(*List1*) \Rightarrow *list*

In Gradian angle mode:

$$\text{csc}^{-1}(1) \quad 100.$$

Returns the angle whose cosecant is *Expr1* or returns a list containing the inverse cosecants of each element of *List1*.**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.**Note:** You can insert this function from the keyboard by typing **arccsc**(...).

In Radian angle mode:

$$\text{csc}^{-1}(\{1, 4, 6\}) \quad \left\{\frac{\pi}{2}, \sin^{-1}\left(\frac{1}{4}\right), \sin^{-1}\left(\frac{1}{6}\right)\right\}$$

csch()**Catalog** > **csch**(*Expr1*) \Rightarrow *expression*

$$\text{csch}(3) \quad \frac{1}{\sinh(3)}$$

csch(*List1*) \Rightarrow *list*

$$\text{csch}(\{1, 2, 1, 4\}) \quad \left\{\frac{1}{\sinh(1)}, 0.248641, \frac{1}{\sinh(4)}\right\}$$

Returns the hyperbolic cosecant of *Expr1* or returns a list of the hyperbolic cosecants of all elements of *List1*.**csch-1()****Catalog** > **csch-1**(*Expr1*) \Rightarrow *expression*

$$\text{csch}^{-1}(1) \quad \sinh^{-1}(1)$$

csch-1(*List1*) \Rightarrow *list*

$$\text{csch}^{-1}(\{1, 2, 1, 3\}) \quad \left\{\sinh^{-1}(1), 0.459815, \sinh^{-1}\left(\frac{1}{3}\right)\right\}$$

Returns the inverse hyperbolic cosecant of *Expr1* or returns a list containing the inverse hyperbolic cosecants of each element of *List1*.**Note:** You can insert this function from the keyboard by typing **arccsch**(...).

cSolve(Equation, Var) ⇒ Boolean expression

cSolve(Equation, Var=Guess) ⇒ Boolean expression

cSolve(Inequality, Var) ⇒ Boolean expression

$\text{cSolve}(x^3=-1,x)$	
$x=\frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i$ or $x=\frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i$ or $x=-1$	
$\text{solve}(x^3=-1,x)$	$x=-1$

Returns candidate complex solutions of an equation or inequality for *Var*. The goal is to produce candidates for all real and non-real solutions. Even if *Equation* is real, **cSolve()** allows non-real results in Real result Complex Format.

Although all undefined variables that do not end with an underscore () are processed as if they were real, **cSolve()** can solve polynomial equations for complex solutions.

cSolve() temporarily sets the domain to complex during the solution even if the current domain is real. In the complex domain, fractional powers having odd denominators use the principal rather than the real branch. Consequently, solutions from **solve()** to equations involving such fractional powers are not necessarily a subset of those from **cSolve()**.

cSolve() starts with exact symbolic methods. **cSolve()** also uses iterative approximate complex polynomial factoring, if necessary.

Note: See also **cZeros()**, **solve()**, and **zeros()**.

$\text{cSolve}\left(x^{\frac{1}{3}}=-1,x\right)$	false
$\text{solve}\left(x^{\frac{1}{3}}=-1,x\right)$	$x=-1$

In Display Digits mode of Fix 2:

$\text{exact}\left(\text{cSolve}\left(x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3=0,x\right)\right)$	
$x\cdot\left(x^4+4\cdot x^3+5\cdot x^2-6\right)=3$	
$\text{cSolve}\left(\text{Ans},x\right)$	
$x=-1.11+1.07\cdot i$ or $x=-1.11-1.07\cdot i$ or $x=-2.9$	

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

cSolve(Eqn1 and Eqn2 [and...], VarOrGuess1, VarOrGuess2 [, ...]) ⇒ Boolean expression

cSolve(*SystemOfEqns*, *VarOrGuess1*,
VarOrGuess2 [, ...]) ⇒
Boolean expression

Returns candidate complex solutions to the simultaneous algebraic equations, where each *varOrGuess* specifies a variable that you want to solve for.

Optionally, you can specify an initial guess for a variable. Each *varOrGuess* must have the form:

variable
– or –
variable = *real or non-real number*

For example, x is valid and so is $x=3+i$.

If all of the equations are polynomials and if you do NOT specify any initial guesses, **cSolve()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** complex solutions.

Complex solutions can include both real and non-real solutions, as in the example to the right.

$$\text{cSolve}(u \cdot v - u = v \text{ and } v^2 = -u, \{u, v\})$$

$$u = \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i \text{ and } v = \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i \text{ or } u = \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i$$

To see the entire result,
press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

Simultaneous polynomial equations can have extra variables that have no values, but represent given numeric values that could be substituted later.

$$\text{cSolve}(u \cdot v - u = c \cdot v \text{ and } v^2 = -u, \{u, v\})$$

$$u = \frac{-(\sqrt{4 \cdot c - 1} \cdot i + 1)^2}{4} \text{ and } v = \frac{\sqrt{4 \cdot c - 1} \cdot i + 1}{2}$$

To see the entire result,
press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

You can also include solution variables that do not appear in the equations. These solutions show how families of solutions might contain arbitrary constants of the form ck , where k is an integer suffix from 1 through 255.

$$\text{cSolve}(u \cdot v - u = v \text{ and } v^2 = -u, \{u, v, w\})$$

$$u = \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i \text{ and } v = \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i \text{ and } w = c\mathbf{d3} \text{ or } \blacktriangleright$$

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list solution variables. If your initial choice exhausts memory or your patience, try rearranging the variables in the equations and/or *varOrGuess* list.

If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in all solution variables, **cSolve()** uses Gaussian elimination to attempt to determine all solutions.

If a system is neither polynomial in all of its variables nor linear in its solution variables, **cSolve()** determines at most one solution using an approximate iterative method. To do so, the number of solution variables must equal the number of equations, and all other variables in the equations must simplify to numbers.

A non-real guess is often necessary to determine a non-real solution. For convergence, a guess might have to be rather close to a solution.

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

$$\text{cSolve}(u+v=e^w \text{ and } u-v=i, \{u,v\})$$

$$u = \frac{e^w + i}{2} \text{ and } v = \frac{e^w - i}{2}$$

$$\text{cSolve}(e^z = w \text{ and } w = z^2, \{w,z\})$$

$$w = 0.494866 \text{ and } z = 0.703467$$

$$\text{cSolve}(e^z = w \text{ and } w = z^2, \{w,z=1+i\})$$

$$w = 0.149606 + 4.8919 \cdot i \text{ and } z = 1.58805 + 1.5402 \cdot i$$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

CubicReg

CubicReg $X, Y[, [Freq] [, Category, Include]]$

Computes the cubic polynomial regression $y = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and Y are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot x^3 + b \cdot x^2 + c \cdot x + d$
stat.a, stat.b, stat.c, stat.d	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

cumulativeSum()

cumulativeSum(List1) \Rightarrow list

cumulativeSum{ {1,2,3,4} } {1,3,6,10}

Returns a list of the cumulative sums of the elements in *List1*, starting at element 1.

cumulativeSum()

Catalog >

cumulativeSum(*Matrix1*) ⇒ *matrix*

Returns a matrix of the cumulative sums of the elements in *Matrix1*. Each element is the cumulative sum of the column from top to bottom.

An empty (void) element in *List1* or *Matrix1* produces a void element in the resulting list or matrix. For more information on empty elements, see page 251.

1 2	→ <i>m1</i>	1 2
3 4		3 4
5 6		5 6
cumulativeSum(<i>m1</i>)		1 2 4 6 9 12

Cycle

Catalog >

Cycle

Transfers control immediately to the next iteration of the current loop (**For**, **While**, or **Loop**).

Cycle is not allowed outside the three looping structures (**For**, **While**, or **Loop**).

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Function listing that sums the integers from 1 to 100 skipping 50.

```
Define g()=Func                                Done
  Local temp,i
  0→temp
  For i,1,100,1
  If i=50
  Cycle
  temp+i→temp
  EndFor
  Return temp
  EndFunc
```

g() 5000

► Cylind

Catalog >

Vector ► **Cylind**

Note: You can insert this operator from the computer keyboard by typing @>**Cylind**.

Displays the row or column vector in cylindrical form [*r*, ∠ *θ*, *z*].

Vector must have exactly three elements. It can be either a row or a column.

[2 2 3]►Cylind	$2\sqrt{2} \angle \frac{\pi}{4} 3$
----------------	------------------------------------

cZeros()

cZeros(*Expr*, *Var*) \Rightarrow *list*

Returns a list of candidate real and non-real values of *Var* that make *Expr*=0. **cZeros()** does this by computing

exp ► **list**(**cSolve**(*Expr*=0,*Var*),*Var*).

Otherwise, **cZeros()** is similar to **zeros()**.

Note: See also **cSolve()**, **solve()**, and **zeros()**.

cZeros({*Expr*1, *Expr*2[, ...] },
{*VarOrGuess*1,*VarOrGuess*2[, ...] })
 \Rightarrow *matrix*

Returns candidate positions where the expressions are zero simultaneously. Each *VarOrGuess* specifies an unknown whose value you seek.

Optionally, you can specify an initial guess for a variable. Each *VarOrGuess* must have the form:

variable

– or –

variable = *real or non-real number*

For example, x is valid and so is x=3+i.

If all of the expressions are polynomials and you do NOT specify any initial guesses, **cZeros()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine **all** complex zeros.

Complex zeros can include both real and non-real zeros, as in the example to the right.

Each row of the resulting matrix represents an alternate zero, with the components ordered the same as the *VarOrGuess* list. To extract a row, index the matrix by [*row*].

In Display Digits mode of Fix 3:

$$\text{cZeros}(x^5+4 \cdot x^4+5 \cdot x^3-6 \cdot x-3, x) \\ \{-1.114+1.073 \cdot i, -1.114-1.073 \cdot i, -2.125, -0.612, 0\}$$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

$$\text{cZeros}(\{u \cdot v - u - v, v^2 + u\}, \{u, v\}) \\ \begin{bmatrix} \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i & \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i \\ \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i & \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i \end{bmatrix}$$

Extract row 2:

$$\text{Ans}[2] \quad \left[\frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i \quad \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i \right]$$

cZeros()

Catalog > 

Simultaneous polynomials can have extra variables that have no values, but represent given numeric values that could be substituted later.

$$\text{cZeros}\left(\left\{u \cdot v - u - c \cdot v^2, v^2 + u\right\}, \{u, v\}\right)$$
$$\begin{bmatrix} 0 & 0 \\ -(c-1)^2 & -(c-1) \end{bmatrix}$$

You can also include unknown variables that do not appear in the expressions. These zeros show how families of zeros might contain arbitrary constants of the form ck , where k is an integer suffix from 1 through 255.

$$\text{cZeros}\left(\left\{u \cdot v - u - v, v^2 + u\right\}, \{u, v, w\}\right)$$
$$\text{cZero}\left(\left\{u \cdot (v-1) - v, u + v^2\right\}, \{u, v, w\}\right)$$
$$\begin{bmatrix} 0 & 0 & c4 \\ \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i & \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i & c4 \\ \frac{1}{2} + \frac{\sqrt{3}}{2} \cdot i & \frac{1}{2} - \frac{\sqrt{3}}{2} \cdot i & c4 \end{bmatrix}$$

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list unknowns. If your initial choice exhausts memory or your patience, try rearranging the variables in the expressions and/or *VarOrGuess* list.

If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in all unknowns, **cZeros()** uses Gaussian elimination to attempt to determine all zeros.

$$\text{cZeros}\left(\left\{u + v - e^{w}, u - v - i\right\}, \{u, v\}\right)$$
$$\begin{bmatrix} e^{w+i} & e^{w-i} \\ 2 & 2 \end{bmatrix}$$

If a system is neither polynomial in all of its variables nor linear in its unknowns, **cZeros()** determines at most one zero using an approximate iterative method. To do so, the number of unknowns must equal the number of expressions, and all other variables in the expressions must simplify to numbers.

$$\text{cZeros}_x\left(\left\{e^z - w, w - z^2\right\}, \{w, z\}\right)$$
$$[0.494866 \quad -0.703467]$$

A non-real guess is often necessary to determine a non-real zero. For convergence, a guess might have to be rather close to a zero.

$$\text{cZeros}\left(\left\{e \cdot z - w, w - z^2\right\}, \{w, z = 1 + i\}\right)$$
$$[0.149606 + 4.8919 \cdot i \quad 1.58805 + 1.54022 \cdot i]$$

D

dbd()

Catalog > 

dbd(date1, date2) ⇒ *value*

Returns the number of days between *date1* and *date2* using the actual-day-count method.

dbd(12.3103,1.0104)	1
dbd(1.0107,6.0107)	151
dbd(3112.03,101.04)	1
dbd(101.07,106.07)	151

date1 and *date2* can be numbers or lists of numbers within the range of the dates on the standard calendar. If both *date1* and *date2* are lists, they must be the same length.

date1 and *date2* must be between the years 1950 through 2049.

You can enter the dates in either of two formats. The decimal placement differentiates between the date formats.

MM.DDYY (format used commonly in the United States)

DDMM.YY (format use commonly in Europe)

▶ DD

Expr1 ▶ DD ⇒ *valueList1*

▶ DD ⇒ *listMatrix1*

▶ DD ⇒ *matrix*

Note: You can insert this operator from the computer keyboard by typing @>DD.

Returns the decimal equivalent of the argument expressed in degrees. The argument is a number, list, or matrix that is interpreted by the Angle mode setting in gradians, radians or degrees.

In Degree angle mode:

(1.5°) ▶DD	1.5°
$(45^\circ 22' 14.3")$ ▶DD	45.3706°
$(\{45^\circ 22' 14.3", 60^\circ 0' 0"\})$ ▶DD	{45.3706°, 60°}

In Gradian angle mode:

1▶DD	$\frac{9}{10}^\circ$
------	----------------------

In Radian angle mode:

(1.5) ▶DD	85.9437°
-------------	----------

▶ Decimal

Expression1 ▶ Decimal ⇒ *expression*

List1 ▶ Decimal ⇒ *expression*

Matrix1 ▶ Decimal ⇒ *expression*

Note: You can insert this operator from the computer keyboard by typing @>Decimal.

$\frac{1}{3}$ ▶Decimal	0.333333
------------------------	----------

Displays the argument in decimal form.
This operator can be used only at the end of the entry line.

Define

Define *Var* = *Expression*

Define *Function*(*Param1*, *Param2*, ...) = *Expression*

Defines the variable *Var* or the user-defined function *Function*.

Parameters, such as *Param1*, provide placeholders for passing arguments to the function. When calling a user-defined function, you must supply arguments (for example, values or variables) that correspond to the parameters. When called, the function evaluates *Expression* using the supplied arguments.

Var and *Function* cannot be the name of a system variable or built-in function or command.

Note: This form of **Define** is equivalent to executing the expression: *expression* → *Function*(*Param1*,*Param2*).

Define *Function*(*Param1*, *Param2*, ...) = **Func**

Block

EndFunc

Define *Program*(*Param1*, *Param2*, ...) = **Prgm**

Block

EndPrgm

In this form, the user-defined function or program can execute a block of multiple statements.

Block can be either a single statement or a series of statements on separate lines. *Block* also can include expressions and instructions (such as **If**, **Then**, **Else**, and **For**).

Define $g(x,y)=2 \cdot x-3 \cdot y$	Done
$g(1,2)$	-4
$1 \rightarrow a: 2 \rightarrow b: g(a,b)$	-4
Define $h(x)=\text{when}(x<2,2 \cdot x-3,-2 \cdot x+3)$	Done
$h(-3)$	-9
$h(4)$	-5

Define $g(x,y)=\text{Func}$	Done
If $x>y$ Then	
Return x	
Else	
Return y	
EndIf	
EndFunc	
$g(3,-7)$	3

Define

Catalog > 

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Note: See also **Define LibPriv**, page 47, and **Define LibPub**, page 47.

```
Define g(x,y)=Prgm
  If x>y Then
    Disp x," greater than ",y
  Else
    Disp x," not greater than ",y
  EndIf
EndPrgm
```

Done

g(3,-7)

3 greater than -7

Done

Define LibPriv

Catalog > 

Define LibPriv *Var = Expression*
Define LibPriv *Function(Param1, Param2, ...)* = *Expression*

Define LibPriv *Function(Param1, Param2, ...)* = **Func**
Block
EndFunc

Define LibPriv *Program(Param1, Param2, ...)* = **Prgm**
Block
EndPrgm

Operates the same as **Define**, except defines a private library variable, function, or program. Private functions and programs do not appear in the Catalog.

Note: See also **Define**, page 46, and **Define LibPub**, page 47.

Define LibPub

Catalog > 

Define LibPub *Var = Expression*
Define LibPub *Function(Param1, Param2, ...)* = *Expression*

Define LibPub *Function(Param1, Param2, ...)* = **Func**
Block
EndFunc

```
Define LibPub Program(Param1, Param2,
...) = Prgm
    Block
EndPrgm
```

Operates the same as **Define**, except defines a public library variable, function, or program. Public functions and programs appear in the Catalog after the library has been saved and refreshed.

Note: See also **Define**, page 46, and **Define LibPriv**, page 47.

deltaList()See **ΔList()**, page 103.**deltaTmpCnv()**See **ΔtmpCnv()**, page 189.**DelVar**

```
DelVar Var1[, Var2] [, Var3] ...
```

$2 \rightarrow a$	2
-------------------	---

```
DelVar Var.
```

$(a+2)^2$	16
-----------	----

Deletes the specified variable or variable group from memory.

DelVar <i>a</i>	Done
-----------------	------

$(a+2)^2$	$(a+2)^2$
-----------	-----------

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See **unLock**, page 197.

DelVar *Var.* deletes all members of the *Var.* variable group (such as the statistics *stat.nn* results or variables created using the **LibShortcut()** function). The dot (.) in this form of the **DelVar** command limits it to deleting a variable group; the simple variable *Var* is not affected.

<i>aa.a</i> :=45	45
------------------	----

<i>aa.b</i> :=5.67	5.67
--------------------	------

<i>aa.c</i> :=78.9	78.9
--------------------	------

getVarInfo()	<table border="1"> <tr> <td><i>aa.a</i></td> <td>"NUM"</td> <td>"[]"</td> </tr> <tr> <td><i>aa.b</i></td> <td>"NUM"</td> <td>"[]"</td> </tr> <tr> <td><i>aa.c</i></td> <td>"NUM"</td> <td>"[]"</td> </tr> </table>	<i>aa.a</i>	"NUM"	"[]"	<i>aa.b</i>	"NUM"	"[]"	<i>aa.c</i>	"NUM"	"[]"
<i>aa.a</i>	"NUM"	"[]"								
<i>aa.b</i>	"NUM"	"[]"								
<i>aa.c</i>	"NUM"	"[]"								

DelVar <i>aa.</i>	Done
-------------------	------

getVarInfo()	"NONE"
--------------	--------

delVoid()

Catalog >

delVoid(List1) ⇒ *list*

delVoid{1,void,3}

 {1,3}Returns a list that has the contents of *List1* with all empty (void) elements removed.

For more information on empty elements, see page 251.

derivative()See *d()*, page 221.**deSolve()**

Catalog >

deSolve(1stOr2ndOrderODE, Var, depVar) ⇒ *a general solution*

$$\text{deSolve}(y''+2\cdot y'+y=x^2, x, y)$$

$$y=(c3\cdot x+c4)\cdot e^{-x}+x^2-4\cdot x+6$$

$$\text{right}(Ans) \rightarrow temp \quad (c3\cdot x+c4)\cdot e^{-x}+x^2-4\cdot x+6$$

$$\frac{d^2}{dx^2}(temp)+2\cdot \frac{d}{dx}(temp)+temp-x^2 \quad 0$$

$$\text{DelVar } temp \quad Done$$

Returns an equation that explicitly or implicitly specifies a general solution to the 1st- or 2nd-order ordinary differential equation (ODE). In the ODE:

- Use a prime symbol (press) to denote the 1st derivative of the dependent variable with respect to the independent variable.
- Use two prime symbols to denote the corresponding second derivative.

The prime symbol is used for derivatives within **deSolve()** only. In other cases, use **d()**.The general solution of a 1st-order equation contains an arbitrary constant of the form *ck*, where *k* is an integer suffix from 1 through 255. The solution of a 2nd-order equation contains two such constants.Apply **solve()** to an implicit solution if you want to try to convert it to one or more equivalent explicit solutions.

$$\text{deSolve}(y'=(\cos(y))^2, x, x, y) \quad \tan(y)=\frac{x^2}{2}+c4$$

$$\text{solve}(Ans, y) \quad y=\tan^{-1}\left(\frac{x^2+2\cdot c4}{2}\right)+n3\cdot \pi$$

$$Ans|c4=c-1 \text{ and } n3=0 \quad y=\tan^{-1}\left(\frac{x^2+2\cdot (c-1)}{2}\right)$$

When comparing your results with textbook or manual solutions, be aware that different methods introduce arbitrary constants at different points in the calculation, which may produce different general solutions.

deSolve(*1stOrderODE* and *initCond*, *Var*, *depVar*) \Rightarrow a particular solution

Returns a particular solution that satisfies *1stOrderODE* and *initCond*. This is usually easier than determining a general solution, substituting initial values, solving for the arbitrary constant, and then substituting that value into the general solution.

initCond is an equation of the form:

depVar (*initialIndependentValue*) = *initialDependentValue*

The *initialIndependentValue* and *initialDependentValue* can be variables such as *x0* and *y0* that have no stored values. Implicit differentiation can help verify implicit solutions.

deSolve(*2ndOrderODE* and *initCond1* and *initCond2*, *Var*, *depVar*) \Rightarrow particular solution

Returns a particular solution that satisfies *2nd Order ODE* and has a specified value of the dependent variable and its first derivative at one point.

For *initCond1*, use the form:

depVar (*initialIndependentValue*) = *initialDependentValue*

For *initCond2*, use the form:

depVar (*initialIndependentValue*) = *initial1stDerivativeValue*

deSolve(*2ndOrderODE* and *bndCond1* and *bndCond2*, *Var*, *depVar*) \Rightarrow a particular solution

Returns a particular solution that satisfies *2ndOrderODE* and has specified values at two different points.

$\sin(y) = (y \cdot e^x + \cos(y)) \cdot y' \rightarrow ode$	
$\sin(y) = (e^x \cdot y + \cos(y)) \cdot y'$	
$deSolve(ode \text{ and } y(0)=0, x, y) \rightarrow soln$	
$\frac{-(2 \cdot \sin(y) + y^2)}{2} = (e^x - 1) \cdot e^{-x} \cdot \sin(y)$	
$soln x=0 \text{ and } y=0$	true
$ode y' = \text{impDif}(soln, x, y)$	true
$DelVar \text{ } ode, soln$	Done

$$deSolve\left(y'' = y^{\frac{3}{2}} \text{ and } y(0)=0 \text{ and } y'(0)=0, t, y\right)$$

$$\frac{2 \cdot y^{\frac{4}{3}}}{3} = t$$

$$solve\left(\frac{2 \cdot y^{\frac{4}{3}}}{3} = t, y\right)$$

$$y = \frac{\frac{1}{3} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot t^3}{4} \text{ and } t \geq 0$$

$$deSolve(y'' = x \text{ and } y(0)=1 \text{ and } y'(2)=3, x, y)$$

$$y = \frac{x^3}{6} + x + 1$$

$$deSolve(y'' = 2 \cdot y' \text{ and } y(3)=1 \text{ and } y'(4)=2, x, y)$$

$$y = e^{2 \cdot x - 8} \cdot e^{-2 + 1}$$

$$\text{deSolve}\left(w'' - \frac{2 \cdot w'}{x} + \left(9 + \frac{2}{x^2}\right) \cdot w = x \cdot e^x \text{ and } w\left(\frac{\pi}{6}\right) = 0 \text{ and } w\left(\frac{\pi}{3}\right) = 0, x, w\right)$$



$$w = \frac{x \cdot e^x}{(\ln(e))^2 + 9} + \frac{e^3 \cdot x \cdot \cos(3 \cdot x)}{(\ln(e))^2 + 9} - \frac{e^6 \cdot x \cdot \sin(3 \cdot x)}{(\ln(e))^2 + 9}$$

det()

det(*squareMatrix* [, *Tolerance*]) ⇒ *expression*

Returns the determinant of *squareMatrix*.

Optionally, any matrix element is treated as zero if its absolute value is less than *Tolerance*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tolerance* is ignored.

- If you use   or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tolerance* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\text{dim}(\text{squareMatrix})) \cdot \text{rowNorm}(\text{squareMatrix})$

$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right)$	$a \cdot d - b \cdot c$
$\det\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$	-2
$\det\left(\text{identity}(3) - x \cdot \begin{bmatrix} 1 & -2 & 3 \\ -2 & 4 & 1 \\ -6 & -2 & 7 \end{bmatrix}\right)$	$-(98 \cdot x^3 - 55 \cdot x^2 + 12 \cdot x - 1)$
$\begin{bmatrix} 1. \text{E}20 & 1 \\ 0 & 1 \end{bmatrix} \rightarrow \text{mat1}$	$\begin{bmatrix} 1. \text{E}20 & 1 \\ 0 & 1 \end{bmatrix}$
$\det(\text{mat1})$	0
$\det(\text{mat1}, 1)$	1. E20

diag()

diag(*List*) ⇒ *matrix*

diag(*rowMatrix*) ⇒ *matrix*

diag(*columnMatrix*) ⇒ *matrix*

Returns a matrix with the values in the argument list or matrix in its main diagonal.

diag(*squareMatrix*) ⇒ *rowMatrix*

Returns a row matrix containing the elements from the main diagonal of *squareMatrix*.

$\text{diag}([2 \ 4 \ 6])$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$
$\begin{bmatrix} 4 & 6 & 8 \\ 1 & 2 & 3 \\ 5 & 7 & 9 \end{bmatrix}$	$\begin{bmatrix} 4 & 6 & 8 \\ 1 & 2 & 3 \\ 5 & 7 & 9 \end{bmatrix}$
$\text{diag}(\text{Ans})$	$\begin{bmatrix} 4 & 2 & 9 \end{bmatrix}$

diag()

Catalog >

squareMatrix must be square.**dim()**

Catalog >

dim(List) ⇒ *integer*

dim({0,1,2}) 3

Returns the dimension of *List*.**dim(Matrix)** ⇒ *list*dim($\begin{pmatrix} 1 & -1 \\ 2 & -2 \\ 3 & 5 \end{pmatrix}$) {3,2}

Returns the dimensions of matrix as a two-element list {rows, columns}.

dim(String) ⇒ *integer*

dim("Hello") 5

Returns the number of characters contained in character string *String*.

dim("Hello "&"there") 11

Disp

Catalog >

Disp *exprOrString1* [, *exprOrString2*] ...Displays the arguments in the *Calculator* history. The arguments are displayed in succession, with thin spaces as separators.

Useful mainly in programs and functions to ensure the display of intermediate calculations.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

```

Define chars(start,end)=Prgm
  For i,start,end
  Disp i," ",char(i)
  EndFor
EndPrgm
Done
chars(240,243)
-----
240 đ
241 ñ
242 ò
243 ó
-----
Done

```

DispAt

Catalog >

DispAt *int,expr1* [,*expr2* ...] ...

DispAt

DispAt allows you to specify the line where the specified expression or string will be displayed on the screen.**Example**

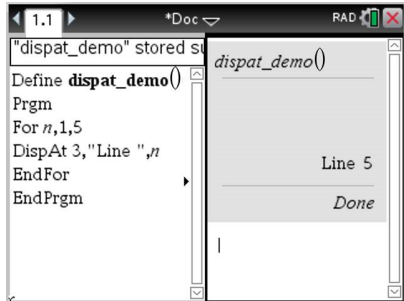
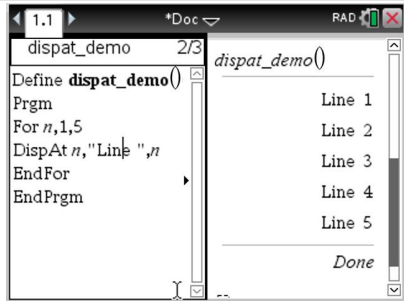
The line number can be specified as an expression.

Please note that the line number is not for the entire screen but for the area immediately following the command/program.

This command allows dashboard-like output from programs where the value of an expression or from a sensor reading is updated on the same line.

DispAt and Disp can be used within the same program.

Note: The maximum number is set to 8 since that matches a screen-full of lines on the handheld screen - as long as the lines don't have 2D math expressions. The exact number of lines depends on the content of the displayed information.



Illustrative examples:

Code	Output
Define z()=	z()
Prgm	
For n,1,3	Iteration 1:
DispAt 1,"N: ",n	Line 1: N:1
Disp "Hello"	Line 2: Hello
EndFor	
EndPrgm	Iteration 2:
	Line 1: N:2
	Line 2: Hello
	Line 3: Hello
	Iteration 3:
	Line 1: N:3
	Line 2: Hello
	Line 3: Hello

	Line 4: Hello
Define z1()=	z1()
Prgm	Line 1: N:3
For n,1,3	Line 2: Hello
DispAt 1,"N: ",n	Line 3: Hello
EndFor	Line 4: Hello
	Line 5: Hello
For n,1,4	
Disp "Hello"	
EndFor	
EndPrgm	

Error conditions:

Error Message	Description
DispAt line number must be between 1 and 8	Expression evaluates the line number outside the range 1-8 (inclusive)
Too few arguments	The function or command is missing one or more arguments.
No arguments	Same as current 'syntax error' dialog
Too many arguments	Limit argument. Same error as Disp.
Invalid data type	First argument must be a number.
Void: DispAt void	"Hello World" Datatype error is thrown for the void (if the callback is defined)
Conversion operator: DispAt 2_ft @> _m, "Hello World"	CAS: Datatype Error is thrown (if the callback is defined) Numeric: Conversion will be evaluated and if the result is a valid argument, DispAt print the string at the result line.

► DMS*Expr* ► DMS

In Degree angle mode:

List ► DMS $\{45.371\}$ ► DMS $45^{\circ}22'15.6''$ *Matrix* ► DMS $\{\{45.371,60\}\}$ ► DMS $\{45^{\circ}22'15.6'',60^{\circ}\}$

Note: You can insert this operator from the computer keyboard by typing @>DMS.

Interprets the argument as an angle and displays the equivalent DMS (DDDDDD°MM'SS.ss") number. See °, ', '' on page 228 for DMS (degree, minutes, seconds) format.

Note: ► DMS will convert from radians to degrees when used in radian mode. If the input is followed by a degree symbol °, no conversion will occur. You can use ► DMS only at the end of an entry line.

domain()

domain(Expr1, Var) ⇒ expression

Returns the domain of *Expr1* with respect to *Var*.

domain() can be used to examine domains of functions. It is restricted to real and finite domain.

This functionality has limitations due to shortcomings of computer algebra simplification and solver algorithms.

Certain functions cannot be used as arguments for **domain()**, regardless of whether they appear explicitly or within user-defined variables and functions. In the following example, the expression cannot be simplified because $\int()$ is a disallowed function.

$$\text{domain}\left(\int \begin{pmatrix} x \\ 1/t \\ 1 \end{pmatrix} dt, x\right) \rightarrow \text{domain}\left(\int \begin{pmatrix} x \\ 1/t \\ 1 \end{pmatrix} dt, x\right)$$

$$\text{domain}\left(\frac{1}{x+y}, y\right) \quad -\infty < y < -x \text{ or } -x < y < \infty$$

$$\text{domain}\left(\frac{x+1}{x^2+2 \cdot x}, x\right) \quad x \neq -2 \text{ and } x \neq 0$$

$$\text{domain}\left(\left(\sqrt{x}\right)^2, x\right) \quad 0 \leq x < \infty$$

$$\text{domain}\left(\frac{1}{x+y}, y\right) \quad -\infty < y < -x \text{ or } -x < y < \infty$$

dominantTerm(Expr1, Var [, Point]) ⇒ expression

dominantTerm(Expr1, Var [, Point]) | Var > Point ⇒ expression

dominantTerm(Expr1, Var [, Point]) | Var < Point ⇒ expression

Returns the dominant term of a power series representation of *Expr1* expanded about *Point*. The dominant term is the one whose magnitude grows most rapidly near *Var = Point*. The resulting power of (*Var - Point*) can have a negative and/or fractional exponent. The coefficient of this power can include logarithms of (*Var - Point*) and other functions of *Var* that are dominated by all powers of (*Var - Point*) having the same exponent sign.

Point defaults to 0. *Point* can be ∞ or -∞, in which cases the dominant term will be the term having the largest exponent of *Var* rather than the smallest exponent of *Var*.

dominantTerm(...) returns “**dominantTerm(...)**” if it is unable to determine such a representation, such as for essential singularities such as $\sin(1/z)$ at $z=0$, $e^{-1/z}$ at $z=0$, or e^z at $z = \infty$ or $-\infty$.

If the series or one of its derivatives has a jump discontinuity at *Point*, the result is likely to contain sub-expressions of the form $\text{sign}(\dots)$ or $\text{abs}(\dots)$ for a real expansion variable or $(-1)^{\text{floor}(\dots \cdot \text{angle}(\dots))}$ for a complex expansion variable, which is one ending with “_”. If you intend to use the dominant term only for values on one side of *Point*, then append to **dominantTerm(...)** the appropriate one of “| *Var > Point*”, “| *Var < Point*”, “| “*Var ≥ Point*”, or “*Var ≤ Point*” to obtain a simpler result.

dominantTerm() distributes over 1st-argument lists and matrices.

$$\text{dominantTerm}(\tan(\sin(x)) - \sin(\tan(x)), x)$$

$$\frac{x^7}{30}$$

$$\text{dominantTerm}\left(\frac{1 - \cos(x-1)}{(x-1)^3}, x, 1\right)$$

$$\frac{1}{2 \cdot (x-1)}$$

$$\text{dominantTerm}\left(x^{-2} \cdot \tan\left(\frac{1}{x^3}\right), x\right)$$

$$\frac{1}{x^3}$$

$$\text{dominantTerm}(\ln(x^x - 1) \cdot x^{-2}, x)$$

$$\frac{\ln(x \cdot \ln(x))}{x^2}$$

$$\text{dominantTerm}\left(e^{\frac{-1}{z}}, z\right)$$

$$\text{dominantTerm}\left(e^{\frac{-1}{z}}, z, 0\right)$$

$$\text{dominantTerm}\left(\left(1 + \frac{1}{n}\right)^n, n, \infty\right)$$

$$e$$

$$\text{dominantTerm}\left(\tan^{-1}\left(\frac{1}{x}\right), x, 0\right)$$

$$\frac{\pi \cdot \text{sign}(x)}{2}$$

$$\text{dominantTerm}\left(\tan^{-1}\left(\frac{1}{x}\right), x, x > 0\right)$$

$$\frac{\pi}{2}$$

dominantTerm() is useful when you want to know the simplest possible expression that is asymptotic to another expression as $Var \rightarrow Point$. **dominantTerm()** is also useful when it isn't obvious what the degree of the first non-zero term of a series will be, and you don't want to iteratively guess either interactively or by a program loop.

Note: See also **series()**, page 161.

dotP()

dotP(List1, List2) ⇒ expression

Returns the “dot” product of two lists.

$\text{dotP}(\{a,b,c\},\{d,e,f\})$	$a \cdot d + b \cdot e + c \cdot f$
$\text{dotP}(\{1,2\},\{5,6\})$	17

dotP(Vector1, Vector2) ⇒ expression

Returns the “dot” product of two vectors.

$\text{dotP}([a \ b \ c],[d \ e \ f])$	$a \cdot d + b \cdot e + c \cdot f$
$\text{dotP}([1 \ 2 \ 3],[4 \ 5 \ 6])$	32

Both must be row vectors, or both must be column vectors.

See Also: [TI-Nspire™ CX II - Draw Commands](#)

E

$e^{\wedge}()$

$e^{\wedge}(Expr1) \Rightarrow expression$

Returns e raised to the $Expr1$ power.

e^1	e
$e^1.$	2.71828
e^3^2	e^9

Note: See also **e exponent template**, page 2.

Note: Pressing to display $e^{\wedge}()$ is different from pressing the character on the keyboard.

You can enter a complex number in $re^{i\theta}$ polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

$e^{\wedge}()$ **e^x key** **$e^{\wedge}(\text{List1}) \Rightarrow \text{list}$** $e^{\{1,1,0.5\}}$ $\{e,2.71828,1.64872\}$ Returns **e** raised to the power of each element in *List1*. **$e^{\wedge}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}$**

1	5	3	782.209	559.617	456.509
4	2	1	680.546	488.795	396.521
6	-2	1	524.929	371.222	307.879

Returns the matrix exponential of *squareMatrix1*. This is not the same as calculating **e** raised to the power of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.**eff()****Catalog >** **$\text{eff}(\text{nominalRate}, \text{CpY}) \Rightarrow \text{value}$** $\text{eff}(5.75,12)$ 5.90398Financial function that converts the nominal interest rate *nominalRate* to an annual effective rate, given *CpY* as the number of compounding periods per year.*nominalRate* must be a real number, and *CpY* must be a real number > 0.**Note:** See also **nom()**, page 123.**eigVc()****Catalog >** **$\text{eigVc}(\text{squareMatrix}) \Rightarrow \text{matrix}$**

In Rectangular Complex Format:

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

-1	2	5	$\rightarrow m1$	-1	2	5
3	-6	9		3	-6	9
2	-5	7		2	-5	7

if $V = [x_1, x_2, \dots, x_n]$ then $x_1^2 + x_2^2 + \dots + x_n^2 = 1$
 $\text{eigVc}(m1)$

-0.800906	0.767947		(
0.484029	0.573804+0.052258 <i>i</i>	0.5738	*
0.352512	0.262687+0.096286 <i>i</i>	0.2626	

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

squareMatrix is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The *squareMatrix* is then reduced to upper Hessenberg form and the eigenvectors are computed via a Schur factorization.

eigVl()

eigVl(*squareMatrix*) ⇒ *list*

Returns a list of the eigenvalues of a real or complex *squareMatrix*.

squareMatrix is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The *squareMatrix* is then reduced to upper Hessenberg form and the eigenvalues are computed from the upper Hessenberg matrix.

In Rectangular complex format mode:

$$\begin{bmatrix} -1 & 2 & 5 \\ 3 & -6 & 9 \\ 2 & -5 & 7 \end{bmatrix} \rightarrow m1 \qquad \begin{bmatrix} -1 & 2 & 5 \\ 3 & -6 & 9 \\ 2 & -5 & 7 \end{bmatrix}$$

eigVl(*m1*)

{-4.40941,2.20471+0.763006*i*,2.20471-0.*i*}

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

Else

See If, page 86.

Elseif

If *BooleanExpr1* Then

Block1

Elseif *BooleanExpr2* Then

Block2

:

Elseif *BooleanExprN* Then

BlockN

Endif

:

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define $g(x)=\text{Func}$

If $x \leq -5$ Then

Return 5

Elseif $x > -5$ and $x < 0$ Then

Return $-x$

Elseif $x \geq 0$ and $x \neq 10$ Then

Return x

Elseif $x = 10$ Then

Return 3

EndIf

EndFunc

Done

EndFor

See For, page 72.

EndFunc

See Func, page 76.

EndIf

See If, page 86.

EndLoop

See Loop, page 110.

EndPrgm

See Prgm, page 137.

EndTry

See Try, page 191.

EndWhile

See While, page 201.

euler ()Catalog > 

euler(*Expr*, *Var*, *depVar*, {*Var0*, *VarMax*},
depVar0, *VarStep* [, *eulerStep*]) ⇒ *matrix*

euler(*SystemOfExpr*, *Var*, *ListOfDepVars*,
{*Var0*, *VarMax*}, *ListOfDepVars0*,
VarStep [, *eulerStep*]) ⇒ *matrix*

euler(*ListOfExpr*, *Var*, *ListOfDepVars*,
{*Var0*, *VarMax*}, *ListOfDepVars0*,
VarStep [, *eulerStep*]) ⇒ *matrix*

Differential equation:

 $y' = 0.001 \cdot y \cdot (100 - y)$ and $y(0) = 10$

euler(0.001·y·(100-y),t,v,{0,100},10,1)				
0.	1.	2.	3.	4.
10.	10.9	11.8712	12.9174	14.042

To see the entire result,
press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the
cursor.

Compare above result with CAS exact
solution obtained using `deSolve()` and
`seqGen()`:

Uses the Euler method to solve the system

$$\frac{d \text{depVar}}{d \text{Var}} = \text{Expr}(\text{Var}, \text{depVar})$$

with $\text{depVar}(\text{Var}0) = \text{depVar}0$ on the interval $[\text{Var}0, \text{VarMax}]$. Returns a matrix whose first row defines the *Var* output values and whose second row defines the value of the first solution component at the corresponding *Var* values, and so on.

Expr is the right-hand side that defines the ordinary differential equation (ODE).

SystemOfExpr is the system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in *ListOfDepVars*).

ListOfExpr is a list of right-hand sides that define the system of ODEs (corresponds to the order of dependent variables in *ListOfDepVars*).

Var is the independent variable.

ListOfDepVars is a list of dependent variables.

$\{\text{Var}0, \text{VarMax}\}$ is a two-element list that tells the function to integrate from *Var*0 to *Var*Max.

*ListOfDepVars*0 is a list of initial values for dependent variables.

VarStep is a nonzero number such that $\text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax} - \text{Var}0)$ and solutions are returned at $\text{Var}0 + i \cdot \text{VarStep}$ for all $i=0,1,2,\dots$ such that $\text{Var}0 + i \cdot \text{VarStep}$ is in $[\text{var}0, \text{VarMax}]$ (there may not be a solution value at *Var*Max).

eulerStep is a positive integer (defaults to 1) that defines the number of euler steps between output values. The actual step size used by the euler method is $\text{VarStep} / \text{eulerStep}$.

$$\text{deSolve}(y' = 0.001 \cdot y \cdot (100 - y) \text{ and } y(0) = 10, t, y)$$

$$y = \frac{100 \cdot (1.10517)^t}{(1.10517)^t + 9}$$

$$\text{seqGen}\left(\frac{100 \cdot (1.10517)^t}{(1.10517)^t + 9}, t, y, \{0, 100\}\right)$$

$$\{10., 10.9367, 11.9494, 13.0423, 14.2189\}$$

System of equations:

$$\begin{cases} y1' = -y1 + 0.1 \cdot y1 \cdot y2 \\ y2' = 3 \cdot y2 - y1 \cdot y2 \end{cases}$$

with $y1(0) = 2$ and $y2(0) = 5$

$$\text{euler}\left(\begin{cases} -y1 + 0.1 \cdot y1 \cdot y2 \\ 3 \cdot y2 - y1 \cdot y2 \end{cases}, t, \{y1, y2\}, \{0, 5\}, \{2, 5\}, 1\right)$$

0.	1.	2.	3.	4.	5.
2.	1.	1.	3.	27.	243.
5.	10.	30.	90.	90.	-2070.

eval(Expr) ⇒ string

eval() is valid only in the TI-Innovator™ Hub Command argument of programming commands **Get**, **GetStr**, and **Send**. The software evaluates expression *Expr* and replaces the **eval()** statement with the result as a character string.

The argument *Expr* must simplify to a real number.

Set the blue element of the RGB LED to half intensity.

<i>lum</i> :=127	127
Send "SET COLOR.BLUE eval(<i>lum</i>) "	Done

Reset the blue element to OFF.

Send "SET COLOR.BLUE OFF"	Done
---------------------------	------

eval() argument must simplify to a real number.

Send "SET LED eval("4") TO ON"	"Error: Invalid data type"
--------------------------------	----------------------------

Program to fade-in the red element

```
Define fadein()=
Prgm
For i,0,255,10
  Send "SET COLOR.RED eval(i)"
  Wait 0.1
EndFor
Send "SET COLOR.RED OFF"
EndPrgm
```

Execute the program.

fadein()	Done
-----------------	------

<i>n</i> :=0.25	0.25
<i>m</i> :=8	8
<i>n</i> · <i>m</i>	2.
Send "SET COLOR.BLUE ON TIME eval(<i>n</i> · <i>m</i>)"	Done
<i>iostr</i> .SendAns	"SET COLOR.BLUE ON TIME 2"

Although **eval()** does not display its result, you can view the resulting Hub command string after executing the command by inspecting any of the following special variables.

iostr.SendAns
iostr.GetAns
iostr.GetStrAns

Note: See also **Get** (page 77), **GetStr** (page 84), and **Send** (page 158).

exact()

Catalog >

exact(*Expr1* [, *Tolerance*]) ⇒ *expression***exact**(*List1* [, *Tolerance*]) ⇒ *list***exact**(*Matrix1* [, *Tolerance*]) ⇒ *matrix*

Uses Exact mode arithmetic to return, when possible, the rational-number equivalent of the argument.

Tolerance specifies the tolerance for the conversion; the default is 0 (zero).

$\text{exact}(0.25)$	$\frac{1}{4}$
$\text{exact}(0.333333)$	$\frac{333333}{1000000}$
$\text{exact}(0.333333, 0.001)$	$\frac{1}{3}$
$\text{exact}(3.5 \cdot x + y)$	$\frac{7 \cdot x + y}{2}$
$\text{exact}(\{0.2, 0.33, 4.125\})$	$\left\{ \frac{1}{5}, \frac{33}{100}, \frac{33}{8} \right\}$

Exit

Catalog >

Exit

Exits the current **For**, **While**, or **Loop** block.

Exit is not allowed outside the three looping structures (**For**, **While**, or **Loop**).

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Function listing:

Define $g()$ = Func	Done
Local <i>temp, i</i>	
0 → <i>temp</i>	
For <i>i</i> , 1, 100, 1	
<i>temp</i> + <i>i</i> → <i>temp</i>	
If <i>temp</i> > 20 Then	
Exit	
EndIf	
EndFor	
EndFunc	
$g()$	21

► exp

Catalog >

Expr ► **exp**

Represents *Expr* in terms of the natural exponential *e*. This is a display conversion operator. It can be used only at the end of the entry line.

Note: You can insert this operator from the computer keyboard by typing @>**exp**.

$\frac{d}{dx}(e^x + e^{-x})$	$2 \cdot \sinh(x)$
$2 \cdot \sinh(x)$ ► exp	$e^x - e^{-x}$

exp()**e^x** key**exp**(*Expr1*) ⇒ *expression*Returns *e* raised to the *Expr1* power.**Note:** See also *e* exponent template, page 2.

You can enter a complex number in re^{iθ} polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

exp(*List1*) ⇒ *list*Returns *e* raised to the power of each element in *List1*.**exp**(*squareMatrix1*) ⇒ *squareMatrix*

Returns the matrix exponential of *squareMatrix1*. This is not the same as calculating *e* raised to the power of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

e^1	e
$e^{1.}$	2.71828
e^{3^2}	e^9

$e^{\{1,1,0.5\}}$	$\{e,2.71828,1.64872\}$
-------------------	-------------------------

$\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$	$\begin{bmatrix} 782.209 & 559.617 & 456.509 \\ 680.546 & 488.795 & 396.521 \\ 524.929 & 371.222 & 307.879 \end{bmatrix}$
--	---

exp ► list()**Catalog** > **exp ► list**(*Expr,Var*) ⇒ *list*

Examines *Expr* for equations that are separated by the word “or,” and returns a list containing the right-hand sides of the equations of the form *Var=Expr*. This gives you an easy way to extract some solution values embedded in the results of the **solve()**, **cSolve()**, **fMin()**, and **fMax()** functions.

Note: **exp ► list()** is not necessary with the **zeros()** and **cZeros()** functions because they return a list of solution values directly.

You can insert this function from the keyboard by typing **exp@>list(...)**.

$\text{solve}(x^2-x-2=0,x)$	$x=-1$ or $x=2$
$\text{exp ► list}(\text{solve}(x^2-x-2=0,x),x)$	$\{-1,2\}$

expand(*Expr1* [, *Var*]) ⇒ *expression*

expand(*List1* [, *Var*]) ⇒ *list*

expand(*Matrix1* [, *Var*]) ⇒ *matrix*

expand(*Expr1*) returns *Expr1* expanded with respect to all its variables. The expansion is polynomial expansion for polynomials and partial fraction expansion for rational expressions.

The goal of **expand()** is to transform *Expr1* into a sum and/or difference of simple terms. In contrast, the goal of **factor()** is to transform *Expr1* into a product and/or quotient of simple factors.

expand(*Expr1*, *Var*) returns *Expr1* expanded with respect to *Var*. Similar powers of *Var* are collected. The terms and their factors are sorted with *Var* as the main variable. There might be some incidental factoring or expansion of the collected coefficients. Compared to omitting *Var*, this often saves time, memory, and screen space, while making the expression more comprehensible.

Even when there is only one variable, using *Var* might make the denominator factorization used for partial fraction expansion more complete.

Hint: For rational expressions, **propFrac()** is a faster but less extreme alternative to **expand()**.

Note: See also **comDenom()** for an expanded numerator over an expanded denominator.

$$\text{expand}\left(\frac{(x+y+1)^2}{x^2+2\cdot x\cdot y+2\cdot x+y^2+2\cdot y+1}\right)$$

$$\text{expand}\left(\frac{x^2-x+y^2-y}{x^2\cdot y^2-x^2\cdot y-x\cdot y^2+x\cdot y}\right) = \frac{1}{x-1} - \frac{1}{x} + \frac{1}{y-1} - \frac{1}{y}$$

$$\text{expand}\left(\frac{(x+y+1)^2\cdot y}{y^2+2\cdot y\cdot (x+1)+(x+1)^2}\right)$$

$$\text{expand}\left(\frac{(x+y+1)^2\cdot x}{x^2+2\cdot x\cdot (y+1)+(y+1)^2}\right)$$

$$\text{expand}\left(\frac{x^2-x+y^2-y}{x^2\cdot y^2-x^2\cdot y-x\cdot y^2+x\cdot y}\right) = \frac{1}{y-1} - \frac{1}{y} + \frac{1}{x\cdot (x-1)}$$

$$\text{expand}(Ans, x) = \frac{1}{x-1} - \frac{1}{x} + \frac{1}{y\cdot (y-1)}$$

$$\text{expand}\left(\frac{x^3+x^2-2}{x^2-2}\right) = \frac{2\cdot x}{x^2-2} + x+1$$

$$\text{expand}(Ans, x) = \frac{1}{x-\sqrt{2}} + \frac{1}{x+\sqrt{2}} + x+1$$

expand()

Catalog >

expand(Expr1,[Var]) also distributes logarithms and fractional powers regardless of *Var*. For increased distribution of logarithms and fractional powers, inequality constraints might be necessary to guarantee that some factors are nonnegative.

expand(Expr1, [Var]) also distributes absolute values, **sign()**, and exponentials, regardless of *Var*.

Note: See also **tExpand()** for trigonometric angle-sum and multiple-angle expansion.

$\ln(2 \cdot x \cdot y) + \sqrt{2 \cdot x \cdot y}$	$\ln(2 \cdot x \cdot y) + \sqrt{2 \cdot x \cdot y}$
$\text{expand}(\text{Ans})$	$\ln(x \cdot y) + \sqrt{2 \cdot \sqrt{x \cdot y}} + \ln(2)$
$\text{expand}(\text{Ans}) y \geq 0$	$\ln(x) + \sqrt{2 \cdot \sqrt{x \cdot y}} + \ln(y) + \ln(2)$
$\text{sign}(x \cdot y) + x \cdot y + e^{2 \cdot x + y}$	$e^{2 \cdot x + y} + \text{sign}(x \cdot y) + x \cdot y $
$\text{expand}(\text{Ans})$	$\text{sign}(x) \cdot \text{sign}(y) + x \cdot y + (e^x)^2 \cdot e^y$

expr()

Catalog >

expr(String) ⇒ expression

Returns the character string contained in *String* as an expression and immediately executes it.

$\text{expr}("1+2+x^2+x")$	x^2+x+3
$\text{expr}("\text{expand}((1+x)^2)")$	$x^2+2 \cdot x+1$
"Define cube(x)=x^3" → <i>funcstr</i>	"Define cube(x)=x^3"
$\text{expr}(\text{funcstr})$	<i>Done</i>
$\text{cube}(2)$	8

ExpReg

Catalog >

ExpReg *X*, *Y* [, [*Freq*] [, *Category*, *Include*]]

Computes the exponential regression $y = a \cdot (b)^x$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding X and Y data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot (b)^x$
stat.a, stat.b	Regression coefficients
stat.r ²	Coefficient of linear determination for transformed data
stat.r	Correlation coefficient for transformed data (x , $\ln(y)$)
stat.Resid	Residuals associated with the exponential model
stat.ResidTrans	Residuals associated with linear fit of transformed data
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

F

factor()

factor(*Expr I* [, *Var*]) \Rightarrow expression

factor(*List I* [, *Var*]) \Rightarrow list

factor(*Matrix I* [, *Var*]) \Rightarrow matrix

factor(*Expr I*) returns *Expr I* factored with respect to all of its variables over a common denominator.

$\text{factor}(a^3 \cdot x^2 - a \cdot x^2 - a^3 + a)$	$a \cdot (a-1) \cdot (a+1) \cdot (x-1) \cdot (x+1)$
$\text{factor}(x^2+1)$	x^2+1
$\text{factor}(x^2-4)$	$(x-2) \cdot (x+2)$
$\text{factor}(x^2-3)$	x^2-3
$\text{factor}(x^2-a)$	x^2-a

$Expr1$ is factored as much as possible toward linear rational factors without introducing new non-real subexpressions. This alternative is appropriate if you want factorization with respect to more than one variable.

factor($Expr1, Var$) returns $Expr1$ factored with respect to variable Var .

$Expr1$ is factored as much as possible toward real factors that are linear in Var , even if it introduces irrational constants or subexpressions that are irrational in other variables.

The factors and their terms are sorted with Var as the main variable. Similar powers of Var are collected in each factor. Include Var if factorization is needed with respect to only that variable and you are willing to accept irrational expressions in any other variables to increase factorization with respect to Var . There might be some incidental factoring with respect to other variables.

For the Auto setting of the **Auto** or **Approximate** mode, including Var permits approximation with floating-point coefficients where irrational coefficients cannot be explicitly expressed concisely in terms of the built-in functions. Even when there is only one variable, including Var might yield more complete factorization.

Note: See also **comDenom**() for a fast way to achieve partial factoring when **factor**() is not fast enough or if it exhausts memory.

Note: See also **cFactor**() for factoring all the way to complex coefficients in pursuit of linear factors.



$$\begin{array}{l} \text{factor}(a^3 \cdot x^2 - a \cdot x^2 - a^3 + a, x) \\ \hline a \cdot (a^2 - 1) \cdot (x - 1) \cdot (x + 1) \\ \text{factor}(x^2 - 3, x) \qquad \qquad (x + \sqrt{3}) \cdot (x - \sqrt{3}) \\ \hline \text{factor}(x^2 - a, x) \qquad \qquad (x + \sqrt{a}) \cdot (x - \sqrt{a}) \end{array}$$

$$\begin{array}{l} \text{factor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3) \\ \hline x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3 \\ \text{factor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x - 3, x) \\ \hline (x - 0.964673) \cdot (x + 0.611649) \cdot (x + 2.12543) \cdot (x + \dots) \end{array}$$

factor(*rationalNumber*) returns the rational number factored into primes. For composite numbers, the computing time grows exponentially with the number of digits in the second-largest factor. For example, factoring a 30-digit integer could take more than a day, and factoring a 100-digit number could take more than a century.

factor(152417172689)	123457 · 1234577
isPrime(152417172689)	false

To stop a calculation manually,

- **Handheld:** Hold down the  key and press  repeatedly.
- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **iPad®:** The app displays a prompt. You can continue waiting or cancel.

If you merely want to determine if a number is prime, use **isPrime()** instead. It is much faster, particularly if *rationalNumber* is not prime and if the second-largest factor has more than five digits.

FCdf

FCdf(*lowBound*, *upBound*, *dfNumer*, *dfDenom*) ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

FCdf

FCdf(*lowBound*, *upBound*, *dfNumer*, *dfDenom*) ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

Computes the F distribution probability between *lowBound* and *upBound* for the specified *dfNumer* (degrees of freedom) and *dfDenom*.

For $P(X \leq upBound)$, set *lowBound* = 0.

Fill *Expr*, *matrixVar* ⇒ *matrix*

Replaces each element in variable *matrixVar* with *Expr*.

matrixVar must already exist.

1 2	→ <i>amatrix</i>	1 2
3 4		3 4

Fill 1.01, <i>amatrix</i>	<i>Done</i>
---------------------------	-------------

<i>amatrix</i>	1.01 1.01
	1.01 1.01

Fill *Expr*, *listVar* ⇒ *list*

Replaces each element in variable *listVar* with *Expr*.

listVar must already exist.

{1,2,3,4,5}	→ <i>alist</i>	{1,2,3,4,5}
-------------	----------------	-------------

Fill 1.01, <i>alist</i>	<i>Done</i>
-------------------------	-------------

<i>alist</i>	{1.01,1.01,1.01,1.01,1.01}
--------------	----------------------------

FiveNumSummary**FiveNumSummary** *X* [, [*Freq*]
[, *Category*, *Include*]]

Provides an abbreviated version of the 1-variable statistics on list *X*. A summary of results is stored in the *stat.results* variable. (See page 176.)

X represents a list containing the data.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1.

Category is a list of numeric category codes for the corresponding *X* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists *X*, *Freq*, or *Category* results in a void for the corresponding element of all those lists. For more information on empty elements, see page 251.

Output variable	Description
stat.MinX	Minimum of x values.
stat.Q ₁ X	1st Quartile of x.

Output variable	Description
stat.MedianX	Median of x.
stat.Q3X	3rd Quartile of x.
stat.MaxX	Maximum of x values.

floor()

Catalog > 

floor(Expr1) ⇒ integer

$$\text{floor}(-2.14) \quad -3.$$

Returns the greatest integer that is \leq the argument. This function is identical to **int()**.

The argument can be a real or a complex number.

floor(List1) ⇒ list

floor(Matrix1) ⇒ matrix

$$\text{floor}\left(\left\{\frac{3}{2}, 0, -5.3\right\}\right) \quad \{1, 0, -6\}$$

Returns a list or matrix of the floor of each element.

$$\text{floor}\left(\begin{array}{cc} 1.2 & 3.4 \\ 2.5 & 4.8 \end{array}\right) \quad \begin{array}{cc} 1. & 3. \\ 2. & 4. \end{array}$$

Note: See also **ceiling()** and **int()**.

fMax()

Catalog > 

fMax(Expr, Var) ⇒ Boolean expression

fMax(Expr, Var, lowBound)

fMax(Expr, Var, lowBound, upBound)

fMax(Expr, Var) |

lowBound ≤ Var ≤ upBound

$$\text{fMax}(1-(x-a)^2-(x-b)^2, x) \quad x = \frac{a+b}{2}$$

$$\text{fMax}(5 \cdot x^3 - x - 2, x) \quad x = \infty$$

Returns a Boolean expression specifying candidate values of *Var* that maximize *Expr* or locate its least upper bound.

You can use the constraint (“|”) operator to restrict the solution interval and/or specify other constraints.

$$\text{fMax}(0.5 \cdot x^3 - x - 2, x) | x \leq 1 \quad x = -0.816497$$

For the Approximate setting of the **Auto** or **Approximate** mode, **fMax()** iteratively searches for one approximate local maximum. This is often faster, particularly if you use the “|” operator to constrain the search to a relatively small interval that contains exactly one local maximum.

Note: See also **fMin()** and **max()**.

fMin()

Catalog >

fMin(*Expr*, *Var*) ⇒ *Boolean expression***fMin**(*Expr*, *Var*, *lowBound*)**fMin**(*Expr*, *Var*, *lowBound*, *upBound*)**fMin**(*Expr*, *Var*) |*lowBound* ≤ *Var* ≤ *upBound*

Returns a Boolean expression specifying candidate values of *Var* that minimize *Expr* or locate its greatest lower bound.

You can use the constraint (“|”) operator to restrict the solution interval and/or specify other constraints.

For the Approximate setting of the **Auto or Approximate** mode, **fMin()** iteratively searches for one approximate local minimum. This is often faster, particularly if you use the “|” operator to constrain the search to a relatively small interval that contains exactly one local minimum.

Note: See also **fMax()** and **min()**.

$fMin(1-(x-a)^2-(x-b)^2, x)$	$x=-\infty$ or $x=\infty$
$fMin(0.5 \cdot x^3 - x - 2, x) x \geq 1$	$x=1.$

For

Catalog >

For *Var*, *Low*, *High* [, *Step*]
*Block***EndFor**

Executes the statements in *Block* iteratively for each value of *Var*, from *Low* to *High*, in increments of *Step*.

Var must not be a system variable.

Step can be positive or negative. The default value is 1.

Block can be either a single statement or a series of statements separated with the “:” character.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define $g()$ =Func	Done
Local <i>tempsum</i> , <i>step</i> , <i>i</i>	
0 → <i>tempsum</i>	
1 → <i>step</i>	
For <i>i</i> , 1, 100, <i>step</i>	
<i>tempsum</i> + <i>i</i> → <i>tempsum</i>	
EndFor	
EndFunc	
$g()$	5050

format()Catalog > **format**(*Expr*[, *formatString*]) ⇒ *string*Returns *Expr* as a character string based on the format template.*Expr* must simplify to a number.*formatString* is a string and must be in the form: "F[n]", "S[n]", "E[n]", "G[n][c]", where [] indicate optional portions.

F[n]: Fixed format. n is the number of digits to display after the decimal point.

S[n]: Scientific format. n is the number of digits to display after the decimal point.

E[n]: Engineering format. n is the number of digits after the first significant digit. The exponent is adjusted to a multiple of three, and the decimal point is moved to the right by zero, one, or two digits.

G[n][c]: Same as fixed format but also separates digits to the left of the radix into groups of three. c specifies the group separator character and defaults to a comma. If c is a period, the radix will be shown as a comma.

[Rc]: Any of the above specifiers may be suffixed with the Rc radix flag, where c is a single character that specifies what to substitute for the radix point.

<code>format(1.234567,"f3")</code>	"1.235"
<code>format(1.234567,"s2")</code>	"1.23E0"
<code>format(1.234567,"e3")</code>	"1.235E0"
<code>format(1.234567,"g3")</code>	"1.235"
<code>format(1234.567,"g3")</code>	"1,234.567"
<code>format(1.234567,"g3,r:")</code>	"1:235"

fPart()Catalog > **fPart**(*Expr1*) ⇒ *expression***fPart**(*List1*) ⇒ *list***fPart**(*Matrix1*) ⇒ *matrix*

Returns the fractional part of the argument.

For a list or matrix, returns the fractional parts of the elements.

The argument can be a real or a complex number.

<code>fPart(-1.234)</code>	-0.234
<code>fPart({1,-2,3,7.003})</code>	{0,-0.3,0.003}

FPdf(*XVal*,*dfNumer*,*dfDenom*) \Rightarrow *number*
if *XVal* is a number, *list* if *XVal* is a list

Computes the F distribution probability at *XVal* for the specified *dfNumer* (degrees of freedom) and *dfDenom*.

freqTable ► list()

freqTable ► **list**(*List1*,*freqIntegerList*) \Rightarrow *list*

Returns a list containing the elements from *List1* expanded according to the frequencies in *freqIntegerList*. This function can be used for building a frequency table for the Data & Statistics application.

List1 can be any valid list.

freqIntegerList must have the same dimension as *List1* and must contain non-negative integer elements only. Each element specifies the number of times the corresponding *List1* element will be repeated in the result list. A value of zero excludes the corresponding *List1* element.

Note: You can insert this function from the computer keyboard by typing **freqTable@>list(...)**.

Empty (void) elements are ignored. For more information on empty elements, see page 251.

```
freqTable►list({1,2,3,4},{1,4,3,1})
                {1,2,2,2,3,3,3,4}
freqTable►list({1,2,3,4},{1,4,0,1})
                {1,2,2,2,4}
```

frequency()

frequency(*List1*,*binsList*) \Rightarrow *list*

Returns a list containing counts of the elements in *List1*. The counts are based on ranges (bins) that you define in *binsList*.

If *binsList* is {b(1), b(2), ..., b(n)}, the specified ranges are { $? \leq b(1)$, $b(1) < ? \leq b(2)$, ..., $b(n-1) < ? \leq b(n)$, $b(n) > ?$ }. The resulting list is one element longer than *binsList*.

```
datalist={1,2,e,3,π,4,5,6,"hello",7}
          {1,2,2.71828,3,3.14159,4,5,6,"hello",7}
frequency(datalist,{2.5,4.5})           {2,4,3}
```

Explanation of result:

2 elements from *Datalist* are ≤ 2.5

Each element of the result corresponds to the number of elements from *List1* that are in the range of that bin. Expressed in terms of the **countif()** function, the result is {countif(list, $? \leq b(1)$), countif(list, $b(1) < ? \leq b(2)$), ..., countif(list, $b(n-1) < ? \leq b(n)$), countif(list, $b(n) > ?$)}.

Elements of *List1* that cannot be “placed in a bin” are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 251.

Within the Lists & Spreadsheet application, you can use a range of cells in place of both arguments.

Note: See also **countif()**, page 35.

4 elements from *Datalist* are >2.5 and ≤ 4.5

3 elements from *Datalist* are >4.5

The element “hello” is a string and cannot be placed in any of the defined bins.

FTest_2Samp

FTest_2Samp *List1, List2[, Freq1[, Freq2[, Hypoth]]]*

FTest_2Samp *List1, List2[, Freq1[, Freq2[, Hypoth]]]*

(Data list input)

FTest_2Samp *sx1, n1, sx2, n2[, Hypoth]*

FTest_2Samp *sx1, n1, sx2, n2[, Hypoth]*

(Summary stats input)

Performs a two-sample F test. A summary of results is stored in the *stat.results* variable. (See page 176.)

For $H_a: \sigma_1 > \sigma_2$, set *Hypoth* > 0

For $H_a: \sigma_1 \neq \sigma_2$ (default), set *Hypoth* = 0

For $H_a: \sigma_1 < \sigma_2$, set *Hypoth* < 0

For information on the effect of empty elements in a list, see *Empty (Void) Elements*, page 251.

Output variable	Description
stat.F	Calculated F statistic for the data sequence

Output variable	Description
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.dfNumer	numerator degrees of freedom = $n1-1$
stat.dfDenom	denominator degrees of freedom = $n2-1$
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.x1_bar stat.x2_bar	Sample means of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Size of the samples

Func

Catalog >

Func

Block

EndFunc

Template for creating a user-defined function.

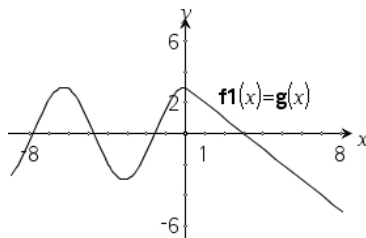
Block can be a single statement, a series of statements separated with the “;” character, or a series of statements on separate lines. The function can use the **Return** instruction to return a specific result.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define a piecewise function:

```
Define g(x)=Func Done
  If x<0 Then
    Return 3*cos(x)
  Else
    Return 3-x
  EndIf
EndFunc
```

Result of graphing $g(x)$



G

gcd()

Catalog >

$\text{gcd}(\text{Number1}, \text{Number2}) \Rightarrow \text{expression}$

$\text{gcd}(18,33)$ 3

Returns the greatest common divisor of the two arguments. The **gcd** of two fractions is the **gcd** of their numerators divided by the **lcm** of their denominators.

gcd()

Catalog > 

In Auto or Approximate mode, the **gcd** of fractional floating-point numbers is 1.0.

gcd(*List1*, *List2*) ⇒ *list*

Returns the greatest common divisors of the corresponding elements in *List1* and *List2*.

$\text{gcd}(\{12,14,16\},\{9,7,5\})$ $\{3,7,1\}$

gcd(*Matrix1*, *Matrix2*) ⇒ *matrix*

Returns the greatest common divisors of the corresponding elements in *Matrix1* and *Matrix2*.

$\text{gcd}\left(\begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}, \begin{pmatrix} 4 & 8 \\ 12 & 16 \end{pmatrix}\right)$ $\begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$

geomCdf()

Catalog > 

geomCdf(*p*,*lowBound*,*upBound*) ⇒ *number*
if *lowBound* and *upBound* are numbers, *list*
if *lowBound* and *upBound* are lists

geomCdf(*p*,*upBound*) for $P(1 \leq X \leq \textit{upBound})$
⇒ *number* if *upBound* is a number, *list* if
upBound is a list

Computes a cumulative geometric probability from *lowBound* to *upBound* with the specified probability of success *p*.

For $P(X \leq \textit{upBound})$, set *lowBound* = 1.

geomPdf()

Catalog > 

geomPdf(*p*,*XVal*) ⇒ *number* if *XVal* is a number, *list* if *XVal* is a list

Computes a probability at *XVal*, the number of the trial on which the first success occurs, for the discrete geometric distribution with the specified probability of success *p*.

Get

Hub Menu

Get [*promptString*,] *var*[, *statusVar*]

Get [*promptString*,] *func*(*arg1*, ...*argn*)
[, *statusVar*]

Example: Request the current value of the hub's built-in light-level sensor. Use **Get** to retrieve the value and assign it to variable *lightval*.

Programming command: Retrieves a value from a connected TI-Innovator™ Hub and assigns the value to variable *var*.

The value must be requested:

- In advance, through a **Send "READ ..."** command.
— or —
- By embedding a **"READ ..."** request as the optional *promptString* argument. This method lets you use a single command to request the value and retrieve it.

Implicit simplification takes place. For example, a received string of "123" is interpreted as a numeric value. To preserve the string, use **GetStr** instead of **Get**.

If you include the optional argument *statusVar*, it is assigned a value based on the success of the operation. A value of zero means that no data was received.

In the second syntax, the *func()* argument allows a program to store the received string as a function definition. This syntax operates as if the program executed the command:

Define *func(arg1, ...argn) = received string*

The program can then use the defined function *func()*.

Note: You can use the **Get** command within a user-defined program but not within a function.

Note: See also **GetStr**, page 84 and **Send**, page 158.

Send "READ BRIGHTNESS"	Done
Get <i>lightval</i>	Done
<i>lightval</i>	0.347922

Embed the READ request within the **Get** command.

Get "READ BRIGHTNESS", <i>lightval</i>	Done
<i>lightval</i>	0.378441

getDenom()

Catalog >

getDenom(*Expr1*) \Rightarrow *expression*

Transforms the argument into an expression having a reduced common denominator, and then returns its denominator.

$\text{getDenom}\left(\frac{x+2}{y-3}\right)$	$y-3$
$\text{getDenom}\left(\frac{2}{7}\right)$	7
$\text{getDenom}\left(\frac{1}{x} + \frac{y^2+y}{y^2}\right)$	$x \cdot y$

getKey()

Catalog >

getKey([0|1]) \Rightarrow *returnString*

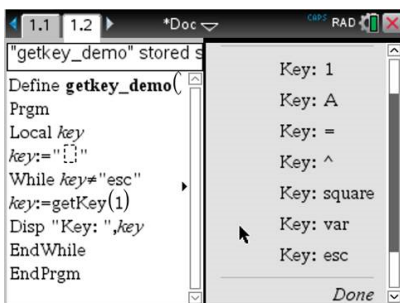
Description: **getKey()** - allows a TI-Basic program to get keyboard input - handheld, desktop and emulator on desktop.

Example:

- keypressed := **getKey()** will return a key or an empty string if no key has been pressed. This call will return immediately.
- keypressed := **getKey(1)** will wait till a key is pressed. This call will pause execution of the program till a key is pressed.

getKey()

Example:



Handling of key presses:

Handheld Device/Emulator Key	Desktop	Return Value
Esc	Esc	"esc"
Touchpad - Top click	n/a	"up"
On	n/a	"home"
Scratchpads	n/a	"scratchpad"
Touchpad - Left click	n/a	"left"
Touchpad - Center click	n/a	"center"
Touchpad - Right click	n/a	"right"
Doc	n/a	"doc"

Handheld Device/Emulator Key	Desktop	Return Value
Tab	Tab	"tab"
Touchpad - Bottom click	Down Arrow	"down"
Menu	n/a	"menu"
Ctrl	Ctrl	no return
Shift	Shift	no return
Var	n/a	"var"
Del	n/a	"del"
=	=	"="
trig	n/a	"trig"
0 through 9	0-9	"0" ... "9"
Templates	n/a	"template"
Catalog	n/a	"cat"
^	^	"^"
X^2	n/a	"square"
/ (division key)	/	"/"
* (multiply key)	*	"*"
e^x	n/a	"exp"
10^x	n/a	"10power"
+	+	"+"
-	-	"_"
(("("
))	")"
.	.	"."
(-)	n/a	"-" (negate sign)
Enter	Enter	"enter"
ee	n/a	"E" (scientific notation E)
a - z	a-z	alpha = letter pressed (lower

Handheld Device/Emulator Key	Desktop	Return Value
		case) ("a" - "z")
shift a-z	shift a-z	alpha = letter pressed "A" - "Z"
		Note: ctrl-shift works to lock caps
?!	n/a	"?!"
pi	n/a	"pi"
Flag	n/a	no return
,	,	","
Return	n/a	"return"
Space	Space	" " (space)
Inaccessible	Special Character Keys like @,!,^, etc.	The character is returned
n/a	Function Keys	No returned character
n/a	Special desktop control keys	No returned character
Inaccessible	Other desktop keys that are not available on the calculator while getKey() is waiting for a keystroke. ({, },,, ;, ...)	Same character you get in Notes (not in a math box)

Note: It is important to note that the presence of **getKey()** in a program changes how certain events are handled by the system. Some of these are described below.

Terminate program and Handle event - Exactly as if the user were to break out of program by pressing the **ON** key

"**Support**" below means - System works as expected - program continues to run.

Event	Device	Desktop - TI-Nspire™ Student Software
Quick Poll	Terminate program, handle event	Same as the handheld (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)
Remote file mgmt	Terminate program, handle event	Same as the handheld. (TI-Nspire™ Student

Event	Device	Desktop - TI-Nspire™ Student Software
(Incl. sending 'Exit Press 2 Test' file from another handheld or desktop-handheld)		Software, TI-Nspire™ Navigator™ NC Teacher Software-only)
End Class	Terminate program, handle event	Support (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)

Event	Device	Desktop - TI-Nspire™ All Versions
TI-Innovator™ Hub connect/disconnect	Support - Can successfully issue commands to the TI-Innovator™ Hub. After you exit the program the TI-Innovator™ Hub is still working with the handheld.	Same as the handheld

getLangInfo()

Catalog > 

getLangInfo() ⇒ *string*

`getLangInfo()`

"en"

Returns a string that corresponds to the short name of the currently active language. You can, for example, use it in a program or function to determine the current language.

English = "en"

Danish = "da"

German = "de"

Finnish = "fi"

French = "fr"

Italian = "it"

Dutch = "nl"

Belgian Dutch = "nl_BE"


Norwegian = "no"

Portuguese = "pt"

Spanish = "es"

Swedish = "sv"

getLockInfo()

Catalog > 

getLockInfo(*Var*) ⇒ *value*

Returns the current locked/unlocked state of variable *Var*.

value =0: *Var* is unlocked or does not exist.

value =1: *Var* is locked and cannot be modified or deleted.

See **Lock**, page 106, and **unLock**, page 197.

<i>a</i> :=65	65
Lock <i>a</i>	Done
getLockInfo(<i>a</i>)	1
<i>a</i> :=75	"Error: Variable is locked."
DelVar <i>a</i>	"Error: Variable is locked."
Unlock <i>a</i>	Done
<i>a</i> :=75	75
DelVar <i>a</i>	Done

getMode()

Catalog > 

getMode(*ModeNameInteger*) ⇒ *value*

getMode(0) ⇒ *list*

getMode(*ModeNameInteger*) returns a value representing the current setting of the *ModeNameInteger* mode.

getMode(0) returns a list containing number pairs. Each pair consists of a mode integer and a setting integer.

For a listing of the modes and their settings, refer to the table below.

If you save the settings with **getMode**(0) → *var*, you can use **setMode**(*var*) in a function or program to temporarily restore the settings within the execution of the function or program only. See **setMode**(), page 162.

getMode(0)	{ 1,7,2,1,3,1,4,1,5,1,6,1,7,1,8,1 }
getMode(1)	7
getMode(8)	1

Mode Name	Mode Integer	Setting Integers
Display Digits	1	1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12, 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12
Angle	2	1=Radian, 2=Degree, 3=Gradian
Exponential Format	3	1=Normal, 2=Scientific, 3=Engineering

Mode Name	Mode Integer	Setting Integers
Real or Complex	4	1=Real, 2=Rectangular, 3=Polar
Auto or Approx.	5	1=Auto, 2=Approximate, 3=Exact
Vector Format	6	1=Rectangular, 2=Cylindrical, 3=Spherical
Base	7	1=Decimal, 2=Hex, 3=Binary
Unit system	8	1=SI, 2=Eng/US

getNum()

Catalog > 

getNum(*Expr1*) ⇒ *expression*

Transforms the argument into an expression having a reduced common denominator, and then returns its numerator.

$\text{getNum}\left(\frac{x+2}{y-3}\right)$	$x+2$
$\text{getNum}\left(\frac{2}{7}\right)$	2
$\text{getNum}\left(\frac{1}{x} + \frac{1}{y}\right)$	$x+y$

GetStr

Hub Menu

GetStr [*promptString*,] *var* [, *statusVar*]

For examples, see **Get**.

GetStr [*promptString*,] *func*(*arg1*, ...*argn*)
[, *statusVar*]

Programming command: Operates identically to the **Get** command, except that the retrieved value is always interpreted as a string. By contrast, the **Get** command interprets the response as an expression unless it is enclosed in quotation marks ("").

Note: See also **Get**, page 77 and **Send**, page 158.

getType()

Catalog > 

getType(var) ⇒ *string*

Returns a string that indicates the data type of variable *var*.

If *var* has not been defined, returns the string "NONE".

$\{1,2,3\} \rightarrow temp$	$\{1,2,3\}$
<code>getType(temp)</code>	"LIST"
$3 \cdot i \rightarrow temp$	$3 \cdot i$
<code>getType(temp)</code>	"EXPR"
<code>DelVar temp</code>	Done
<code>getType(temp)</code>	"NONE"

getVarInfo()

Catalog > 

getVarInfo() ⇒ *matrix* or *string*

getVarInfo(LibNameString) ⇒ *matrix* or *string*

getVarInfo() returns a matrix of information (variable name, type, library accessibility, and locked/unlocked state) for all variables and library objects defined in the current problem.

If no variables are defined, **getVarInfo()** returns the string "NONE".

getVarInfo(LibNameString) returns a matrix of information for all library objects defined in library *LibNameString*. *LibNameString* must be a string (text enclosed in quotation marks) or a string variable.

If the library *LibNameString* does not exist, an error occurs.

Note the example, in which the result of **getVarInfo()** is assigned to variable *vs*. Attempting to display row 2 or row 3 of *vs* returns an "Invalid list or matrix" error because at least one of elements in those rows (variable *b*, for example) reevaluates to a matrix.

This error could also occur when using *Ans* to reevaluate a **getVarInfo()** result.

The system gives the above error because the current version of the software does not support a generalized matrix structure where an element of a matrix can be either a matrix or a list.

<code>getVarInfo()</code>	"NONE"												
Define $x=5$	Done												
Lock x	Done												
Define LibPriv $y=\{1,2,3\}$	Done												
Define LibPub $z(x)=3 \cdot x^2 - x$	Done												
<code>getVarInfo()</code>	<table border="1"><tr><td>x</td><td>"NUM"</td><td>"{"</td><td>1</td></tr><tr><td>y</td><td>"LIST"</td><td>"LibPriv"</td><td>0</td></tr><tr><td>z</td><td>"FUNC"</td><td>"LibPub"</td><td>0</td></tr></table>	x	"NUM"	"{"	1	y	"LIST"	"LibPriv"	0	z	"FUNC"	"LibPub"	0
x	"NUM"	"{"	1										
y	"LIST"	"LibPriv"	0										
z	"FUNC"	"LibPub"	0										
<code>getVarInfo(tmp3)</code>	"Error: Argument must be a string"												
<code>getVarInfo("tmp3")</code>	<table border="1"><tr><td>$[volcy12$</td><td>"NONE"</td><td>"LibPub"</td><td>0]</td></tr></table>	$[volcy12$	"NONE"	"LibPub"	0]								
$[volcy12$	"NONE"	"LibPub"	0]										

$a:=1$	1												
$b:=[1 \ 2]$	[1 2]												
$c:=[1 \ 3 \ 7]$	[1 3 7]												
$vs:=\text{getVarInfo}()$	<table border="1"><tr><td>a</td><td>"NUM"</td><td>"{"</td><td>0</td></tr><tr><td>b</td><td>"MAT"</td><td>"{"</td><td>0</td></tr><tr><td>c</td><td>"MAT"</td><td>"{"</td><td>0</td></tr></table>	a	"NUM"	"{"	0	b	"MAT"	"{"	0	c	"MAT"	"{"	0
a	"NUM"	"{"	0										
b	"MAT"	"{"	0										
c	"MAT"	"{"	0										
$vs[1]$	[1 "NUM" "{" 0]												
$vs[1,1]$	1												
$vs[2]$	"Error: Invalid list or matrix"												
$vs[2,1]$	[1 2]												

Goto

Catalog >

Goto *labelName*Transfers control to the label *labelName*.*labelName* must be defined in the same function using a **Lbl** instruction.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define $g()$ =Func	<i>Done</i>
Local <i>temp,i</i>	
$0 \rightarrow temp$	
$1 \rightarrow i$	
Lbl <i>top</i>	
$temp+i \rightarrow temp$	
If $i < 10$ Then	
$i+1 \rightarrow i$	
Goto <i>top</i>	
EndIf	
Return <i>temp</i>	
EndFunc	
$g()$	55

► Grad

Catalog >

Expr1 ► **Grad** ⇒ *expression*Converts *Expr1* to gradian angle measure.

Note: You can insert this operator from the computer keyboard by typing @>**Grad**.

In Degree angle mode:	
$(1.5) \blacktriangleright \text{Grad}$	$(1.66667)^{\circ}$
In Radian angle mode:	
$(1.5) \blacktriangleright \text{Grad}$	$(95.493)^{\circ}$

/

identity()

Catalog >

identity(*Integer*) ⇒ *matrix*Returns the identity matrix with a dimension of *Integer*.*Integer* must be a positive integer.

identity(4)	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
-------------	--

If

Catalog >

If *BooleanExpr*
*Statement***If** *BooleanExpr* **Then**
*Block***EndIf**

Define $g(x)$ =Func	<i>Done</i>
If $x < 0$ Then	
Return x^2	
EndIf	
EndFunc	
$g(-2)$	4

If *BooleanExpr* evaluates to true, executes the single statement *Statement* or the block of statements *Block* before continuing execution.

If *BooleanExpr* evaluates to false, continues execution without executing the statement or block of statements.

Block can be either a single statement or a sequence of statements separated with the “.” character.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

If *BooleanExpr* Then
Block1

Else
Block2

EndIf

If *BooleanExpr* evaluates to true, executes *Block1* and then skips *Block2*.

If *BooleanExpr* evaluates to false, skips *Block1* but executes *Block2*.

Block1 and *Block2* can be a single statement.

If *BooleanExpr1* Then
Block1

Elseif *BooleanExpr2* Then
Block2

:

Elseif *BooleanExprN* Then
BlockN

EndIf

Allows for branching. If *BooleanExpr1* evaluates to true, executes *Block1*. If *BooleanExpr1* evaluates to false, evaluates *BooleanExpr2*, and so on.

```
Define g(x)=Func                               Done
  If x<0 Then
  Return -x
  Else
  Return x
  EndIf
  EndFunc
```

$g(12)$	12
$g(-12)$	12

```
Define g(x)=Func
  If x<-5 Then
  Return 5
  ElseIf x>-5 and x<0 Then
  Return -x
  ElseIf x≥0 and x≠10 Then
  Return x
  ElseIf x=10 Then
  Return 3
  EndIf
  EndFunc
```

$g(-4)$	4
$g(10)$	3

ifFn()

Catalog >

$\text{ifFn}(\text{BooleanExpr}, \text{Value_If_true } [, \text{Value_If_false } [, \text{Value_If_unknown}]]) \Rightarrow$
expression, list, or matrix

Evaluates the boolean expression *BooleanExpr* (or each element from *BooleanExpr*) and produces a result based on the following rules:

- *BooleanExpr* can test a single value, a list, or a matrix.
- If an element of *BooleanExpr* evaluates to true, returns the corresponding element from *Value_If_true*.
- If an element of *BooleanExpr* evaluates to false, returns the corresponding element from *Value_If_false*. If you omit *Value_If_false*, returns undef.
- If an element of *BooleanExpr* is neither true nor false, returns the corresponding element *Value_If_unknown*. If you omit *Value_If_unknown*, returns undef.
- If the second, third, or fourth argument of the **ifFn()** function is a single expression, the Boolean test is applied to every position in *BooleanExpr*.

Note: If the simplified *BooleanExpr* statement involves a list or matrix, all other list or matrix arguments must have the same dimension(s), and the result will have the same dimension(s).

$$\text{ifFn}(\{1,2,3\} < 2.5, \{5,6,7\}, \{8,9,10\})$$

$$\{5,6,10\}$$

Test value of **1** is less than 2.5, so its corresponding

Value_If_True element of **5** is copied to the result list.

Test value of **2** is less than 2.5, so its corresponding

Value_If_True element of **6** is copied to the result list.

Test value of **3** is not less than 2.5, so its corresponding *Value_If_False* element of **10** is copied to the result list.

$$\text{ifFn}(\{1,2,3\} < 2.5, 4, \{8,9,10\})$$

$$\{4,4,10\}$$

Value_If_true is a single value and corresponds to any selected position.

$$\text{ifFn}(\{1,2,3\} < 2.5, \{5,6,7\})$$

$$\{5,6,\text{undef}\}$$

Value_If_false is not specified. Undef is used.

$$\text{ifFn}(\{2, "a" \} < 2.5, \{6,7\}, \{9,10\}, "err")$$

$$\{6, "err" \}$$

One element selected from *Value_If_true*.
 One element selected from *Value_If_undefined*.

imag()

Catalog >

$\text{imag}(\text{Expr } I) \Rightarrow$ *expression*

Returns the imaginary part of the argument.

$$\text{imag}(1+2 \cdot i)$$

$$2$$

$$\text{imag}(z)$$

$$0$$

$$\text{imag}(x+i \cdot y)$$

$$y$$

imag()Catalog > 

Note: All undefined variables are treated as real variables. See also `real()`, page 146

imag(List l) ⇒ list

$\text{imag}(\{-3,4-i,i\})$	$\{0,-1,1\}$
-----------------------------	--------------

Returns a list of the imaginary parts of the elements.

imag(Matrix l) ⇒ matrix

$\text{imag}\left(\begin{pmatrix} a & b \\ i\cdot c & i\cdot d \end{pmatrix}\right)$	$\begin{bmatrix} 0 & 0 \\ c & d \end{bmatrix}$
--	--

Returns a matrix of the imaginary parts of the elements.

impDif()Catalog > 

impDif(Equation, Var, dependVar[,Ord]) ⇒ expression

$\text{impDif}(x^2+y^2=100,x,y)$	$\frac{-x}{y}$
----------------------------------	----------------

where the order *Ord* defaults to 1.

Computes the implicit derivative for equations in which one variable is defined implicitly in terms of another.

Indirection

See #(), page 226.

inString()Catalog > 

inString(srcString, subString[, Start]) ⇒ integer

$\text{inString}(\text{"Hello there"}, \text{"the"})$	7
$\text{inString}(\text{"ABCEFG"}, \text{"D"})$	0

Returns the character position in string *srcString* at which the first occurrence of string *subString* begins.

Start, if included, specifies the character position within *srcString* where the search begins. Default = 1 (the first character of *srcString*).

If *srcString* does not contain *subString* or *Start* is > the length of *srcString*, returns zero.

int()

Catalog >

int(*Expr*) ⇒ *integer* $\text{int}(-2.5)$ -3.**int**(*List1*) ⇒ *list* $\text{int}([-1.234 \ 0 \ 0.37])$ [-2. 0 0.]**int**(*Matrix1*) ⇒ *matrix*

Returns the greatest integer that is less than or equal to the argument. This function is identical to **floor()**.

The argument can be a real or a complex number.

For a list or matrix, returns the greatest integer of each of the elements.

intDiv()

Catalog >

intDiv(*Number1*, *Number2*) ⇒ *integer* $\text{intDiv}(-7,2)$ -3**intDiv**(*List1*, *List2*) ⇒ *list* $\text{intDiv}(4,5)$ 0**intDiv**(*Matrix1*, *Matrix2*) ⇒ *matrix* $\text{intDiv}(\{12, -14, -16\}, \{5, 4, -3\})$ {2, -3, 5}

Returns the signed integer part of (*Number1* ÷ *Number2*).

For lists and matrices, returns the signed integer part of (argument 1 ÷ argument 2) for each element pair.

integralSee **∫()**, page 221.**interpolate ()**

Catalog >

interpolate(*xValue*, *xList*, *yList*, *yPrimeList*) ⇒ *list*

Differential equation:

 $y' = -3 \cdot y + 6 \cdot t + 5$ and $y(0) = 5$

This function does the following:

$$rk := rk23(-3 \cdot y + 6 \cdot t + 5, t, y, \{0, 10\}, 5, 1)$$

0.	1.	2.	3.	4.
5.	3.19499	5.00394	6.99957	9.00593

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

interpolate ()

Catalog > 

Given $xList$, $yList=f(xList)$, and $yPrimeList=f'(xList)$ for some unknown function f , a cubic interpolant is used to approximate the function f at $xValue$. It is assumed that $xList$ is a list of monotonically increasing or decreasing numbers, but this function may return a value even when it is not. This function walks through $xList$ looking for an interval $[xList[i], xList[i+1]]$ that contains $xValue$. If it finds such an interval, it returns an interpolated value for $f(xValue)$; otherwise, it returns **undef**.

$xList$, $yList$, and $yPrimeList$ must be of equal dimension ≥ 2 and contain expressions that simplify to numbers.

$xValue$ can be an undefined variable, a number, or a list of numbers.

Use the interpolate() function to calculate the function values for the $xvalueList$:

```
xvalueList:=seq(i,i,0,10,0.5)
{0,0.5,1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
xlist:=mat▶list(rk[1])
{0.,1.,2.,3.,4.,5.,6.,7.,8.,9.,10.}
ylist:=mat▶list(rk[2])
{5.,3.19499,5.00394,6.99957,9.00593,10.9979}
yprimeList:=-3*y+6*t+5|y=yList and t=xList
{-10.,1.41503,1.98819,2.00129,1.98221,2.00601}
interpolate(xvalueList,xlist,yList,yprimeList)
{5.,2.67062,3.19499,4.02782,5.00394,6.00011}
```

invχ²()

Catalog > 

$inv\chi^2(Area,df)$

$invChi2(Area,df)$

Computes the Inverse cumulative χ^2 (chi-square) probability function specified by degree of freedom, df for a given $Area$ under the curve.

invF()

Catalog > 

$invF(Area,dfNumer,dfDenom)$

$invF(Area,dfNumer,dfDenom)$

computes the Inverse cumulative F distribution function specified by $dfNumer$ and $dfDenom$ for a given $Area$ under the curve.

invBinom()

Catalog > 

invBinom

(CumulativeProb, NumTrials, Prob, OutputForm) \Rightarrow scalar or matrix

Inverse binomial. Given the number of trials (*NumTrials*) and the probability of success of each trial (*Prob*), this function returns the minimum number of successes, *k*, such that the value, *k*, is greater than or equal to the given cumulative probability (*CumulativeProb*).

OutputForm=0, displays result as a scalar (default).

OutputForm=1, displays result as a matrix.

Example: Mary and Kevin are playing a dice game. Mary has to guess the maximum number of times 6 shows up in 30 rolls. If the number 6 shows up that many times or less, Mary wins. Furthermore, the smaller the number that she guesses, the greater her winnings. What is the smallest number Mary can guess if she wants the probability of winning to be greater than 77%?

$\text{invBinom}\left(0.77, 30, \frac{1}{6}\right)$	6
$\text{invBinom}\left(0.77, 30, \frac{1}{6}, 1\right)$	$\begin{bmatrix} 5 & 0.616447 \\ 6 & 0.776537 \end{bmatrix}$

invBinomN()

Catalog > 

invBinomN(CumulativeProb, Prob, NumSuccess, OutputForm) \Rightarrow scalar or matrix

Inverse binomial with respect to N. Given the probability of success of each trial (*Prob*), and the number of successes (*NumSuccess*), this function returns the minimum number of trials, *N*, such that the value, *N*, is less than or equal to the given cumulative probability (*CumulativeProb*).

OutputForm=0, displays result as a scalar (default).

OutputForm=1, displays result as a matrix.

Example: Monique is practicing goal shots for netball. She knows from experience that her chance of making any one shot is 70%. She plans to practice until she scores 50 goals. How many shots must she attempt to ensure that the probability of making at least 50 goals is more than 0.99?

$\text{invBinomN}(0.01, 0.7, 49)$	86
$\text{invBinomN}(0.01, 0.7, 49, 1)$	$\begin{bmatrix} 85 & 0.010451 \\ 86 & 0.00709 \end{bmatrix}$

invNorm()

Catalog > 

invNorm(Area, μ , σ)

Computes the inverse cumulative normal distribution function for a given *Area* under the normal distribution curve specified by μ and σ .

invt()

Catalog > 

invt(Area, df)

invT()

Catalog >

Computes the inverse cumulative student-t probability function specified by degree of freedom, df for a given *Area* under the curve.

iPart()

Catalog >

iPart(*Number*) \Rightarrow *integer***iPart**(*List1*) \Rightarrow *list***iPart**(*Matrix1*) \Rightarrow *matrix*

iPart (-1.234)	-1.
-----------------------	-----

iPart ($\left\{\left\{\frac{3}{2}, -2.3, 7.003\right\}\right\}$)	{1,-2,.7}
---	-----------

Returns the integer part of the argument.

For lists and matrices, returns the integer part of each element.

The argument can be a real or a complex number.

irr()

Catalog >

irr(*CF0*,*CFList* [,*CFFreq*]) \Rightarrow *value*

Financial function that calculates internal rate of return of an investment.

CF0 is the initial cash flow at time 0; it must be a real number.

CFList is a list of cash flow amounts after the initial cash flow *CF0*.

CFFreq is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.

Note: See also **mirr()**, page 115.

<i>list1</i> := {6000,-8000,2000,-3000}	{6000,-8000,2000,-3000}
---	-------------------------

<i>list2</i> := {2,2,2,1}	{2,2,2,1}
---------------------------	-----------

irr (5000, <i>list1</i> , <i>list2</i>)	-4.64484
---	----------

isPrime()

Catalog >

isPrime(*Number*) \Rightarrow *Boolean constant expression*

isPrime (5)	true
--------------------	------

isPrime (6)	false
--------------------	-------

isPrime()

Catalog > 

Returns true or false to indicate if *number* is a whole number ≥ 2 that is evenly divisible only by itself and 1.

If *Number* exceeds about 306 digits and has no factors ≤ 1021 , **isPrime(*Number*)** displays an error message.

If you merely want to determine if *Number* is prime, use **isPrime()** instead of **factor()**. It is much faster, particularly if *Number* is not prime and has a second-largest factor that exceeds about five digits.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Function to find the next prime after a specified number:

Define <i>nextprim</i> (<i>n</i>)=Func	Done
Loop	
$n+1 \rightarrow n$	
If isPrime(<i>n</i>)	
Return <i>n</i>	
EndLoop	
EndFunc	
<i>nextprim</i> (7)	11

isVoid()

Catalog > 

isVoid(*Var*) \Rightarrow Boolean constant expression

isVoid(*Expr*) \Rightarrow Boolean constant expression

isVoid(*List*) \Rightarrow list of Boolean constant expressions

Returns true or false to indicate if the argument is a void data type.

For more information on void elements, see page 251.

<i>a</i> :=_	_
isVoid(<i>a</i>)	true
isVoid({1,_,3})	{ false,true,false }

L

Lbl

Catalog > 

Lbl *labelName*

Defines a label with the name *labelName* within a function.

You can use a **Goto** *labelName* instruction to transfer control to the instruction immediately following the label.

labelName must meet the same naming requirements as a variable name.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define $g()$ =Func	<i>Done</i>
Local <i>temp,i</i>	
$0 \rightarrow temp$	
$1 \rightarrow i$	
Lbl <i>top</i>	
$temp+i \rightarrow temp$	
If $i < 10$ Then	
$i+1 \rightarrow i$	
Goto <i>top</i>	
EndIf	
Return <i>temp</i>	
EndFunc	

$g()$	55
-------	----

lcm()

Catalog > 

$lcm(\text{Number1}, \text{Number2}) \Rightarrow \text{expression}$

$lcm(\text{List1}, \text{List2}) \Rightarrow \text{list}$


$lcm(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix}$

Returns the least common multiple of the two arguments. The **lcm** of two fractions is the **lcm** of their numerators divided by the **gcd** of their denominators. The **lcm** of fractional floating-point numbers is their product.

For two lists or matrices, returns the least common multiples of the corresponding elements.

$lcm(6,9)$	18
$lcm\left(\left\{\frac{1}{3}, -14, 16\right\}, \left\{\frac{2}{15}, 7, 5\right\}\right)$	$\left\{\frac{2}{3}, 14, 80\right\}$

left()

Catalog > 

$left(\text{sourceString}[, \text{Num}]) \Rightarrow \text{string}$

Returns the leftmost *Num* characters contained in character string *sourceString*.

If you omit *Num*, returns all of *sourceString*.

$left(\text{List1}[, \text{Num}]) \Rightarrow \text{list}$

$left(\text{"Hello"}, 2)$	"He"
$left(\{1, 3, 2, 4\}, 3)$	$\{1, 3, 2\}$

left()

Catalog > 

Returns the leftmost *Num* elements contained in *List1*.

If you omit *Num*, returns all of *List1*.

left(Comparison) ⇒ *expression*

Returns the left-hand side of an equation or inequality.

$$\text{left}(x < 3) \quad x$$

libShortcut()

Catalog > 

libShortcut(*LibNameString*,
ShortcutNameString
[, *LibPrivFlag*]) ⇒ *list of variables*

Creates a variable group in the current problem that contains references to all the objects in the specified library document *libNameString*. Also adds the group members to the Variables menu. You can then refer to each object using its *ShortcutNameString*.

Set *LibPrivFlag*=0 to exclude private library objects (default)

Set *LibPrivFlag*=1 to include private library objects

To copy a variable group, see **CopyVar** on page 29.

To delete a variable group, see **DelVar** on page 48.

This example assumes a properly stored and refreshed library document named **linalg2** that contains objects defined as *clearmat*, *gauss1*, and *gauss2*.

```
getVarInfo("linalg2")
  [clearmat "FUNC" "LibPub "]
  [gauss1  "PRGM" "LibPriv "]
  [gauss2  "FUNC" "LibPub "]
libShortcut("linalg2", "la")
  {la.clearmat, la.gauss2}
libShortcut("linalg2", "la", 1)
  {la.clearmat, la.gauss1, la.gauss2}
```

limit() or lim()

Catalog > 

limit(*Expr1*, *Var*, *Point* [, *Direction*]) ⇒ *expression*

limit(*List1*, *Var*, *Point* [, *Direction*]) ⇒ *list*

limit(*Matrix1*, *Var*, *Point* [, *Direction*]) ⇒ *matrix*

Returns the limit requested.

Note: See also **Limit template**, page 6.

Direction: negative=from left, positive=from right, otherwise=both. (If omitted, *Direction* defaults to both.)

$\lim_{x \rightarrow 5} (2 \cdot x + 3)$	13
$\lim_{x \rightarrow 0^+} \left(\frac{1}{x} \right)$	∞
$\lim_{x \rightarrow 0} \left(\frac{\sin(x)}{x} \right)$	1
$\lim_{h \rightarrow 0} \left(\frac{\sin(x+h) - \sin(x)}{h} \right)$	$\cos(x)$
$\lim_{n \rightarrow \infty} \left(\left(1 + \frac{1}{n} \right)^n \right)$	e

Limits at positive ∞ and at negative ∞ are always converted to one-sided limits from the finite side.

Depending on the circumstances, **limit()** returns itself or `undef` when it cannot determine a unique limit. This does not necessarily mean that a unique limit does not exist. `undef` means that the result is either an unknown number with finite or infinite magnitude, or it is the entire set of such numbers.

limit() uses methods such as L'Hopital's rule, so there are unique limits that it cannot determine. If *Expr1* contains undefined variables other than *Var*, you might have to constrain them to obtain a more concise result.

Limits can be very sensitive to rounding error. When possible, avoid the Approximate setting of the **Auto or Approximate** mode and approximate numbers when computing limits. Otherwise, limits that should be zero or have infinite magnitude probably will not, and limits that should have finite non-zero magnitude might not.

$\lim_{x \rightarrow \infty} (a^x)$	undef
$\lim_{x \rightarrow \infty} (a^x) a > 1$	∞
$\lim_{x \rightarrow \infty} (a^x) a > 0 \text{ and } a < 1$	0

LinRegBx

LinRegBx *X*,*Y*, [*Freq*][, *Category*, *Include*]

Computes the linear regression $y = a + b \cdot x$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $a+b \cdot x$
stat.a, stat.b	Regression coefficients
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

LinRegMx *X*,*Y*,[*Freq*],[*Category*,*Include*]

Computes the linear regression $y = m \cdot x + b$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and Y are lists of independent and dependent variables.

$Freq$ is an optional list of frequency values. Each element in $Freq$ specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

$Category$ is a list of category codes for the corresponding X and Y data.

$Include$ is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $y = m \cdot x + b$
stat.m, stat.b	Regression coefficients
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.FreqReg	List of frequencies corresponding to $stat.XReg$ and $stat.YReg$

LinRegtIntervals

LinRegtIntervals $X, Y, F[, 0, CLev]]$

For Slope. Computes a level C confidence interval for the slope.

LinRegtIntervals $X, Y, F[, 1, Xval[, CLev]]$

For Response. Computes a predicted y -value, a level C prediction interval for a single observation, and a level C confidence interval for the mean response.

A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension.

X and Y are lists of independent and dependent variables.

F is an optional list of frequency values. Each element in F specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $a+b \cdot x$
stat.a, stat.b	Regression coefficients
stat.df	Degrees of freedom
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression

For Slope type only

Output variable	Description
[stat.CLower, stat.CUpper]	Confidence interval for the slope
stat.ME	Confidence interval margin of error
stat.SESlope	Standard error of slope
stat.s	Standard error about the line

For Response type only

Output variable	Description
[stat.CLower, stat.CUpper]	Confidence interval for the mean response
stat.ME	Confidence interval margin of error
stat.SE	Standard error of mean response
[stat.LowerPred, stat.UpperPred]	Prediction interval for a single observation
stat.MEPred	Prediction interval margin of error
stat.SEPred	Standard error for prediction
stat. \hat{y}	$a + b \cdot X_{\text{Val}}$

LinRegtTest

Catalog > 

LinRegtTest $X, Y[, Freq[, Hypoth]]$

Computes a linear regression on the X and Y lists and a t test on the value of slope β and the correlation coefficient ρ for the equation $y = \alpha + \beta x$. It tests the null hypothesis $H_0: \beta = 0$ (equivalently, $\rho = 0$) against one of three alternative hypotheses.

All the lists must have equal dimension.

X and Y are lists of independent and dependent variables.

$Freq$ is an optional list of frequency values. Each element in $Freq$ specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

$Hypoth$ is an optional value specifying one of three alternative hypotheses against which the null hypothesis ($H_0: \beta = \rho = 0$) will be tested.

For $H_a: \beta \neq 0$ and $\rho \neq 0$ (default), set $Hypoth = 0$

For $H_a: \beta < 0$ and $\rho < 0$, set $Hypoth < 0$

For $H_a: \beta > 0$ and $\rho > 0$, set $Hypoth > 0$

A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a + b \cdot x$
stat.t	t -Statistic for significance test
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom
stat.a, stat.b	Regression coefficients
stat.s	Standard error about the line
stat.SESlope	Standard error of slope
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression

linSolve()

linSolve(*SystemOfLinearEqns*, *Var1*, *Var2*, ...) \Rightarrow list

$$\text{linSolve}\left(\left\{\begin{array}{l} 2 \cdot x + 4 \cdot y = 3 \\ 5 \cdot x - 3 \cdot y = 7 \end{array}, \{x, y\}\right\}, \left\{\frac{37}{26}, \frac{1}{26}\right\}\right)$$

linSolve(*LinearEqn1* and *LinearEqn2* and ..., *Var1*, *Var2*, ...) \Rightarrow list

$$\text{linSolve}\left(\left\{\begin{array}{l} 2 \cdot x = 3 \\ 5 \cdot x - 3 \cdot y = 7 \end{array}, \{x, y\}\right\}, \left\{\frac{3}{2}, \frac{1}{6}\right\}\right)$$

linSolve({*LinearEqn1*, *LinearEqn2*, ...}, *Var1*, *Var2*, ...) \Rightarrow list

$$\text{linSolve}\left(\left\{\begin{array}{l} \text{apple} + 4 \cdot \text{pear} = 23 \\ 5 \cdot \text{apple} - \text{pear} = 17 \end{array}, \{\text{apple}, \text{pear}\}\right\}, \left\{\frac{13}{3}, \frac{14}{3}\right\}\right)$$

linSolve(*SystemOfLinearEqns*, {*Var1*, *Var2*, ...}) \Rightarrow list

$$\text{linSolve}\left(\left\{\begin{array}{l} \text{apple} \cdot 4 + \frac{\text{pear}}{3} = 14 \\ -\text{apple} + \text{pear} = 6 \end{array}, \{\text{apple}, \text{pear}\}\right\}, \left\{\frac{36}{13}, \frac{114}{13}\right\}\right)$$

linSolve(*LinearEqn1* and *LinearEqn2* and ..., {*Var1*, *Var2*, ...}) \Rightarrow list

linSolve({*LinearEqn1*, *LinearEqn2*, ...}, {*Var1*, *Var2*, ...}) \Rightarrow list

Returns a list of solutions for the variables *Var1*, *Var2*, ...

linSolve()

Catalog > 

The first argument must evaluate to a system of linear equations or a single linear equation. Otherwise, an argument error occurs.

For example, evaluating `linSolve(x=1 and x=2, x)` produces an "Argument Error" result.

ΔList()

Catalog > 

$\Delta\text{List}(List1) \Rightarrow list$

$\Delta\text{List}(\{20,30,45,70\})$	$\{10,15,25\}$
--------------------------------------	----------------

Note: You can insert this function from the keyboard by typing `deltaList(...)`.

Returns a list containing the differences between consecutive elements in *List1*. Each element of *List1* is subtracted from the next element of *List1*. The resulting list is always one element shorter than the original *List1*.

list ► mat()

Catalog > 

$\text{list} \blacktriangleright \text{mat}(List [, elementsPerRow]) \Rightarrow matrix$

$\text{list} \blacktriangleright \text{mat}(\{1,2,3\})$	$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$
$\text{list} \blacktriangleright \text{mat}(\{1,2,3,4,5\},2)$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 0 \end{bmatrix}$

Returns a matrix filled row-by-row with the elements from *List*.

elementsPerRow, if included, specifies the number of elements per row. Default is the number of elements in *List* (one row).

If *List* does not fill the resulting matrix, zeros are added.

Note: You can insert this function from the computer keyboard by typing `list@>mat(...)`.

► ln

Catalog > 

$Expr \blacktriangleright \ln \Rightarrow expression$

$\left(\log_{10}(x)\right) \blacktriangleright \ln$	$\frac{\ln(x)}{\ln(10)}$
---	--------------------------

Causes the input *Expr* to be converted to an expression containing only natural logs (ln).

Note: You can insert this operator from the computer keyboard by typing @>1n.

ln()ctrl  keys

$\ln(\text{Expr}1) \Rightarrow \text{expression}$

$\ln(2.)$ 0.693147

$\ln(\text{List}1) \Rightarrow \text{list}$

Returns the natural logarithm of the argument.

If complex format mode is Real:

$\ln(\{-3,1,2,5\})$
"Error: Non-real calculation"

For a list, returns the natural logarithms of the elements.

If complex format mode is Rectangular:

$\ln(\{-3,1,2,5\})$ { $\ln(3)+\pi \cdot i, 0.182322, \ln(5)$ }

$\ln(\text{squareMatrix}1) \Rightarrow \text{squareMatrix}$


Returns the matrix natural logarithm of *squareMatrix1*. This is not the same as calculating the natural logarithm of each element. For information about the calculation method, refer to **cos()** on.

In Radian angle mode and Rectangular complex format:

$\ln\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$
 $\begin{bmatrix} 1.83145+1.73485 \cdot i & 0.009193-1.49086 \\ 0.448761-0.725533 \cdot i & 1.06491+0.623491 \cdot i \\ -0.266891-2.08316 \cdot i & 1.12436+1.79018 \cdot i \end{bmatrix}$

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

LnRegCatalog > 

LnReg *X*, *Y* [, [*Freq*] [, [*Category*, *Include*]]]

Computes the logarithmic regression $y = a+b \cdot \ln(x)$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a+b \cdot \ln(x)$
stat.a, stat.b	Regression coefficients
stat.r ²	Coefficient of linear determination for transformed data
stat.r	Correlation coefficient for transformed data ($\ln(x)$, y)
stat.Resid	Residuals associated with the logarithmic model
stat.ResidTrans	Residuals associated with linear fit of transformed data
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

Local *Var1* [, *Var2*] [, *Var3*] ...

Declares the specified *vars* as local variables. Those variables exist only during evaluation of a function and are deleted when the function finishes execution.

Note: Local variables save memory because they only exist temporarily. Also, they do not disturb any existing global variable values. Local variables must be used for **For** loops and for temporarily saving values in a multi-line function since modifications on global variables are not allowed in a function.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define *rollcount()*=FuncLocal *i**1* → *i*

Loop

If *randInt(1,6)*=*randInt(1,6)*Goto *end**i*+1 → *i*

EndLoop

Lbl *end*Return *i*

EndFunc

*Done**rollcount()*

16

rollcount()

3

Lock**Lock** *Var1* [, *Var2*] [, *Var3*] ...**Lock** *Var*.

Locks the specified variables or variable group. Locked variables cannot be modified or deleted.

You cannot lock or unlock the system variable *Ans*, and you cannot lock the system variable groups *stat.* or *tvm.*

Note: The **Lock** command clears the Undo/Redo history when applied to unlocked variables.

See **unLock**, page 197, and **getLockInfo()**, page 83.

a:=65

65

Lock *a**Done*getLockInfo(*a*)

1

a:=75

"Error: Variable is locked."

DelVar *a*

"Error: Variable is locked."

Unlock *a**Done**a*:=75

75

DelVar *a**Done*

log()

ctrl **10^x** **keys**

$\log(\text{Expr1}[, \text{Expr2}]) \Rightarrow \text{expression}$

$$\log_{10} (2.) \quad 0.30103$$

$\log(\text{List1}[, \text{Expr2}]) \Rightarrow \text{list}$

$$\log_4 (2.) \quad 0.5$$

Returns the base-*Expr2* logarithm of the first argument.

$$\log_3 (10) - \log_3 (5) \quad \log_3 (2)$$

Note: See also **Log template**, page 2.

If complex format mode is Real:

For a list, returns the base-*Expr2* logarithm of the elements.

$$\log_{10} (\{-3, 1.2, 5\}) \quad \text{Error: Non-real result}$$

If the second argument is omitted, 10 is used as the base.

If complex format mode is Rectangular:

$$\log_{10} (\{-3, 1.2, 5\}) \\ \left\{ \log_{10} (3) + 1.36438 \cdot i, 0.079181, \log_{10} (5) \right\}$$

$\log(\text{squareMatrix1}[, \text{Expr}]) \Rightarrow \text{squareMatrix}$

In Radian angle mode and Rectangular complex format:

Returns the matrix base-*Expr* logarithm of *squareMatrix1*. This is not the same as calculating the base-*Expr* logarithm of each element. For information about the calculation method, refer to **cos()**.

$$\log_{10} \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \\ \begin{bmatrix} 0.795387 + 0.753438 \cdot i & 0.003993 - 0.6474 \cdot i \\ 0.194895 - 0.315095 \cdot i & 0.462485 + 0.2707 \cdot i \\ -0.115909 - 0.904706 \cdot i & 0.488304 + 0.7774 \cdot i \end{bmatrix}$$

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

If the base argument is omitted, 10 is used as base.

► logbase

Catalog >

$\text{Expr} \blacktriangleright \logbase(\text{Expr1}) \Rightarrow \text{expression}$

$$\log_3 (10) - \log_5 (5) \blacktriangleright \logbase(5) \quad \frac{\log_5 (10)}{\log_5 (3)}$$

Causes the input Expression to be simplified to an expression using base *Expr1*.

Note: You can insert this operator from the computer keyboard by typing **@>logbase (...)**.

Logistic $X, Y, [Freq] [, Category, Include]$

Computes the logistic regression $y = c / (1 + a \cdot e^{-bx})$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and Y are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding X and Y data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

Output variable	Description
stat.RegEqn	Regression equation: $c / (1 + a \cdot e^{-bx})$
stat.a, stat.b, stat.c	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

LogisticD *X*, *Y* [, [*Iterations*] , [*Freq*] [, *Category*, *Include*]]

Computes the logistic regression $y = (c / (1+a \cdot e^{-bx}) + d)$ on lists *X* and *Y* with frequency *Freq*, using a specified number of *Iterations*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

Output variable	Description
stat.RegEqn	Regression equation: $c / (1+a \cdot e^{-bx}) + d$
stat.a, stat.b, stat.c, stat.d	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

Loop

Block

EndLoop

Repeatedly executes the statements in *Block*. Note that the loop will be executed endlessly, unless a **Goto** or **Exit** instruction is executed within *Block*.

Block is a sequence of statements separated with the “.” character.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

```
Define rollcount()=Func
    Local i
    1 → i
    Loop
    If randInt(1,6)=randInt(1,6)
    Goto end
    i+1 → i
    EndLoop
    Lbl end
    Return i
EndFunc
Done
rollcount() 16
rollcount() 3
```

LU

lMatrix, *lMatrix*, *uMatrix*, *pMatrix* [,*Tol*]

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

$$lMatrix \cdot uMatrix = pMatrix \cdot matrix$$

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

$\begin{bmatrix} 6 & 12 & 18 \\ 5 & 14 & 31 \\ 3 & 8 & 18 \end{bmatrix}$	$\rightarrow m1$	$\begin{bmatrix} 6 & 12 & 18 \\ 5 & 14 & 31 \\ 3 & 8 & 18 \end{bmatrix}$
LU <i>m1</i> , <i>lower</i> , <i>upper</i> , <i>perm</i>		Done
<i>lower</i>		$\begin{bmatrix} 1 & 0 & 0 \\ \frac{5}{6} & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$
<i>upper</i>		$\begin{bmatrix} 6 & 12 & 18 \\ 0 & 4 & 16 \\ 0 & 0 & 1 \end{bmatrix}$
<i>perm</i>		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- If you use or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\dim(Matrix)) \cdot \text{rowNorm}(Matrix)$

The **LU** factorization algorithm uses partial pivoting with row interchanges.

$\begin{bmatrix} m & n \\ o & p \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} m & n \\ o & p \end{bmatrix}$
LU <i>m1,lower,upper,perm</i>	Done
<i>lower</i>	$\begin{bmatrix} 1 & 0 \\ m & 1 \\ o & \end{bmatrix}$
<i>upper</i>	$\begin{bmatrix} o & p \\ 0 & n - \frac{m \cdot p}{o} \end{bmatrix}$
<i>perm</i>	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

M

mat ▶ list()

mat ▶ list(*Matrix*) ⇒ *list*

Returns a list filled with the elements in *Matrix*. The elements are copied from *Matrix* row by row.

Note: You can insert this function from the computer keyboard by typing **mat@>list** (...).

mat▶list([1 2 3])	{1,2,3}
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
mat▶list(m1)	{1,2,3,4,5,6}

max()

max(*Expr1, Expr2*) ⇒ *expression*

max(*List1, List2*) ⇒ *list*

max(*Matrix1, Matrix2*) ⇒ *matrix*

Returns the maximum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the maximum value of each pair of corresponding elements.

max(*List*) ⇒ *expression*

Returns the maximum element in *list*.

max(*Matrix1*) ⇒ *matrix*

Returns a row vector containing the maximum element of each column in *Matrix1*.

max(2.3,1.4)	2.3
max({1,2},{-4,3})	{1,3}
max({0,1,-7,1.3,0.5})	1.3
max($\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}$)	$\begin{bmatrix} 1 & 0 & 7 \end{bmatrix}$

max()Catalog > 

Empty (void) elements are ignored. For more information on empty elements, see page 251.

Note: See also **fMax()** and **min()**.

mean()Catalog > 

mean(List[,freqList]) ⇒ *expression*

Returns the mean of the elements in *List*.

Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.

mean(MatrixI[,freqMatrix]) ⇒ *matrix*

Returns a row vector of the means of all the columns in *MatrixI*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *MatrixI*.

Empty (void) elements are ignored. For more information on empty elements, see page 251.

$\text{mean}\{\{0.2,0,1,-0.3,0.4\}\}$	0.26
$\text{mean}\{\{1,2,3\},\{3,2,1\}\}$	$\frac{5}{3}$

In Rectangular vector format:

$\text{mean}\left(\begin{bmatrix} 0.2 & 0 \\ -1 & 3 \\ 0.4 & -0.5 \end{bmatrix}\right)$	$[-0.133333 \quad 0.833333]$
$\text{mean}\left(\begin{bmatrix} \frac{1}{5} & 0 \\ -1 & 3 \\ \frac{2}{5} & \frac{-1}{2} \end{bmatrix}\right)$	$\left[\frac{-2}{15} \quad \frac{5}{6}\right]$
$\text{mean}\left(\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 4 & 4 & 1 \\ 5 & 6 & 6 & 2 \end{bmatrix}\right)$	$\left[\frac{47}{15} \quad \frac{11}{3}\right]$

median()Catalog > 

median(List[,freqList]) ⇒ *expression*

Returns the median of the elements in *List*.

Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.

median(MatrixI[,freqMatrix]) ⇒ *matrix*

Returns a row vector containing the medians of the columns in *MatrixI*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *MatrixI*.

$\text{median}\{\{0.2,0,1,-0.3,0.4\}\}$	0.2
---	-----

$\text{median}\left(\begin{bmatrix} 0.2 & 0 \\ 1 & -0.3 \\ 0.4 & -0.5 \end{bmatrix}\right)$	$[0.4 \quad -0.3]$
---	--------------------

Notes:

- All entries in the list or matrix must simplify to numbers.
- Empty (void) elements in the list or matrix are ignored. For more information on empty elements, see page 251.

MedMed**MedMed** X, Y [, *Freq*] [, *Category*, *Include*]

Computes the median-median line $y = (m \cdot x + b)$ on lists X and Y with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and Y are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding X and Y data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Median-median line equation: $m \cdot x + b$
stat.m, stat.b	Model coefficients

Output variable	Description
stat.Resid	Residuals from the median-median line
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

mid()

Catalog > 

mid(sourceString, Start[, Count]) ⇒ *string*

Returns *Count* characters from character string *sourceString*, beginning with character number *Start*.

If *Count* is omitted or is greater than the dimension of *sourceString*, returns all characters from *sourceString*, beginning with character number *Start*.

Count must be ≥ 0 . If *Count* = 0, returns an empty string.

mid(sourceList, Start [, Count]) ⇒ *list*

Returns *Count* elements from *sourceList*, beginning with element number *Start*.

If *Count* is omitted or is greater than the dimension of *sourceList*, returns all elements from *sourceList*, beginning with element number *Start*.

Count must be ≥ 0 . If *Count* = 0, returns an empty list.

mid(sourceStringList, Start[, Count]) ⇒ *list*

Returns *Count* strings from the list of strings *sourceStringList*, beginning with element number *Start*.

mid("Hello there",2)	"ello there"
mid("Hello there",7,3)	"the"
mid("Hello there",1,5)	"Hello"
mid("Hello there",1,0)	" "

mid({9,8,7,6},3)	{7,6}
mid({9,8,7,6},2,2)	{8,7}
mid({9,8,7,6},1,2)	{9,8}
mid({9,8,7,6},1,0)	{ }

mid({"A","B","C","D"},2,2)	{"B","C"}
----------------------------	-----------

min()Catalog > **min**(*Expr1*, *Expr2*) ⇒ *expression* $\text{min}(2.3, 1.4)$ 1.4**min**(*List1*, *List2*) ⇒ *list* $\text{min}(\{1,2\}, \{-4,3\})$ $\{-4,2\}$ **min**(*Matrix1*, *Matrix2*) ⇒ *matrix*

Returns the minimum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the minimum value of each pair of corresponding elements.

min(*List*) ⇒ *expression* $\text{min}(\{0,1,-7,1.3,0.5\})$ -7Returns the minimum element of *List*.**min**(*Matrix1*) ⇒ *matrix* $\text{min}\left(\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}\right)$ $\begin{bmatrix} -4 & -3 & 0.3 \end{bmatrix}$

Returns a row vector containing the minimum element of each column in *Matrix1*.

Note: See also **fMin()** and **max()**.**mirr()**Catalog > **mirr****(***financeRate*, *reinvestRate*, *CF0*, *CFList* [*CFFreq*]) $\text{list1} := \{6000, -8000, 2000, -3000\}$
 $\{6000, -8000, 2000, -3000\}$

Financial function that returns the modified internal rate of return of an investment.

 $\text{list2} := \{2, 2, 2, 1\}$ $\{2, 2, 2, 1\}$

financeRate is the interest rate that you pay on the cash flow amounts.

 $\text{mirr}(4.65, 12, 5000, \text{list1}, \text{list2})$ 13.41608607

reinvestRate is the interest rate at which the cash flows are reinvested.

CF0 is the initial cash flow at time 0; it must be a real number.

CFList is a list of cash flow amounts after the initial cash flow *CF0*.

CFFreq is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.

Note: See also irr(), page 93.

mod()

mod(*Expr1*, *Expr2*) \Rightarrow *expression*

mod(7,0)	7
----------	---

mod(*List1*, *List2*) \Rightarrow *list*

mod(7,3)	1
----------	---

mod(*Matrix1*, *Matrix2*) \Rightarrow *matrix*

mod(-7,3)	2
-----------	---

Returns the first argument modulo the second argument as defined by the identities:

mod(7,-3)	-2
-----------	----

$\text{mod}(x,0) = x$

mod(-7,-3)	-1
------------	----

$\text{mod}(x,y) = x - y \text{ floor}(x/y)$

mod({12,-14,16},{9,7,-5})	{3,0,-4}
---------------------------	----------

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

If the arguments are two lists or two matrices, returns a list or matrix containing the modulo of each pair of corresponding elements.

Note: See also remain(), page 149

mRow()

mRow(*Expr*, *Matrix1*, *Index*) \Rightarrow *matrix*

Returns a copy of *Matrix1* with each element in row *Index* of *Matrix1* multiplied by *Expr*.

mRow($\frac{1}{3}$, $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, 2)	$\begin{bmatrix} 1 & 2 \\ -1 & -4 \\ 3 & 3 \end{bmatrix}$
---	---

mRowAdd()

mRowAdd(*Expr*, *Matrix1*, *Index1*, *Index2*) \Rightarrow *matrix*

Returns a copy of *Matrix1* with each element in row *Index2* of *Matrix1* replaced with:

mRowAdd(-3, $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, 1, 2)	$\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix}$
--	---

mRowAdd(n , $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, 1, 2)	$\begin{bmatrix} a & b \\ a \cdot n + c & b \cdot n + d \end{bmatrix}$
---	--

$\text{Expr} \cdot \text{row } \text{Index1} + \text{row } \text{Index2}$

MultReg $Y, X1[,X2[,X3,...[,X10]]]$

Calculates multiple linear regression of list Y on lists $X1, X2, \dots, X10$. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $b_0+b_1 \cdot x_1+b_2 \cdot x_2+ \dots$
stat.b0, stat.b1, ...	Regression coefficients
stat.R ²	Coefficient of multiple determination
stat. \hat{y} List	\hat{y} List = $b_0+b_1 \cdot x_1+ \dots$
stat.Resid	Residuals from the regression

MultRegIntervals**MultRegIntervals** $Y, X1[, X2[, X3,...[, X10]]], XValList[, CLevel]$

Computes a predicted y -value, a level C prediction interval for a single observation, and a level C confidence interval for the mean response.

A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $b_0+b_1 \cdot x_1+b_2 \cdot x_2+ \dots$
stat. \hat{y}	A point estimate: $\hat{y} = b_0 + b_1 \cdot x_1 + \dots$ for <i>XValList</i>
stat.dfError	Error degrees of freedom

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for a mean response
stat.ME	Confidence interval margin of error
stat.SE	Standard error of mean response
stat.LowerPred, stat.UpperrPred	Prediction interval for a single observation
stat.MEPred	Prediction interval margin of error
stat.SEPred	Standard error for prediction
stat.bList	List of regression coefficients, {b0,b1,b2,...}
stat.Resid	Residuals from the regression

MultRegTests

Catalog > 

MultRegTests $Y, X1[, X2[, X3, \dots[, X10]]]$

Multiple linear regression test computes a multiple linear regression on the given data and provides the global F test statistic and t test statistics for the coefficients.

A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Outputs

Output variable	Description
stat.RegEqn	Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots$
stat.F	Global F test statistic
stat.PVal	P-value associated with global F statistic
stat.R ²	Coefficient of multiple determination
stat.AdjR ²	Adjusted coefficient of multiple determination
stat.s	Standard deviation of the error
stat.DW	Durbin-Watson statistic; used to determine whether first-order auto correlation is present in the model

Output variable	Description
stat.dfReg	Regression degrees of freedom
stat.SSReg	Regression sum of squares
stat.MSReg	Regression mean square
stat.dfError	Error degrees of freedom
stat.SSError	Error sum of squares
stat.MSError	Error mean square
stat.bList	{b0,b1,...} List of coefficients
stat.tList	List of t statistics, one for each coefficient in the bList
stat.PList	List P-values for each t statistic
stat.SEList	List of standard errors for coefficients in bList
stat.yList	\hat{y} List = $b_0 + b_1 \cdot x_1 + \dots$
stat.Resid	Residuals from the regression
stat.sResid	Standardized residuals; obtained by dividing a residual by its standard deviation
stat.CookDist	Cook's distance; measure of the influence of an observation based on the residual and leverage
stat.Leverage	Measure of how far the values of the independent variable are from their mean values

N

nand

  **keys**

BooleanExpr1 **nand** *BooleanExpr2* returns
Boolean expression

$x \geq 3$ and $x \geq 4$ $x \geq 4$

BooleanList1 **nand** *BooleanList2* returns
Boolean list

$x \geq 3$ nand $x \geq 4$ $x < 4$

BooleanMatrix1 **nand** *BooleanMatrix2*
returns *Boolean matrix*

Returns the negation of a logical **and** operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

nand**ctrl** **=** **keys***Integer1 nand Integer2* ⇒ *integer*

Compares two real integers bit-by-bit using a **nand** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 0 if both bits are 1; otherwise, the result is 1. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

3 and 4	0
3 nand 4	-1
{1,2,3} and {3,2,1}	{1,2,1}
{1,2,3} nand {3,2,1}	{-2,-3,-2}

nCr()**Catalog** > *nCr(Expr1, Expr2)* ⇒ *expression*

For integer *Expr1* and *Expr2* with *Expr1* ≥ *Expr2* ≥ 0, **nCr()** is the number of combinations of *Expr1* things taken *Expr2* at a time. (This is also known as a binomial coefficient.) Both arguments can be integers or symbolic expressions.

$nCr(z,3)$	$\frac{z \cdot (z-2) \cdot (z-1)}{6}$
$Ans z=5$	10
$nCr(z,c)$	$\frac{z!}{c! \cdot (z-c)!}$
$\frac{Ans}{nPr(z,c)}$	$\frac{1}{c!}$

nCr(Expr, 0) ⇒ **1***nCr(Expr, negInteger)* ⇒ **0***nCr(Expr, posInteger)* ⇒ *Expr*•(*Expr*-1) ... (*Expr*-*posInteger*+1) / *posInteger*!*nCr(Expr, nonInteger)* ⇒ *expression*! / ((*Expr*-*nonInteger*)!•*nonInteger*!)*nCr(List1, List2)* ⇒ *list*

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

$nCr(\{5,4,3\},\{2,4,2\})$	{10,1,3}
----------------------------	----------

nCr(Matrix1, Matrix2) ⇒ *matrix*

$nCr\left(\begin{bmatrix} 6 & 5 \\ 4 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}\right)$	$\begin{bmatrix} 15 & 10 \\ 6 & 3 \end{bmatrix}$
--	--

nCr()

Catalog >

Returns a matrix of combinations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

nDerivative()

Catalog >

nDerivative(*Expr1*, *Var*=*Value*[, *Order*])
 \Rightarrow *value*

$nDerivative(x , x=1)$	1
-------------------------	---

$nDerivative(x , x) _{x=0}$	undef
------------------------------	-------

nDerivative(*Expr1*, *Var*[, *Order*])
 | *Var*=*Value* \Rightarrow *value*

$nDerivative(\sqrt{x-1}, x) _{x=1}$	undef
-------------------------------------	-------

Returns the numerical derivative calculated using auto differentiation methods.

When *Value* is specified, it overrides any prior variable assignment or any current “|” substitution for the variable.

Order of the derivative must be 1 or 2.

newList()

Catalog >

newList(*numElements*) \Rightarrow *list*

$newList(4)$	{0,0,0,0}
--------------	-----------

Returns a list with a dimension of *numElements*. Each element is zero.

newMat()

Catalog >

newMat(*numRows*, *numColumns*) \Rightarrow
matrix

$newMat(2,3)$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
---------------	--

Returns a matrix of zeros with the dimension *numRows* by *numColumns*.

nfMax()

Catalog >

nfMax(*Expr*, *Var*) \Rightarrow *value*

nfMax(*Expr*, *Var*, *lowBound*) \Rightarrow *value*

nfMax(*Expr*, *Var*, *lowBound*, *upBound*) \Rightarrow
value

nfMax(*Expr*, *Var*) |

lowBound \leq *Var* \leq *upBound* \Rightarrow *value*

$nfMax(-x^2-2 \cdot x-1, x)$	-1.
------------------------------	-----

$nfMax(0.5 \cdot x^3-x-2, x, -5, 5)$	5.
--------------------------------------	----

nfMax()Catalog > 

Returns a candidate numerical value of variable *Var* where the local maximum of *Expr* occurs.

If you supply *lowBound* and *upBound*, the function looks in the closed interval [*lowBound*,*upBound*] for the local maximum.

Note: See also **fMax()** and **d()**.

nfMin()Catalog > 

nfMin(*Expr*, *Var*) ⇒ *value*

nfMin(*Expr*, *Var*, *lowBound*) ⇒ *value*

nfMin(*Expr*, *Var*, *lowBound*, *upBound*) ⇒ *value*

nfMin(*Expr*, *Var*) |

lowBound ≤ *Var* ≤ *upBound* ⇒ *value*

Returns a candidate numerical value of variable *Var* where the local minimum of *Expr* occurs.

If you supply *lowBound* and *upBound*, the function looks in the closed interval [*lowBound*,*upBound*] for the local minimum.

Note: See also **fMin()** and **d()**.

$\text{nfMin}(x^2 + 2 \cdot x + 5, x)$	-1.
$\text{nfMin}(0.5 \cdot x^3 - x - 2, x, -5, 5)$	-5.

nInt()Catalog > 

nInt(*Expr1*, *Var*, *Lower*, *Upper*) ⇒ *expression*

$\text{nInt}(e^{-x^2}, x, -1, 1)$	1.49365
-----------------------------------	---------

If the integrand *Expr1* contains no variable other than *Var*, and if *Lower* and *Upper* are constants, positive ∞ , or negative ∞ , then **nInt()** returns an approximation of $\int (\text{Expr1}, \text{Var}, \text{Lower}, \text{Upper})$. This approximation is a weighted average of some sample values of the integrand in the interval *Lower* < *Var* < *Upper*.

nInt()

Catalog > 

The goal is six significant digits. The adaptive algorithm terminates when it seems likely that the goal has been achieved, or when it seems unlikely that additional samples will yield a worthwhile improvement.

$$\text{nInt}(\cos(x), x, \pi, \pi+1. \epsilon^{-12}) \quad -1.04144 \epsilon^{-12}$$

$$\int_{-\pi}^{\pi+10^{-12}} \cos(x) dx \quad -\sin\left(\frac{1}{1000000000000}\right)$$

A warning is displayed (“Questionable accuracy”) when it seems that the goal has not been achieved.

Nest **nInt()** to do multiple numeric integration. Integration limits can depend on integration variables outside them.

$$\text{nInt}\left(\text{nInt}\left(\frac{e^{-x \cdot y}}{\sqrt{x^2 - y^2}}, y, -x, x\right), x, 0, 1\right) \quad 3.30423$$

Note: See also **∫()**, page 221.

nom()

Catalog > 

nom(*effectiveRate*, *CpY*) ⇒ *value*

$$\text{nom}(5.90398, 12) \quad 5.75$$

Financial function that converts the annual effective interest rate *effectiveRate* to a nominal rate, given *CpY* as the number of compounding periods per year.

effectiveRate must be a real number, and *CpY* must be a real number > 0.

Note: See also **eff()**, page 58.

nor

  **keys**

BooleanExpr1 **nor** *BooleanExpr2* returns *Boolean expression*

$$x \geq 3 \text{ or } x \geq 4 \quad x \geq 3$$

BooleanList1 **nor** *BooleanList2* returns *Boolean list*

$$x \geq 3 \text{ nor } x \geq 4 \quad x < 3$$

BooleanMatrix1 **nor** *BooleanMatrix2* returns *Boolean matrix*

Returns the negation of a logical **or** operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

norctrl  keys*Integer1* **nor** *Integer2* ⇒ *integer*

3 or 4	7
3 nor 4	-8
{1,2,3} or {3,2,1}	{3,2,3}
{1,2,3} nor {3,2,1}	{-4,-3,-4}

Compares two real integers bit-by-bit using a **nor** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

norm()Catalog > **norm**(*Matrix*) ⇒ *expression*

$$\text{norm}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) \quad \sqrt{a^2+b^2+c^2+d^2}$$

norm(*Vector*) ⇒ *expression*

$$\text{norm}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right) \quad \sqrt{30}$$

Returns the Frobenius norm.

$$\text{norm}\left(\begin{bmatrix} 1 & 2 \end{bmatrix}\right) \quad \sqrt{5}$$

$$\text{norm}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) \quad \sqrt{5}$$

normalLine()Catalog > **normalLine**(*Expr1*,*Var*,*Point*) ⇒ *expression*

$$\text{normalLine}(x^2,x,1) \quad \frac{3}{2} \frac{x}{2}$$

normalLine(*Expr1*,*Var=Point*) ⇒ *expression*

$$\text{normalLine}((x-3)^2-4,x,3) \quad x=3$$

Returns the normal line to the curve represented by *Expr1* at the point specified in *Var=Point*.

$$\text{normalLine}\left(x^{\frac{1}{3}},x=0\right) \quad 0$$

$$\text{normalLine}(\sqrt{|x|},x=0) \quad \text{undef}$$

Make sure that the independent variable is not defined. For example, if $f_1(x):=5$ and $x:=3$, then **normalLine**($f_1(x),x,2$) returns "false."

normCdf()

Catalog > 

normCdf(*lowBound*,*upBound* [, μ [, σ]]) \Rightarrow *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

Computes the normal distribution probability between *lowBound* and *upBound* for the specified μ (default=0) and σ (default=1).

For $P(X \leq \textit{upBound})$, set *lowBound* = $-\infty$.

normPdf()

Catalog > 

normPdf(*XVal* [, μ [, σ]]) \Rightarrow *number* if *XVal* is a number, *list* if *XVal* is a list

Computes the probability density function for the normal distribution at a specified *XVal* value for the specified μ and σ .

not

Catalog > 

not *BooleanExpr* \Rightarrow *Boolean expression*

Returns true, false, or a simplified form of the argument.

not *Integer1* \Rightarrow *integer*

Returns the one's complement of a real integer. Internally, *Integer1* is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one's complement. Results are displayed according to the Base mode.

You can enter the integer in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, the integer is treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see **► Base2**, page 17.

not(2 \geq 3)	true
not(x<2)	x \geq 2
not not <i>innocent</i>	<i>innocent</i>

In Hex base mode:

Important: Zero, not the letter O.

not 0h7AC36	0hFFFFFFFFFFFF853C9
-------------	---------------------

In Bin base mode:

0b100101	►Base10	37
not 0b100101		
0b11111111111111111111111111111111	►	
not 0b100101	►Base10	-38

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.

Note: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

nPr()

Catalog >

nPr(Expr1, Expr2) ⇒ expression

For integer *Expr1* and *Expr2* with *Expr1* ≥ *Expr2* ≥ 0, **nPr()** is the number of permutations of *Expr1* things taken *Expr2* at a time. Both arguments can be integers or symbolic expressions.

nPr(z,3)	$z \cdot (z-2) \cdot (z-1)$
Ans z=5	60
nPr(z,-3)	$\frac{1}{(z+1) \cdot (z+2) \cdot (z+3)}$
nPr(z,c)	$\frac{z!}{(z-c)!}$
Ans·nPr(z-c,-c)	1

nPr(Expr, 0) ⇒ 1**nPr(Expr, negInteger) ⇒ 1 / ((Expr+1)·(Expr+2) ... (expression-negInteger))****nPr(Expr, posInteger) ⇒ Expr·(Expr-1) ... (Expr-posInteger+1)****nPr(Expr, nonInteger) ⇒ Expr! / (Expr-nonInteger)!****nPr(List1, List2) ⇒ list**

Returns a list of permutations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

nPr({5,4,3},{2,4,2})	{20,24,6}
----------------------	-----------

nPr(Matrix1, Matrix2) ⇒ matrix

Returns a matrix of permutations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

nPr($\begin{bmatrix} 6 & 5 \\ 4 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$)	$\begin{bmatrix} 30 & 20 \\ 12 & 6 \end{bmatrix}$
---	---

npv()

Catalog >

npv(InterestRate,CFO,CFList[,CFFreq])

Financial function that calculates net present value; the sum of the present values for the cash inflows and outflows. A positive result for npv indicates a profitable investment.

list1:= {6000,-8000,2000,-3000}	{6000,-8000,2000,-3000}
list2:= {2,2,2,1}	{2,2,2,1}
npv(10,5000,list1,list2)	4769.91

InterestRate is the rate by which to discount the cash flows (the cost of money) over one period.

CFO is the initial cash flow at time 0; it must be a real number.

CFList is a list of cash flow amounts after the initial cash flow *CFO*.

CFFreq is a list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.

nSolve()

nSolve(*Equation*,*Var*[=*Guess*]) ⇒ *number* or *error_string*

$$\text{nSolve}(x^2+5\cdot x-25=9,x) \quad 3.84429$$

nSolve(*Equation*,*Var*[=*Guess*],*lowBound*) ⇒ *number* or *error_string*

$$\text{nSolve}(x^2=4,x=-1) \quad -2.$$

$$\text{nSolve}(x^2=4,x=1) \quad 2.$$

nSolve(*Equation*,*Var*[=*Guess*],*lowBound*,*upBound*) ⇒ *number* or *error_string*

Note: If there are multiple solutions, you can use a guess to help find a particular solution.

nSolve(*Equation*,*Var*[=*Guess*]) | *lowBound* ≤ *Var* ≤ *upBound* ⇒ *number* or *error_string*

Iteratively searches for one approximate real numeric solution to *Equation* for its one variable. Specify the variable as:

variable

– or –

variable = *real number*

For example, x is valid and so is x=3.

nSolve() is often much faster than **solve()** or **zeros()**, particularly if the “|” operator is used to constrain the search to a small interval containing exactly one simple solution.

$$\text{nSolve}(x^2+5\cdot x-25=9,x)|x<0 \quad -8.84429$$

$$\text{nSolve}\left(\frac{(1+r)^{24}-1}{r}=26,r\right)|r>0 \text{ and } r<0.25$$

0.006886

$$\text{nSolve}(x^2=-1,x) \quad \text{"No solution found"}$$

nSolve() attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string “no solution found.”

Note: See also `cSolve()`, `cZeros()`, `solve()`, and `zeros()`.

O

OneVar

OneVar [**1**],*X*],[*Freq*],[*Category*],[*Include*]]

OneVar [*n*],*X1*,*X2*[*X3*[...[,*X20*]]]

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of numeric category codes for the corresponding *X* values.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists *X*, *Freq*, or *Category* results in a void for the corresponding element of all those lists. An empty element in any of the lists *X1* through *X20* results in a void for the corresponding element of all those lists. For more information on empty elements, see page 251.

Output variable	Description
stat. \bar{x}	Mean of <i>x</i> values
stat. Σx	Sum of <i>x</i> values
stat. Σx^2	Sum of <i>x</i> ² values

Output variable	Description
stat.sx	Sample standard deviation of x
stat.σx	Population standard deviation of x
stat.n	Number of data points
stat.MinX	Minimum of x values
stat.Q ₁ X	1st Quartile of x
stat.MedianX	Median of x
stat.Q ₃ X	3rd Quartile of x
stat.MaxX	Maximum of x values
stat.SSX	Sum of squares of deviations from the mean of x

or

Catalog > 

BooleanExpr1 or *BooleanExpr2* returns *Boolean expression*

BooleanList1 or *BooleanList2* returns *Boolean list*

BooleanMatrix1 or *BooleanMatrix2* returns *Boolean matrix*

Returns true or false or a simplified form of the original entry.

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See *xor*.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Integer1 or *Integer2* ⇒ *integer*

$x \geq 3$ or $x \geq 4$ $x \geq 3$

```
Define g(x)=Func Done
  If x≤0 or x≥5
  Goto end
  Return x·3
 Lbl end
EndFunc
```

$g(3)$ 9

$g(0)$ *A function did not return a value*

In Hex base mode:

0h7AC36 or 0h3D5F 0h7BD7F

Important: Zero, not the letter O.

In Bin base mode:

0b100101 or 0b100 0b100101

Compares two real integers bit-by-bit using an or operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ► **Base2**, page 17.

Note: See **xor**.

ord()

ord(*String*) \Rightarrow *integer*

ord(*List l*) \Rightarrow *list*

Returns the numeric code of the first character in character string *String*, or a list of the first characters of each list element.

<code>ord("hello")</code>	104
<code>char(104)</code>	"h"
<code>ord(char(24))</code>	24
<code>ord({"alpha", "beta"})</code>	{97,98}

P

P ► Rx()

P ► **Rx**(*rExpr*, θ *Expr*) \Rightarrow *expression*

P ► **Rx**(*rList*, θ *List*) \Rightarrow *list*

P ► **Rx**(*rMatrix*, θ *Matrix*) \Rightarrow *matrix*

Returns the equivalent x-coordinate of the (*r*, θ) pair.

In Radian angle mode:

<code>P ► Rx(r, θ)</code>	$\cos(\theta) \cdot r$
<code>P ► Rx(4,60°)</code>	2
<code>P ► Rx({-3,10,1.3}, {$\frac{\pi}{3}, \frac{\pi}{4}, 0$})</code>	$\left\{ \frac{-3}{2}, 5\sqrt{2}, 1.3 \right\}$

Note: The θ argument is interpreted as either a degree, gradian or radian angle, according to the current angle mode. If the argument is an expression, you can use $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode setting temporarily.

Note: You can insert this function from the computer keyboard by typing $\text{P@>Rx}(\dots)$.

P ▶ Ry(*rExpr*, θ *Expr*) \Rightarrow *expression*

In Radian angle mode:

P ▶ Ry(*rList*, θ *List*) \Rightarrow *list*

P ▶ Ry(*rMatrix*, θ *Matrix*) \Rightarrow *matrix*

P ▶ Ry(<i>r</i> , θ)	$\sin(\theta) \cdot r$
P ▶ Ry(4, 60°)	$2 \cdot \sqrt{3}$
P ▶ Ry($\left\{ \{-3, 10, 1.3\}, \left\{ \frac{\pi}{3}, \frac{\pi}{4}, 0 \right\} \right\}$)	$\left\{ \frac{-3 \cdot \sqrt{3}}{2}, -5 \cdot \sqrt{2}, 0 \right\}$

Returns the equivalent y-coordinate of the (*r*, θ) pair.

Note: The θ argument is interpreted as either a degree, radian or gradian angle, according to the current angle mode. If the argument is an expression, you can use $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode setting temporarily.

Note: You can insert this function from the computer keyboard by typing $\text{P@>Ry}(\dots)$.

PassErr

For an example of **PassErr**, See Example 2 under the **Try** command, page 191.

Passes an error to the next level.

If system variable *errCode* is zero, **PassErr** does not do anything.

The **Else** clause of the **Try...Else...EndTry** block should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. If there are no more pending **Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.

Note: See also **ClrErr**, page 25, and **Try**, page 191.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

piecewise()

piecewise(*Expr1* [, *Cond1* [, *Expr2* [, *Cond2* [, ...]]])

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

Note: See also **Piecewise template**, page 3.

Define $p(x) = \begin{cases} x, & x > 0 \\ \text{undef}, & x \leq 0 \end{cases}$	Done
$p(1)$	1
$p(-1)$	undef

poissCdf()

poissCdf(λ , *lowBound*, *upBound*) \Rightarrow *number*
if *lowBound* and *upBound* are numbers, *list*
if *lowBound* and *upBound* are lists

poissCdf(λ , *upBound*) for $P(0 \leq X \leq \text{upBound}) \Rightarrow$
number if *upBound* is a number, *list* if
upBound is a list

Computes a cumulative probability for the discrete Poisson distribution with specified mean λ .

For $P(X \leq \text{upBound})$, set *lowBound*=0

poissPdf()

poissPdf(λ , *XVal*) \Rightarrow *number* if *XVal* is a
number, *list* if *XVal* is a list

Computes a probability for the discrete Poisson distribution with the specified mean λ .

Vector ► Polar

$$\left[\begin{array}{c} 1 \\ 3 \end{array} \right] \text{►Polar} \quad \left[3.16228 \quad \angle 1.24905 \right]$$

Note: You can insert this operator from the computer keyboard by typing @>Polar.

Displays *vector* in polar form $[r \angle \theta]$. The vector must be of dimension 2 and can be a row or a column.

Note: ►Polar is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update *ans*.

Note: See also ►Rect, page 146.

complexValue ► Polar

Displays *complexVector* in polar form.

- Degree angle mode returns $(r \angle \theta)$.
- Radian angle mode returns $re^{i\theta}$.

complexValue can have any complex form. However, an $re^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use the parentheses for an $(r \angle \theta)$ polar entry.

In Radian angle mode:

$$\left((3+4i) \right) \text{►Polar} \quad e^{i \cdot \left(\frac{\pi}{2} - \tan^{-1} \left(\frac{3}{4} \right) \right)} \cdot 5$$

$$\left(\left(4 \angle \frac{\pi}{3} \right) \right) \text{►Polar} \quad \frac{i \cdot \pi}{e^3 \cdot 4}$$

In Gradian angle mode:

$$(4i) \text{►Polar} \quad (4 \angle 100.)$$

In Degree angle mode:

$$(3+4i) \text{►Polar} \quad \left(5 \angle 90 - \tan^{-1} \left(\frac{3}{4} \right) \right)$$

polyCoeffs()**polyCoeffs**(*Poly* [, *Var*]) ⇒ *list*

$$\text{polyCoeffs}(4x^2-3x+2,x) \quad \{4,-3,2\}$$

polyCoeffs()Catalog > 

Returns a list of the coefficients of polynomial *Poly* with respect to variable *Var*.

Poly must be a polynomial expression in *Var*. We recommend that you do not omit *Var* unless *Poly* is an expression in a single variable.

$$\text{polyCoeffs}\left((x-1)^2 \cdot (x+2)^3\right) \\ \{1, 4, 1, -10, -4, 8\}$$

Expands the polynomial and selects *x* for the omitted *Var*.

$$\text{polyCoeffs}\left((x+y+z)^2, x\right) \\ \{1, 2 \cdot (y+z), (y+z)^2\}$$

$$\text{polyCoeffs}\left((x+y+z)^2, y\right) \\ \{1, 2 \cdot (x+z), (x+z)^2\}$$

$$\text{polyCoeffs}\left((x+y+z)^2, z\right) \\ \{1, 2 \cdot (x+y), (x+y)^2\}$$

polyDegree()Catalog > 

polyDegree(*Poly* [, *Var*]) \Rightarrow *value*

Returns the degree of polynomial expression *Poly* with respect to variable *Var*. If you omit *Var*, the **polyDegree()** function selects a default from the variables contained in the polynomial *Poly*.

Poly must be a polynomial expression in *Var*. We recommend that you do not omit *Var* unless *Poly* is an expression in a single variable.

$$\text{polyDegree}(5) \quad 0$$

$$\text{polyDegree}(\ln(2) + \pi, x) \quad 0$$

Constant polynomials

$$\text{polyDegree}(4 \cdot x^2 - 3 \cdot x + 2, x) \quad 2$$

$$\text{polyDegree}\left((x-1)^2 \cdot (x+2)^3\right) \quad 5$$

$$\text{polyDegree}\left((x+y^2+z^3)^2, x\right) \quad 2$$

$$\text{polyDegree}\left((x+y^2+z^3)^2, y\right) \quad 4$$

$$\text{polyDegree}\left((x-1)^{10000}, x\right) \quad 10000$$

The degree can be extracted even though the coefficients cannot. This is because the degree can be extracted without expanding the polynomial.

polyEval()Catalog > **polyEval**(*List1*, *Expr1*) \Rightarrow *expression***polyEval**(*List1*, *List2*) \Rightarrow *expression*

Interprets the first argument as the coefficient of a descending-degree polynomial, and returns the polynomial evaluated for the value of the second argument.

$\text{polyEval}(\{a,b,c\},x)$	$a \cdot x^2 + b \cdot x + c$
$\text{polyEval}(\{1,2,3,4\},2)$	26
$\text{polyEval}(\{1,2,3,4\},\{2,-7\})$	$\{26,-262\}$

polyGcd()Catalog > **polyGcd**(*Expr1*, *Expr2*) \Rightarrow *expression*

Returns greatest common divisor of the two arguments.

Expr1 and *Expr2* must be polynomial expressions.

List, matrix, and Boolean arguments are not allowed.

$\text{polyGcd}(100,30)$	10
$\text{polyGcd}(x^2-1,x-1)$	$x-1$
$\text{polyGcd}(x^3-6 \cdot x^2+11 \cdot x-6,x^2-6 \cdot x+8)$	$x-2$

polyQuotient()Catalog > **polyQuotient**(*Poly1*, *Poly2* [, *Var*]) \Rightarrow *expression*

Returns the quotient of polynomial *Poly1* divided by polynomial *Poly2* with respect to the specified variable *Var*.

Poly1 and *Poly2* must be polynomial expressions in *Var*. We recommend that you do not omit *Var* unless *Poly1* and *Poly2* are expressions in the same single variable.

$\text{polyQuotient}(x-1,x-3)$	1
$\text{polyQuotient}(x-1,x^2-1)$	0
$\text{polyQuotient}(x^2-1,x-1)$	$x+1$
$\text{polyQuotient}(x^3-6 \cdot x^2+11 \cdot x-6,x^2-6 \cdot x+8)$	x
$\text{polyQuotient}((x-y) \cdot (y-z), x+y+z, x)$	$y-z$
$\text{polyQuotient}((x-y) \cdot (y-z), x+y+z, y)$	$2 \cdot x - y + 2 \cdot z$
$\text{polyQuotient}((x-y) \cdot (y-z), x+y+z, z)$	$-(x-y)$

polyRemainder()

Catalog > 

polyRemainder(*Poly1*,*Poly2* [,*Var*]) ⇒ *expression*

Returns the remainder of polynomial *Poly1* divided by polynomial *Poly2* with respect to the specified variable *Var*.

Poly1 and *Poly2* must be polynomial expressions in *Var*. We recommend that you do not omit *Var* unless *Poly1* and *Poly2* are expressions in the same single variable.

$\text{polyRemainder}(x-1, x-3)$	2
$\text{polyRemainder}(x-1, x^2-1)$	$x-1$
$\text{polyRemainder}(x^2-1, x-1)$	0
<hr/>	
$\text{polyRemainder}((x-y) \cdot (y-z), x+y+z, x)$	$-(y-z) \cdot (2 \cdot y+z)$
$\text{polyRemainder}((x-y) \cdot (y-z), x+y+z, y)$	$-2 \cdot x^2 - 5 \cdot x \cdot z - 2 \cdot z^2$
$\text{polyRemainder}((x-y) \cdot (y-z), x+y+z, z)$	$(x-y) \cdot (x+2 \cdot y)$

polyRoots()

Catalog > 

polyRoots(*Poly*,*Var*) ⇒ *list*

polyRoots(*ListOfCoeffs*) ⇒ *list*

The first syntax, **polyRoots**(*Poly*,*Var*), returns a list of real roots of polynomial *Poly* with respect to variable *Var*. If no real roots exist, returns an empty list: {}.

Poly must be a polynomial in one variable.

The second syntax, **polyRoots**(*ListOfCoeffs*), returns a list of real roots for the coefficients in *ListOfCoeffs*.

Note: See also **cPolyRoots()**, page 36.

$\text{polyRoots}(y^3+1, y)$	{-1}
$\text{cPolyRoots}(y^3+1, y)$	$\left\{-1, \frac{1}{2} - \frac{\sqrt{3}}{2}i, \frac{1}{2} + \frac{\sqrt{3}}{2}i\right\}$
$\text{polyRoots}(x^2+2 \cdot x+1, x)$	{-1, -1}
$\text{polyRoots}\{1, 2, 1\}$	{-1, -1}

PowerReg

Catalog > 

PowerReg *X*,*Y*, [*Freq*][, *Category*, *Include*]

Computes the power regression $y = a \cdot (x)^b$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot (x)^b$
stat.a, stat.b	Regression coefficients
stat.r ²	Coefficient of linear determination for transformed data
stat.r	Correlation coefficient for transformed data ($\ln(x)$, $\ln(y)$)
stat.Resid	Residuals associated with the power model
stat.ResidTrans	Residuals associated with linear fit of transformed data
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

Prgm
Block
EndPrgm

Calculate GCD and display intermediate results.

Template for creating a user-defined program. Must be used with the **Define**, **Define LibPub**, or **Define LibPriv** command.

Block can be a single statement, a series of statements separated with the “.” character, or a series of statements on separate lines.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

```
Define proggcd(a,b)=Prgm
  Local d
  While b≠0
  d:=mod(a,b)
  a:=b
  b:=d
  Disp a," ",b
  EndWhile
  Disp "GCD=",a
EndPrgm
```

Done

```
proggcd(4560,450)
-----
450 60
60 30
30 0
-----
GCD=30
-----
Done
```

prodSeq()See Π (), page 223.**Product (P1)**See Π (), page 223.**product()**

product(List[, Start[, End]]) \Rightarrow *expression*

Returns the product of the elements contained in *List*. *Start* and *End* are optional. They specify a range of elements.

product(Matrix1[, Start[, End]]) \Rightarrow *matrix*

Returns a row vector containing the products of the elements in the columns of *Matrix1*. *Start* and *end* are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 251.

product({{1,2,3,4}})	24
product({{2,x,y}})	2·x·y
product({{4,5,8,9}},2,3)	40
product($\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$)	[28 80 162]
product($\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$,1,2)	[4 10 18]

propFrac()

Catalog >

propFrac(*Expr1* [, *Var*]) \Rightarrow *expression*

propFrac(*rational_number*) returns *rational_number* as the sum of an integer and a fraction having the same sign and a greater denominator magnitude than numerator magnitude.

propFrac(*rational_expression*, *Var*) returns the sum of proper ratios and a polynomial with respect to *Var*. The degree of *Var* in the denominator exceeds the degree of *Var* in the numerator in each proper ratio. Similar powers of *Var* are collected. The terms and their factors are sorted with *Var* as the main variable.

If *Var* is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.

For rational expressions, **propFrac()** is a faster but less extreme alternative to **expand()**.

You can use the **propFrac()** function to represent mixed fractions and demonstrate addition and subtraction of mixed fractions.

$\text{propFrac}\left(\frac{4}{3}\right)$	$1 + \frac{1}{3}$
$\text{propFrac}\left(-\frac{4}{3}\right)$	$-1 - \frac{1}{3}$

$\text{propFrac}\left(\frac{x^2+x+1}{x+1} + \frac{y^2+y+1}{y+1}, x\right)$	
	$\frac{1}{x+1} + x + \frac{y^2+y+1}{y+1}$
$\text{propFrac}(\text{Ans})$	$\frac{1}{x+1} + x + \frac{1}{y+1} + y$

$\text{propFrac}\left(\frac{11}{7}\right)$	$1 + \frac{4}{7}$
$\text{propFrac}\left(3 + \frac{1}{11} + 5 + \frac{3}{4}\right)$	$8 + \frac{37}{44}$
$\text{propFrac}\left(3 + \frac{1}{11} - \left(5 + \frac{3}{4}\right)\right)$	$-2 - \frac{29}{44}$

Q**QR**

Catalog >

QR *Matrix*, *qMatrix*, *rMatrix* [, *Tol*]

Calculates the Householder QR factorization of a real or complex matrix. The resulting Q and R matrices are stored to the specified *Matrix*. The Q matrix is unitary. The R matrix is upper triangular.

The floating-point number (9.) in m1 causes results to be calculated in floating-point form.

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

- If you use or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\dim(\text{Matrix})) \cdot \text{rowNorm}(\text{Matrix})$

The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in *qMatName* are the orthonormal basis vectors that span the space defined by *matrix*.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
QR <i>m1,qm,rm</i>	Done
<i>qm</i>	$\begin{bmatrix} 0.123091 & 0.904534 & 0.408248 \\ 0.492366 & 0.301511 & -0.816497 \\ 0.86164 & -0.301511 & 0.408248 \end{bmatrix}$
<i>rm</i>	$\begin{bmatrix} 8.12404 & 9.60114 & 11.0782 \\ 0. & 0.904534 & 1.80907 \\ 0. & 0. & 0. \end{bmatrix}$

$\begin{bmatrix} m & n \\ o & p \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} m & n \\ o & p \end{bmatrix}$
QR <i>m1,qm,rm</i>	Done
<i>qm</i>	$\begin{bmatrix} \frac{m}{\sqrt{m^2+o^2}} & \frac{-\text{sign}(m \cdot p - n \cdot o) \cdot o}{\sqrt{m^2+o^2}} \\ \frac{o}{\sqrt{m^2+o^2}} & \frac{m \cdot \text{sign}(m \cdot p - n \cdot o)}{\sqrt{m^2+o^2}} \end{bmatrix}$
<i>rm</i>	$\begin{bmatrix} \sqrt{m^2+o^2} & \frac{m \cdot n + o \cdot p}{\sqrt{m^2+o^2}} \\ 0 & \frac{m \cdot p - n \cdot o}{\sqrt{m^2+o^2}} \end{bmatrix}$

QuadReg

QuadReg *X,Y[, Freq][, Category, Include]*

Computes the quadratic polynomial regression $y = a \cdot x^2 + b \cdot x + c$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot x^2 + b \cdot x + c$
stat.a, stat.b, stat.c	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

QuartReg

QuartReg *X*,*Y*[, *Freq*][, *Category*, *Include*]]

Computes the quartic polynomial regression $y = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and Y are lists of independent and dependent variables.

$Freq$ is an optional list of frequency values. Each element in $Freq$ specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0 .

$Category$ is a list of category codes for the corresponding X and Y data.

$Include$ is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 251.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$
stat.a, stat.b, stat.c, stat.d, stat.e	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.FreqReg	List of frequencies corresponding to $stat.XReg$ and $stat.YReg$

R

R▶ Pθ()

R▶ Pθ ($xExpr, yExpr$) \Rightarrow expression

In Degree angle mode:

R▶ Pθ ($xList, yList$) \Rightarrow list

R▶ Pθ ($xMatrix, yMatrix$) \Rightarrow matrix

$R \blacktriangleright P\theta(x, y)$

$$90 \cdot \text{sign}(y) - \tan^{-1}\left(\frac{x}{y}\right)$$

R ▶ Pθ()

Catalog >

Returns the equivalent θ -coordinate of the (x,y) pair arguments.

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

Note: You can insert this function from the computer keyboard by typing **R@>Ptheta** (...).

In Gradian angle mode:

$$\text{R}\blacktriangleright\text{P}\theta(x,y) \quad 100 \cdot \text{sign}(y) - \tan^{-1}\left(\frac{x}{y}\right)$$

In Radian angle mode:

$$\text{R}\blacktriangleright\text{P}\theta(3,2) \quad \tan^{-1}\left(\frac{2}{3}\right)$$

$$\text{R}\blacktriangleright\text{P}\theta\left(\left[3 \quad -4 \quad 2\right], \left[0 \quad \frac{\pi}{4} \quad 1.5\right]\right) \\ \left[0 \quad \tan^{-1}\left(\frac{16}{\pi}\right) + \frac{\pi}{2} \quad 0.643501\right]$$

R ▶ Pr()

Catalog >

R ▶ Pr ($xExpr, yExpr$) \Rightarrow *expression*

R ▶ Pr ($xList, yList$) \Rightarrow *list*

R ▶ Pr ($xMatrix, yMatrix$) \Rightarrow *matrix*

Returns the equivalent r -coordinate of the (x,y) pair arguments.

Note: You can insert this function from the computer keyboard by typing **R@>Pr** (...).

In Radian angle mode:

$$\text{R}\blacktriangleright\text{Pr}(3,2) \quad \sqrt{13}$$

$$\text{R}\blacktriangleright\text{Pr}(x,y) \quad \sqrt{x^2+y^2}$$

$$\text{R}\blacktriangleright\text{Pr}\left(\left[3 \quad -4 \quad 2\right], \left[0 \quad \frac{\pi}{4} \quad 1.5\right]\right) \\ \left[3 \quad \frac{\sqrt{\pi^2+256}}{4} \quad 2.5\right]$$

▶ Rad

Catalog >

Expr1 ▶ **Rad** \Rightarrow *expression*

Converts the argument to radian angle measure.

Note: You can insert this operator from the computer keyboard by typing **@>Rad**.

In Degree angle mode:

$$(1.5)\blacktriangleright\text{Rad} \quad (0.02618)^{\circ}$$

In Gradian angle mode:

$$(1.5)\blacktriangleright\text{Rad} \quad (0.023562)^{\text{g}}$$

rand()

Catalog >

rand() \Rightarrow *expression*

rand(#Trials) \Rightarrow *list*

Set the random-number seed.

rand()Catalog > 

rand() returns a random value between 0 and 1.

rand(*#Trials*) returns a list containing *#Trials* random values between 0 and 1.

RandSeed 1147

Done

rand(2)

{0.158206,0.717917}

randBin()Catalog > 

randBin(*n*, *p*) ⇒ *expression*
randBin(*n*, *p*, *#Trials*) ⇒ *list*

randBin(*n*, *p*) returns a random real number from a specified Binomial distribution.

randBin(*n*, *p*, *#Trials*) returns a list containing *#Trials* random real numbers from a specified Binomial distribution.

randBin(80,0.5)

42

randBin(80,0.5,3)

{41,32,39}

randInt()Catalog > 

randInt
(*lowBound*,*upBound*)
⇒ *expression*
randInt
(*lowBound*,*upBound*
,*#Trials*) ⇒ *list*

randInt
(*lowBound*,*upBound*)
returns a random integer within the range specified by *lowBound* and *upBound* integer bounds.

randInt
(*lowBound*,*upBound*
,*#Trials*) returns a list containing *#Trials* random integers within the specified range.

randInt(3,10)

5

randInt(3,10,4)

{9,7,5,8}

randMat()

Catalog > 

randMat(*numRows*, *numColumns*) ⇒ *matrix*

Returns a matrix of integers between -9 and 9 of the specified dimension.

Both arguments must simplify to integers.

RandSeed 1147	Done
randMat(3,3)	$\begin{bmatrix} 8 & -3 & 6 \\ -2 & 3 & -6 \\ 0 & 4 & -6 \end{bmatrix}$

Note: The values in this matrix will change each time you press **enter**.

randNorm()

Catalog > 

randNorm(μ , σ) ⇒ *expression*
randNorm(μ , σ , #Trials) ⇒ *list*

randNorm(μ , σ) returns a decimal number from the specified normal distribution. It could be any real number but will be heavily concentrated in the interval $[\mu-3\sigma, \mu+3\sigma]$.

randNorm(μ , σ , #Trials) returns a list containing #Trials decimal numbers from the specified normal distribution.

RandSeed 1147	Done
randNorm(0,1)	0.492541
randNorm(3,4.5)	-3.54356

randPoly()

Catalog > 

randPoly(*Var*, *Order*) ⇒ *expression*

Returns a polynomial in *Var* of the specified *Order*. The coefficients are random integers in the range -9 through 9. The leading coefficient will not be zero.

Order must be 0-99.

RandSeed 1147	Done
randPoly(x,5)	$-2 \cdot x^5 + 3 \cdot x^4 - 6 \cdot x^3 + 4 \cdot x - 6$

randSamp()

Catalog > 

randSamp(*List*, #Trials[,*noRepl*]) ⇒ *list*

Returns a list containing a random sample of #Trials trials from *List* with an option for sample replacement (*noRepl*=0), or no sample replacement (*noRepl*=1). The default is with sample replacement.

Define list3={1,2,3,4,5}	Done
Define list4=randSamp(list3,6)	Done
list4	{2,3,4,3,1,2}

RandSeed *Number*

If *Number* = 0, sets the seeds to the factory defaults for the random-number generator. If *Number* ≠ 0, it is used to generate two seeds, which are stored in system variables seed1 and seed2.

RandSeed 1147	Done
rand()	0.158206

real()

real(*Expr1*) ⇒ *expression*

Returns the real part of the argument.

Note: All undefined variables are treated as real variables. See also **imag()**, page 88.

real(*List1*) ⇒ *list*

Returns the real parts of all elements.

real(*Matrix1*) ⇒ *matrix*

Returns the real parts of all elements.

$\text{real}(2+3\cdot i)$	2
$\text{real}(z)$	z
$\text{real}(x+i\cdot y)$	x

$\text{real}(\{a+i\cdot b, 3, i\})$	$\{a, 3, 0\}$
-------------------------------------	---------------

$\text{real}\left(\begin{bmatrix} a+i\cdot b & 3 \\ c & i \end{bmatrix}\right)$	$\begin{bmatrix} a & 3 \\ c & 0 \end{bmatrix}$
---	--

► Rect

Vector ► **Rect**

Note: You can insert this operator from the computer keyboard by typing **@>Rect**.

Displays *Vector* in rectangular form [x, y, z]. The vector must be of dimension 2 or 3 and can be a row or a column.

Note: ► **Rect** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update *ans*.

Note: See also ► **Polar**, page 133.

complexValue ► **Rect**

Displays *complexValue* in rectangular form a+bi. The *complexValue* can have any complex form. However, an $\text{re}^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use parentheses for an ($r\angle\theta$) polar entry.

$\left(3 \angle \frac{\pi}{4} \quad \angle \frac{\pi}{6}\right) \text{►Rect}$	$\begin{bmatrix} 3\sqrt{2} & 3\sqrt{2} & 3\sqrt{3} \\ 4 & 4 & 2 \end{bmatrix}$
$[a \angle b \angle c]$	$[a\cdot\cos(b)\cdot\sin(c) \quad a\cdot\sin(b)\cdot\sin(c) \quad a\cdot\cos(c)]$

In Radian angle mode:

$\left(4\cdot e^{\frac{\pi}{3}}\right) \text{►Rect}$	$4\cdot e^{\frac{\pi}{3}}$
$\left(\left(4 \angle \frac{\pi}{3}\right)\right) \text{►Rect}$	$2+2\sqrt{3}\cdot i$

In Gradian angle mode:

$$\left((1 \angle 100) \right) \blacktriangleright \text{Rect} \quad i$$

In Degree angle mode:

$$\left((4 \angle 60) \right) \blacktriangleright \text{Rect} \quad 2+2\sqrt{3}\cdot i$$

Note: To type \angle , select it from the symbol list in the Catalog.

ref()

$\text{ref}(\text{Matrix } I[, Tol]) \Rightarrow \text{matrix}$

Returns the row echelon form of *Matrix I*.

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

- If you use or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\text{dim}(\text{Matrix } I)) \cdot \text{rowNorm}(\text{Matrix } I)$

Avoid undefined elements in *Matrix I*. They can lead to unexpected results.

For example, if *a* is undefined in the following expression, a warning message appears and the result is shown as:

$$\text{ref} \left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \quad \begin{bmatrix} 1 & \frac{1}{a} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{ref} \left(\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix} \right) \quad \begin{bmatrix} 1 & \frac{-2}{5} & \frac{-4}{5} & \frac{4}{5} \\ 0 & 1 & \frac{4}{7} & \frac{11}{7} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow m1 \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\text{ref}(m1) \quad \begin{bmatrix} 1 & \frac{d}{c} \\ 0 & 1 \end{bmatrix}$$

Warning message: *a* is undefined.

The warning appears because the generalized element $1/a$ would not be valid for $a=0$.

You can avoid this by storing a value to a beforehand or by using the constraint ("|") operator to substitute a value, as shown in the following example.


$$\text{ref} \left(\begin{array}{ccc} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) | a=0 \quad \begin{array}{ccc} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}$$

Note: See also `rref()`, page 156.

RefreshProbeVars

RefreshProbeVars

Allows you to access sensor data from all connected sensor probes in your TI-Basic program.

StatusVar Value	Status
<i>statusVar</i> =0	Normal (continue with the program) The Vernier DataQuest™ application is in data collection mode.
<i>statusVar</i> =1	Note: The Vernier DataQuest™ application must be in meter mode for this command to work. 
<i>statusVar</i> =2	The Vernier DataQuest™ application is not launched.
<i>statusVar</i> =3	The Vernier DataQuest™ application is launched, but you have not connected any probes.

Example

```

Define temp()=
Prgm
© Check if system is ready
RefreshProbeVars status
If status=0 Then
Disp "ready"
For n,1,50
RefreshProbeVars status
temperature:=meter.temperature
Disp "Temperature:
",temperature
If temperature>30 Then
Disp "Too hot"
EndIf
© Wait for 1 second between
samples
Wait 1
EndFor

```



```
Else
Disp "Not ready. Try again
later"
EndIf
EndPrgm
```

Note: This can also be used with TI-Innovator™ Hub.

remain()

remain(*Expr1*, *Expr2*) ⇒ *expression*

remain(*List1*, *List2*) ⇒ *list*
remain(*Matrix1*, *Matrix2*) ⇒ *matrix*

Returns the remainder of the first argument with respect to the second argument as defined by the identities:

$\text{remain}(x,0) = x$
 $\text{remain}(x,y) = x - y \cdot \text{iPart}(x/y)$

As a consequence, note that **remain**(-*x*,*y*) = **remain**(*x*,*y*). The result is either zero or it has the same sign as the first argument.

Note: See also **mod()**, page 116.

$\text{remain}(7,0)$	7
$\text{remain}(7,3)$	1
$\text{remain}(-7,3)$	-1
$\text{remain}(7,-3)$	1
$\text{remain}(-7,-3)$	-1
$\text{remain}(\{12,-14,16\},\{9,7,-5\})$	{3,0,1}

$\text{remain}\left(\begin{pmatrix} 9 & -7 \\ 6 & 4 \end{pmatrix}, \begin{pmatrix} 4 & 3 \\ 4 & -3 \end{pmatrix}\right)$	$\begin{pmatrix} 1 & -1 \\ 2 & 1 \end{pmatrix}$
--	---

Request

Request *promptString*, *var*[, *DispFlag* [, *statusVar*]]

Request *promptString*, *func*(*arg1*, ...*argn*) [, *DispFlag* [, *statusVar*]]

Programming command: Pauses the program and displays a dialog box containing the message *promptString* and an input box for the user's response.

When the user types a response and clicks **OK**, the contents of the input box are assigned to variable *var*.

Define a program:

```
Define request_demo()=Prgm
Request "Radius: ",r
Disp "Area = ",pi*r^2
EndPrgm
```

Run the program and type a response:

```
request_demo()
```

If the user clicks **Cancel**, the program proceeds without accepting any input. The program uses the previous value of *var* if *var* was already defined.

The optional *DispFlag* argument can be any expression.

- If *DispFlag* is omitted or evaluates to **1**, the prompt message and user's response are displayed in the Calculator history.
- If *DispFlag* evaluates to **0**, the prompt and response are not displayed in the history.

The optional *statusVar* argument gives the program a way to determine how the user dismissed the dialog box. Note that *statusVar* requires the *DispFlag* argument.

- If the user clicked **OK** or pressed **Enter** or **Ctrl+Enter**, variable *statusVar* is set to a value of **1**.
- Otherwise, variable *statusVar* is set to a value of **0**.



The *func()* argument allows a program to store the user's response as a function definition. This syntax operates as if the user executed the command:

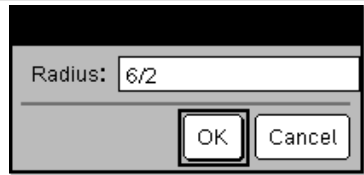
Define *func(arg1, ...argn) = user's response*

The program can then use the defined function *func()*. The *promptString* should guide the user to enter an appropriate *user's response* that completes the function definition.

Note: You can use the Request command within a user-defined program but not within a function.

To stop a program that contains a **Request** command inside an infinite loop:

- **Handheld:** Hold down the  key and press  repeatedly.



Result after selecting **OK**:

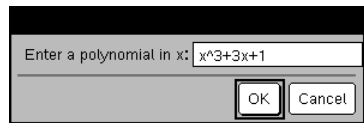
Radius: 6/2
Area= 28.2743

Define a program:

```
Define polynomial()=Prgm
  Request "Enter a polynomial in
  x:",p(x)
  Disp "Real roots are:",polyRoots
  (p(x),x)
EndPrgm
```

Run the program and type a response:

polynomial()



Result after entering x^3+3x+1 and selecting **OK**:

Real roots are: $\{-0.322185\}$

- **Windows®**: Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®**: Hold down the **F5** key and press **Enter** repeatedly.
- **iPad®**: The app displays a prompt. You can continue waiting or cancel.

Note: See also **RequestStr**, page 151.



RequestStr

RequestStr *promptString*, *var*[, *DispFlag*]

Programming command: Operates identically to the first syntax of the **Request** command, except that the user's response is always interpreted as a string. By contrast, the **Request** command interprets the response as an expression unless the user encloses it in quotation marks ("").

Note: You can use the **RequestStr** command within a user-defined program but not within a function.

To stop a program that contains a **RequestStr** command inside an infinite loop:

- **Handheld**: Hold down the  key and press  repeatedly.
- **Windows®**: Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®**: Hold down the **F5** key and press **Enter** repeatedly.
- **iPad®**: The app displays a prompt. You can continue waiting or cancel.

Note: See also **Request**, page 149.

Define a program:

```
Define requestStr_demo()=Prgm
  RequestStr "Your name:",name,0
  Disp "Response has ",dim(name),"
  characters."
EndPrgm
```

Run the program and type a response:

```
requestStr_demo()
```



Result after selecting **OK** (Note that the *DispFlag* argument of **0** omits the prompt and response from the history):

```
requestStr_demo()
```

Response has 5 characters.

Return [*Expr*]

Returns *Expr* as the result of the function.
Use within a **Func...EndFunc** block.

Note: Use **Return** without an argument within a **Prgm...EndPrgm** block to exit a program.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

```
Define factorial (nn)=
Func
Local answer,counter
1 → answer
For counter,1,nn
answer · counter → answer
EndFor
Return answer
EndFunc
```

factorial (3)

6

right()**right**(*List1*[, *Num*]) ⇒ *list*

Returns the rightmost *Num* elements contained in *List1*.

If you omit *Num*, returns all of *List1*.

right(*sourceString*[, *Num*]) ⇒ *string*

Returns the rightmost *Num* characters contained in character string *sourceString*.

If you omit *Num*, returns all of *sourceString*.

right(*Comparison*) ⇒ *expression*

Returns the right side of an equation or inequality.

right({1,3,-2,4},3)

{3,-2,4}

right("Hello",2)

"lo"

right($x < 3$)

3

rk23 ()

rk23(*Expr*, *Var*, *depVar*, {*Var0*, *VarMax*}, *depVar0*, *VarStep* [, *difTol*]) ⇒ *matrix*

Differential equation:

$y' = 0.001 \cdot y \cdot (100 - y)$ and $y(0) = 10$

rk23(*SystemOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep* [, *difTol*]) ⇒ *matrix*

rk23($0.001 \cdot y \cdot (100 - y), t, y, \{0, 100\}, 10, 1$)

0.	1.	2.	3.	4.
10.	10.9367	11.9493	13.042	14.2

rk23(*ListOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep* [, *difTol*]) ⇒ *matrix*

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

Uses the Runge-Kutta method to solve the system

$$\frac{d \text{depVar}}{d \text{Var}} = \text{Expr}(\text{Var}, \text{depVar})$$

with $\text{depVar}(\text{Var}0) = \text{depVar}0$ on the interval $[\text{Var}0, \text{VarMax}]$. Returns a matrix whose first row defines the Var output values as defined by VarStep . The second row defines the value of the first solution component at the corresponding Var values, and so on.

Expr is the right hand side that defines the ordinary differential equation (ODE).

SystemOfExpr is a system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).

ListOfExpr is a list of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).

Var is the independent variable.

ListOfDepVars is a list of dependent variables.

$\{\text{Var}0, \text{VarMax}\}$ is a two-element list that tells the function to integrate from $\text{Var}0$ to VarMax .

$\text{ListOfDepVars}0$ is a list of initial values for dependent variables.

If VarStep evaluates to a nonzero number: $\text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax} - \text{Var}0)$ and solutions are returned at $\text{Var}0 + i * \text{VarStep}$ for all $i=0,1,2,\dots$ such that $\text{Var}0 + i * \text{VarStep}$ is in $[\text{var}0, \text{VarMax}]$ (may not get a solution value at VarMax).

if VarStep evaluates to zero, solutions are returned at the "Runge-Kutta" Var values.

dftol is the error tolerance (defaults to 0.001).

Same equation with dftol set to $1.E-6$

$$\text{rk23}\left(\left\{0.001 \cdot y \cdot \{100-y\}, t, y, \{0,100\}, 10, 1, 1.E-6\right\}\right)$$

0.	1.	2.	3.	4.
10.	10.9367	11.9495	13.0423	14.2189

Compare above result with CAS exact solution obtained using $\text{deSolve}()$ and $\text{seqGen}()$:

$$\text{deSolve}(y' = 0.001 \cdot y \cdot \{100-y\} \text{ and } y(0) = 10, t, y)$$

$$y = \frac{100 \cdot \{1.10517\}^t}{\{1.10517\}^t + 9}$$

$$\text{seqGen}\left(\frac{100 \cdot \{1.10517\}^t}{\{1.10517\}^t + 9}, t, y, \{0,100\}\right)$$

$$\{10., 10.9367, 11.9494, 13.0423, 14.2189, 15.4\}$$

System of equations:

$$\begin{cases} y1' = -y1 + 0.1 \cdot y1 \cdot y2 \\ y2' = 3 \cdot y2 - y1 \cdot y2 \end{cases}$$

with $y1(0) = 2$ and $y2(0) = 5$

$$\text{rk23}\left(\left\{\begin{matrix} -y1 + 0.1 \cdot y1 \cdot y2 \\ 3 \cdot y2 - y1 \cdot y2 \end{matrix}, t, \{y1, y2\}, \{0, 5\}, \{2, 5\}, 1\right\}\right)$$

0.	1.	2.	3.	4.
2.	1.94103	4.78694	3.25253	1.82848
5.	16.8311	12.3133	3.51112	6.27245

root()

Catalog >

root(*Expr*) ⇒ *root***root**(*Expr1*, *Expr2*) ⇒ *root***root**(*Expr*) returns the square root of *Expr*.**root**(*Expr1*, *Expr2*) returns the *Expr2* root of *Expr1*. *Expr1* can be a real or complex floating point constant, an integer or complex rational constant, or a general symbolic expression.**Note:** See also **Nth root template**, page 1.

$\sqrt[3]{8}$	2
$\sqrt[3]{3}$	$\frac{1}{3^3}$
$\sqrt[3]{3}$	1.44225

rotate()

Catalog >

rotate(*Integer1* [, *#ofRotations*]) ⇒ *integer*Rotates the bits in a binary integer. You can enter *Integer1* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *Integer1* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see ► **Base2**, page 17.If *#ofRotations* is positive, the rotation is to the left. If *#ofRotations* is negative, the rotation is to the right. The default is -1 (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

0b00000000000001111010110000110101

Rightmost bit rotates to leftmost.

produces:

0b10000000000000111101011000011010

The result is displayed according to the Base mode.

rotate(*List1* [, *#ofRotations*]) ⇒ *list*Returns a copy of *List1* rotated right or left by *#ofRotations* elements. Does not alter *List1*.

In Bin base mode:

rotate (0b11111111111111111111111111111111)	
0b10000000000000000000000000000001	▶
rotate (256,1)	0b1000000000

To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

In Hex base mode:

rotate (0h78E)	0h3C7
rotate (0h78E,-2)	0h80000000000001E3
rotate (0h78E,2)	0h1E38

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

In Dec base mode:

rotate ({1,2,3,4})	{4,1,2,3}
rotate ({1,2,3,4},-2)	{3,4,1,2}
rotate ({1,2,3,4},1)	{2,3,4,1}

rotate()

Catalog > 

If *#ofRotations* is positive, the rotation is to the left. If *#ofRotations* is negative, the rotation is to the right. The default is -1 (rotate right one element).

rotate(*String1* [, *#ofRotations*]) \Rightarrow *string*

Returns a copy of *String1* rotated right or left by *#ofRotations* characters. Does not alter *String1*.

If *#ofRotations* is positive, the rotation is to the left. If *#ofRotations* is negative, the rotation is to the right. The default is -1 (rotate right one character).

rotate("abcd")	"dabc"
rotate("abcd",-2)	"cdab"
rotate("abcd",1)	"bcda"

round()

Catalog > 

round(*Expr1* [, *digits*]) \Rightarrow *expression*

Returns the argument rounded to the specified number of digits after the decimal point.

digits must be an integer in the range 0–12. If *digits* is not included, returns the argument rounded to 12 significant digits.

Note: Display digits mode may affect how this is displayed.

round(*List1* [, *digits*]) \Rightarrow *list*

Returns a list of the elements rounded to the specified number of digits.

round(*Matrix1* [, *digits*]) \Rightarrow *matrix*


Returns a matrix of the elements rounded to the specified number of digits.

round(1.234567,3)	1.235
-------------------	-------

round({ π , $\sqrt{2}$,ln(2)},4)	{ 3.1416,1.4142,0.6931 }
---------------------------------------	--------------------------

round($\begin{bmatrix} \ln(5) & \ln(3) \\ \pi & e^1 \end{bmatrix}$,1)	$\begin{bmatrix} 1.6 & 1.1 \\ 3.1 & 2.7 \end{bmatrix}$
---	--

rowAdd()

Catalog > 

rowAdd(*Matrix1*, *rIndex1*, *rIndex2*) \Rightarrow *matrix*

Returns a copy of *Matrix1* with row *rIndex2* replaced by the sum of rows *rIndex1* and *rIndex2*.

rowAdd($\begin{bmatrix} 3 & 4 \\ -3 & -2 \end{bmatrix}$,1,2)	$\begin{bmatrix} 3 & 4 \\ 0 & 2 \end{bmatrix}$
rowAdd($\begin{bmatrix} a & b \\ c & d \end{bmatrix}$,1,2)	$\begin{bmatrix} a & b \\ a+c & b+d \end{bmatrix}$

rowDim()

Catalog >

rowDim(*Matrix*) ⇒ *expression*Returns the number of rows in *Matrix*.**Note:** See also **colDim()**, page 26.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$
rowDim (<i>m1</i>)	3

rowNorm()

Catalog >

rowNorm(*Matrix*) ⇒ *expression*Returns the maximum of the sums of the absolute values of the elements in the rows in *Matrix*.**Note:** All matrix elements must simplify to numbers. See also **colNorm()**, page 26.

rowNorm $\left(\begin{bmatrix} -5 & 6 & -7 \\ 3 & 4 & 9 \\ 9 & -9 & -7 \end{bmatrix}\right)$	25
---	----

rowSwap()

Catalog >

rowSwap(*Matrix1*, *rIndex1*, *rIndex2*) ⇒ *matrix*Returns *Matrix1* with rows *rIndex1* and *rIndex2* exchanged.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow mat$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$
rowSwap (<i>mat</i> ,1,3)	$\begin{bmatrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \end{bmatrix}$

rref()

Catalog >

rref(*Matrix1*[, *Tol*]) ⇒ *matrix*Returns the reduced row echelon form of *Matrix1*.

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

- If you use or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.

rref $\left(\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix}\right)$	$\begin{bmatrix} 1 & 0 & 0 & \frac{66}{71} \\ 0 & 1 & 0 & \frac{147}{71} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$
---	---

rref $\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right)$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
---	--

- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\dim(\text{Matrix } I)) \cdot \text{rowNorm}(\text{Matrix } I)$

Note: See also `ref()`, page 147.

S

sec()

 key

`sec(Expr1) ⇒ expression`

In Degree angle mode:

`sec(List1) ⇒ list`

Returns the secant of *Expr1* or returns a list containing the secants of all elements in *List1*.

$$\frac{\sec(45)}{\sec(\{1,2,3,4\})} = \frac{\sqrt{2}}{\left\{ \frac{1}{\cos(1)}, 1.00081, \frac{1}{\cos(4)} \right\}}$$

Note: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode temporarily.

sec-1()

 key

`sec-1(Expr1) ⇒ expression`

In Degree angle mode:

`sec-1(List1) ⇒ list`

Returns the angle whose secant is *Expr1* or returns a list containing the inverse secants of each element of *List1*.

$$\sec^{-1}(1) = 0$$

In Gradian angle mode:

Note: The result is returned as a degree, gradian, or radian angle, according to the current angle mode setting.

$$\sec^{-1}(\sqrt{2}) = 50$$

In Radian angle mode:

Note: You can insert this function from the keyboard by typing `arcsec(...)`.

$$\sec^{-1}(\{1,2,5\}) = \left\{ 0, \frac{\pi}{3}, \cos^{-1}\left(\frac{1}{5}\right) \right\}$$

sech()

Catalog > 

sech(*Expr1*) ⇒ *expression*

sech(3)	$\frac{1}{\cosh(3)}$
---------	----------------------

sech(*List1*) ⇒ *list*

Returns the hyperbolic secant of *Expr1* or returns a list containing the hyperbolic secants of the *List1* elements.

sech({1,2,3,4})	$\left\{ \frac{1}{\cosh(1)}, 0.198522, \frac{1}{\cosh(4)} \right\}$
-----------------	---

sech-1()

Catalog > 

sech-1(*Expr1*) ⇒ *expression*

In Radian angle and Rectangular complex mode:

sech-1(*List1*) ⇒ *list*

Returns the inverse hyperbolic secant of *Expr1* or returns a list containing the inverse hyperbolic secants of each element of *List1*.

sech ⁻¹ (1)	0
sech ⁻¹ {1,2,2.1}	$\left\{ 0, \frac{2 \cdot \pi}{3} \cdot i, 8. \text{E} - 15 + 1.07448 \cdot i \right\}$

Note: You can insert this function from the keyboard by typing **arcsech** (...).

Send

Hub Menu

Send *exprOrString1* [, *exprOrString2*] ...

Example: Turn on the blue element of the built-in RGB LED for 0.5 seconds.

Programming command: Sends one or more TI-Innovator™ Hub commands to a connected hub.

Send "SET COLOR.BLUE ON TIME .5"	Done
----------------------------------	------

exprOrString must be a valid TI-Innovator™ Hub Command. Typically, *exprOrString* contains a "SET ..." command to control a device or a "READ ..." command to request data.

Example: Request the current value of the hub's built-in light-level sensor. A **Get** command retrieves the value and assigns it to variable *lightval*.

The arguments are sent to the hub in succession.

Send "READ BRIGHTNESS"	Done
Get <i>lightval</i>	Done
<i>lightval</i>	0.347922

Note: You can use the **Send** command within a user-defined program but not within a function.

Example: Send a calculated frequency to the hub's built-in speaker. Use special variable *iostr.SendAns* to show the hub command with the expression evaluated.

Note: See also **Get** (page 77), **GetStr** (page 84), and **eval()** (page 62).

$n:=50$	50
$m:=4$	4
Send "SET SOUND eval(m·n)"	Done
<i>iostr.SendAns</i>	"SET SOUND 200"

seq()

Catalog >

seq(Expr, Var, Low, High[, Step]) ⇒ list

Increments *Var* from *Low* through *High* by an increment of *Step*, evaluates *Expr*, and returns the results as a list. The original contents of *Var* are still there after **seq()** is completed.

The default value for *Step* = 1.

$\text{seq}\left(n^2, n, 1, 6\right)$	$\{1, 4, 9, 16, 25, 36\}$
$\text{seq}\left(\frac{1}{n}, n, 1, 10, 2\right)$	$\left\{1, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}\right\}$
$\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right)$	$\frac{1968329}{1270080}$

Note: To force an approximate result,

Handheld: Press .

Windows®: Press **Ctrl+Enter**.

Macintosh®: Press **⌘+Enter**.

iPad®: Hold **enter**, and select .

$\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right)$	1.54977
--	---------

seqGen()

Catalog >

seqGen(Expr, Var, depVar, {Var0, VarMax}[, ListOfInitTerms [, VarStep[, CeilingValue]]) ⇒ list

Generates a list of terms for sequence $\text{depVar}(Var)=Expr$ as follows: Increments independent variable *Var* from *Var0* through *VarMax* by *VarStep*, evaluates $\text{depVar}(Var)$ for corresponding values of *Var* using the *Expr* formula and *ListOfInitTerms*, and returns the results as a list.

seqGen(ListOrSystemOfExpr, Var, ListOfDepVars, {Var0, VarMax} [, MatrixOfInitTerms[, VarStep[, CeilingValue]]) ⇒ matrix

Generate the first 5 terms of the sequence $u(n) = u(n-1)^2/2$, with $u(1)=2$ and $VarStep=1$.

$\text{seqGen}\left(\frac{(u(n-1))^2}{n}, n, u, \{1, 5\}, \{2\}\right)$	$\left\{2, 2, \frac{4}{3}, \frac{4}{9}, \frac{16}{405}\right\}$
---	---

Example in which $Var0=2$:

$\text{seqGen}\left(\frac{u(n-1)+1}{n}, n, u, \{2, 5\}, \{3\}\right)$	$\left\{3, \frac{4}{3}, \frac{7}{12}, \frac{19}{60}\right\}$
---	--

Generates a matrix of terms for a system (or list) of sequences $ListOfDepVars(Var)=ListOrSystemOfExpr$ as follows: Increments independent variable Var from $Var0$ through $VarMax$ by $VarStep$, evaluates $ListOfDepVars(Var)$ for corresponding values of Var using $ListOrSystemOfExpr$ formula and $MatrixOfInitTerms$, and returns the results as a matrix.

The original contents of Var are unchanged after **seqGen()** is completed.

The default value for $VarStep = 1$.

Example in which initial term is symbolic:

$$\text{seqGen}\left\{u(n-1)+2n,u,\{1,5\},\{a\}\right\}$$

$$\left\{a,a+2,a+4,a+6,a+8\right\}$$

System of two sequences:

$$\text{seqGen}\left\{\left\{\frac{1}{n},\frac{u2(n-1)}{2}+u1(n-1)\right\},n,\{u1,u2\},\{1,5\},\left[-\frac{1}{2}\right]\right\}$$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ 2 & 2 & \frac{3}{2} & \frac{13}{2} & \frac{19}{2} \\ 2 & 2 & 2 & 12 & 24 \end{bmatrix}$$

Note: The Void () in the initial term matrix above is used to indicate that the initial term for $u1(n)$ is calculated using the explicit sequence formula $u1(n)=1/n$.

seqn()

seqn($Expr(u, n[, ListOfInitTerms[, nMax[, CeilingValue]])$) \Rightarrow list

Generates a list of terms for a sequence $u(n)=Expr(u, n)$ as follows: Increments n from 1 through $nMax$ by 1, evaluates $u(n)$ for corresponding values of n using the $Expr(u, n)$ formula and $ListOfInitTerms$, and returns the results as a list.

seqn($Expr(n[, nMax[, CeilingValue]])$) \Rightarrow list

Generates a list of terms for a non-recursive sequence $u(n)=Expr(n)$ as follows: Increments n from 1 through $nMax$ by 1, evaluates $u(n)$ for corresponding values of n using the $Expr(n)$ formula, and returns the results as a list.

If $nMax$ is missing, $nMax$ is set to 2500

If $nMax=0$, $nMax$ is set to 2500

Note: **seqn()** calls **seqGen()** with $n0=1$ and $nstep=1$

Generate the first 6 terms of the sequence $u(n) = u(n-1)/2$, with $u(1)=2$.

$$\text{seqn}\left\{\frac{u(n-1)}{n},\{2\},6\right\}$$

$$\left\{2,1,\frac{1}{3},\frac{1}{12},\frac{1}{60},\frac{1}{360}\right\}$$

$$\text{seqn}\left\{\frac{1}{n^2},6\right\}$$

$$\left\{1,\frac{1}{4},\frac{1}{9},\frac{1}{16},\frac{1}{25},\frac{1}{36}\right\}$$

series(Expr1, Var, Order[, Point]) ⇒
expression

**series(Expr1, Var, Order[, Point] |
Var > Point)** ⇒ expression

**series(Expr1, Var, Order[, Point] |
Var < Point)** ⇒ expression

Returns a generalized truncated power series representation of *Expr1* expanded about *Point* through degree *Order*. *Order* can be any rational number. The resulting powers of (*Var* - *Point*) can include negative and/or fractional exponents. The coefficients of these powers can include logarithms of (*Var* - *Point*) and other functions of *Var* that are dominated by all powers of (*Var* - *Point*) having the same exponent sign.

Point defaults to 0. *Point* can be ∞ or $-\infty$, in which cases the expansion is through degree *Order* in $1/(Var - Point)$.

series(...) returns “**series(...)**” if it is unable to determine such a representation, such as for essential singularities such as $\sin(1/z)$ at $z=0$, $e^{-1/z}$ at $z=0$, or e^z at $z = \infty$ or $-\infty$.

If the series or one of its derivatives has a jump discontinuity at *Point*, the result is likely to contain sub-expressions of the form $\text{sign}(\dots)$ or $\text{abs}(\dots)$ for a real expansion variable or $(-1)^{\text{floor}(\dots \text{angle}(\dots))}$ for a complex expansion variable, which is one ending with “_”. If you intend to use the series only for values on one side of *Point*, then append the appropriate one of “| *Var* > *Point*”, “| *Var* < *Point*”, “| *Var* ≥ *Point*”, or “*Var* ≤ *Point*” to obtain a simpler result.

series() can provide symbolic approximations to indefinite integrals and definite integrals for which symbolic solutions otherwise can't be obtained.

$$\text{series}\left(\frac{1-\cos(x-1)}{(x-1)^2}, x, 4, 1\right) \quad \frac{1}{2} \frac{(x-1)^2}{24} + \frac{(x-1)^4}{720}$$

$$\text{series}\left(\frac{-1}{e^z}, z, 1\right) \quad z_{-1}$$

$$\text{series}\left(\left(1+\frac{1}{n}\right)^n, n, 2, \infty\right) \quad e - \frac{e}{2 \cdot n} + \frac{11 \cdot e}{24 \cdot n^2}$$

$$\text{series}\left(\tan^{-1}\left(\frac{1}{x}\right), x, 5\right), x > 0 \quad \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5}$$

$$\text{series}\left(\int \frac{\sin(x)}{x} dx, x, 6\right) \quad x - \frac{x^3}{18} + \frac{x^5}{600}$$

$$\text{series}\left(\int_0^x \sin(x \cdot \sin(t)) dt, x, 7\right) \quad \frac{x^3}{2} - \frac{x^5}{24} - \frac{29 \cdot x^7}{720}$$

$$\text{series}\left(\left(1+e^x\right)^2, x, 2, 1\right) \quad (e+1)^2 + 2 \cdot e \cdot (e+1) \cdot (x-1) + e \cdot (2 \cdot e+1) \cdot (x-1)^2$$

series() distributes over 1st-argument lists and matrices.

series() is a generalized version of **taylor()**.

As illustrated by the last example to the right, the display routines downstream of the result produced by **series(...)** might rearrange terms so that the dominant term is not the leftmost one.

Note: See also **dominantTerm()**, page 56.

setMode()

setMode(modeNameInteger, settingInteger) ⇒ integer
setMode(list) ⇒ integer list

Valid only within a function or program.

setMode(modeNameInteger, settingInteger) temporarily sets mode *modeNameInteger* to the new setting *settingInteger*, and returns an integer corresponding to the original setting of that mode. The change is limited to the duration of the program/function's execution.

modeNameInteger specifies which mode you want to set. It must be one of the mode integers from the table below.

settingInteger specifies the new setting for the mode. It must be one of the setting integers listed below for the specific mode you are setting.

setMode(list) lets you change multiple settings. *list* contains pairs of mode integers and setting integers. **setMode(list)** returns a similar list whose integer pairs represent the original modes and settings.

If you have saved all mode settings with **getMode(0)** → *var*, you can use **setMode(var)** to restore those settings until the function or program exits. See **getMode()**, page 83.

Display approximate value of π using the default setting for Display Digits, and then display π with a setting of Fix2. Check to see that the default is restored after the program executes.

Define <i>prog1()</i> =Prgm	<i>Done</i>
Disp approx(π)	
setMode(1,16)	
Disp approx(π)	
EndPrgm	
<hr/>	
<i>prog1()</i>	
	3.14159
	3.14
	<i>Done</i>

Note: The current mode settings are passed to called subroutines. If any subroutine changes a mode setting, the mode change will be lost when control returns to the calling routine.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Mode Name	Mode Integer	Setting Integers
Display Digits	1	1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12, 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12
Angle	2	1=Radian, 2=Degree, 3=Gradian
Exponential Format	3	1=Normal, 2=Scientific, 3=Engineering
Real or Complex	4	1=Real, 2=Rectangular, 3=Polar
Auto or Approx.	5	1=Auto, 2=Approximate, 3=Exact
Vector Format	6	1=Rectangular, 2=Cylindrical, 3=Spherical
Base	7	1=Decimal, 2=Hex, 3=Binary
Unit system	8	1=SI, 2=Eng/US

shift()

shift(Integer I[, #ofShifts]) ⇒ *integer*

Shifts the bits in a binary integer. You can enter *Integer I* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *Integer I* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see ► **Base2**, page 17.

In Bin base mode:

```

shift(0b1111010110000110101)
                                0b111101011000011010
shift(256,1)                      0b1000000000

```

In Hex base mode:

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is -1 (shift right one bit).

shift(0h78E)	0h3C7
shift(0h78E,-2)	0h1E3
shift(0h78E,2)	0h1E38

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

For example, in a right shift:

Each bit shifts right.

0b0000000000000111101011000011010

Inserts 0 if leftmost bit is 0,
or 1 if leftmost bit is 1.

produces:

0b00000000000000111101011000011010

The result is displayed according to the Base mode. Leading zeros are not shown.

shift(List I[,#ofShifts]) ⇒ *list*

Returns a copy of *List I* shifted right or left by *#ofShifts* elements. Does not alter *List I*.

In Dec base mode:

shift({ 1,2,3,4 })	{ undef,1,2,3 }
shift({ 1,2,3,4 },-2)	{ undef,undef,1,2 }
shift({ 1,2,3,4 },2)	{ 3,4,undef,undef }

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is -1 (shift right one element).

Elements introduced at the beginning or end of *list* by the shift are set to the symbol "undef".

shift(String I[,#ofShifts]) ⇒ *string*

Returns a copy of *String I* shifted right or left by *#ofShifts* characters. Does not alter *String I*.

shift("abcd")	" abc"
shift("abcd",-2)	" ab"
shift("abcd",1)	"bcd "

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is -1 (shift right one character).

shift()

Characters introduced at the beginning or end of *string* by the shift are set to a space.

sign()

sign(*Expr1*) ⇒ *expression*

$$\text{sign}(-3.2) \quad -1.$$

sign(*List1*) ⇒ *list*

$$\text{sign}\{2,3,4,-5\} \quad \{1,1,1,-1\}$$

sign(*Matrix1*) ⇒ *matrix*

$$\text{sign}(1+|x|) \quad 1$$

For real and complex *Expr1*, returns *Expr1*/**abs**(*Expr1*) when *Expr1* ≠ 0.

If complex format mode is Real:

Returns 1 if *Expr1* is positive. Returns -1 if *Expr1* is negative.

$$\text{sign}\begin{bmatrix} -3 & 0 & 3 \end{bmatrix} \quad \begin{bmatrix} -1 & \neq 1 & 1 \end{bmatrix}$$

sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

simult()

simult(*coeffMatrix*, *constVector* [, *Tol*]) ⇒ *matrix*

Solve for x and y:

$$x + 2y = 1$$

$$3x + 4y = -1$$

Returns a column vector that contains the solutions to a system of linear equations.

$$\text{simult}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) \quad \begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

Note: See also **linSolve()**, page 102.

coeffMatrix must be a square matrix that contains the coefficients of the equations.

The solution is x=-3 and y=2.

constVector must have the same number of rows (same dimension) as *coeffMatrix* and contain the constants.

Solve:

$$ax + by = 1$$

$$cx + dy = 2$$

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*.


$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \text{mat}1 \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

$$\text{simult}\left(\text{mat}1, \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) \quad \begin{bmatrix} -(2 \cdot b - d) \\ a \cdot d - b \cdot c \\ 2 \cdot a - c \\ a \cdot d - b \cdot c \end{bmatrix}$$

- If you set the **Auto** or **Approximate** mode to Approximate, computations are done using floating-point arithmetic.

simult()

Catalog > 

- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\dim(\text{coeffMatrix}))$
 $\cdot \text{rowNorm}(\text{coeffMatrix})$

simult(coeffMatrix, constMatrix[, Tol]) ⇒
matrix

Solves multiple systems of linear equations, where each system has the same equation coefficients but different constants.

Each column in *constMatrix* must contain the constants for a system of equations. Each column in the resulting matrix contains the solution for the corresponding system.

Solve:

$$x + 2y = 1$$

$$3x + 4y = -1$$

$$x + 2y = 2$$

$$3x + 4y = -3$$

$$\text{simult}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ -1 & -3 \end{bmatrix}\right) \quad \begin{bmatrix} -3 & -7 \\ 2 & 9 \\ 2 \end{bmatrix}$$

For the first system, $x=-3$ and $y=2$. For the second system, $x=-7$ and $y=9/2$.

► sin

Catalog > 

Expr ► sin

Note: You can insert this operator from the computer keyboard by typing `@>sin`.

Represents *Expr* in terms of sine. This is a display conversion operator. It can be used only at the end of the entry line.

► sin reduces all powers of $\cos(\dots)$ modulo $1-\sin(\dots)^2$ so that any remaining powers of $\sin(\dots)$ have exponents in the range (0, 2). Thus, the result will be free of $\cos(\dots)$ if and only if $\cos(\dots)$ occurs in the given expression only to even powers.

Note: This conversion operator is not supported in Degree or Gradian Angle modes. Before using it, make sure that the Angle mode is set to Radians and that *Expr* does not contain explicit references to degree or gradian angles.

$$\frac{(\cos(x))^2 \text{► sin}}{1 - (\sin(x))^2}$$

sin()

 key

sin(Expr1) ⇒ *expression*

In Degree angle mode:

sin() **key****sin(List1)** ⇒ *list***sin(Expr1)** returns the sine of the argument as an expression.**sin(List1)** returns a list of the sines of all elements in *List1*.**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, g, or r to override the angle mode setting temporarily.

$\sin\left(\frac{\pi}{4}\right)$	$\frac{\sqrt{2}}{2}$
$\sin(45)$	$\frac{\sqrt{2}}{2}$
$\sin(\{0,60,90\})$	$\left\{0, \frac{\sqrt{3}}{2}, 1\right\}$

In Gradian angle mode:

$\sin(50)$	$\frac{\sqrt{2}}{2}$
------------	----------------------

In Radian angle mode:

$\sin\left(\frac{\pi}{4}\right)$	$\frac{\sqrt{2}}{2}$
$\sin(45^\circ)$	$\frac{\sqrt{2}}{2}$

sin(squareMatrix1) ⇒ *squareMatrix*Returns the matrix sine of *squareMatrix1*. This is not the same as calculating the sine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

$\sin\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$	$\begin{bmatrix} 0.9424 & -0.04542 & -0.031999 \\ -0.045492 & 0.949254 & -0.020274 \\ -0.048739 & -0.00523 & 0.961051 \end{bmatrix}$
---	--

sin-1() **key****sin-1(Expr1)** ⇒ *expression***sin-1(List1)** ⇒ *list***sin-1(Expr1)** returns the angle whose sine is *Expr1* as an expression.**sin-1(List1)** returns a list of the inverse sines of each element of *List1*.**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

In Degree angle mode:

$\sin^{-1}(1)$	90
----------------	----

In Gradian angle mode:

$\sin^{-1}(1)$	100
----------------	-----

In Radian angle mode:

$\sin^{-1}(\{0,0.2,0.5\})$	$\{0,0.201358,0.523599\}$
----------------------------	---------------------------

sin-1()



Note: You can insert this function from the keyboard by typing `arcsin(...)`.

sin-1(*squareMatrix1*) \Rightarrow *squareMatrix*

Returns the matrix inverse sine of *squareMatrix1*. This is not the same as calculating the inverse sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

$$\sin^{-1}\left(\begin{bmatrix} 1 & 5 \\ 4 & 2 \end{bmatrix}\right)$$

$$\begin{bmatrix} -0.174533-0.12198 \cdot i & 1.74533-2.35591 \cdot i \\ 1.39626-1.88473 \cdot i & 0.174533-0.593162 \cdot i \end{bmatrix}$$

sinh()

Catalog >

sinh(*Expr1*) \Rightarrow *expression*

$\sinh(1.2)$	1.50946
--------------	---------

sinh(*List1*) \Rightarrow *list*

$\sinh(\{0,1,2,3\})$	$\{0,1.50946,10.0179\}$
----------------------	-------------------------

sinh (*Expr1*) returns the hyperbolic sine of the argument as an expression.

sinh (*List1*) returns a list of the hyperbolic sines of each element of *List1*.

sinh(*squareMatrix1*) \Rightarrow *squareMatrix*

Returns the matrix hyperbolic sine of *squareMatrix1*. This is not the same as calculating the hyperbolic sine of each element. For information about the calculation method, refer to **cos()**.

In Radian angle mode:

$$\sinh\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$$

360.954	305.708	239.604
352.912	233.495	193.564
298.632	154.599	140.251

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

sinh-1()

Catalog >

sinh-1(*Expr1*) \Rightarrow *expression*

$\sinh^{-1}(0)$	0
-----------------	---

sinh-1(*List1*) \Rightarrow *list*

$\sinh^{-1}(\{0,2,1,3\})$	$\{0,1.48748,\sinh^{-1}(3)\}$
---------------------------	-------------------------------

sinh-1(*Expr1*) returns the inverse hyperbolic sine of the argument as an expression.

sinh-1(List1) returns a list of the inverse hyperbolic sines of each element of *List1*.

Note: You can insert this function from the keyboard by typing **arcsinh(...)**.

sinh-1(squareMatrix1) \Rightarrow *squareMatrix*

Returns the matrix inverse hyperbolic sine of *squareMatrix1*. This is not the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

$$\sinh^{-1} \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \begin{bmatrix} 0.041751 & 2.15557 & 1.1582 \\ 1.46382 & 0.926568 & 0.112557 \\ 2.75079 & -1.5283 & 0.57268 \end{bmatrix}$$

SinReg

SinReg X, Y, [Iterations],[Period],[Category, Include]]

Computes the sinusoidal regression on lists *X* and *Y*. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Iterations is a value that specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

Period specifies an estimated period. If omitted, the difference between values in *X* should be equal and in sequential order. If you specify *Period*, the differences between *x* values can be unequal.

Category is a list of category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

The output of **SinReg** is always in radians, regardless of the angle mode setting.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.RegEqn	Regression Equation: $a \cdot \sin(bx+c)+d$
stat.a, stat.b, stat.c, stat.d	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

solve()

solve(Equation, Var) \Rightarrow Boolean expression

solve(Equation, Var=Guess) \Rightarrow Boolean expression

solve(Inequality, Var) \Rightarrow Boolean expression

Returns candidate real solutions of an equation or an inequality for *Var*. The goal is to return candidates for all solutions. However, there might be equations or inequalities for which the number of solutions is infinite.

Solution candidates might not be real finite solutions for some combinations of values for undefined variables.

$$\text{solve}(a \cdot x^2 + b \cdot x + c = 0, x)$$

$$x = \frac{\sqrt{b^2 - 4 \cdot a \cdot c} - b}{2 \cdot a} \text{ or } x = \frac{-\left(\sqrt{b^2 - 4 \cdot a \cdot c} + b\right)}{2 \cdot a}$$

Ans|a=1 and b=1 and c=1

$$x = \frac{-1 + \sqrt{3}}{2} \cdot i \text{ or } x = \frac{-1 - \sqrt{3}}{2} \cdot i$$

solve()

For the Auto setting of the **Auto** or **Approximate** mode, the goal is to produce exact solutions when they are concise, and supplemented by iterative searches with approximate arithmetic when exact solutions are impractical.

Due to default cancellation of the greatest common divisor from the numerator and denominator of ratios, solutions might be solutions only in the limit from one or both sides.

For inequalities of types \geq , \leq , $<$, or $>$, explicit solutions are unlikely unless the inequality is linear and contains only *Var*.

For the Exact mode, portions that cannot be solved are returned as an implicit equation or inequality.

Use the constraint (“|”) operator to restrict the solution interval and/or other variables that occur in the equation or inequality. When you find a solution in one interval, you can use the inequality operators to exclude that interval from subsequent searches.

false is returned when no real solutions are found. true is returned if **solve()** can determine that any finite real value of *Var* satisfies the equation or inequality.

Since **solve()** always returns a Boolean result, you can use “and,” “or,” and “not” to combine results from **solve()** with each other or with other Boolean expressions.

Solutions might contain a unique new undefined constant of the form nj with j being an integer in the interval 1–255. Such variables designate an arbitrary integer.

$$\text{solve}((x-a) \cdot e^x = x \cdot (x-a), x) \quad x=a \text{ or } x=-0.567143$$

$$(x+1) \cdot \frac{x-1}{x-1} + x-3 \quad 2 \cdot x-2$$

$$\text{solve}(5 \cdot x-2 \geq 2 \cdot x, x) \quad x \geq \frac{2}{3}$$

$$\text{exact}(\text{solve}((x-a) \cdot e^x = x \cdot (x-a), x)) \quad e^x + x = 0 \text{ or } x = a$$

In Radian angle mode:

$$\text{solve}(\tan(x) = \frac{1}{x}, x) | x > 0 \text{ and } x < 1 \quad x = 0.860334$$

$$\begin{array}{ll} \text{solve}(x=x+1, x) & \text{false} \\ \text{solve}(x=x, x) & \text{true} \end{array}$$

$$2 \cdot x-1 \leq 1 \text{ and solve}(x^2 \neq 9, x) \quad x \neq -3 \text{ and } x \leq 1$$

In Radian angle mode:

$$\text{solve}(\sin(x)=0, x) \quad x = n1 \cdot \pi$$

In Real mode, fractional powers having odd denominators denote only the real branch. Otherwise, multiple branched expressions such as fractional powers, logarithms, and inverse trigonometric functions denote only the principal branch. Consequently, **solve()** produces only solutions corresponding to that one real or principal branch.

Note: See also **cSolve()**, **cZeros()**, **nSolve()**, and **zeros()**.

solve(*Eqn1* and *Eqn2*[and ...],
VarOrGuess1, *VarOrGuess2*[, ...])

⇒ *Boolean expression*

solve(*SystemOfEqns*, *VarOrGuess1*,
VarOrGuess2[, ...])

⇒ *Boolean expression*

solve{*Eqn1*, *Eqn2* [...]}
{*VarOrGuess1*, *VarOrGuess2* [, ...]}

⇒ *Boolean expression*

Returns candidate real solutions to the simultaneous algebraic equations, where each *VarOrGuess* specifies a variable that you want to solve for.

You can separate the equations with the **and** operator, or you can enter a *SystemOfEqns* using a template from the Catalog. The number of *VarOrGuess* arguments must match the number of equations. Optionally, you can specify an initial guess for a variable. Each *VarOrGuess* must have the form:

variable

– or –

variable = *real or non-real number*

For example, *x* is valid and so is *x=3*.

$\text{solve}\left(x^{\frac{1}{3}}=-1,x\right)$	$x=-1$
$\text{solve}\left(\sqrt{x=2},x\right)$	false
$\text{solve}\left(-\sqrt{x=2},x\right)$	$x=4$

$\text{solve}\left(y=x^2-2 \text{ and } x+2y=1, \{x,y\}\right)$	
$x=\frac{-3}{2} \text{ and } y=\frac{1}{4} \text{ or } x=1 \text{ and } y=-1$	

If all of the equations are polynomials and if you do NOT specify any initial guesses, **solve()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine all real solutions.

For example, suppose you have a circle of radius r at the origin and another circle of radius r centered where the first circle crosses the positive x -axis. Use **solve()** to find the intersections.

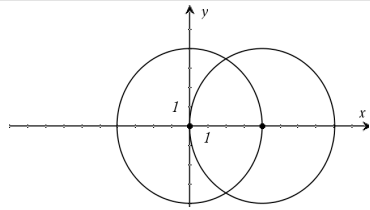
As illustrated by r in the example to the right, simultaneous polynomial equations can have extra variables that have no values, but represent given numeric values that could be substituted later.

You can also (or instead) include solution variables that do not appear in the equations. For example, you can include z as a solution variable to extend the previous example to two parallel intersecting cylinders of radius r .

The cylinder solutions illustrate how families of solutions might contain arbitrary constants of the form ck , where k is an integer suffix from 1 through 255.

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list solution variables. If your initial choice exhausts memory or your patience, try rearranging the variables in the equations and/or *varOrGuess* list.

If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in the solution variables, **solve()** uses Gaussian elimination to attempt to determine all real solutions.



$$\text{solve}\left(x^2+y^2=r^2 \text{ and } (x-r)^2+y^2=r^2, \{x,y\}\right)$$

$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2} \text{ or } x=\frac{r}{2} \text{ and } y=-\frac{\sqrt{3}\cdot r}{2}$$

$$\text{solve}\left(x^2+y^2=r^2 \text{ and } (x-r)^2+y^2=r^2, \{x,y,z\}\right)$$

$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2} \text{ and } z=c1 \text{ or } x=\frac{r}{2} \text{ and } y=-\frac{\sqrt{3}\cdot r}{2} \text{ and } z=c1$$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

$$\text{solve}\left(x+e^z\cdot y=1 \text{ and } x-y=\sin(z), \{x,y\}\right)$$

$$x=\frac{e^z\cdot \sin(z)+1}{e^z+1} \text{ and } y=\frac{-(\sin(z)-1)}{e^z+1}$$

solve()

Catalog > 

If a system is neither polynomial in all of its variables nor linear in its solution variables, **solve()** determines at most one solution using an approximate iterative method. To do so, the number of solution variables must equal the number of equations, and all other variables in the equations must simplify to numbers.

Each solution variable starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional solutions one by one. For convergence, a guess may have to be rather close to a solution.

```
solve(e^z*y=1 and y=sin(z),{y,z})
y=2.812E-10 and z=21.9911 or y=0.001871P
```

To see the entire result,
press **▲** and then use **◀** and **▶** to move the
cursor.

```
solve(e^z*y=1 and y=sin(z),{y,z=2*pi})
y=0.001871 and z=6.28131
```

SortA

Catalog > 

SortA *List1* [, *List2*] [, *List3*]...

SortA *Vector1* [, *Vector2*] [, *Vector3*]...

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 251.

```
{2,1,4,3} → list1      {2,1,4,3}
SortA list1             Done
list1                  {1,2,3,4}
{4,3,2,1} → list2      {4,3,2,1}
SortA list2,list1      Done
list2                  {1,2,3,4}
list1                  {4,3,2,1}
```

SortD

Catalog >

SortD $List1[, List2][, List3]...$

SortD $Vector1[, Vector2][, Vector3]...$

Identical to **SortA**, except **SortD** sorts the elements in descending order.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 251.

$\{2,1,4,3\} \rightarrow list1$

$\{2,1,4,3\}$

$\{1,2,3,4\} \rightarrow list2$

$\{1,2,3,4\}$

SortD $list1, list2$

Done

$list1$

$\{4,3,2,1\}$

$list2$

$\{3,4,1,2\}$

► Sphere

Catalog >

Vector ► **Sphere**

Note: You can insert this operator from the computer keyboard by typing @>**Sphere**.

Displays the row or column vector in spherical form $[\rho \angle \theta \angle \phi]$.

Vector must be of dimension 3 and can be either a row or a column vector.

Note: ► **Sphere** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

Note: To force an approximate result,

Handheld: Press .

Windows®: Press **Ctrl+Enter**.

Macintosh®: Press **⌘+Enter**.

iPad®: Hold **enter**, and select .

$\left[\begin{array}{ccc} 1 & 2 & 3 \end{array} \right] \text{►Sphere}$

$[3.74166 \angle 1.10715 \angle 0.640522]$

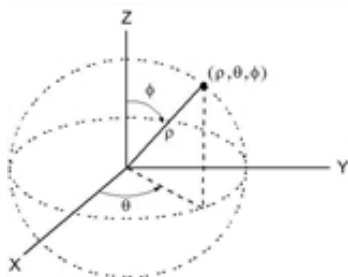
$\left(\left[\begin{array}{ccc} 2 & \angle \frac{\pi}{4} & 3 \end{array} \right] \right) \text{►Sphere}$

$[3.60555 \angle 0.785398 \angle 0.588003]$

Press

$\left(\left[\begin{array}{ccc} 2 & \angle \frac{\pi}{4} & 3 \end{array} \right] \right) \text{►Sphere}$

$\left[\sqrt{13} \angle \frac{\pi}{4} \angle \sin^{-1} \left(\frac{2 \cdot \sqrt{13}}{13} \right) \right]$



sqrt()

sqrt(*Expr1*) ⇒ *expression*

$$\sqrt{4} \qquad 2$$

sqrt(*List1*) ⇒ *list*

$$\sqrt{\{9,a,4\}} \qquad \{3,\sqrt{a},2\}$$

Returns the square root of the argument.

For a list, returns the square roots of all the elements in *List1*.

Note: See also **Square root template**, page 1.

stat.results

stat.results

Displays results from a statistics calculation.

The results are displayed as a set of name-value pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

Note: Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

$$xlist=\{1,2,3,4,5\} \qquad \{1,2,3,4,5\}$$

$$ylist=\{4,8,11,14,17\} \qquad \{4,8,11,14,17\}$$

LinRegMx *xlist,ylist,1: stat.results*

"Title"	"Linear Regression (mx+b)"
"RegEqn"	"m*x+b"
"m"	3.2
"b"	1.2
"r ² "	0.996109
"t"	0.998053
"Resid"	" {... } "

<i>stat.values</i>	"Linear Regression (mx+b)"
	"m*x+b"
	3.2
	1.2
	0.996109
	0.998053
	" {-0.4,0.4,0.2,0,-0.2} "

stat.a	stat.dfDenom	stat.MedianY	stat.Q3X	stat.SSBLOCK
stat.AdjR ²	stat.dfBlock	stat.MEPred	stat.Q3Y	stat.SSCol
stat.b	stat.dfCol	stat.MinX	stat.r	stat.SSX
stat.b0	stat.dfError	stat.MinY	stat.r ²	stat.SSY
stat.b1	stat.dfInteract	stat.MS	stat.RegEqn	stat.SSError
stat.b2	stat.dfReg	stat.MSBlock	stat.Resid	stat.SSInteract
stat.b3	stat.dfNumer	stat.MSCol	stat.ResidTrans	stat.SSReg
stat.b4	stat.dfRow	stat.MSError	stat.σ _x	stat.SSRow
stat.b5	stat.DW	stat.MSInteract	stat.σ _y	stat.tList
stat.b6	stat.e	stat.MSReg	stat.σ _{x1}	stat.UpperPred
stat.b7	stat.ExpMatrix	stat.MSRow	stat.σ _{x2}	stat.UpperVal
stat.b8	stat.F	stat.n	stat.Σ _x	stat.̄ _x
stat.b9	stat.FBlock	stat.̂ _p	stat.Σ _x ²	stat.̄ _{x1}
stat.b10	stat.Fcol	stat.̂ _{p1}	stat.Σ _{xy}	stat.̄ _{x2}
stat.bList	stat.FInteract	stat.̂ _{p2}	stat.Σ _y	stat.̄ _x Diff
stat.χ ²	stat.FreqReg	stat.̂ _p Diff	stat.Σ _y ²	stat.̄ _x List
stat.c	stat.Frow	stat.PList	stat.s	stat.XReg
stat.CLower	stat.Leverage	stat.PVal	stat.SE	stat.XVal
stat.CLowerList	stat.LowerPred	stat.PValBlock	stat.SEList	stat.XValList
stat.CompList	stat.LowerVal	stat.PValCol	stat.SEPred	stat.ȳ
stat.CompMatrix	stat.m	stat.PValInteract	stat.sResid	stat.ŷ
stat.CookDist	stat.MaxX	stat.PValRow	stat.SEslope	stat.ŷList
stat.CUpper	stat.MaxY	stat.Q1X	stat.sp	stat.YReg
stat.CUpperList	stat.ME	stat.Q1Y	stat.SS	
stat.d	stat.MedianX			

Note: Each time the Lists & Spreadsheet application calculates statistical results, it copies the “stat.” group variables to a “stat#.” group, where # is a number that is incremented automatically. This lets you maintain previous results while performing multiple calculations.

stat.values

Catalog > 

stat.values

See the **stat.results** example.

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike **stat.results**, **stat.values** omits the names associated with the values.

You can copy a value and paste it into other locations.

stDevPop()**stDevPop**(*List* [, *freqList*]) ⇒ *expression*Returns the population standard deviation of the elements in *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**Note:** *List* must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 251.**stDevPop**(*Matrix1* [, *freqMatrix*]) ⇒ *matrix*Returns a row vector of the population standard deviations of the columns in *Matrix1*.Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.**Note:** *Matrix1* must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 251.

In Radian angle and auto modes:

$$\text{stDevPop}\{\{a,b,c\}\} = \frac{\sqrt{2 \cdot (a^2 - a \cdot (b+c) + b^2 - b \cdot c + c^2)}}{3}$$

$$\text{stDevPop}\{\{1,2,5,-6,3,-2\}\} = \frac{\sqrt{465}}{6}$$

$$\text{stDevPop}\{\{1.3,2.5,-6.4\},\{3,2,5\}\} = 4.11107$$

$$\text{stDevPop}\left(\begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{bmatrix}, \begin{bmatrix} 4 \cdot \sqrt{6} & \sqrt{78} & 2 \cdot \sqrt{6} \\ 3 & 3 & 3 \end{bmatrix}\right)$$

$$\text{stDevPop}\left(\begin{bmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{bmatrix}, \begin{bmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{bmatrix}\right) = [2.52608 \quad 5.21506]$$

stDevSamp()**stDevSamp**(*List* [, *freqList*]) ⇒ *expression*Returns the sample standard deviation of the elements in *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**Note:** *List* must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 251.

$$\text{stDevSamp}\{\{a,b,c\}\} = \frac{\sqrt{3 \cdot (a^2 - a \cdot (b+c) + b^2 - b \cdot c + c^2)}}{3}$$

$$\text{stDevSamp}\{\{1,2,5,-6,3,-2\}\} = \frac{\sqrt{62}}{2}$$

$$\text{stDevSamp}\{\{1.3,2.5,-6.4\},\{3,2,5\}\} = 4.33345$$

stDevSamp()

Catalog >

stDevSamp(*Matrix1*[, *freqMatrix*]) ⇒ *matrix*

Returns a row vector of the sample standard deviations of the columns in *Matrix1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

Note: *Matrix1* must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 251.

$\text{stDevSamp}\left(\begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{bmatrix}\right)$	$[4 \ \sqrt{13} \ 2]$
$\text{stDevSamp}\left(\begin{bmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{bmatrix}, \begin{bmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{bmatrix}\right)$	$[2.7005 \ 5.44695]$

Stop

Catalog >

Stop

Programming command: Terminates the program.

Stop is not allowed in functions.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

<i>i</i> :=0	0
Define <i>prog1</i> ()=Prgm	<i>Done</i>
For <i>i</i> ,1,10,1	
If <i>i</i> =5	
Stop	
EndFor	
EndPrgm	
<i>prog1</i> ()	<i>Done</i>
<i>i</i>	5

Store

See →(store), page 233.

string()

Catalog >

string(*Expr*) ⇒ *string*

Simplifies *Expr* and returns the result as a character string.

$\text{string}(1.2345)$	"1.2345"
$\text{string}(1+2)$	"3"
$\text{string}(\cos(x)+\sqrt{3})$	"cos(x)+√(3)"

subMat()

Catalog >

subMat(*MatrixI*[, *startRow*][, *startCol*][, *endRow*][, *endCol*]) ⇒ *matrix*Returns the specified submatrix of *MatrixI*.Defaults: *startRow*=1, *startCol*=1, *endRow*=last row, *endCol*=last column.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
<code>subMat(m1,2,1,3,2)</code>	$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$
<code>subMat(m1,2,2)</code>	$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$

Sum (Sigma)See $\Sigma()$, page 224.**sum()**

Catalog >

sum(*List*[, *Start*[, *End*]]) ⇒ *expression*Returns the sum of all elements in *List*.*Start* and *End* are optional. They specify a range of elements.Any void argument produces a void result. Empty (void) elements in *List* are ignored. For more information on empty elements, see page 251.**sum**(*MatrixI*[, *Start*[, *End*]]) ⇒ *matrix*Returns a row vector containing the sums of all elements in the columns in *MatrixI*.*Start* and *End* are optional. They specify a range of rows.Any void argument produces a void result. Empty (void) elements in *MatrixI* are ignored. For more information on empty elements, see page 251.

<code>sum({1,2,3,4,5})</code>	15
<code>sum({a,2·a,3·a})</code>	6·a
<code>sum(seq(n,n,1,10))</code>	55
<code>sum({1,3,5,7,9},3)</code>	21

<code>sum($\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix})$</code>	$[5 \ 7 \ 9]$
<code>sum($\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix})$</code>	$[12 \ 15 \ 18]$
<code>sum($\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, 2, 3)$</code>	$[11 \ 13 \ 15]$

sumIf()

Catalog >

sumIf(*List*,*Criteria*[, *SumList*]) ⇒ *value*Returns the accumulated sum of all elements in *List* that meet the specified *Criteria*. Optionally, you can specify an alternate list, *sumList*, to supply the elements to accumulate.

<code>sumIf({1,2,e,3,π,4,5,6},2.5<?<4.5)</code>	$e + \pi + 7$
<code>sumIf({1,2,3,4},2<?<5,{10,20,30,40})</code>	70

List can be an expression, list, or matrix.
SumList, if specified, must have the same dimension(s) as *List*.

Criteria can be:

- A value, expression, or string. For example, **34** accumulates only those elements in *List* that simplify to the value 34.
- A Boolean expression containing the symbol ? as a placeholder for each element. For example, **?<10** accumulates only those elements in *List* that are less than 10.

When a *List* element meets the *Criteria*, the element is added to the accumulating sum. If you include *sumList*, the corresponding element from *sumList* is added to the sum instead.

Within the Lists & Spreadsheet application, you can use a range of cells in place of *List* and *sumList*.

Empty (void) elements are ignored. For more information on empty elements, see page 251.

Note: See also **countIf()**, page 35.

system(*Eqn1*[, *Eqn2*[, *Eqn3*[, ...]]])

system(*Expr1*[, *Expr2*[, *Expr3*[, ...]]])

$$\text{solve} \left(\begin{cases} x+y=0 \\ x-y=8 \end{cases}, x, y \right) \quad x=4 \text{ and } y=-4$$

Returns a system of equations, formatted as a list. You can also create a system by using a template.

Note: See also **System of equations**, page 3.

T (transpose)

Catalog >  $MatrixIT \Rightarrow matrix$ Returns the complex conjugate transpose of $Matrix1$.**Note:** You can insert this operator from the computer keyboard by typing @t.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T$	$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^T$	$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$
$\begin{bmatrix} 1+i & 2+i \\ 3+i & 4+i \end{bmatrix}^T$	$\begin{bmatrix} 1-i & 3-i \\ 2-i & 4-i \end{bmatrix}$

tan()

 key $\tan(Expr1) \Rightarrow expression$ $\tan(List1) \Rightarrow list$ $\tan(Expr1)$ returns the tangent of the argument as an expression. $\tan(List1)$ returns a list of the tangents of all elements in $List1$.**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, g or r to override the angle mode setting temporarily.

In Degree angle mode:

$\tan\left(\frac{\pi}{4}\right)$	1
$\tan(45)$	1
$\tan(\{0,60,90\})$	$\{0,\sqrt{3},undef\}$

In Gradian angle mode:

$\tan\left(\frac{\pi}{4}\right)$	1
$\tan(50)$	1
$\tan(\{0,50,100\})$	$\{0,1,undef\}$

In Radian angle mode:

$\tan\left(\frac{\pi}{4}\right)$	1
$\tan(45^\circ)$	1
$\tan\left(\left\{\pi, \frac{\pi}{3}, \pi, \frac{\pi}{4}\right\}\right)$	$\{0,\sqrt{3},0,1\}$

In Radian angle mode:

$\tan\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$	$\begin{bmatrix} -28.2912 & 26.0887 & 11.1142 \\ 12.1171 & -7.83536 & -5.48138 \\ 36.8181 & -32.8063 & -10.4594 \end{bmatrix}$
---	--

 $\tan(squareMatrix1) \Rightarrow squareMatrix$ Returns the matrix tangent of $squareMatrix1$. This is not the same as calculating the tangent of each element. For information about the calculation method, refer to **cos()**.

tan()



squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

tan-1()



tan-1(*Expr1*) ⇒ *expression*

In Degree angle mode:

tan-1(*List1*) ⇒ *list*

$\tan^{-1}(1)$	45
----------------	----

tan-1(*Expr1*) returns the angle whose tangent is *Expr1* as an expression.

In Gradian angle mode:

tan-1(*List1*) returns a list of the inverse tangents of each element of *List1*.

$\tan^{-1}(1)$	50
----------------	----

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

In Radian angle mode:

Note: You can insert this function from the keyboard by typing **arctan(...)**.

$\tan^{-1}\{0,0.2,0.5\}$	$\{0,0.197396,0.463648\}$
--------------------------	---------------------------

tan-1(*squareMatrix1*) ⇒ *squareMatrix*

In Radian angle mode:

Returns the matrix inverse tangent of *squareMatrix1*. This is not the same as calculating the inverse tangent of each element. For information about the calculation method, refer to **cos()**.

$\tan^{-1}\begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$	$\begin{bmatrix} -0.083658 & 1.26629 & 0.62263 \\ 0.748539 & 0.630015 & -0.070012 \\ 1.68608 & -1.18244 & 0.455126 \end{bmatrix}$
---	---

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

tangentLine()



tangentLine(*Expr1*,*Var*,*Point*) ⇒ *expression*

$\text{tangentLine}(x^2, x, 1)$	$2 \cdot x - 1$
---------------------------------	-----------------

tangentLine(*Expr1*,*Var*=*Point*) ⇒ *expression*

$\text{tangentLine}((x-3)^2-4, x=3)$	-4
--------------------------------------	----

Returns the tangent line to the curve represented by *Expr1* at the point specified in *Var*=*Point*.

$\text{tangentLine}\left(\frac{1}{x^3}, x=0\right)$	$x=0$
---	-------

$\text{tangentLine}(\sqrt{x^2-4}, x=2)$	undef
---	-------

$x:=3: \text{tangentLine}(x^2, x, 1)$	5
---------------------------------------	---

tangentLine()

Catalog > 

Make sure that the independent variable is not defined. For example, if $f1(x):=5$ and $x:=3$, then **tangentLine**($f1(x),x,2$) returns "false."

tanh()

Catalog > 

tanh(*Expr1*) \Rightarrow *expression*

$\tanh(1.2)$ 0.833655

tanh(*List1*) \Rightarrow *list*

$\tanh(\{0,1\})$ $\{0,\tanh(1)\}$

tanh(*Expr1*) returns the hyperbolic tangent of the argument as an expression.

tanh(*List1*) returns a list of the hyperbolic tangents of each element of *List1*.

tanh(*squareMatrix1*) \Rightarrow *squareMatrix*

In Radian angle mode:

Returns the matrix hyperbolic tangent of *squareMatrix1*. This is not the same as calculating the hyperbolic tangent of each element. For information about the calculation method, refer to **cos()**.

$\tanh\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$

-0.097966	0.933436	0.425972
0.488147	0.538881	-0.129382
1.28295	-1.03425	0.428817

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

tanh-1()

Catalog > 

tanh-1(*Expr1*) \Rightarrow *expression*

In Rectangular complex format:

tanh-1(*List1*) \Rightarrow *list*

$\tanh^{-1}(0)$ 0

tanh-1(*Expr1*) returns the inverse hyperbolic tangent of the argument as an expression.

$\tanh^{-1}(\{1,2,3\})$

$$\left\{ \text{undef}, 0.518046 - 1.5708 \cdot i, \frac{\ln(2)}{2} - \frac{\pi}{2} \cdot i \right\}$$

tanh-1(*List1*) returns a list of the inverse hyperbolic tangents of each element of *List1*.

Note: You can insert this function from the keyboard by typing **arctanh**(...).

tanh-1(*squareMatrix1*) \Rightarrow *squareMatrix*

In Radian angle mode and Rectangular complex format:

tanh-1()

Catalog >

Returns the matrix inverse hyperbolic tangent of *squareMatrix1*. This is not the same as calculating the inverse hyperbolic tangent of each element. For information about the calculation method, refer to **cos()**.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

$$\tanh^{-1}\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$$

$$\begin{bmatrix} -0.099353+0.164058\cdot i & 0.267834-1.4908 \\ -0.087596-0.725533\cdot i & 0.479679-0.94730 \\ 0.511463-2.08316\cdot i & -0.878563+1.7901 \end{bmatrix}$$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

taylor()

Catalog >

taylor(Expr1, Var, Order[, Point]) \Rightarrow *expression*

Returns the requested Taylor polynomial. The polynomial includes non-zero terms of integer degrees from zero through *Order* in (*Var* minus *Point*). **taylor()** returns itself if there is no truncated power series of this order, or if it would require negative or fractional exponents. Use substitution and/or temporary multiplication by a power of (*Var* minus *Point*) to determine more general power series.

Point defaults to zero and is the expansion point.

taylor($e^{\sqrt{x}}$, x , 2)	taylor($e^{\sqrt{x}}$, x , 2, 0)
taylor(e^t , t , 4) $t = \sqrt{x}$	$\frac{3}{24} + \frac{x}{6} + \frac{x}{2} + \sqrt{x} + 1$
taylor($\frac{1}{x \cdot (x-1)}$, x , 3)	taylor($\frac{1}{x \cdot (x-1)}$, x , 3, 0)
expand($\frac{\text{taylor}(\frac{x}{x \cdot (x-1)}, x, 4)}{x}$, x)	$-x^3 - x^2 - x - \frac{1}{x} - 1$

tCdf()


Catalog >

tCdf(lowBound, upBound, df) \Rightarrow *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

Computes the Student-*t* distribution probability between *lowBound* and *upBound* for the specified degrees of freedom *df*.

For $P(X \leq \text{upBound})$, set *lowBound* = $-\infty$.

tCollect()

Catalog > 

tCollect(*Expr1*) \Rightarrow *expression*

Returns an expression in which products and integer powers of sines and cosines are converted to a linear combination of sines and cosines of multiple angles, angle sums, and angle differences. The transformation converts trigonometric polynomials into a linear combination of their harmonics.

Sometimes **tCollect()** will accomplish your goals when the default trigonometric simplification does not. **tCollect()** tends to reverse transformations done by **tExpand()**. Sometimes applying **tExpand()** to a result from **tCollect()**, or vice versa, in two separate steps simplifies an expression.

$$\begin{array}{l} \text{tCollect}(\{\cos(\alpha)\}^2) \quad \frac{\cos(2 \cdot \alpha)+1}{2} \\ \text{tCollect}(\sin(\alpha) \cdot \cos(\beta)) \quad \frac{\sin(\alpha-\beta)+\sin(\alpha+\beta)}{2} \end{array}$$

tExpand()

Catalog > 

tExpand(*Expr1*) \Rightarrow *expression*

Returns an expression in which sines and cosines of integer-multiple angles, angle sums, and angle differences are expanded. Because of the identity $(\sin(x))^2+(\cos(x))^2=1$, there are many possible equivalent results. Consequently, a result might differ from a result shown in other publications.

Sometimes **tExpand()** will accomplish your goals when the default trigonometric simplification does not. **tExpand()** tends to reverse transformations done by **tCollect()**. Sometimes applying **tCollect()** to a result from **tExpand()**, or vice versa, in two separate steps simplifies an expression.

Note: Degree-mode scaling by $\pi/180$ interferes with the ability of **tExpand()** to recognize expandable forms. For best results, **tExpand()** should be used in Radian mode.

$$\begin{array}{l} \text{tExpand}(\sin(3 \cdot \phi)) \quad 4 \cdot \sin(\phi) \cdot \{\cos(\phi)\}^2 - \sin(\phi) \\ \text{tExpand}(\cos(\alpha - \beta)) \quad \cos(\alpha) \cdot \cos(\beta) + \sin(\alpha) \cdot \sin(\beta) \end{array}$$

Text*promptString*[, *DispFlag*]

Programming command: Pauses the program and displays the character string *promptString* in a dialog box.

When the user selects **OK**, program execution continues.


The optional *flag* argument can be any expression.

- If *DispFlag* is omitted or evaluates to **1**, the text message is added to the Calculator history.
- If *DispFlag* evaluates to **0**, the text message is not added to the history.

If the program needs a typed response from the user, refer to **Request**, page 149, or **RequestStr**, page 151.

Note: You can use this command within a user-defined program but not within a function.

Define a program that pauses to display each of five random numbers in a dialog box.

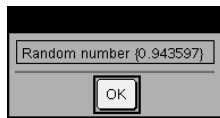
Within the Prgm...EndPrgm template, complete each line by pressing  instead of **enter**. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define text_demo()=Prgm
  For i,1,5
    strinfo:="Random number " &
string(rand(i))
    Text strinfo
  EndFor
EndPrgm
```

Run the program:

```
text_demo()
```

Sample of one dialog box:

**Interval****Interval** *List*[, *Freq*[, *CLevel*]]

(Data list input)

Interval \bar{x} , *sx*, *n*[, *CLevel*]

(Summary stats input)

Computes a *t* confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
stat. \bar{x}	Sample mean of the data sequence from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom
stat.sx	Sample standard deviation
stat.n	Length of the data sequence with sample mean

Interval_2Samp

Interval_2Samp *List1, List2, Freq1[, Freq2*
[, CLevel[, Pooled]]]

(Data list input)

Interval_2Samp $\bar{x}1, sx1, n1, \bar{x}2, sx2, n2$
[, CLevel[, Pooled]]

(Summary stats input)

Computes a two-sample *t* confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

Pooled=1 pools variances; *Pooled=0* does not pool variances.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. $\bar{x}1 - \bar{x}2$	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom

Output variable	Description
stat. \bar{x} 1, stat. \bar{x} 2	Sample means of the data sequences from the normal random distribution
stat. σ x1, stat. σ x2	Sample standard deviations for <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Number of samples in data sequences
stat.sp	The pooled standard deviation. Calculated when <i>Pooled</i> = YES

tmpCnv()

Catalog > 

tmpCnv(*Expr* °*tempUnit*, °*tempUnit2*)
 \Rightarrow *expression* °*tempUnit2*

Converts a temperature value specified by *Expr* from one unit to another. Valid temperature units are:

°C Celsius
 °F Fahrenheit
 °K Kelvin
 °R Rankine

To type °, select it from the Catalog symbols.

to type `_`, press  .

For example, 100 °C converts to 212 °F.

To convert a temperature range, use **ΔtmpCnv()** instead.

tmpCnv(100 °C, °F)	212 °F
tmpCnv(32 °F, °C)	0 °C
tmpCnv(0 °C, °K)	273.15 °K
tmpCnv(0 °F, °R)	459.67 °R

Note: You can use the Catalog to select temperature units.

ΔtmpCnv()

Catalog > 

ΔtmpCnv(*Expr* °*tempUnit*, °*tempUnit2*)
 \Rightarrow *expression* °*tempUnit2*

Note: You can insert this function from the keyboard by typing **deltaTmpCnv (...)**.

Converts a temperature range (the difference between two temperature values) specified by *Expr* from one unit to another. Valid temperature units are:

°C Celsius
 °F Fahrenheit
 °K Kelvin
 °R Rankine



ΔtmpCnv(100 °C, °F)	180 °F
ΔtmpCnv(180 °F, °C)	100 °C
ΔtmpCnv(100 °C, °K)	100 °K
ΔtmpCnv(100 °F, °R)	100 °R
ΔtmpCnv(1 °C, °F)	1.8 °F

Note: You can use the Catalog to select temperature units.

Δ tmpCnv()

Catalog > 

To enter °, select it from the Symbol Palette or type @d.

To type °, press  .

1_°C and 1_°K have the same magnitude, as do 1_°F and 1_°R. However, 1_°C is 9/5 as large as 1_°F.

For example, a 100_°C range (from 0_°C to 100_°C) is equivalent to a 180_°F range.

To convert a particular temperature value instead of a range, use **tmpCnv()**.

tPdf()

Catalog > 

tPdf(XVal,df) \Rightarrow *number* if *XVal* is a number, *list* if *XVal* is a list

Computes the probability density function (pdf) for the Student-*t* distribution at a specified *x* value with specified degrees of freedom *df*.

trace()

Catalog > 

trace(squareMatrix) \Rightarrow *expression*

Returns the trace (sum of all the elements on the main diagonal) of *squareMatrix*.

$\text{trace}\left(\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}\right)$	15
$\text{trace}\left(\begin{pmatrix} a & 0 \\ 1 & a \end{pmatrix}\right)$	$2 \cdot a$

Try

block1

Else

block2

EndTry

Executes *block1* unless an error occurs. Program execution transfers to *block2* if an error occurs in *block1*. System variable *errCode* contains the error code to allow the program to perform error recovery. For a list of error codes, see "Error codes and messages," page 261.

block1 and *block2* can be either a single statement or a series of statements separated with the ":" character.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

To see the commands **Try**, **ClrErr**, and **PassErr** in operation, enter the `eigenvals()` program shown at the right. Run the program by executing each of the following expressions.

$$\text{eigenvals}\left(\begin{bmatrix} -3 \\ -41 \\ 5 \end{bmatrix}, \begin{bmatrix} 1 & 2 & -3.1 \end{bmatrix}\right)$$

$$\text{eigenvals}\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

Note: See also **ClrErr**, page 25, and **PassErr**, page 131.

Define `prog1()`=Prgm

Try

`z:=z+1`

Disp "z incremented."

Else

Disp "Sorry, z undefined."

EndTry

EndPrgm

Done

`z:=1:prog1()`

z incremented.

Done

DelVar `z:prog1()`

Sorry, z undefined.

Done

Define `eigenvals(a,b)`=Prgm© Program `eigenvals(A,B)` displays eigenvalues of $A \cdot B$

Try

Disp "A= ",a

Disp "B= ",b

Disp ""

Disp "Eigenvalues of $A \cdot B$ are:",`eigVl(a*b)`

Else

If `errCode=230` ThenDisp "Error: Product of $A \cdot B$ must be a square matrix"

ClrErr

Else

PassErr

Endif

EndTry

EndPrgm

tTest $\mu_0, List[, Freq[, Hypoth]]$

(Data list input)

tTest $\mu_0, \bar{x}, sx, n, [Hypoth]$

(Summary stats input)

Performs a hypothesis test for a single unknown population mean μ when the population standard deviation σ is unknown. A summary of results is stored in the *stat.results* variable. (See page 176.)

Test $H_0: \mu = \mu_0$, against one of the following:

For $H_a: \mu < \mu_0$, set *Hypoth*<0

For $H_a: \mu \neq \mu_0$ (default), set *Hypoth*=0

For $H_a: \mu > \mu_0$, set *Hypoth*>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.t	$(\bar{x} - \mu_0) / (\text{stdev} / \sqrt{n})$
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom
stat. \bar{x}	Sample mean of the data sequence in <i>List</i>
stat.sx	Sample standard deviation of the data sequence
stat.n	Size of the sample

tTest_2Samp

tTest_2Samp *List1, List2[, Freq1[, Freq2*
[, Hypoth[, Pooled]]]

(Data list input)

tTest_2Samp $\bar{x}1, sx1, n1, \bar{x}2, sx2, n2[, Hypoth$
 $[, Pooled]]$

(Summary stats input)

Computes a two-sample t test. A summary of results is stored in the *stat.results* variable. (See page 176.)

Test $H_0: \mu_1 = \mu_2$, against one of the following:

For $H_a: \mu_1 < \mu_2$, set *Hypo* $\theta < 0$

For $H_a: \mu_1 \neq \mu_2$ (default), set *Hypo* $\theta = 0$

For $H_a: \mu_1 > \mu_2$, set *Hypo* $\theta > 0$

Pooled $=1$ pools variances

Pooled $=0$ does not pool variances

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.t	Standard normal value computed for the difference of means
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom for the t-statistic
stat.X1, stat.X2	Sample means of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Size of the samples
stat.sp	The pooled standard deviation. Calculated when <i>Pooled</i> $=1$.

tvfV()

tvfV(*N,I,PV,Pmt,[PpY],[CpY],[PmtAt]*)
 \Rightarrow *value*

$\text{tvfV}(120,5,0,-500,12,12)$

77641.1

Financial function that calculates the future value of money.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 195. See also **amortTbl()**, page 8.

tvml()

tvml(*N,PV,Pmt,FV,[PpY],[CpY],[PmtAt]*)
 \Rightarrow *value*

$\text{tvml}(240,100000,-1000,0,12,12)$

10.5241

tvmI()Catalog > 

Financial function that calculates the interest rate per year.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 195. See also **amortTbl()**, page 8.

tvmN()Catalog > 

tvmN($I, PV, Pmt, FV, [PpY], [CpY], [PmtAt]$)
 \Rightarrow *value*

tvmN(5,0,-500,77641,12,12)	120.
----------------------------	------

Financial function that calculates the number of payment periods.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 195. See also **amortTbl()**, page 8.

tvmPmt()Catalog > 

tvmPmt($N, I, PV, FV, [PpY], [CpY], [PmtAt]$)
 \Rightarrow *value*

tvmPmt(60,4,30000,0,12,12)	-552.496
----------------------------	----------

Financial function that calculates the amount of each payment.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 195. See also **amortTbl()**, page 8.

tvmPV()Catalog > 

tvmPV($N, I, Pmt, FV, [PpY], [CpY], [PmtAt]$)
 \Rightarrow *value*

tvmPV(48,4,-500,30000,12,12)	-3426.7
------------------------------	---------

Financial function that calculates the present value.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 195. See also **amortTbl()**, page 8.

TVM argument*	Description	Data type
N	Number of payment periods	real number
I	Annual interest rate	real number
PV	Present value	real number
Pmt	Payment amount	real number
FV	Future value	real number
PpY	Payments per year, default=1	integer > 0
CpY	Compounding periods per year, default=1	integer > 0
PmtAt	Payment due at the end or beginning of each period, default=end	integer (0=end, 1=beginning)

* These time-value-of-money argument names are similar to the TVM variable names (such as **tvm.pv** and **tvm.pmt**) that are used by the *Calculator* application's finance solver. Financial functions, however, do not store their argument values or results to the TVM variables.

TwoVar

Catalog > 

TwoVar *X*, *Y* [, [*Freq*] [, *Category*, *Include*]]

Calculates the TwoVar statistics. A summary of results is stored in the *stat.results* variable. (See page 176.)

All the lists must have equal dimension except for *Include*.

X and *Y* are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in *Freq* specifies the frequency of occurrence for each corresponding *X* and *Y* data point. The default value is 1. All elements must be integers ≥ 0 .

Category is a list of numeric category codes for the corresponding *X* and *Y* data.

Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists X , $Freq$, or $Category$ results in a void for the corresponding element of all those lists.

An empty element in any of the lists $X1$ through $X20$ results in a void for the corresponding element of all those lists. For more information on empty elements, see page 251.

Output variable	Description
stat. \bar{x}	Mean of x values
stat. Σx	Sum of x values
stat. Σx^2	Sum of x ² values
stat.sx	Sample standard deviation of x
stat. σx	Population standard deviation of x
stat.n	Number of data points
stat. \bar{y}	Mean of y values
stat. Σy	Sum of y values
stat. Σy^2	Sum of y ² values
stat.sy	Sample standard deviation of y
stat. σy	Population standard deviation of y
stat. Σxy	Sum of x*y values
stat.r	Correlation coefficient
stat.MinX	Minimum of x values
stat.Q ₁ X	1st Quartile of x
stat.MedianX	Median of x
stat.Q ₃ X	3rd Quartile of x
stat.MaxX	Maximum of x values
stat.MinY	Minimum of y values
stat.Q ₁ Y	1st Quartile of y
stat.MedY	Median of y
stat.Q ₃ Y	3rd Quartile of y

Output variable	Description
stat.MaxY	Maximum of y values
stat. $\Sigma(x-\bar{x})^2$	Sum of squares of deviations from the mean of x
stat. $\Sigma(y-\bar{y})^2$	Sum of squares of deviations from the mean of y

U

unitV()

Catalog >

unitV(*Vector1*) \Rightarrow vector

Returns either a row- or column-unit vector, depending on the form of *Vector1*.

Vector1 must be either a single-row matrix or a single-column matrix.

unitV ([a b c])	$\left[\frac{a}{\sqrt{a^2+b^2+c^2}} \quad \frac{b}{\sqrt{a^2+b^2+c^2}} \quad \frac{c}{\sqrt{a^2+b^2+c^2}} \right]$
unitV ([1 2 1])	$\left[\frac{\sqrt{6}}{6} \quad \frac{\sqrt{6}}{3} \quad \frac{\sqrt{6}}{6} \right]$
unitV $\left(\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\right)$	$\begin{bmatrix} \frac{\sqrt{14}}{14} \\ \frac{14}{\sqrt{14}} \\ 7 \\ 3 \cdot \frac{\sqrt{14}}{14} \\ 14 \end{bmatrix}$

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

unLock

Catalog >

unLock *Var1* [, *Var2*] [, *Var3*] ...

unLock *Var*.

Unlocks the specified variables or variable group. Locked variables cannot be modified or deleted.

See **Lock**, page 106, and **getLockInfo()**, page 83.

a:=65	65
Lock a	Done
getLockInfo(a)	1
a:=75	"Error: Variable is locked."
DelVar a	"Error: Variable is locked."
Unlock a	Done
a:=75	75
DelVar a	Done

varPop()Catalog > **varPop(List[, freqList])** ⇒ *expression*

$\text{varPop}\{\{5,10,15,20,25,30\}\}$	<u>875</u>
	12
<i>Ans</i> : 1.	<u>72.9167</u>

Returns the population variance of *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**Note:** *List* must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 251.

varSamp()Catalog > **varSamp(List[, freqList])** ⇒ *expression*

$\text{varSamp}\{\{a,b,c\}\}$	
	$\frac{a^2 - a \cdot (b+c) + b^2 - b \cdot c + c^2}{3}$
$\text{varSamp}\{\{1,2,5,6,3,-2\}\}$	<u>31</u>
	2
$\text{varSamp}\{\{1,3,5\},\{4,6,2\}\}$	<u>68</u>
	33

Returns the sample variance of *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**Note:** *List* must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 251.

varSamp(MatrixI[, freqMatrix]) ⇒ *matrix*

$\text{varSamp}\left(\begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ .5 & .7 & 3 \end{bmatrix}\right)$	<u>[4.75 1.03 4]</u>
$\text{varSamp}\left(\begin{bmatrix} -1.1 & 2.2 \\ 3.4 & 5.1 \\ -2.3 & 4.3 \end{bmatrix}, \begin{bmatrix} 6 & 3 \\ 2 & 4 \\ 5 & 1 \end{bmatrix}\right)$	<u>[3.91731 2.08411]</u>

Returns a row vector containing the sample variance of each column in *MatrixI*.Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *MatrixI*.

If an element in either matrix is empty (void), that element is ignored, and the corresponding element in the other matrix is also ignored. For more information on empty elements, see page 251.

Note: *Matrix1* must contain at least two rows.

W

Wait



Wait *timeInSeconds*

Suspends execution for a period of *timeInSeconds* seconds.

Wait is particularly useful in a program that needs a brief delay to allow requested data to become available.

The argument *timeInSeconds* must be an expression that simplifies to a decimal value in the range 0 through 100. The command rounds this value up to the nearest 0.1 seconds.

To cancel a **Wait** that is in progress,

- **Handheld:** Hold down the  key and press  repeatedly.
- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **iPad®:** The app displays a prompt. You can continue waiting or cancel.

Note: You can use the **Wait** command within a user-defined program but not within a function.

To wait 4 seconds:

```
Wait 4
```

To wait 1/2 second:

```
Wait 0.5
```

To wait 1.3 seconds using the variable *seccount*:

```
seccount:=1.3
Wait seccount
```

This example switches a green LED on for 0.5 seconds and then switches it off.

```
Send "SET GREEN 1 ON"
Wait 0.5
Send "SET GREEN 1 OFF"
```

warnCodes ()

Catalog > 

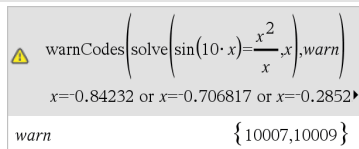
warnCodes(*Expr1*, *StatusVar*) \Rightarrow
expression

Evaluates expression *Expr1*, returns the result, and stores the codes of any generated warnings in the *StatusVar* list variable. If no warnings are generated, this function assigns *StatusVar* an empty list.

Expr1 can be any valid TI-Nspire™ or TI-Nspire™ CAS math expression. You cannot use a command or assignment as *Expr1*.

StatusVar must be a valid variable name.

For a list of warning codes and associated messages, see page 269.



The screenshot shows the TI-Nspire CAS interface. At the top, a yellow warning triangle icon is visible. The main display area contains the command: `warnCodes(solve(sin(10*x)=x^2/x),warn)`. Below the command, the results are displayed: `x=-0.84232 or x=-0.706817 or x=-0.2852`. At the bottom, a list box shows the output for the `warn` variable: `{10007,10009}`.

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

when()

Catalog > 

when(*Condition*, *trueResult* [, *falseResult*] [, *unknownResult*]) \Rightarrow *expression*

Returns *trueResult*, *falseResult*, or *unknownResult*, depending on whether *Condition* is true, false, or unknown.

Returns the input if there are too few arguments to specify the appropriate result.

Omit both *falseResult* and *unknownResult* to make an expression defined only in the region where *Condition* is true.

Use an **undef** *falseResult* to define an expression that graphs only on an interval.

when() is helpful for defining recursive functions.

<code>when(x<0,x+3)</code>	<code>x=5</code>	<code>undef</code>
-------------------------------	------------------	--------------------

<code>when(n>0,n*factorial(n-1),1)</code>	<code>factorial(n)</code>	<i>Done</i>
<code>factorial(3)</code>		6
3!		6

While *Condition**Block***EndWhile**

Executes the statements in *Block* as long as *Condition* is true.

Block can be either a single statement or a sequence of statements separated with the ":" character.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

```
Define sum_of_recip(n)=Func
  Local i,tempsum
  1 → i
  0 → tempsum
  While i ≤ n
    tempsum + 1/i → tempsum
  i + 1 → i
EndWhile
Return tempsum
EndFunc
```

<i>sum_of_recip</i> (3)	Done
	11
	6

X**xor**

BooleanExpr1 **xor** *BooleanExpr2* returns *Boolean expression*
BooleanList1 **xor** *BooleanList2* returns *Boolean list*
BooleanMatrix1 **xor** *BooleanMatrix2* returns *Boolean matrix*

true xor true	false
5 > 3 xor 3 > 5	true

Returns true if *BooleanExpr1* is true and *BooleanExpr2* is false, or vice versa.

Returns false if both arguments are true or if both are false. Returns a simplified Boolean expression if either of the arguments cannot be resolved to true or false.

Note: See **or**, page 129.

Integer1 **xor** *Integer2* ⇒ *integer*

Compares two real integers bit-by-bit using an **xor** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

In Hex base mode:

Important: Zero, not the letter O.

0h7AC36 xor 0h3D5F	0h79169
--------------------	---------

In Bin base mode:

0b100101 xor 0b100	0b100001
--------------------	----------

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ► **Base2**, page 17.

Note: See **or**, page 129.

Z

zeros()

$\text{zeros}(\text{Expr}, \text{Var}) \Rightarrow \text{list}$

$\text{zeros}(\text{Expr}, \text{Var}=\text{Guess}) \Rightarrow \text{list}$

Returns a list of candidate real values of *Var* that make *Expr*=0. **zeros()** does this by computing **exp** ► **list** ► **solve** (*Expr*=0, *Var*), *Var*).

For some purposes, the result form for **zeros()** is more convenient than that of **solve()**. However, the result form of **zeros()** cannot express implicit solutions, solutions that require inequalities, or solutions that do not involve *Var*.

Note: See also **cSolve()**, **cZeros()**, and **solve()**.

$\text{zeros}(\{\text{Expr}1, \text{Expr}2, \{\text{VarOrGuess}1, \text{VarOrGuess}2 [, \dots]\}) \Rightarrow \text{matrix}$

Returns candidate real zeros of the simultaneous algebraic expressions, where each *VarOrGuess* specifies an unknown whose value you seek.

Optionally, you can specify an initial guess for a variable. Each *VarOrGuess* must have the form:

Note: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

$$\frac{\text{zeros}(a \cdot x^2 + b \cdot x + c, x)}{\left\{ \frac{\sqrt{b^2 - 4 \cdot a \cdot c} - b}{2 \cdot a}, \frac{-\left(\sqrt{b^2 - 4 \cdot a \cdot c} + b\right)}{2 \cdot a} \right\}}$$

$$\frac{a \cdot x^2 + b \cdot x + c | x = \text{Ans}[2]}{0}$$

$$\frac{\text{exact}\left(\text{zeros}\left(a \cdot \left(e^x + x\right) \cdot \left(\text{sign}(x) - 1\right), x\right)\right)}{\text{exact}\left(\text{solve}\left(a \cdot \left(e^x + x\right) \cdot \left(\text{sign}(x) - 1\right) = 0, x\right)\right)}$$

$$e^x + x = 0 \text{ or } x > 0 \text{ or } a = 0$$

variable

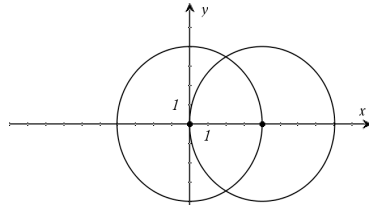
– or –

variable = real or non-real number

For example, x is valid and so is x=3.

If all of the expressions are polynomials and if you do NOT specify any initial guesses, **zeros()** uses the lexical Gröbner/Buchberger elimination method to attempt to determine all real zeros.

For example, suppose you have a circle of radius r at the origin and another circle of radius r centered where the first circle crosses the positive x-axis. Use **zeros()** to find the intersections.



As illustrated by r in the example to the right, simultaneous polynomial expressions can have extra variables that have no values, but represent given numeric values that could be substituted later.

Each row of the resulting matrix represents an alternate zero, with the components ordered the same as the *varOrGuess* list. To extract a row, index the matrix by [row].

$$\text{zeros}\left(\left\{x^2+y^2-r^2, (x-r)^2+y^2-r^2\right\}, \{x,y\}\right)$$

$\frac{r}{2}$	$\frac{-\sqrt{3}\cdot r}{2}$
$\frac{r}{2}$	$\frac{\sqrt{3}\cdot r}{2}$

Extract row 2:

$$\text{Ans}[2]$$

$\frac{r}{2}$	$\frac{\sqrt{3}\cdot r}{2}$
---------------	-----------------------------

You can also (or instead) include unknowns that do not appear in the expressions. For example, you can include z as an unknown to extend the previous example to two parallel intersecting cylinders of radius r. The cylinder zeros illustrate how families of zeros might contain arbitrary constants in the form ck, where k is an integer suffix from 1 through 255.

$$\text{zeros}\left(\left\{x^2+y^2-r^2, (x-r)^2+y^2-r^2\right\}, \{x,y,z\}\right)$$

$\frac{r}{2}$	$\frac{-\sqrt{3}\cdot r}{2}$	c1
$\frac{r}{2}$	$\frac{\sqrt{3}\cdot r}{2}$	c1

For polynomial systems, computation time or memory exhaustion may depend strongly on the order in which you list unknowns. If your initial choice exhausts memory or your patience, try rearranging the variables in the expressions and/or *varOrGuess* list.

zeros()

Catalog > 

If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in the unknowns, **zeros()** uses Gaussian elimination to attempt to determine all real zeros.

$$\text{zeros}\left(\left\{x+e^z \cdot y-1, x-y-\sin(z)\right\}, \{x, y\}\right)$$

$$\left[\begin{array}{cc} \frac{e^z \cdot \sin(z)+1}{e^z+1} & \frac{-(\sin(z)-1)}{e^z+1} \end{array} \right]$$

If a system is neither polynomial in all of its variables nor linear in its unknowns, **zeros()** determines at most one zero using an approximate iterative method. To do so, the number of unknowns must equal the number of expressions, and all other variables in the expressions must simplify to numbers.

$$\text{zeros}\left(\left\{e^z \cdot y-1, y-\sin(z)\right\}, \{y, z\}\right)$$

$$\left[\begin{array}{cc} 0.041458 & 3.18306 \\ 0.001871 & 6.28131 \\ 4.76\text{E-}11 & 1796.99 \\ 2.\text{E-}13 & 254.469 \end{array} \right]$$


Each unknown starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional zeros one by one. For convergence, a guess may have to be rather close to a zero.

$$\text{zeros}\left(\left\{e^z \cdot y-1, y-\sin(z)\right\}, \{y, z=2 \cdot \pi\}\right)$$

$$\left[\begin{array}{cc} 0.001871 & 6.28131 \end{array} \right]$$

zInterval

Catalog > 

zInterval $\sigma, \text{List}, \text{Freq}, \text{CLevel}$]]

(Data list input)

zInterval $\sigma, \bar{x}, n [, \text{CLevel}]$

(Summary stats input)

Computes a z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
stat. \bar{x}	Sample mean of the data sequence from the normal random distribution
stat.ME	Margin of error
stat.sx	Sample standard deviation

Output variable	Description
stat.n	Length of the data sequence with sample mean
stat.σ	Known population standard deviation for data sequence <i>List</i>

zInterval_1Prop

Catalog > 

zInterval_1Prop $x, n [, CLevel]$

Computes a one-proportion z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

x is a non-negative integer.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. \hat{p}	The calculated proportion of successes
stat.ME	Margin of error
stat.n	Number of samples in data sequence

zInterval_2Prop

Catalog > 

zInterval_2Prop $x1, n1, x2, n2 [, CLevel]$

Computes a two-proportion z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

$x1$ and $x2$ are non-negative integers.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. \hat{p} Diff	The calculated difference between proportions
stat.ME	Margin of error

Output variable	Description
stat. \hat{p} 1	First sample proportion estimate
stat. \hat{p} 2	Second sample proportion estimate
stat.n1	Sample size in data sequence one
stat.n2	Sample size in data sequence two

zInterval_2Samp

Catalog > 

zInterval_2Samp $\sigma_1, \sigma_2, List1, List2, Freq1$
 $[, Freq2, [CLevel]]]$

(Data list input)

zInterval_2Samp $\sigma_1, \sigma_2, \bar{x}1, n1, \bar{x}2, n2$
 $[, CLevel]$

(Summary stats input)

Computes a two-sample z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 176.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. $\bar{x}1$ - $\bar{x}2$	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat. $\bar{x}1$, stat. $\bar{x}2$	Sample means of the data sequences from the normal random distribution
stat. $\sigma x1$, stat. $\sigma x2$	Sample standard deviations for <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Number of samples in data sequences
stat.r1, stat.r2	Known population standard deviations for data sequence <i>List 1</i> and <i>List 2</i>

zTest

Catalog > 

zTest $\mu_0, \sigma, List, [Freq[, Hypoth]]$

(Data list input)

zTest $\mu_0, \sigma, \bar{x}, n[, Hypoth]$

(Summary stats input)

Performs a z test with frequency *freqlist*. A summary of results is stored in the *stat.results* variable. (See page 176.)

Test $H_0: \mu = \mu_0$, against one of the following:

For $H_a: \mu < \mu_0$, set *Hypoth*<0

For $H_a: \mu \neq \mu_0$ (default), set *Hypoth*=0

For $H_a: \mu > \mu_0$, set *Hypoth*>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.z	$(\bar{x} - \mu_0) / (\sigma / \sqrt{n})$
stat.P Value	Least probability at which the null hypothesis can be rejected
stat. \bar{x}	Sample mean of the data sequence in <i>List</i>
stat.sx	Sample standard deviation of the data sequence. Only returned for <i>Data</i> input.
stat.n	Size of the sample

zTest_1Prop

Output variable	Description
stat.p0	Hypothesized population proportion
stat.z	Standard normal value computed for the proportion
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat. \hat{p}	Estimated sample proportion
stat.n	Size of the sample

zTest_2Prop

zTest_2Prop $x1, n1, x2, n2[, Hypoth]$

Computes a two-proportion z test. A summary of results is stored in the *stat.results* variable. (See page 176.)

$x1$ and $x2$ are non-negative integers.

Test $H_0: p1 = p2$, against one of the following:

For $H_a: p1 > p2$, set *Hypoth*>0

For $H_a: p1 \neq p2$ (default), set *Hypoth*=0

For $H_a: p < p0$, set *Hypoth*<0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.z	Standard normal value computed for the difference of proportions
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat. $\hat{p}1$	First sample proportion estimate
stat. $\hat{p}2$	Second sample proportion estimate
stat. \hat{p}	Pooled sample proportion estimate
stat.n1, stat.n2	Number of samples taken in trials 1 and 2

zTest_2Samp $\sigma_1, \sigma_2, List1, List2[, Freq1 [, Freq2[, Hypoth]]]$

(Data list input)

zTest_2Samp $\sigma_1, \sigma_2, \bar{x}1, n1, \bar{x}2, n2[, Hypoth]$

(Summary stats input)

Computes a two-sample z test. A summary of results is stored in the *stat.results* variable. (See page 176.)

Test $H_0: \mu1 = \mu2$, against one of the following:

For $H_a: \mu1 < \mu2$, set *Hypoth*<0

For $H_a: \mu1 \neq \mu2$ (default), set *Hypoth*=0

For $H_a: \mu1 > \mu2$, *Hypoth*>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 251.

Output variable	Description
stat.z	Standard normal value computed for the difference of means
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat. $\bar{x}1$, stat. $\bar{x}2$	Sample means of the data sequences in <i>List1</i> and <i>List2</i>
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in <i>List1</i> and <i>List2</i>
stat.n1, stat.n2	Size of the samples

Symbols

+ (add)



$Expr1 + Expr2 \Rightarrow expression$

Returns the sum of the two arguments.

56	56
56+4	60
60+4	64
64+4	68
68+4	72

$List1 + List2 \Rightarrow list$

$Matrix1 + Matrix2 \Rightarrow matrix$

Returns a list (or matrix) containing the sums of corresponding elements in *List1* and *List2* (or *Matrix1* and *Matrix2*).

Dimensions of the arguments must be equal.

$\left\{22,\pi,\frac{\pi}{2}\right\} \rightarrow I1$	$\left\{22,\pi,\frac{\pi}{2}\right\}$
$\left\{10,5,\frac{\pi}{2}\right\} \rightarrow I2$	$\left\{10,5,\frac{\pi}{2}\right\}$
$I1+I2$	$\{32,\pi+5,\pi\}$
$Ans+\{\pi,5,\pi\}$	$\{\pi+32,\pi,0\}$
$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} a+1 & b \\ c & d+1 \end{bmatrix}$

$Expr + List1 \Rightarrow list$

$List1 + Expr \Rightarrow list$

Returns a list containing the sums of *Expr* and each element in *List1*.

$Expr + Matrix1 \Rightarrow matrix$

$Matrix1 + Expr \Rightarrow matrix$

Returns a matrix with *Expr* added to each element on the diagonal of *Matrix1*. *Matrix1* must be square.

$15+\{10,15,20\}$	$\{25,30,35\}$
$\{10,15,20\}+15$	$\{25,30,35\}$

$20+\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 21 & 2 \\ 3 & 24 \end{bmatrix}$
---	--

Note: Use $\cdot+$ (dot plus) to add an expression to each element.

- (subtract)



$Expr1 - Expr2 \Rightarrow expression$

Returns *Expr1* minus *Expr2*.

6-2	4
$\pi - \frac{\pi}{6}$	$\frac{5\cdot\pi}{6}$

$List1 - List2 \Rightarrow list$

$Matrix1 - Matrix2 \Rightarrow matrix$

$\left\{22,\pi,\frac{\pi}{2}\right\} - \left\{10,5,\frac{\pi}{2}\right\}$	$\{12,\pi-5,0\}$
$\begin{bmatrix} 3 & 4 \\ -1 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$

– (subtract)



Subtracts each element in *List2* (or *Matrix2*) from the corresponding element in *List1* (or *Matrix1*), and returns the results.

Dimensions of the arguments must be equal.

$$\text{Expr} - \text{List1} \Rightarrow \text{list}$$

$$15 - \{10, 15, 20\} \quad \{5, 0, -5\}$$

$$\{10, 15, 20\} - 15 \quad \{-5, 0, 5\}$$

$$\text{List1} - \text{Expr} \Rightarrow \text{list}$$

Subtracts each *List1* element from *Expr* or subtracts *Expr* from each *List1* element, and returns a list of the results.

$$\text{Expr} - \text{Matrix1} \Rightarrow \text{matrix}$$

$$20 - \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \begin{bmatrix} 19 & -2 \\ -3 & 16 \end{bmatrix}$$

$$\text{Matrix1} - \text{Expr} \Rightarrow \text{matrix}$$

Expr - *Matrix1* returns a matrix of *Expr* times the identity matrix minus *Matrix1*. *Matrix1* must be square.

Matrix1 - *Expr* returns a matrix of *Expr* times the identity matrix subtracted from *Matrix1*. *Matrix1* must be square.

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

• (multiply)



$$\text{Expr1} \cdot \text{Expr2} \Rightarrow \text{expression}$$

$$2 \cdot 3 \cdot 4 \quad 6 \cdot 9$$

Returns the product of the two arguments.

$$x \cdot y \cdot x \quad x^2 \cdot y$$

$$\text{List1} \cdot \text{List2} \Rightarrow \text{list}$$

$$\{1, 2, 3\} \cdot \{4, 5, 6\} \quad \{4, 10, 18\}$$

Returns a list containing the products of the corresponding elements in *List1* and *List2*.

$$\left\{ \frac{2}{a}, \frac{3}{2} \right\} \cdot \left\{ a^2, \frac{b}{3} \right\} \quad \left\{ 2 \cdot a, \frac{b}{2} \right\}$$

Dimensions of the lists must be equal.

$$\text{Matrix1} \cdot \text{Matrix2} \Rightarrow \text{matrix}$$

Returns the matrix product of *Matrix1* and *Matrix2*.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \quad \begin{bmatrix} a+2 \cdot b+3 \cdot c & d+2 \cdot e+3 \cdot f \\ 4 \cdot a+5 \cdot b+6 \cdot c & 4 \cdot d+5 \cdot e+6 \cdot f \end{bmatrix}$$

The number of columns in *Matrix1* must equal the number of rows in *Matrix2*.

• (multiply)

$\boxed{\times}$ key

$Expr \bullet List1 \Rightarrow list$

$$\pi \cdot \{4,5,6\} \qquad \{4 \cdot \pi, 5 \cdot \pi, 6 \cdot \pi\}$$

$List1 \bullet Expr \Rightarrow list$

Returns a list containing the products of $Expr$ and each element in $List1$.

$Expr \bullet Matrix1 \Rightarrow matrix$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 0.01 \qquad \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & 0.04 \end{bmatrix}$$

$Matrix1 \bullet Expr \Rightarrow matrix$

Returns a matrix containing the products of $Expr$ and each element in $Matrix1$.

$$\lambda \cdot \text{identity}(3) \qquad \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}$$

Note: Use \bullet (dot multiply) to multiply an expression by each element.

/ (divide)

$\boxed{\div}$ key

$Expr1 / Expr2 \Rightarrow expression$

$$\frac{2}{3.45} \qquad 0.57971$$

Returns the quotient of $Expr1$ divided by $Expr2$.

$$\frac{x^3}{x} \qquad x^2$$

Note: See also **Fraction template**, page 1.

$List1 / List2 \Rightarrow list$

Returns a list containing the quotients of $List1$ divided by $List2$.

$$\frac{\{1,2,3\}}{\{4,5,6\}} \qquad \left\{0.25, \frac{2}{5}, \frac{1}{2}\right\}$$

Dimensions of the lists must be equal.

$Expr / List1 \Rightarrow list$

$$\frac{a}{\{3, a, \sqrt{a}\}} \qquad \left\{\frac{a}{3}, 1, \sqrt{a}\right\}$$

$List1 / Expr \Rightarrow list$

Returns a list containing the quotients of $Expr$ divided by $List1$ or $List1$ divided by $Expr$.

$$\frac{\{a, b, c\}}{a \cdot b \cdot c} \qquad \left\{\frac{1}{b \cdot c}, \frac{1}{a \cdot c}, \frac{1}{a \cdot b}\right\}$$

$Matrix1 / Expr \Rightarrow matrix$

Returns a matrix containing the quotients of $Matrix1 / Expr$.

$$\frac{\begin{bmatrix} a & b & c \end{bmatrix}}{a \cdot b \cdot c} \qquad \begin{bmatrix} \frac{1}{b \cdot c} & \frac{1}{a \cdot c} & \frac{1}{a \cdot b} \end{bmatrix}$$

$Matrix1 / Value \Rightarrow matrix$

/ (divide)

 **key**

Note: Use ./ (dot divide) to divide an expression by each element.

^ (power)

 **key**

$Expr1 \wedge Expr2 \Rightarrow expression$

$$4^2 \qquad 16$$

$List1 \wedge List2 \Rightarrow list$

$$\{a, 2, c\} \{1, b, 3\} \qquad \{a, 2^b, c^3\}$$

Returns the first argument raised to the power of the second argument.

Note: See also **Exponent template**, page 1.

For a list, returns the elements in *List1* raised to the power of the corresponding elements in *List2*.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

$Expr \wedge List1 \Rightarrow list$

$$p \{a, 2, 3\} \qquad \left\{ p^a, p^2, \frac{1}{p^3} \right\}$$

Returns *Expr* raised to the power of the elements in *List1*.

$List1 \wedge Expr \Rightarrow list$

$$\{1, 2, 3, 4\}^{-2} \qquad \left\{ 1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16} \right\}$$

Returns the elements in *List1* raised to the power of *Expr*.

$squareMatrix1 \wedge integer \Rightarrow matrix$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^2 \qquad \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

Returns *squareMatrix1* raised to the *integer* power.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} \qquad \begin{bmatrix} -2 & 1 \\ 3 & -1 \\ 2 & 2 \end{bmatrix}$$

squareMatrix1 must be a square matrix.

If *integer* = -1, computes the inverse matrix.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-2} \qquad \begin{bmatrix} 11 & -5 \\ 2 & 2 \\ -15 & 7 \\ 4 & 4 \end{bmatrix}$$

If *integer* < -1, computes the inverse matrix to an appropriate positive power.

x² (square)**[x²] key***Expr*1² ⇒ *expression*

Returns the square of the argument.

4 ²	16
----------------	----

*List*1² ⇒ *list*Returns a list containing the squares of the elements in *List*1.

{2,4,6} ²	{4,16,36}
----------------------	-----------

*squareMatrix*1² ⇒ *matrix*Returns the matrix square of *squareMatrix*1. This is not the same as calculating the square of each element. Use *.^2* to calculate the square of each element.

$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}^2$	$\begin{bmatrix} 40 & 64 & 88 \\ 49 & 79 & 109 \\ 58 & 94 & 130 \end{bmatrix}$
---	--

$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix} \wedge 2$	$\begin{bmatrix} 4 & 16 & 36 \\ 9 & 25 & 49 \\ 16 & 36 & 64 \end{bmatrix}$
--	--

.+ (dot add)**[.] [+] keys***Matrix*1 .+ *Matrix*2 ⇒ *matrix**Expr* .+ *Matrix*1 ⇒ *matrix**Matrix*1.+*Matrix*2 returns a matrix that is the sum of each pair of corresponding elements in *Matrix*1 and *Matrix*2.*Expr* .+ *Matrix*1 returns a matrix that is the sum of *Expr* and each element in *Matrix*1.

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix} .+ \begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} a+c & 6 \\ b+5 & d+3 \end{bmatrix}$
--	--

$x .+ \begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} x+c & x+4 \\ x+5 & x+d \end{bmatrix}$
---	--

.- (dot sub.)**[.] [-] keys***Matrix*1 .- *Matrix*2 ⇒ *matrix**Expr* .- *Matrix*1 ⇒ *matrix**Matrix*1.- *Matrix*2 returns a matrix that is the difference between each pair of corresponding elements in *Matrix*1 and *Matrix*2.*Expr* .- *Matrix*1 returns a matrix that is the difference of *Expr* and each element in *Matrix*1.

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix} .- \begin{bmatrix} c & 4 \\ d & 5 \end{bmatrix}$	$\begin{bmatrix} a-c & -2 \\ b-d & -2 \end{bmatrix}$
--	--

$x .- \begin{bmatrix} c & 4 \\ d & 5 \end{bmatrix}$	$\begin{bmatrix} x-c & x-4 \\ x-d & x-5 \end{bmatrix}$
---	--

.•(dot mult.)

.	x	keys
---	---	------

Matrix1 .• *Matrix2* ⇒ *matrix*

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix}$.	$\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} a \cdot c & 8 \\ 5 \cdot b & 3 \cdot d \end{bmatrix}$
--	---	--	--

Expr .• *Matrix1* ⇒ *matrix*

<i>x</i> .	$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$	$\begin{bmatrix} a \cdot x & b \cdot x \\ c \cdot x & d \cdot x \end{bmatrix}$
------------	--	--

Matrix1 .• *Matrix2* returns a matrix that is the product of each pair of corresponding elements in *Matrix1* and *Matrix2*.

Expr .• *Matrix1* returns a matrix containing the products of *Expr* and each element in *Matrix1*.

./ (dot divide)

.	÷	keys
---	---	------

Matrix1 ./ *Matrix2* ⇒ *matrix*

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix}$./	$\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} \frac{a}{c} & \frac{1}{2} \\ \frac{b}{5} & \frac{3}{d} \end{bmatrix}$
--	----	--	--

Expr ./ *Matrix1* ⇒ *matrix*

<i>x</i> ./	$\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} \frac{x}{c} & \frac{x}{4} \\ \frac{x}{5} & \frac{x}{d} \end{bmatrix}$
-------------	--	--

Matrix1 ./ *Matrix2* returns a matrix that is the quotient of each pair of corresponding elements in *Matrix1* and *Matrix2*.

Expr ./ *Matrix1* returns a matrix that is the quotient of *Expr* and each element in *Matrix1*.

.^ (dot power)

.	^	keys
---	---	------

Matrix1 .^ *Matrix2* ⇒ *matrix*

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix}$./	$\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} a^c & 16 \\ b^5 & 3^d \end{bmatrix}$
--	----	--	---

Expr .^ *Matrix1* ⇒ *matrix*

<i>x</i> .^	$\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} x^c & x^4 \\ x^5 & x^d \end{bmatrix}$
-------------	--	--

Matrix1 .^ *Matrix2* returns a matrix where each element in *Matrix2* is the exponent for the corresponding element in *Matrix1*.

Expr .^ *Matrix1* returns a matrix where each element in *Matrix1* is the exponent for *Expr*.

- (negate)

(-)	key
-----	-----

-Expr1 ⇒ *expression*

-2.43	-2.43
-------	-------

-List1 ⇒ *list*

$\{-1, 0.4, 1.2 \text{E} 19\}$	$\{1, -0.4, -1.2 \text{E} 19\}$
--------------------------------	---------------------------------

-Matrix1 ⇒ *matrix*

$-a \cdot b$	$a \cdot b$
--------------	-------------

- (negate)

 key

Returns the negation of the argument.

For a list or matrix, returns all the elements negated.

If the argument is a binary or hexadecimal integer, the negation gives the two's complement.

In Bin base mode:

Important: Zero, not the letter O.

```
-0b100101
0b11111111111111111111111111111111▶
```

To see the entire result, press \blacktriangle and then use \blacktriangleleft and \blacktriangleright to move the cursor.

% (percent)

  keys

$Expr1\% \Rightarrow expression$

$List1\% \Rightarrow list$

$Matrix1\% \Rightarrow matrix$

Returns $\frac{argument}{100}$

For a list or matrix, returns a list or matrix with each element divided by 100.

Note: To force an approximate result,

Handheld: Press  .

Windows®: Press **Ctrl+Enter**.

Macintosh®: Press $\mathcal{C}+\mathcal{E}$.

iPad®: Hold **enter**, and select .

13% 0.13

$\{ \{1,10,100\} \}\%$ $\{0.01,0.1,1.\}$

= (equal)

 key

$Expr1=Expr2 \Rightarrow Boolean\ expression$

$List1=List2 \Rightarrow Boolean\ list$

$Matrix1=Matrix2 \Rightarrow Boolean\ matrix$

Returns true if $Expr1$ is determined to be equal to $Expr2$.

Returns false if $Expr1$ is determined to not be equal to $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Example function that uses math test symbols: =, \neq , <, \leq , >, \geq

Define $g(x)=Func$

```
If  $x \leq -5$  Then
  Return 5
ElseIf  $x > -5$  and  $x < 0$  Then
  Return  $-x$ 
ElseIf  $x \geq 0$  and  $x \neq 10$  Then
  Return  $x$ 
ElseIf  $x = 10$  Then
  Return 3
EndIf
EndFunc
```

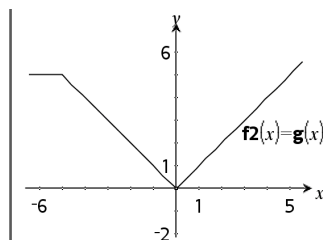
Done

= (equal)



Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Result of graphing $g(x)$



$f2(x) = g(x)$

≠ (not equal)



$Expr1 \neq Expr2 \Rightarrow$ Boolean expression

See “=” (equal) example.

$List1 \neq List2 \Rightarrow$ Boolean list

$Matrix1 \neq Matrix2 \Rightarrow$ Boolean matrix

Returns true if $Expr1$ is determined to be not equal to $Expr2$.

Returns false if $Expr1$ is determined to be equal to $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing \neq

< (less than)



$Expr1 < Expr2 \Rightarrow$ Boolean expression



See “=” (equal) example.

$List1 < List2 \Rightarrow$ Boolean list

$Matrix1 < Matrix2 \Rightarrow$ Boolean matrix

Returns true if $Expr1$ is determined to be less than $Expr2$.

< (less than)

  keys

Returns false if *Expr1* is determined to be greater than or equal to *Expr2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≤ (less or equal)

  keys

$Expr1 \leq Expr2 \Rightarrow$ Boolean expression

See “=” (equal) example.

$List1 \leq List2 \Rightarrow$ Boolean list

$Matrix1 \leq Matrix2 \Rightarrow$ Boolean matrix

Returns true if *Expr1* is determined to be less than or equal to *Expr2*.

Returns false if *Expr1* is determined to be greater than *Expr2*.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing <=

> (greater than)

  keys

$Expr1 > Expr2 \Rightarrow$ Boolean expression

See “=” (equal) example.

$List1 > List2 \Rightarrow$ Boolean list

$Matrix1 > Matrix2 \Rightarrow$ Boolean matrix

Returns true if *Expr1* is determined to be greater than *Expr2*.

Returns false if *Expr1* is determined to be less than or equal to *Expr2*.

Anything else returns a simplified form of the equation.

> (greater than)

ctrl = keys

For lists and matrices, returns comparisons element by element.

≥ (greater or equal)

ctrl = keys

$Expr1 \geq Expr2 \Rightarrow$ Boolean expression

See “=” (equal) example.

$List1 \geq List2 \Rightarrow$ Boolean list

$Matrix1 \geq Matrix2 \Rightarrow$ Boolean matrix

Returns true if $Expr1$ is determined to be greater than or equal to $Expr2$.

Returns false if $Expr1$ is determined to be less than $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing >=

⇒ (logical implication)

ctrl = keys

$BooleanExpr1 \Rightarrow BooleanExpr2$ returns Boolean expression

$5 > 3$ or $3 > 5$	true
--------------------	------

$5 > 3 \Rightarrow 3 > 5$	false
---------------------------	-------

$BooleanList1 \Rightarrow BooleanList2$ returns Boolean list

3 or 4	7
--------	---

$3 \Rightarrow 4$	-4
-------------------	----

$BooleanMatrix1 \Rightarrow BooleanMatrix2$ returns Boolean matrix

$\{1,2,3\}$ or $\{3,2,1\}$	$\{3,2,3\}$
----------------------------	-------------

$\{1,2,3\} \Rightarrow \{3,2,1\}$	$\{-1,-1,-3\}$
-----------------------------------	----------------

$Integer1 \Rightarrow Integer2$ returns Integer

Evaluates the expression **not** <argument1> or <argument2> and returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing =>

⇔ (logical double implication, XNOR)**ctrl** **=** **keys***BooleanExpr1* ⇔ *BooleanExpr2* returns *Boolean expression*

5>3 xor 3>5 true

BooleanList1 ⇔ *BooleanList2* returns *Boolean list*

5>3 ⇔ 3>5 false

BooleanMatrix1 ⇔ *BooleanMatrix2* returns *Boolean matrix*

3 xor 4 7

3 ⇔ 4 -8

 $\{1,2,3\}$ xor $\{3,2,1\}$ $\{2,0,2\}$ $\{1,2,3\}$ ⇔ $\{3,2,1\}$ $\{-3,-1,-3\}$ *Integer1* ⇔ *Integer2* returns *Integer*Returns the negation of an **XOR** Boolean operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing <=>**! (factorial)****?!>** **key***Expr1!* ⇒ *expression*

5! 120

List1! ⇒ *list* $\{\{5,4,3\}\}!$ $\{120,24,6\}$ *Matrix1!* ⇒ *matrix* $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}!$ $\begin{bmatrix} 1 & 2 \\ 6 & 24 \end{bmatrix}$

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

& (append)**ctrl** **⌨** **keys***String1* & *String2* ⇒ *string*

"Hello "&"Nick" "Hello Nick"

Returns a text string that is *String2* appended to *String1*.

$d()$ (derivative)

Catalog >  $d(\text{Expr1}, \text{Var}[, \text{Order}]) \Rightarrow \text{expression}$ $d(\text{List1}, \text{Var}[, \text{Order}]) \Rightarrow \text{list}$ $d(\text{Matrix1}, \text{Var}[, \text{Order}]) \Rightarrow \text{matrix}$

Returns the first derivative of the first argument with respect to variable *Var*.

Order, if included, must be an integer. If the order is less than zero, the result will be an anti-derivative.

Note: You can insert this function from the keyboard by typing **derivative (...)**.

$d()$ does not follow the normal evaluation mechanism of fully simplifying its arguments and then applying the function definition to these fully simplified arguments. Instead, $d()$ performs the following steps:

1. Simplify the second argument only to the extent that it does not lead to a non-variable.
2. Simplify the first argument only to the extent that it does recall any stored value for the variable determined by step 1.
3. Determine the symbolic derivative of the result of step 2 with respect to the variable from step 1.

If the variable from step 1 has a stored value or a value specified by the constraint ("|") operator, substitute that value into the result from step 3.

Note: See also **First derivative**, page 5; **Second derivative**, page 6; or **Nth derivative**, page 6.

$\frac{d}{dx}(f(x) \cdot g(x))$	$\frac{d}{dx}(f(x)) \cdot g(x) + \frac{d}{dx}(g(x)) \cdot f(x)$
$\frac{d}{dy}\left(\frac{d}{dx}(x^2 \cdot y^3)\right)$	$6 \cdot y^2 \cdot x$
$\frac{d}{dx}\left(\left\{x^2, x^3, x^4\right\}\right)$	$\left\{2 \cdot x, 3 \cdot x^2, 4 \cdot x^3\right\}$

$\int()$ (integral)

Catalog >  $\int(\text{Expr1}, \text{Var}[, \text{Lower}, \text{Upper}]) \Rightarrow \text{expression}$ $\int(\text{Expr1}, \text{Var}[, \text{Constant}]) \Rightarrow \text{expression}$

$\int_a^b x^2 dx$	$\frac{b^3}{3} - \frac{a^3}{3}$
-------------------	---------------------------------

Returns the integral of *Expr1* with respect to the variable *Var* from *Lower* to *Upper*.

Note: See also **Definite** or **Indefinite integral template**, page 6.

Note: You can insert this function from the keyboard by typing `integral (...)`.

If *Lower* and *Upper* are omitted, returns an anti-derivative. A symbolic constant of integration is omitted unless you provide the *Constant* argument.

$$\int x^2 dx = \frac{x^3}{3}$$

$$\int (a \cdot x^2, x, c) = \frac{a \cdot x^3}{3} + c$$

Equally valid anti-derivatives might differ by a numeric constant. Such a constant might be disguised—particularly when an anti-derivative contains logarithms or inverse trigonometric functions. Moreover, piecewise constant expressions are sometimes added to make an anti-derivative valid over a larger interval than the usual formula.

$\int()$ returns itself for pieces of *Expr1* that it cannot determine as an explicit finite combination of its built-in functions and operators.

$$\int b \cdot e^{-x^2} + \frac{a}{x^2+a^2} dx \quad b \cdot \int e^{-x^2} dx + \tan^{-1}\left(\frac{x}{a}\right)$$

When you provide *Lower* and *Upper*, an attempt is made to locate any discontinuities or discontinuous derivatives in the interval $Lower < Var < Upper$ and to subdivide the interval at those places.

For the Auto setting of the **Auto or Approximate** mode, numerical integration is used where applicable when an anti-derivative or a limit cannot be determined.

For the Approximate setting, numerical integration is tried first, if applicable. Anti-derivatives are sought only where such numerical integration is inapplicable or fails.

Note: To force an approximate result,

Handheld: Press `ctrl` `enter`.

Windows®: Press `Ctrl+Enter`.

Macintosh®: Press `⌘+Enter`.

iPad®: Hold `enter`, and select

$\int()$ (integral)

Catalog > 


$$\int_{-1}^1 e^{-x^2} dx \quad 1.49365$$

$\int()$ can be nested to do multiple integrals. Integration limits can depend on integration variables outside them.

Note: See also `nint()`, page 122.

$$\int_0^a \int_0^x \ln(x+y) dy dx$$
$$\frac{a^2 \cdot \ln(a)}{2} + \frac{a^2 \cdot (4 \cdot \ln(2) - 3)}{4}$$

$\sqrt{}$ (square root)

ctrl  keys

$\sqrt{Expr1} \Rightarrow expression$

$$\sqrt{4} \quad 2$$
$$\sqrt{\{9,a,4\}} \quad \{3,\sqrt{a},2\}$$

$\sqrt{List1} \Rightarrow list$

Returns the square root of the argument.

For a list, returns the square roots of all the elements in *List1*.

Note: You can insert this function from the keyboard by typing `sqrt (...)`

Note: See also **Square root template**, page 1.

$\prod()$ (prodSeq)

Catalog > 

$\prod(Expr1, Var, Low, High) \Rightarrow expression$

$$\prod_{n=1}^5 \left(\frac{1}{n}\right) \quad \frac{1}{120}$$

Note: You can insert this function from the keyboard by typing `prodSeq (...)`.

Evaluates *Expr1* for each value of *Var* from *Low* to *High*, and returns the product of the results.

$$\prod_{k=1}^n (k^2) \quad (n!)^2$$

Note: See also **Product template** (\prod), page 5.

$$\prod_{n=1}^5 \left(\left\{\frac{1}{n}, n, 2\right\}\right) \quad \left\{\frac{1}{120}, 120, 32\right\}$$

$\prod()$ (prodSeq)

Catalog > 

$$\prod(\text{Expr1}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 1$$

$$\prod(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \mathbf{1} / \prod(\text{Expr1}, \text{Var}, \text{High}+1, \text{Low}-1) \text{ if } \text{High} < \text{Low}-1$$

$$\prod_{k=4}^3 (k) \quad 1$$

The product formulas used are derived from the following reference:

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

$$\prod_{k=4}^1 \left(\frac{1}{k}\right) \quad 6$$

$$\prod_{k=4}^1 \left(\frac{1}{k}\right) \cdot \prod_{k=2}^4 \left(\frac{1}{k}\right) \quad \frac{1}{4}$$

$\Sigma()$ (sumSeq)

Catalog > 

$$\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \text{expression}$$

Note: You can insert this function from the keyboard by typing `sumSeq (...)`.

Evaluates *Expr1* for each value of *Var* from *Low* to *High*, and returns the sum of the results.

Note: See also **Sum template**, page 5.

$$\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 0$$

$$\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \mu$$

$$\Sigma(\text{Expr1}, \text{Var}, \text{High}+1, \text{Low}-1) \text{ if } \text{High} < \text{Low}-1$$

$$\sum_{n=1}^5 \left(\frac{1}{n}\right) \quad \frac{137}{60}$$

$$\sum_{k=1}^n (k^2) \quad \frac{n \cdot (n+1) \cdot (2 \cdot n+1)}{6}$$

$$\sum_{n=1}^{\infty} \left(\frac{1}{n^2}\right) \quad \frac{\pi^2}{6}$$

$$\sum_{k=4}^3 (k) \quad 0$$

$$\sum_{k=4}^1 (k) \quad -5$$

The summation formulas used are derived from the following reference:

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

$$\sum_{k=4}^1 (k) + \sum_{k=2}^4 (k) \quad 4$$

$\Sigma\text{Int}()$

Catalog >

$\Sigma\text{Int}(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue])$
 \Rightarrow value

$\Sigma\text{Int}(1,3,12,4.75,20000,,12,12)$ -213.48

$\Sigma\text{Int}(NPmt1, NPmt2, amortTable) \Rightarrow$ value

Amortization function that calculates the sum of the interest during a specified range of payments.

$NPmt1$ and $NPmt2$ define the start and end boundaries of the payment range.

$N, I, PV, Pmt, FV, PpY, CpY,$ and $PmtAt$ are described in the table of TVM arguments, page 195.

- If you omit Pmt , it defaults to $Pmt=tvmpmt(N, I, PV, FV, PpY, CpY, PmtAt)$.
- If you omit FV , it defaults to $FV=0$.
- The defaults for $PpY, CpY,$ and $PmtAt$ are the same as for the TVM functions.

$roundValue$ specifies the number of decimal places for rounding. Default=2.

$\Sigma\text{Int}(NPmt1, NPmt2, amortTable)$ calculates the sum of the interest based on amortization table $amortTable$. The $amortTable$ argument must be a matrix in the form described under $amortTbl()$, page 8.

Note: See also $\Sigma Prn()$, below, and $Bal()$, page 17.

$tbl:=amortTbl(12,12,4.75,20000,,12,12)$

0	0.	0.	20000.
1	-77.49	-1632.43	18367.6
2	-71.17	-1638.75	16728.8
3	-64.82	-1645.1	15083.7
4	-58.44	-1651.48	13432.2
5	-52.05	-1657.87	11774.4
6	-45.62	-1664.3	10110.1
7	-39.17	-1670.75	8439.32
8	-32.7	-1677.22	6762.1
9	-26.2	-1683.72	5078.38
10	-19.68	-1690.24	3388.14
11	-13.13	-1696.79	1691.35
12	-6.55	-1703.37	-12.02

$\Sigma\text{Int}(1,3,tbl)$ -213.48

 $\Sigma Prn()$

Catalog >

$\Sigma Prn(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) \Rightarrow$ value

$\Sigma Prn(1,3,12,4.75,20000,,12,12)$ -4916.28

$\Sigma Prn(NPmt1, NPmt2, amortTable) \Rightarrow$ value

Amortization function that calculates the sum of the principal during a specified range of payments.

$NPmt1$ and $NPmt2$ define the start and end boundaries of the payment range.

N , I , PV , Pmt , FV , PpY , CpY , and $PmtAt$ are described in the table of TVM arguments, page 195.

- If you omit Pmt , it defaults to $Pmt=\text{tvmPmt}(N,I,PV,FV,PpY,CpY,PmtAt)$.
- If you omit FV , it defaults to $FV=0$.
- The defaults for PpY , CpY , and $PmtAt$ are the same as for the TVM functions.

$roundValue$ specifies the number of decimal places for rounding. Default=2.

$\Sigma\text{Prn}(NPmt1, NPmt2, amortTable)$ calculates the sum of the principal paid based on amortization table $amortTable$. The $amortTable$ argument must be a matrix in the form described under $\text{amortTbl}()$, page 8.

Note: See also $\Sigma\text{Int}()$, above, and $\text{Bal}()$, page 17.

$tbl:=\text{amortTbl}(12,12,4.75,20000.,12,12)$			
0	0.	0.	20000.
1	-77.49	-1632.43	18367.57
2	-71.17	-1638.75	16728.82
3	-64.82	-1645.1	15083.72
4	-58.44	-1651.48	13432.24
5	-52.05	-1657.87	11774.37
6	-45.62	-1664.3	10110.07
7	-39.17	-1670.75	8439.32
8	-32.7	-1677.22	6762.1
9	-26.2	-1683.72	5078.38
10	-19.68	-1690.24	3388.14
11	-13.13	-1696.79	1691.35
12	-6.55	-1703.37	-12.02
$\Sigma\text{Prn}(1,3,tbl)$			-4916.28

(indirection)

  keys

$varNameString$

#{"x"&"y"&"z"} xyz

Refers to the variable whose name is $varNameString$. This lets you use strings to create variable names from within a function.

Creates or refers to the variable xyz .

$10 \rightarrow r$	10
"r" $\rightarrow s1$	"r"
#s1	10

Returns the value of the variable (r) whose name is stored in variable s1.

E (scientific notation)**EE** key*mantissa*E*exponent*

23000. 23000.

Enters a number in scientific notation. The number is interpreted as *mantissa* × 10^{*exponent*}.

23000000000.+4.1E15 4.1E15

3·10⁴ 30000

Hint: If you want to enter a power of 10 without causing a decimal value result, use 10^{integer}.

Note: You can insert this operator from the computer keyboard by typing @E. for example, type 2.3@E4 to enter 2.3E4.

g (gradian)**1** key*Expr1*g ⇒ *expression*

In Degree, Gradian or Radian mode:

*List1*g ⇒ *list*

$$\cos(50^g) \quad \frac{\sqrt{2}}{2}$$
*Matrix1*g ⇒ *matrix*

$$\cos(\{0,100^g,200^g\}) \quad \{1,0,-1\}$$

This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies *Expr1* by $\pi/200$.

In Degree angle mode, multiplies *Expr1* by $g/100$.

In Gradian mode, returns *Expr1* unchanged.

Note: You can insert this symbol from the computer keyboard by typing @g.

r (radian)**1** key*Expr1*r ⇒ *expression*

In Degree, Gradian or Radian angle mode:

*List1*r ⇒ *list*

$$\cos\left(\frac{\pi}{4^r}\right) \quad \frac{\sqrt{2}}{2}$$
*Matrix1*r ⇒ *matrix*

$$\cos\left(\left\{0^r, \frac{\pi}{12}^r, (\pi)^r\right\}\right) \quad \left\{1, \frac{(\sqrt{3}+1)\cdot\sqrt{2}}{4}, -1\right\}$$

r(radian)

1 key

This function gives you a way to specify a radian angle while in Degree or Gradian mode.

In Degree angle mode, multiplies the argument by $180/\pi$.

In Radian angle mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by $200/\pi$.

Hint: Use r if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

Note: You can insert this symbol from the computer keyboard by typing @r.

° (degree)

1 key

$ExprI^\circ \Rightarrow expression$

$ListI^\circ \Rightarrow list$

$MatrixI^\circ \Rightarrow matrix$

This function gives you a way to specify a degree angle while in Gradian or Radian mode.

In Radian angle mode, multiplies the argument by $\pi/180$.

In Degree angle mode, returns the argument unchanged.

In Gradian angle mode, multiplies the argument by $10/9$.

Note: You can insert this symbol from the computer keyboard by typing @d.

In Degree, Gradian or Radian angle mode:

$$\cos(45^\circ) \quad \frac{\sqrt{2}}{2}$$

In Radian angle mode:

Note: To force an approximate result,

Handheld: Press $\boxed{\text{ctrl}} \boxed{\text{enter}}$.

Windows®: Press **Ctrl+Enter**.

Macintosh®: Press $\boxed{\text{⌘}} + \text{Enter}$.

iPad®: Hold **enter**, and select $\boxed{\approx}$.

$$\cos\left\{\left\{0, \frac{\pi}{4}, 90^\circ, 30.12^\circ\right\}\right\} \\ \{1., 0.707107, 0., 0.864976\}$$

° , ' , " (degree/minute/second)

ctrl $\boxed{\text{⌘}}$ keys

$dd^\circ mm' ss.ss'' \Rightarrow expression$

In Degree angle mode:

°, ', " (degree/minute/second)

  **keys**

dd A positive or negative number

mm A non-negative number

ss.ss A non-negative number

25°13'17.5"

25.2215

25°30'

$\frac{51}{2}$

Returns $dd+(mm/60)+(ss.ss/3600)$.

This base-60 entry format lets you:

- Enter an angle in degrees/minutes/seconds without regard to the current angle mode.
- Enter time as hours/minutes/seconds.

Note: Follow *ss.ss* with two apostrophes ('), not a quote symbol (").

∠ (angle)

  **keys**

[*Radius*, ∠ *θ* *Angle*] ⇒ *vector*
(polar input)

In Radian mode and vector format set to:
rectangular

[*Radius*, ∠ *θ* *Angle*, *Z* *Coordinate*] ⇒
vector
(cylindrical input)

$$\left[5 \angle 60^\circ \angle 45^\circ \right] \left[\frac{5 \cdot \sqrt{2}}{4} \quad \frac{5 \cdot \sqrt{6}}{4} \quad \frac{5 \cdot \sqrt{2}}{2} \right]$$

[*Radius*, ∠ *θ* *Angle*, ∠ *θ* *Angle*] ⇒ *vector*
(spherical input)

cylindrical

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

$$\left[5 \angle 60^\circ \angle 45^\circ \right] \left[\frac{5 \cdot \sqrt{2}}{2} \quad \angle \frac{\pi}{3} \quad \frac{5 \cdot \sqrt{2}}{2} \right]$$

Note: You can insert this symbol from the computer keyboard by typing @<.

spherical

(*Magnitude* ∠ *Angle*) ⇒ *complexValue*
(polar input)

In Radian angle mode and Rectangular complex format:

Enters a complex value in (*r* ∠ *θ*) polar form. The *Angle* is interpreted according to the current Angle mode setting.

$$5+3 \cdot i - \left(10 \angle \frac{\pi}{4} \right) \quad 5-5 \cdot \sqrt{2} + (3-5 \cdot \sqrt{2}) \cdot i$$

Note: To force an approximate result,

Handheld: Press  .

Windows®: Press **Ctrl+Enter**.

Macintosh®: Press **⌘+Enter**.

iPad®: Hold **enter**, and select .

∠ (angle)

ctrl  keys

$$5+3 \cdot i - \left(10 \angle \frac{\pi}{4} \right) \quad -2.07107 - 4.07107 \cdot i$$

' (prime)

 key

variable '
variable ''

Enters a prime symbol in a differential equation. A single prime symbol denotes a 1st-order differential equation, two prime symbols denote a 2nd-order, and so on.

$$\text{deSolve} \left(y'' = y^{-2} \text{ and } y(0) = 0 \text{ and } y'(0) = 0, t, y \right)$$

$$\frac{2 \cdot y^4}{3} = t$$

_ (underscore as an empty element)

See “Empty (Void) Elements,”
page 251.

_ (underscore as unit designator)

ctrl  keys

*Expr*_Unit

$$3 \cdot \text{m} \blacktriangleright \text{ft} \quad 9.84252 \cdot \text{ft}$$

Designates the units for an *Expr*. All unit names must begin with an underscore.


You can use pre-defined units or create your own units. For a list of pre-defined units, open the Catalog and display the Unit Conversions tab. You can select unit names from the Catalog or type the unit names directly.

*Variable*_

When *Variable* has no value, it is treated as though it represents a complex number. By default, without the _, the variable is treated as real.

If *Variable* has a value, the _ is ignored and *Variable* retains its original data type.

Note: You can store a complex number to a variable without using _. However, for best results in calculations such as **cSolve()** and **cZeros()**, the _ is recommended.

Note: You can find the conversion symbol, \blacktriangleright , in the Catalog. Click , and then click **Math Operators**.

Assuming z is undefined:

$\text{real}(z)$	z
$\text{real}(z_)$	$\text{real}(z_)$
$\text{imag}(z)$	0
$\text{imag}(z_)$	$\text{imag}(z_)$

► (convert)

ctrl  keys

$Expr_Unit1 \blacktriangleright _Unit2 \Rightarrow Expr_Unit2$

3·_m►_ft



9.84252·_ft

Converts an expression from one unit to another.

The _ underscore character designates the units. The units must be in the same category, such as Length or Area.

For a list of pre-defined units, open the Catalog and display the Unit Conversions tab:

- You can select a unit name from the list.
- You can select the conversion operator, \blacktriangleright , from the top of the list.

You can also type unit names manually. To type “_” when typing unit names on the handheld, press  .

Note: To convert temperature units, use **tmpCnv()** and **ΔtmpCnv()**. The \blacktriangleright conversion operator does not handle temperature units.

10[^]()

Catalog > 

$10^{\wedge}(Expr1) \Rightarrow expression$

$10^{1.5}$

31.6228

$10^{\wedge}(List1) \Rightarrow list$

$10^{\{0, -2.2, a\}}$

$\left\{1, \frac{1}{100}, 100, 10^a\right\}$

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in *List1*.

$10^{\wedge}(squareMatrix1) \Rightarrow squareMatrix$

$10^{\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}}$

Returns 10 raised to the power of *squareMatrix1*. This is not the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to **cos()**.

$\begin{bmatrix} 1.14336\text{E}7 & 8.17155\text{E}6 & 6.67589\text{E}6 \\ 9.95651\text{E}6 & 7.11587\text{E}6 & 5.81342\text{E}6 \\ 7.65298\text{E}6 & 5.46952\text{E}6 & 4.46845\text{E}6 \end{bmatrix}$

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

Expr1 ^-1 ⇒ *expression*

$(3.1)^{-1}$	0.322581
--------------	----------

List1 ^-1 ⇒ *list*

$\{a,4,-0.1,x,-2\}^{-1}$	$\left\{\frac{1}{a},\frac{1}{4},-10,\frac{1}{x},\frac{-1}{2}\right\}$
--------------------------	---

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in *List1*.

squareMatrix1 ^-1 ⇒ *squareMatrix*

Returns the inverse of *squareMatrix1*.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1}$	$\begin{bmatrix} -2 & 1 \\ 3 & -1 \\ 2 & 2 \end{bmatrix}$
---	---

squareMatrix1 must be a non-singular square matrix.

$\begin{bmatrix} 1 & 2 \\ a & 4 \end{bmatrix}^{-1}$	$\begin{bmatrix} \frac{-2}{a-2} & \frac{1}{a-2} \\ \frac{a}{2 \cdot (a-2)} & \frac{-1}{2 \cdot (a-2)} \end{bmatrix}$
---	--

| (constraint operator)

Expr | BooleanExpr1[and BooleanExpr2]...

$x+1 x=3$	4
-----------	---

Expr | BooleanExpr1[or BooleanExpr2]...

$x+y x=\sin(y)$	$\sin(y)+y$
-----------------	-------------

$x+y \sin(y)=x$	$x+y$
-----------------	-------

The constraint (“|”) symbol serves as a binary operator. The operand to the left of | is an expression. The operand to the right of | specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after | must be joined by logical “and” or “or” operators.

The constraint operator provides three basic types of functionality:

- Substitutions
- Interval constraints
- Exclusions

Substitutions are in the form of an equality, such as $x=3$ or $y=\sin(x)$. To be most effective, the left side should be a simple variable. *Expr | Variable = value* will substitute *value* for every occurrence of *Variable* in *Expr*.

$x^3-2 \cdot x+7 \rightarrow f(x)$	Done
------------------------------------	------

$f(x) x=\sqrt{3}$	$\sqrt{3}+7$
-------------------	--------------

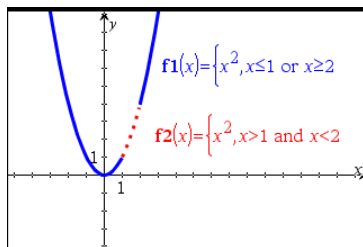
$(\sin(x))^2+2 \cdot \sin(x)-6 \sin(x)=d$	$d^2+2 \cdot d-6$
---	-------------------

| (constraint operator)

ctrl  keys

Interval constraints take the form of one or more inequalities joined by logical “and” or “or” operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

$\text{solve}(x^2-1=0,x) x>0 \text{ and } x<2$	$x=1$
$\sqrt{x} \cdot \sqrt{\frac{1}{x}} x>0$	1
$\sqrt{x} \cdot \sqrt{\frac{1}{x}}$	$\sqrt{\frac{1}{x}} \cdot \sqrt{x}$



Exclusions use the “not equals” (\neq or \neq) relational operator to exclude a specific value from consideration. They are used primarily to exclude an exact solution when using `cSolve()`, `cZeros()`, `fMax()`, `fMin()`, `solve()`, `zeros()`, and so on.

$\text{solve}(x^2-1=0,x) x\neq 1$	$x=-1$
-----------------------------------	--------

→ (store)

ctrl var key

Expr → *Var*

$\frac{\pi}{4} \rightarrow \text{myvar}$	$\frac{\pi}{4}$
--	-----------------

List → *Var*

$2 \cdot \cos(x) \rightarrow y1(x)$	<i>Done</i>
-------------------------------------	-------------

Matrix → *Var*

$\{1,2,3,4\} \rightarrow \text{lst5}$	$\{1,2,3,4\}$
---------------------------------------	---------------

Expr → *Function(Param1,...)*

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow \text{matg}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
--	--

List → *Function(Param1,...)*

$\text{"Hello"} \rightarrow \text{str1}$	"Hello"
--	------------------

Matrix → *Function(Param1,...)*

If the variable *Var* does not exist, creates it and initializes it to *Expr*, *List*, or *Matrix*.

If the variable *Var* already exists and is not locked or protected, replaces its contents with *Expr*, *List*, or *Matrix*.

→ (store)

ctrl var **key**

Hint: If you plan to do symbolic computations using undefined variables, avoid storing anything into commonly used, one-letter variables such as *a*, *b*, *c*, *x*, *y*, *z*, and so on.

Note: You can insert this operator from the keyboard by typing `=:` as a shortcut. For example, type `pi/4 =: myvar`.

:= (assign)

ctrl ⌈ **keys**

Var := *Expr*

Var := *List*

Var := *Matrix*

Function(*Param1*,...) := *Expr*

Function(*Param1*,...) := *List*

Function(*Param1*,...) := *Matrix*

If variable *Var* does not exist, creates *Var* and initializes it to *Expr*, *List*, or *Matrix*.

If *Var* already exists and is not locked or protected, replaces its contents with *Expr*, *List*, or *Matrix*.

Hint: If you plan to do symbolic computations using undefined variables, avoid storing anything into commonly used, one-letter variables such as *a*, *b*, *c*, *x*, *y*, *z*, and so on.

<i>myvar</i> := $\frac{\pi}{4}$	$\frac{\pi}{4}$
<i>y1</i> (<i>x</i>) := $2 \cdot \cos(x)$	<i>Done</i>
<i>lst5</i> := { 1,2,3,4 }	{ 1,2,3,4 }
<i>matg</i> := $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
<i>str1</i> := "Hello"	"Hello"

© (comment)

ctrl  keys

© [*text*]

© processes *text* as a comment line, allowing you to annotate functions and programs that you create.

© can be at the beginning or anywhere in the line. Everything to the right of ©, to the end of the line, is the comment.

Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

Define $g(n)=\text{Func}$

© *Declare variables*

Local *i,result*

result:=0

For *i,1,n,1* © *Loop n times*

result:=result+i²

EndFor

Return *result*

EndFunc

Done

$g(3)$

14

0b, 0h

  keys,   keys

0b *binaryNumber*

0h *hexadecimalNumber*

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

In Dec base mode:

0b10+0hF+10

27

In Bin base mode:

0b10+0hF+10

0b11011

In Hex base mode:

0b10+0hF+10

0h1B

TI-Nspire™ CX II - Draw Commands

This is a supplemental document for the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide. All TI-Nspire™ CX II commands will be incorporated and published in version 5.1 of the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide.

Graphics Programming

New commands have been added on TI-Nspire™ CX II Handhelds and TI-Nspire™ desktop applications for graphics programming.

The TI-Nspire™ CX II Handhelds will switch into this graphics mode while executing graphics commands and switch back to the context in which the program was executed after completion of the program.

The screen will display “Running...” in the top bar while the program is being executed. It will show “Finished” when the program completes. Any key-press will transition the system out of the graphics mode.

- The transition to graphics mode is triggered automatically when one of the Draw (graphics) commands is encountered during execution of the TI-Basic program.
- This transition will only happen when executing a program from calculator; in a document or calculator in scratchpad.
- The transition out of graphics mode happens upon termination of the program.
- The graphics mode is only available on the TI-Nspire™ CX II Handhelds and the desktop TI-Nspire™ CX II Handhelds view. This means it is not available in the computer document view or PublishView (.tnsp) on the desktop nor on iOS.
 - If a graphics command is encountered while executing a TI-Basic program from the incorrect context, an error message is displayed and the TI-Basic program is terminated.

Graphics Screen

The graphics screen will contain a header at the top of the screen that cannot be written to by graphics commands.

The graphics screen drawing area will be cleared (color = 255,255,255) when the graphics screen is initialized.

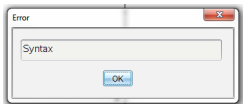



Graphics Screen	Default
Height	212
Width	318
Color	white: 255,255,255

Default View and Settings

- The status icons in the top bar (battery status, press-to-test status, network indicator etc.) will not be visible while a graphics program is running.
- Default drawing color: Black (0,0,0)
- Default pen style - normal, smooth
 - Thickness: 1 (thin), 2 (normal), 3 (thickest)
 - Style: 1 (smooth), 2 (dotted), 3 (dashed)
- All drawing commands will use the current color and pen settings; either default values or those which were set via TI-Basic commands.
- Text font is fixed and cannot be changed.
- Any output to the graphics screen will be drawn within a clipping window which is the size of the graphics screen drawing area. Any drawn output that extends outside of this clipped graphics screen drawing area will not be drawn. No error message will be displayed.
- All x,y coordinates specified for drawing commands are defined such that 0,0 is at the top left corner of the graphics screen drawing area.
 - **Exceptions:**
 - **DrawText** uses the coordinates as the bottom left corner of the bounding box for the text.
 - **SetWindow** uses the bottom left corner of the screen
- All parameters for the commands can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.

Graphics Screen Errors Messages

If the validation fails, an error message will display.

Error Message	Description	View
Error Syntax	If the syntax checker finds any syntax errors, it displays an error message and tries to position the cursor near the first error so you can correct it.	
Error Too few arguments	The function or command is missing one or more arguments	Error Too few arguments The function or command is missing one or more arguments. 
Error Too many arguments	The function or command contains an excessive number of arguments and cannot be evaluated.	Error Too many arguments The function or command contains an excessive number of arguments and cannot be evaluated. 
Error Invalid data type	An argument is of the wrong data type.	Error Invalid data type An argument is of the wrong data type. 

Invalid Commands While in Graphics Mode

Some commands are not allowed once the program switches to graphics mode. If these commands are encountered while in graphics mode and error will be displayed and the program will be terminated.

Disallowed Command	Error Message
Request	Request cannot be executed in graphics mode
RequestStr	RequestStr cannot be executed in graphics mode
Text	Text cannot be executed in graphics mode

The commands that print text to the calculator - **disp** and **dispAt** - will be supported commands in the graphics context. The text from these commands will be sent to the Calculator screen (not on Graphics) and will be visible after the program exits and the system switches back to the Calculator app

C

Clear

Catalog > 
CXII

Clear *x, y, width, height*

Clear

Clears entire screen if no parameters are specified.

Clears entire screen

If *x, y, width* and *height* are specified, the rectangle defined by the parameters will be cleared.

Clear 10,10,100,50

Clears a rectangle area with top left corner on (10, 10) and with width 100, height 50

DrawArcCatalog > 
CXII**DrawArc** *x, y, width, height, startAngle, arcAngle*

Draw an arc within the defined bounding rectangle with the provided start and arc angles.

x, y: upper left coordinate of bounding rectangle

width, height: dimensions of bounding rectangle

The "arc angle" defines the sweep of the arc.

These parameters can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.

DrawArc 20,20,100,100,0,90



DrawArc 50,50,100,100,0,180



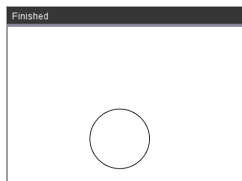
See Also: [FillArc](#)

DrawCircleCatalog > 
CXII**DrawCircle** *x, y, radius*

x, y: coordinate of center

radius: radius of the circle

DrawCircle 150,150,40



See Also: [FillCircle](#)

DrawLine

Catalog > 
CXII

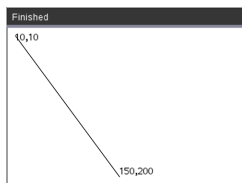
DrawLine $x1, y1, x2, y2$

Draw a line from $x1, y1, x2, y2$.

Expressions that evaluate to a number which is then rounded to the nearest integer.

Screen bounds: If the specified coordinates causes any part of the line to be drawn outside of the graphics screen, that part of the line will be clipped and no error message will be displayed.

DrawLine 10,10,150,200



DrawPoly

Catalog > 
CXII

The commands have two variants:

DrawPoly $xlist, ylist$

or

DrawPoly $x1, y1, x2, y2, x3, y3...xn, yn$

Note: DrawPoly $xlist, ylist$

Shape will connect $x1, y1$ to $x2, y2, x2, y2$ to $x3, y3$ and so on.

Note: DrawPoly $x1, y1, x2, y2, x3, y3...xn, yn$

xn, yn will **NOT** be automatically connected to $x1, y1$.

Expressions that evaluate to a list of real floats

$xlist, ylist$

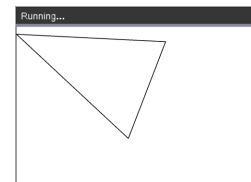
Expressions that evaluate to a single real float

$x1, y1...xn, yn$ = coordinates for vertices of polygon

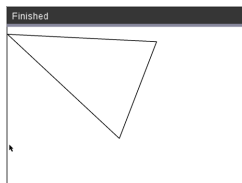
$xlist:=\{0,200,150,0\}$

$ylist:=\{10,20,150,10\}$

DrawPoly $xlist,ylist$



DrawPoly 0,10,200,20,150,150,0,10



Note: DrawPoly: Input size dimensions (width/height) relative to drawn lines. The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn polygon will be larger than the width and height.

See Also: [FillPoly](#)

DrawRect

DrawRect *x, y, width, height*

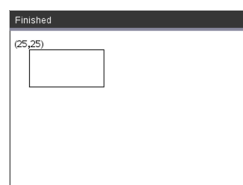
x, y: upper left coordinate of rectangle

width, height: width and height of rectangle (rectangle drawn down and right from starting coordinate).

Note: The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn rectangle will be larger than the width and height indicate.

See Also: [FillRect](#)

DrawRect 25,25,100,50



DrawText

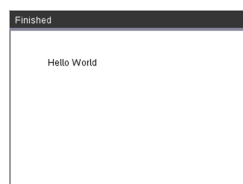
DrawText *x, y, exprOrString1*
[,exprOrString2]...

x, y: coordinate of text output

Draws the text in *exprOrString* at the specified *x, y* coordinate location.

The rules for *exprOrString* are the same as for **Disp** – **DrawText** can take multiple arguments.

DrawText 50,50,"Hello World"



FillArc

Catalog > 
CXII

FillArc *x, y, width, height startAngle, arcAngle*

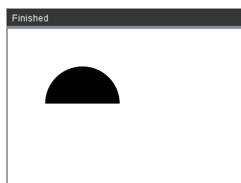
x, y: upper left coordinate of bounding rectangle

Draw and fill an arc within the defined bounding rectangle with the provided start and arc angles.

Default fill color is black. The fill color can be set by the [SetColor](#) command

The "arc angle" defines the sweep of the arc

```
FillArc 50,50,100,100,0,180
```



FillCircle

Catalog > 
CXII

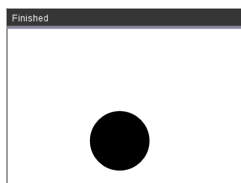
FillCircle *x, y, radius*

x, y: coordinate of center

Draw and fill a circle at the specified center with the specified radius.

Default fill color is black. The fill color can be set by the [SetColor](#) command.

```
FillCircle 150,150,40
```



Here!

FillPoly

Catalog > 
CXII

FillPoly *xlist, ylist*

or

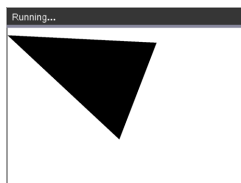
FillPoly *x1, y1, x2, y2, x3, y3...xn, yn*

Note: The line and color are specified by [SetColor](#) and [SetPen](#)

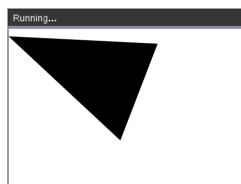
```
xlist:={0,200,150,0}
```

```
ylist:={10,20,150,10}
```

```
FillPoly xlist,ylist
```



```
FillPoly 0,10,200,20,150,150,0,10
```



FillRect

```
FillRect x, y, width, height
```

```
FillRect 25,25,100,50
```

x, y : upper left coordinate of rectangle

$width, height$: width and height of rectangle

Draw and fill a rectangle with the top left corner at the coordinate specified by (x,y)

Default fill color is black. The fill color can be set by the [SetColor](#) command

Note: The line and color are specified by [SetColor](#) and [SetPen](#)



G

getPlatform()

Catalog > 
CXII

getPlatform()

getPlatform()

"dt"

Returns:

"dt" on desktop software applications

"hh" on TI-Nspire™ CX handhelds

"ios" on TI-Nspire™ CX iPad® app

PaintBuffer

Paint graphics buffer to screen

This command is used in conjunction with UseBuffer to increase the speed of display on the screen when the program generates multiple graphical objects.

UseBuffer

For n,1,10

x:=randInt(0,300)

y:=randInt(0,200)

radius:=randInt(10,50)

Wait 0.5

DrawCircle x,y,radius

EndFor

PaintBuffer

This program will display all the 10 circles at once.

If the "UseBuffer" command is removed, each circle will be displayed as it is drawn.

See Also: [UseBuffer](#)

PlotXY $x, y, shape$ x, y : coordinate to plot shape $shape$: a number between 1 and 13 specifying the shape

- 1 - Filled circle
- 2 - Empty circle
- 3 - Filled square
- 4 - Empty square
- 5 - Cross
- 6 - Plus
- 7 - Thin
- 8 - medium point, solid
- 9 - medium point, empty
- 10 - larger point, solid
- 11 - larger point, empty
- 12 - largest point, solid
- 13 - largest point, empty

PlotXY 100,100,1

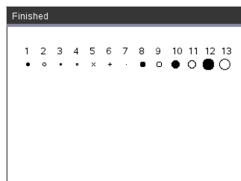


For n,1,13

DrawText 1+22*n,40,n

PlotXY 5+22*n,50,n

EndFor



SetColorCatalog > 
CXII**SetColor**

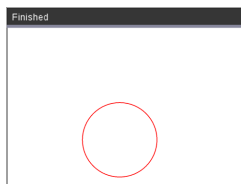
Red-value, Green-value, Blue-value

Valid values for red, green and blue are between 0 and 255

Sets the color for subsequent Draw commands

SetColor 255,0,0

DrawCircle 150,150,100

**SetPen**Catalog > 
CXII**SetPen**

thickness, style

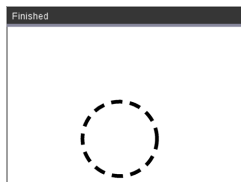
thickness: 1 <= thickness <= 3 | 1 is thinnest, 3 is thickest

style: 1 = Smooth, 2 = Dotted, 3 = Dashed

Sets the pen style for subsequent Draw commands

SetPen 3,3

DrawCircle 150,150,50

**SetWindow**Catalog > 
CXII**SetWindow**

xMin, xMax, yMin, yMax

Establishes a logical window that maps to the graphics drawing area. All parameters are required.

If the part of drawn object is outside the window, the output will be clipped (not shown) and no error message is displayed.

SetWindow 0,160,0,120

will set the output window to have 0,0 in the bottom left corner with a width of 160 and a height of 120

DrawLine 0,0,100,100

SetWindow 0,160,0,120

SetPen 3,3

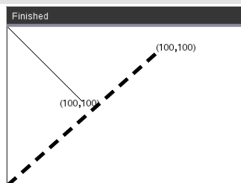
DrawLine 0,0,100,100

If x_{\min} is greater than or equal to x_{\max} or y_{\min} is greater than or equal to y_{\max} , an error message is shown.

Any objects drawn before a SetWindow command will not be re-drawn in the new configuration.

To reset the window parameters to the default, use:

SetWindow 0,0,0,0



UseBuffer

Draw to an off screen graphics buffer instead of screen (to increase performance)

This command is used in conjunction with PaintBuffer to increase the speed of display on the screen when the program generates multiple graphical objects.

With UseBuffer, all the graphics are displayed only after the next PaintBuffer command is executed.

UseBuffer only needs to be called once in the program i.e. every use of PaintBuffer does not need a corresponding UseBuffer

```
UseBuffer
```

```
For n,1,10
```

```
x:=randInt(0,300)
```

```
y:=randInt(0,200)
```

```
radius:=randInt(10,50)
```

```
Wait 0.5
```

```
DrawCircle x,y,radius
```

```
EndFor
```

```
PaintBuffer
```

This program will display all the 10 circles at once.

If the "UseBuffer" command is removed, each circle will be displayed as it is drawn.

See Also: [PaintBuffer](#)

Empty (Void) Elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ CAS Software allows empty, or void, data elements so you can proceed with the nearly complete data rather than having to start over or discard the incomplete cases.

You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “*Graphing spreadsheet data.*”

The **delVoid()** function lets you remove empty elements from a list. The **isVoid()** function lets you test for an empty element. For details, see **delVoid()**, page 49, and **isVoid()**, page 94.

Note: To enter an empty element manually in a math expression, type “_” or the keyword **void**. The keyword **void** is automatically converted to a “_” symbol when the expression is evaluated. To type “_” on the handheld, press **ctrl** **[]**.

Calculations involving void elements

The majority of calculations involving a void input will produce a void result. See special cases below.

$\lfloor _ \rfloor$	–
$\gcd(100, _)$	–
$3 + _$	–
$\{5, _, 10\} - \{3, 6, 9\}$	$\{2, _, 1\}$

List arguments containing void elements

The following functions and commands ignore (skip) void elements found in list arguments.

count, **countIf**, **cumulativeSum**, **freqTable** ► **list**, **frequency**, **max**, **mean**, **median**, **product**, **stDevPop**, **stDevSamp**, **sum**, **sumIf**, **varPop**, and **varSamp**, as well as regression calculations, **OneVar**, **TwoVar**, and **FiveNumSummary** statistics, confidence intervals, and stat tests

$\text{sum}(\{2, _, 3, 5, 6, 6\})$	16.6
$\text{median}(\{1, 2, _, _, 3\})$	2
$\text{cumulativeSum}(\{1, 2, _, 4, 5\})$	$\{1, 3, _, 7, 12\}$
$\text{cumulativeSum} \left(\begin{pmatrix} 1 & 2 \\ 3 & _ \\ 5 & 6 \end{pmatrix} \right)$	$\begin{pmatrix} 1 & 2 \\ 4 & _ \\ 9 & 8 \end{pmatrix}$

SortA and **SortD** move all void elements within the first argument to the bottom.

$\{5, 4, 3, _, 1\} \rightarrow \text{list1}$	$\{5, 4, 3, _, 1\}$
$\{5, 4, 3, 2, 1\} \rightarrow \text{list2}$	$\{5, 4, 3, 2, 1\}$
$\text{SortA list1, list2}$	Done
list1	$\{1, 3, 4, 5, _ \}$
list2	$\{1, 3, 4, 5, 2\}$

List arguments containing void elements

$\{1,2,3,_,5\} \rightarrow list1$	$\{1,2,3,_,5\}$
$\{1,2,3,4,5\} \rightarrow list2$	$\{1,2,3,4,5\}$
SortD list1,list2	Done
list1	$\{5,3,2,1,_\}$
list2	$\{5,3,2,1,4\}$

In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

$l1:=\{1,2,3,4,5\}; l2:=\{2,_,3,5,6,6\}$	$\{2,_,3,5,6,6\}$
LinRegMx l1,l2	Done
stat.Resid	$\{0.434286,_,-0.862857,-0.011429,0.44\}$
stat.XReg	$\{1,_,3,4,5\}$
stat.YReg	$\{2,_,3,5,6,6\}$
stat.FreqReg	$\{1,_,1,1,1,1\}$

An omitted category in regressions introduces a void for the corresponding element of the residual.

$l1:=\{1,3,4,5\}; l2:=\{2,3,5,6,6\}$	$\{2,3,5,6,6\}$
cat:={"M","M","F","F"}; incl:={"F"}	$\{"F"\}$
LinRegMx l1,l2,1,cat,incl	Done
stat.Resid	$\{_,_,0,0,0\}$
stat.XReg	$\{_,_,4,5\}$
stat.YReg	$\{_,_,5,6,6\}$
stat.FreqReg	$\{_,_,1,1,1\}$

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

$l1:=\{1,3,4,5\}; l2:=\{2,3,5,6,6\}$	$\{2,3,5,6,6\}$
LinRegMx l1,l2,{1,0,1,1}	Done
stat.Resid	$\{0.069231,_,-0.276923,0.207692\}$
stat.XReg	$\{1,_,4,5\}$
stat.YReg	$\{2,_,5,6,6\}$
stat.FreqReg	$\{1,_,1,1,1\}$

Shortcuts for Entering Math Expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type `sqrt` (6) on the entry line. When you press `[enter]`, the expression `sqrt(6)` is changed to $\sqrt{6}$. Some shortcuts are useful from both the handheld and the computer keyboard. Others are useful primarily from the computer keyboard.

From the Handheld or Computer Keyboard

To enter this:	Type this shortcut:
π	<code>pi</code>
θ	<code>theta</code>
∞	<code>infinity</code>
\leq	<code><=</code>
\geq	<code>>=</code>
\neq	<code>/=</code>
\Rightarrow (logical implication)	<code>=></code>
\Leftrightarrow (logical double implication, XNOR)	<code><=></code>
\rightarrow (store operator)	<code>=:</code>
$ $ (absolute value)	<code>abs (...)</code>
$\sqrt{()}$	<code>sqrt (...)</code>
$d()$	<code>derivative (...)</code>
$\int()$	<code>integral (...)</code>
$\Sigma()$ (Sum template)	<code>sumSeq (...)</code>
$\Pi()$ (Product template)	<code>prodSeq (...)</code>
<code>sin-1()</code> , <code>cos-1()</code> , ...	<code>arcsin (...)</code> , <code>arccos (...)</code> , ...
<code>ΔList()</code>	<code>deltaList (...)</code>
<code>ΔtmpCnv()</code>	<code>deltaTmpCnv (...)</code>

From the Computer Keyboard

To enter this:	Type this shortcut:
<code>c1</code> , <code>c2</code> , ... (constants)	<code>@c1</code> , <code>@c2</code> , ...

To enter this:	Type this shortcut:
$n1, n2, \dots$ (integer constants)	@n1, @n2, ...
i (imaginary constant)	@i
e (natural log base e)	@e
E (scientific notation)	@E
T (transpose)	@t
r (radians)	@r
$^\circ$ (degrees)	@d
g (gradians)	@g
\sphericalangle (angle)	@<
► (conversion)	@>
►Decimal, ►approxFraction(), and so on.	@>Decimal, @>approxFraction(), and so on.

EOS™ (Equation Operating System) Hierarchy

This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire™ CAS math and science learning technology. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS™ software evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

Order of Evaluation

Level	Operator
1	Parentheses (), brackets [], braces { }
2	Indirection (#)
3	Function calls
4	Post operators: degrees-minutes-seconds ([°] ,',"), factorial (!), percentage (%), radian (r), subscript ([]), transpose (T)
5	Exponentiation, power operator (^)
6	Negation (-)
7	String concatenation (&)
8	Multiplication (*), division (/)
9	Addition (+), subtraction (-)
10	Equality relations: equal (=), not equal (\neq or \neq), less than (<), less than or equal (\leq or \leq), greater than (>), greater than or equal (\geq or \geq)
11	Logical not
12	Logical and
13	Logical or
14	xor, nor, nand
15	Logical implication (\Rightarrow)
16	Logical double implication, XNOR (\Leftrightarrow)
17	Constraint operator (" ")
18	Store (\rightarrow)

Parentheses, Brackets, and Braces

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression $4(1+2)$, EOS™ software first evaluates the portion of the expression inside the parentheses, $1+2$, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, $(1+2)/(3+4$ will display the error message "Missing)."

Note: Because the TI-Nspire™ CAS software allows you to define your own functions, a variable name followed by an expression in parentheses is considered a "function call" instead of implied multiplication. For example $a(b+c)$ is the function a evaluated by $b+c$. To multiply the expression $b+c$ by the variable a , use explicit multiplication: $a \cdot (b+c)$.

Indirection

The indirection operator (#) converts a string to a variable or function name. For example, $\#("x"&"y"&"z")$ creates the variable name xyz . Indirection also allows the creation and modification of variables from inside a program. For example, if $10 \rightarrow r$ and $"r" \rightarrow s1$, then $\#s1=10$.

Post Operators

Post operators are operators that come directly after an argument, such as $5!$, 25% , or $60^\circ 15' 45''$. Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression $4^3!$, $3!$ is evaluated first. The result, 6, then becomes the exponent of 4 to yield 4096.

Exponentiation

Exponentiation (^) and element-by-element exponentiation (.^) are evaluated from right to left. For example, the expression 2^3^2 is evaluated the same as $2^{(3^2)}$ to produce 512. This is different from $(2^3)^2$, which is 64.

Negation

To enter a negative number, press $\boxed{-}$ followed by the number. Post operations and exponentiation are performed before negation. For example, the result of $-x^2$ is a negative number, and $-9^2 = -81$. Use parentheses to square a negative number such as $(-9)^2$ to produce 81.

Constraint ("|")

The argument following the constraint ("|") operator provides a set of constraints that affect the evaluation of the argument preceding the operator.

TI-Nspire CX II - TI-Basic Programming Features

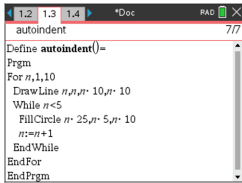
Auto-indentation in Programming Editor

The TI-Nspire™ program editor now auto-indent statements inside a block command.

Block commands are If/EndIf, For/EndFor, While/EndWhile, Loop/EndLoop, Try/EndTry

The editor will automatically prepend spaces to program commands inside a block command. The closing command of the block will be aligned with the opening command.

The example below shows auto-indentation in nested block commands.



```
autoident
777
Define autoident()=
Prgm
For n,1,10
DrawLine n,n,n-10,n-10
While n<5
FillCircle n-25,n-5,n-10
n:=n+1
EndWhile
EndFor
EndPrgm
```

Code fragments that are copied and pasted will retain the original indentation.

Opening a program created in an earlier version of the software will retain the original indentation.

Improved Error Messages for TI-Basic

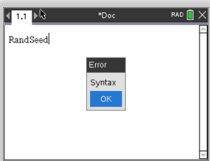
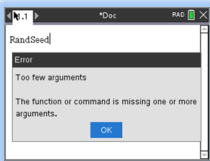
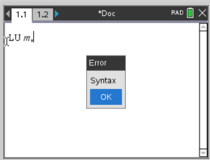
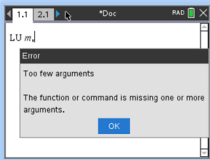
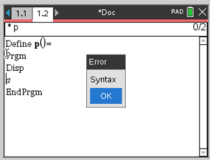
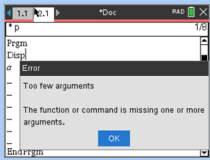
Errors

Error Condition	New message
Error in condition statement (If/While)	A conditional statement did not resolve to TRUE or FALSE NOTE: With the change to place the cursor on the line with the error, we no longer need to specify if the error is in an "If" statement or a "While" statement.
Missing EndIf	Expected EndIf but found a different end statement
Missing EndFor	Expected EndFor but found a different end statement
Missing EndWhile	Expected EndWhile but found a different end statement
Missing EndLoop	Expected EndLoop but found a different end statement

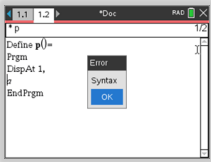
Error Condition	New message
Missing EndTry	Expected EndTry but found a different end statement
"Then" omitted after If <condition>	Missing If..Then
"Then" omitted after Elseif <condition>	Then missing in block: Elseif .
When "Then" , "Else" and "Elseif" were encountered outside of control blocks	Else invalid outside of blocks: If..Then..Endif or Try..EndTry
"Elseif" appears outside of "If..Then..Endif" block	Elseif invalid outside of block: If..Then..Endif
"Then" appears outside of "If...Endif" block	Then invalid outside of block: If..Endif

Syntax Errors

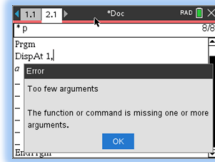
In case commands that expect one or more arguments are called with an incomplete list of arguments, a **"Too few argument error"** will be issued instead of **"syntax"** error

Current behavior	New CX II behavior
	
	
	

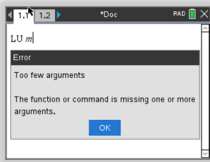
Current behavior



New CX II behavior




Note: When an incomplete list of arguments is not followed by a comma, the error message is: “too few arguments”. This is the same as previous releases.



Constants and Values

The following table lists the constants and their values that are available when performing unit conversions. They can be typed in manually or selected from the

Constants list in **Utilities > Unit Conversions** (Handheld: Press  **3**).



Constant	Name	Value
_c	Speed of light	299792458 _m/_s
_Cc	Coulomb constant	8987551787.3682 _m/_F
_Fc	Faraday constant	96485.33289 _coul/_mol
_g	Acceleration of gravity	9.80665 _m/_s ²
_Gc	Gravitational constant	6.67408E-11 _m ³ /_kg/_s ²
_h	Planck's constant	6.626070040E-34 _J _s
_k	Boltzmann's constant	1.38064852E-23 _J/_°K
_μ0	Permeability of a vacuum	1.2566370614359E-6 _N/_A ²
_μb	Bohr magneton	9.274009994E-24 _J _m ² /_Wb
_Me	Electron rest mass	9.10938356E-31 _kg
_Mμ	Muon mass	1.883531594E-28 _kg
_Mn	Neutron rest mass	1.674927471E-27 _kg
_Mp	Proton rest mass	1.672621898E-27 _kg
_Na	Avogadro's number	6.022140857E23 /_mol
_q	Electron charge	1.6021766208E-19 _coul
_Rb	Bohr radius	5.2917721067E-11 _m
_Rc	Molar gas constant	8.3144598 _J/_mol/_°K
_Rdb	Rydberg constant	10973731.568508/_m
_Re	Electron radius	2.8179403227E-15 _m
_u	Atomic mass	1.660539040E-27 _kg
_Vm	Molar volume	2.2413962E-2 _m ³ /_mol
_ε0	Permittivity of a vacuum	8.8541878176204E-12 _F/_m
_σ	Stefan-Boltzmann constant	5.670367E-8 _W/_m ² /_°K ⁴
_φ0	Magnetic flux quantum	2.067833831E-15 _Wb

Error Codes and Messages

When an error occurs, its code is assigned to variable *errCode*. User-defined programs and functions can examine *errCode* to determine the cause of an error. For an example of using *errCode*, See Example 2 under the **Try** command, page 191.

Note: Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.

Error code	Description
10	A function did not return a value
20	A test did not resolve to TRUE or FALSE. Generally, undefined variables cannot be compared. For example, the test <code>If a<b</code> will cause this error if either <code>a</code> or <code>b</code> is undefined when the <code>If</code> statement is executed.
30	Argument cannot be a folder name.
40	Argument error
50	Argument mismatch Two or more arguments must be of the same type.
60	Argument must be a Boolean expression or integer
70	Argument must be a decimal number
90	Argument must be a list
100	Argument must be a matrix
130	Argument must be a string
140	Argument must be a variable name. Make sure that the name: <ul style="list-style-type: none">• does not begin with a digit• does not contain spaces or special characters• does not use underscore or period in invalid manner• does not exceed the length limitations See the Calculator section in the documentation for more details.
160	Argument must be an expression
165	Batteries too low for sending or receiving Install new batteries before sending or receiving.
170	Bound The lower bound must be less than the upper bound to define the search interval.

Error code	Description
180	Break The  or  key was pressed during a long calculation or during program execution.
190	Circular definition This message is displayed to avoid running out of memory during infinite replacement of variable values during simplification. For example, $a+1 \rightarrow a$, where a is an undefined variable, will cause this error.
200	Constraint expression invalid For example, $\text{solve}(3x^2-4=0,x) \mid x < 0 \text{ or } x > 5$ would produce this error message because the constraint is separated by “or” instead of “and.”
210	Invalid Data type An argument is of the wrong data type.
220	Dependent limit
230	Dimension A list or matrix index is not valid. For example, if the list $\{1,2,3,4\}$ is stored in $L1$, then $L1[5]$ is a dimension error because $L1$ only contains four elements.
235	Dimension Error. Not enough elements in the lists.
240	Dimension mismatch Two or more arguments must be of the same dimension. For example, $[1,2]+[1,2,3]$ is a dimension mismatch because the matrices contain a different number of elements.
250	Divide by zero
260	Domain error An argument must be in a specified domain. For example, $\text{rand}(0)$ is not valid.
270	Duplicate variable name
280	Else and Elseif invalid outside of If...Endif block
290	EndTry is missing the matching Else statement
295	Excessive iteration
300	Expected 2 or 3-element list or matrix
310	The first argument of nSolve must be an equation in a single variable. It cannot contain a non-valued variable other than the variable of interest.
320	First argument of solve or cSolve must be an equation or inequality For example, $\text{solve}(3x^2-4,x)$ is invalid because the first argument is not an equation.

Error code	Description
345	Inconsistent units
350	Index out of range
360	Indirection string is not a valid variable name
380	Undefined Ans Either the previous calculation did not create Ans, or no previous calculation was entered.
390	Invalid assignment
400	Invalid assignment value
410	Invalid command
430	Invalid for the current mode settings
435	Invalid guess
440	Invalid implied multiply For example, $x(x+1)$ is invalid; whereas, $x*(x+1)$ is the correct syntax. This is to avoid confusion between implied multiplication and function calls.
450	Invalid in a function or current expression Only certain commands are valid in a user-defined function.
490	Invalid in Try..EndTry block
510	Invalid list or matrix
550	Invalid outside function or program A number of commands are not valid outside a function or program. For example, Local cannot be used unless it is in a function or program.
560	Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks For example, the Exit command is valid only inside these loop blocks.
565	Invalid outside program
570	Invalid pathname For example, \backslash var is invalid.
575	Invalid polar complex
580	Invalid program reference Programs cannot be referenced within functions or expressions such as $1+p(x)$ where p is a program.

Error code	Description
600	Invalid table
605	Invalid use of units
610	Invalid variable name in a Local statement
620	Invalid variable or function name
630	Invalid variable reference
640	Invalid vector syntax
650	Link transmission A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends.
665	Matrix not diagonalizable
670	Low Memory 1. Delete some data in this document 2. Save and close this document If 1 and 2 fail, pull out and re-insert batteries
672	Resource exhaustion
673	Resource exhaustion
680	Missing (
690	Missing)
700	Missing “
710	Missing]
720	Missing }
730	Missing start or end of block syntax
740	Missing Then in the If..EndIf block
750	Name is not a function or program
765	No functions selected
780	No solution found
800	Non-real result For example, if the software is in the Real setting, $\sqrt{-1}$ is invalid.

Error code	Description
	To allow complex results, change the "Real or Complex" Mode Setting to RECTANGULAR or POLAR.
830	Overflow
850	Program not found A program reference inside another program could not be found in the provided path during execution.
855	Rand type functions not allowed in graphing
860	Recursion too deep
870	Reserved name or system variable
900	Argument error Median-median model could not be applied to data set.
910	Syntax error
920	Text not found
930	Too few arguments The function or command is missing one or more arguments.
940	Too many arguments The expression or equation contains an excessive number of arguments and cannot be evaluated.
950	Too many subscripts
955	Too many undefined variables
960	Variable is not defined No value is assigned to variable. Use one of the following commands: <ul style="list-style-type: none"> • sto → • := • Define to assign values to variables.
965	Unlicensed OS
970	Variable in use so references or changes are not allowed
980	Variable is protected
990	Invalid variable name Make sure that the name does not exceed the length limitations

Error code	Description
1000	Window variables domain
1010	Zoom
1020	Internal error
1030	Protected memory violation
1040	Unsupported function. This function requires Computer Algebra System. Try TI-Nspire™ CAS.
1045	Unsupported operator. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.
1050	Unsupported feature. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.
1060	Input argument must be numeric. Only inputs containing numeric values are allowed.
1070	Trig function argument too big for accurate reduction
1080	Unsupported use of Ans. This application does not support Ans.
1090	<p>Function is not defined. Use one of the following commands:</p> <ul style="list-style-type: none"> • Define • := • sto → <p>to define a function.</p>
1100	<p>Non-real calculation</p> <p>For example, if the software is in the Real setting, $\sqrt{-1}$ is invalid.</p> <p>To allow complex results, change the “Real or Complex” Mode Setting to RECTANGULAR or POLAR.</p>
1110	Invalid bounds
1120	No sign change
1130	Argument cannot be a list or matrix
1140	<p>Argument error</p> <p>The first argument must be a polynomial expression in the second argument. If the second argument is omitted, the software attempts to select a default.</p>
1150	<p>Argument error</p> <p>The first two arguments must be polynomial expressions in the third argument. If the third argument is omitted, the software attempts to select a default.</p>
1160	Invalid library pathname

Error code	Description
	<p>A pathname must be in the form <code>xxx\yyy</code>, where:</p> <ul style="list-style-type: none"> • The <code>xxx</code> part can have 1 to 16 characters. • The <code>yyy</code> part can have 1 to 15 characters. <p>See the Library section in the documentation for more details.</p>
1170	<p>Invalid use of library pathname</p> <ul style="list-style-type: none"> • A value cannot be assigned to a pathname using Define, <code>:=</code>, or <code>sto →</code>. • A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition.
1180	<p>Invalid library variable name.</p> <p>Make sure that the name:</p> <ul style="list-style-type: none"> • Does not contain a period • Does not begin with an underscore • Does not exceed 15 characters <p>See the Library section in the documentation for more details.</p>
1190	<p>Library document not found:</p> <ul style="list-style-type: none"> • Verify library is in the MyLib folder. • Refresh Libraries. <p>See the Library section in the documentation for more details.</p>
1200	<p>Library variable not found:</p> <ul style="list-style-type: none"> • Verify library variable exists in the first problem in the library. • Make sure library variable has been defined as LibPub or LibPriv. • Refresh Libraries. <p>See the Library section in the documentation for more details.</p>
1210	<p>Invalid library shortcut name.</p> <p>Make sure that the name:</p> <ul style="list-style-type: none"> • Does not contain a period • Does not begin with an underscore • Does not exceed 16 characters • Is not a reserved name <p>See the Library section in the documentation for more details.</p>
1220	<p>Domain error:</p> <p>The <code>tangentLine</code> and <code>normalLine</code> functions support real-valued functions only.</p>
1230	<p>Domain error.</p>

Error code	Description
	Trigonometric conversion operators are not supported in Degree or Gradian angle modes.
1250	<p>Argument Error</p> <p>Use a system of linear equations.</p> <p>Example of a system of two linear equations with variables x and y:</p> $3x+7y=5$ $2y-5x=-1$
1260	<p>Argument Error:</p> <p>The first argument of nfMin or nfMax must be an expression in a single variable. It cannot contain a non-valued variable other than the variable of interest.</p>
1270	<p>Argument Error</p> <p>Order of the derivative must be equal to 1 or 2.</p>
1280	<p>Argument Error</p> <p>Use a polynomial in expanded form in one variable.</p>
1290	<p>Argument Error</p> <p>Use a polynomial in one variable.</p>
1300	<p>Argument Error</p> <p>The coefficients of the polynomial must evaluate to numeric values.</p>
1310	<p>Argument error:</p> <p>A function could not be evaluated for one or more of its arguments.</p>
1380	<p>Argument error:</p> <p>Nested calls to <code>domain()</code> function are not allowed.</p>

Warning Codes and Messages

You can use the `warnCodes()` function to store the codes of warnings generated by evaluating an expression. This table lists each numeric warning code and its associated message. For an example of storing warning codes, see `warnCodes()`, page 200.

Warning code	Message
10000	Operation might introduce false solutions. When applicable, try using graphical methods to verify the results.
10001	Differentiating an equation may produce a false equation.
10002	Questionable solution When applicable, try using graphical methods to verify the results.
10003	Questionable accuracy When applicable, try using graphical methods to verify the results.
10004	Operation might lose solutions. When applicable, try using graphical methods to verify the results.
10005	<code>cSolve</code> might specify more zeros.
10006	<code>Solve</code> may specify more zeros. When applicable, try using graphical methods to verify the results.
10007	More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess. Examples using <code>solve()</code> : <ul style="list-style-type: none"><code>solve(Equation, Var=Guess) lowBound<Var<upBound</code><code>solve(Equation, Var) lowBound<Var<upBound</code><code>solve(Equation, Var=Guess)</code> When applicable, try using graphical methods to verify the results.
10008	Domain of the result might be smaller than the domain of the input.
10009	Domain of the result might be larger than the domain of the input.
10012	Non-real calculation
10013	∞^0 or <code>undef^0</code> replaced by 1
10014	<code>undef^0</code> replaced by 1
10015	1^∞ or 1^{undef} replaced by 1
10016	1^{undef} replaced by 1

Warning code	Message
10017	Overflow replaced by ∞ or $-\infty$
10018	Operation requires and returns 64 bit value.
10019	Resource exhaustion, simplification might be incomplete.
10020	Trig function argument too big for accurate reduction.
10021	Input contains an undefined parameter. Result might not be valid for all possible parameter values.
10022	Specifying appropriate lower and upper bounds might produce a solution.
10023	Scalar has been multiplied by the identity matrix.
10024	Result obtained using approximate arithmetic.
10025	Equivalence cannot be verified in EXACT mode.
10026	Constraint might be ignored. Specify constraint in the form " <code>\</code> " 'Variable MathTestSymbol Constant' or a conjunct of these forms, for example ' <code>x<3 and x>-12</code> '

General Information

Online Help

education.ti.com/eguide

Select your country for more product information.

Contact TI Support

education.ti.com/ti-cares

Select your country for technical and other support resources.

Service and Warranty Information

education.ti.com/warranty

Select your country for information about the length and terms of the warranty or about product service.

Limited Warranty. This warranty does not affect your statutory rights.

Texas Instruments Incorporated

12500 TI Blvd.

Dallas, TX 75243

Index

-		
-, subtract	210	
!		
!, factorial	220	
"		
", second notation	228	
#		
#, indirection	226	
#, indirection operator	256	
%		
%, percent	216	
&		
&, append	220	
*		
*, multiply	211	
.		
., dot subtraction	214	
*, dot multiplication	215	
./, dot division	215	
.^, dot power	215	
., dot addition	214	
/		
/, divide	212	
:		
:=, assign	234	
^		
\wedge^{-1} , reciprocal	232	
\wedge , power	213	
—		
_, unit designation	230	
, constraint operator	232	
,		
' minute notation	228	
', prime	230	
+		
+, add	210	
=		
\neq , not equal	217	
\leq , less than or equal	218	
\geq , greater than or equal	219	
$>$, greater than	218	
$=$, equal	216	
Π		
Π , product	223	
Σ		
$\Sigma()$, sum	224	
$\Sigma\text{Int}()$	225	
$\Sigma\text{Prn}()$	225	
$\sqrt{\quad}$		
$\sqrt{\quad}$, square root	223	
\angle		
\angle (angle)	229	
\int		
\int , integral	221	

	▶			1	
▶, convert units		231	$10^{()}$, power of ten		231
▶approxFraction()		14		2	
▶Base10, display as decimal integer		18	2-sample F Test		75
▶Base16, display as hexadecimal		19		A	
▶Base2, display as binary		17	abs(), absolute value		8
▶cos, display in terms of cosine		29	absolute value template for		3-4
▶Cylind, display as cylindrical vector		42	add, +		210
▶DD, display as decimal angle		45	amortization table, amortTbl()		8, 17
▶Decimal, display result as decimal		45	amortTbl(), amortization table		8, 17
▶DMS, display as degree/minute/second		54	and, Boolean operator		9
▶exp, display in terms of e		63	angle(), angle		10
▶Grad, convert to gradian angle		86	angle, angle()		10
▶Polar, display as polar vector		133	ANOVA, one-way variance analysis		10
▶Rad, convert to radian angle		143	ANOVA2way, two-way variance analysis		11
▶Rect, display as rectangular vector		146	Ans, last answer		13
▶sin, display in terms of sine		166	answer (last), Ans		13
▶Sphere, display as spherical vector		175	append, &		220
	⇒		approx(), approximate		13, 15
⇒, logical implication		219, 253	approximate, approx()		13, 15
	→		approxRational()		14
→, store variable		233	arc length, arcLen()		15
	↔		arccos(), $\cos^{-1}()$		14
↔, logical double implication		220, 253	arccosh(), $\cosh^{-1}()$		14
	©		arccot(), $\cot^{-1}()$		14
©, comment		235	arcoth(), $\coth^{-1}()$		14
	°		arccsc(), $\csc^{-1}()$		15
°, degree notation		228	arccsch(), $\operatorname{csch}^{-1}()$		15
°, degrees/minutes/seconds		228	arcLen(), arc length		15
	0		arcsec(), $\sec^{-1}()$		15
Ob, binary indicator		235	arcsech(), $\operatorname{csech}^{-1}()$		15
Oh, hexadecimal indicator		235	arcsin(), $\sin^{-1}()$		15
			arcsinh(), $\sinh^{-1}()$		15
			arctan(), $\tan^{-1}()$		15
			arctanh(), $\tanh^{-1}()$		16
			arguments in TVM functions		195
			augment(), augment/concatenate		16
			augment/concatenate, augment()		16

average rate of change, avgRC()	16	comment, ©	235
avgRC(), average rate of change	16	common denominator, comDenom ()	26
B			
binary		completeSquare(), complete square	27
display, ▶Base2	17	complex	
indicator, Ob	235	conjugate, conj()	28
binomCdf()	20, 92	factor, cFactor()	21
binomPdf()	20	solve, cSolve()	38
Boolean operators		zeros, cZeros()	43
⇒	219, 253	conj(), complex conjugate	28
⇔	220	constant	
and	9	in solve()	171
nand	119	constants	
nor	123	in cSolve()	39
not	125	in cZeros()	44
or	129	in deSolve()	49
xor	201	in solve()	173
C			
Cdf()	69	in zeros()	203
ceiling(), ceiling	20	shortcuts for	253
ceiling, ceiling()	20-21, 36	constraint operator " "	232
centralDiff()	21	constraint operator, order of evaluation	255
cFactor(), complex factor	21	construct matrix, constructMat() ..	28
char(), character string	22	constructMat(), construct matrix ..	28
character string, char()	22	convert	
characters		▶Grad	86
numeric code, ord()	130	▶Rad	143
string, char()	22	units	231
charPoly()	23	copy variable or function, CopyVar ..	29
χ^2 way	23	correlation matrix, corrMat()	29
clear		corrMat(), correlation matrix	29
error, ClrErr	25	\cos^{-1} , arccosine	31
Clear	239	cos(), cosine	30
ClearAZ	25	cosh ⁻¹ (), hyperbolic arccosine	32
ClrErr, clear error	25	cosh(), hyperbolic cosine	32
colAugment	26	cosine	
colDim(), matrix column dimension ..	26	display expression in terms of ..	29
colNorm(), matrix column norm	26	cosine, cos()	30
combinations, nCr()	120	cot ⁻¹ (), arccotangent	33
comDenom(), common		cot(), cotangent	33
denominator	26	cotangent, cot()	33
		coth ⁻¹ (), hyperbolic arccotangent ..	34
		coth(), hyperbolic cotangent	34
		count days between dates, dbd() ..	44

count items in a list conditionally ,			
countif()	35		
count items in a list, count()	34		
count(), count items in a list	34		
countif(), conditionally count items			
in a list	35		
cPolyRoots()	36		
cross product, crossP()	36		
crossP(), cross product	36		
csc ⁻¹ (), inverse cosecant	37		
csc(), cosecant	36		
csch ⁻¹ (), inverse hyperbolic cosecant			
csch(), hyperbolic cosecant	37		
cSolve(), complex solve	38		
cubic regression, CubicReg	40		
CubicReg, cubic regression	40		
cumulative sum, cumulativeSum() .			
cumulativeSum(), cumulative sum .	41		
cycle, Cycle	42		
Cycle, cycle	42		
cylindrical vector display, ►Cylind ...	42		
cZeros(), complex zeros	43		
D			
d(), first derivative	221		
days between dates, dbd()	44		
dbd(), days between dates	44		
decimal			
angle display, ►DD	45		
integer display, ►Base10	18		
Define	46		
Define LibPriv	47		
Define LibPub	47		
define, Define	46		
Define, define	46		
defining			
private function or program ...	47		
public function or program	47		
definite integral			
template for	6		
degree notation, °	228		
degree/minute/second display,			
►DMS	54		
degree/minute/second notation ...	228		
delete			
void elements from list	49		
deleting			
variable, DelVar	48		
deltaList()	48		
deltaTmpConv()	48		
DelVar, delete variable	48		
delVoid(), remove void elements ...	49		
denominator	26		
derivative or nth derivative			
template for	6		
derivative()	49		
derivatives			
first derivative, d()	221		
numeric derivative, nDeriv() ...	121-122		
numeric derivative, nDerivative(
)	121		
deSolve(), solution	49		
det(), matrix determinant	51		
diag(), matrix diagonal	51		
dim(), dimension	52		
dimension, dim()	52		
Disp, display data	52, 158		
DispAt	52		
display as			
binary, ►Base2	17		
cylindrical vector, ►Cylind ...	42		
decimal angle, ►DD	45		
decimal integer, ►Base10	18		
degree/minute/second, ►DMS .	54		
hexadecimal, ►Base16	19		
polar vector, ►Polar	133		
rectangular vector, ►Rect	146		
spherical vector, ►Sphere	175		
display data, Disp	52, 158		
distribution functions			
binomCdf()	20, 92		
binomPdf()	20		
invNorm()	92		
invt()	92		
Invχ ² ()	91		
normCdf()	125		
normPdf()	125		

poissCdf()	132	function, EndFunc	76
poissPdf()	132	if, EndIf	86
tCdf()	185	loop, EndLoop	110
tPdf()	190	program, EndPrgm	137
χ^2 2way()	23	try, EndTry	191
χ^2 Cdf()	24	while, EndWhile	201
χ^2 GOF()	24	end function, EndFunc	76
χ^2 Pdf()	24	end if, EndIf	86
divide, /	212	end loop, EndLoop	110
domain function, domain()	55	end while, EndWhile	201
domain(), domain function	55	EndTry, end try	191
dominant term, dominantTerm()	56	EndWhile, end while	201
dominantTerm(), dominant term	56	EOS (Equation Operating System)	255
dot		equal, =	216
addition, .+	214	Equation Operating System (EOS)	255
division, ./	215	error codes and messages	261, 269
multiplication, .*	215	errors and troubleshooting	
power, .^	215	clear error, ClrErr	25
product, dotP()	57	pass error, PassErr	131
subtraction, .-	214	euler(), Euler function	60
dotP(), dot product	57	evaluate polynomial, polyEval()	135
draw	240-242	evaluation, order of	255
E			
e exponent		exact(), exact	63
template for	2	exact, exact()	63
e to a power, e^()	57, 64	exclusion with " " operator	232
e, display expression in terms of	63	exit, Exit	63
E, exponent	227	Exit, exit	63
e^(), e to a power	57	exp(), e to a power	64
eff(), convert nominal to effective		exp*list(), expression to list	64
rate	58	expand(), expand	65
effective rate, eff()	58	expand, expand()	65
eigenvalue, eigVl()	59	exponent, E	227
eigenvector, eigVc()	58	exponential regression, ExpReg	66
eigVc(), eigenvector	58	exponents	
eigVl(), eigenvalue	59	template for	1
else if, Elseif	59	expr(), string to expression	66, 107
else, Else	86	ExpReg, exponential regression	66
Elseif, else if	59	expressions	
empty (void) elements	251	expression to list, exp*list()	64
end		string to expression, expr()	66, 107
for, EndFor	72	F	
		factor(), factor	67

factor, factor()	67	geomPdf()	77
factorial, !	220	Get	77, 245
fill	243-244	get/return	
Fill, matrix fill	70	denominator, getDenom()	79
financial functions, tvmFV()	193	number, getNum()	84
financial functions, tvml()	193	variables in information,	
financial functions, tvmN()	194	getVarInfo()	82, 85
financial functions, tvmPmt()	194	getDenom(), get/return	
financial functions, tvmPV()	194	denominator	79
first derivative		getKey()	79
template for	5	getLangInfo(), get/return language	
FiveNumSummary	70	information	82
floor(), floor	71	getLockInfo(), tests lock status of	
floor, floor()	71	variable or variable group	83
fMax(), function maximum	71	getMode(), get mode settings	83
fMin(), function minimum	72	getNum(), get/return number	84
For	72	GetStr	84
for, For	72	getType(), get type of variable	85
For, for	72	getVarInfo(), get/return variables	
format string, format()	73	information	85
format(), format string	73	go to, Goto	86
fpart(), function part	73	Goto, go to	86
fractions		gradian notation, g	227
propFrac	139	greater than or equal, \geq	219
template for	1	greater than, $>$	218
freqTable()	74	greatest common divisor, gcd()	76
frequency()	74	groups, locking and unlocking	106, 197
Frobenius norm, norm()	124	groups, testing lock status	83
Func, function	76		
Func, program function	76	H	
functions		hexadecimal	
maximum, fMax()	71	display, ►Base16	19
minimum, fMin()	72	indicator, 0h	235
part, fpart()	73	hyperbolic	
program function, Func	76	arccosine, $\cosh^{-1}()$	32
user-defined	46	arcsine, $\sinh^{-1}()$	168
functions and variables		arctangent, $\tanh^{-1}()$	184
copying	29	cosine, $\cosh()$	32
		sine, $\sinh()$	168
G		tangent, $\tanh()$	184
g, gradians	227		
gcd(), greatest common divisor	76	I	
geomCdf()	77	identity matrix, identity()	86

identity(), identity matrix	86	left(), left	95
if, if	86	left, left()	95
If, if	86	length of string	52
ifFn()	88	less than or equal, \leq	218
imag(), imaginary part	88	LibPriv	47
imaginary part, imag()	88	LibPub	47
ImpDif(), implicit derivative	89	library	
implicit derivative, Impdif()	89	create shortcuts to objects	96
indefinite integral		libShortcut(), create shortcuts to	
template for	6	library objects	96
indirection operator (#)	256	limit	
indirection, #	226	lim()	96
input, Input	89	limit()	96
Input, input	89	template for	6
inString(), within string	89	limit() or lim(), limit	96
int(), integer	90	linear regression, LinRegAx	98
intDiv(), integer divide	90	linear regression, LinRegBx	97, 99
integer divide, intDiv()	90	LinRegBx, linear regression	97
integer part, iPart()	93	LinRegMx, linear regression	98
integer, int()	90	LinRegtIntervals, linear regression	99
integral, \int	221	LinRegtTest	101
interpolate(), interpolate	90	linSolve()	102
inverse cumulative normal		Δ list(), list difference	103
distribution (invNorm())	92	list to matrix, list►mat()	103
inverse, \wedge^{-1}	232	list, conditionally count items in	35
invF()	91	list, count items in	34
invNorm(), inverse cumulative		list►mat(), list to matrix	103
normal distribution)	92	lists	
invt()	92	augment/concatenate,	
Invx ² ()	91	augment()	16
iPart(), integer part	93	cross product, crossP()	36
irr(), internal rate of return		cumulative sum,	
internal rate of return, irr()	93	cumulativeSum()	41
isPrime(), prime test	93	differences in a list, Δ list()	103
isVoid(), test for void	94	dot product, dotP()	57
		empty elements in	251
		expression to list, exp►list()	64
		list to matrix, list►mat()	103
		matrix to list, mat►list()	111
		maximum, max()	111
		mid-string, mid()	114
		minimum, min()	115
		new, newList()	121
		product, product()	138
L			
label, Lbl	95		
language			
get language information	82		
Lbl, label	95		
lcm, least common multiple	95		
least common multiple, lcm	95		

sort ascending, SortA	174	eigenvector, eigVc()	58
sort descending, SortD	175	filling, Fill	70
summation, sum()	180	identity, identity()	86
ln(), natural logarithm	104	list to matrix, list►mat()	103
LnReg, logarithmic regression	104	lower-upper decomposition, LU	110
local variable, Local	106	matrix to list, mat►list()	111
local, Local	106	maximum, max()	111
Local, local variable	106	minimum, min()	115
Lock, lock variable or variable group	106	new, newMat()	121
locking variables and variable groups	106	product, product()	138
Log		QR factorization, QR	139
template for	2	random, randMat()	145
logarithmic regression, LnReg	104	reduced row echelon form, rref(
logarithms	104)	156
logical double implication, ⇔	220	row addition, rowAdd()	155
logical implication, ⇒	219, 253	row dimension, rowDim()	156
logistic regression, Logistic	108	row echelon form, rref()	147
logistic regression, LogisticD	109	row multiplication and addition,	
Logistic, logistic regression	108	mRowAdd()	116
LogisticD, logistic regression	109	row norm, rowNorm()	156
loop, Loop	110	row operation, mRow()	116
Loop, loop	110	row swap, rowSwap()	156
LU, matrix lower-upper		submatrix, subMat()	180-181
decomposition	110	summation, sum()	180
		transpose, T	182
		matrix (1 × 2)	
		template for	4
		matrix (2 × 1)	
		template for	4
		matrix (2 × 2)	
		template for	4
		matrix (m × n)	
		template for	4
		matrix to list, mat►list()	111
		max(), maximum	111
		maximum, max()	111
		mean(), mean	112
		mean, mean()	112
		median(), median	112
		median, median()	112
		medium-medium line regression,	
		MedMed	113
		MedMed, medium-medium line	
		regression	113
M			
mat►list(), matrix to list	111		
matrices			
augment/concatenate,			
augment()	16		
column dimension, colDim() ..	26		
column norm, colNorm()	26		
cumulative sum,			
cumulativeSum()	41		
determinant, det()	51		
diagonal, diag()	51		
dimension, dim()	52		
dot addition, .+	214		
dot division, ./	215		
dot multiplication, .*	215		
dot power, .^	215		
dot subtraction, .-	214		
eigenvalue, eigVl()	59		

mid-string, mid()	114	nor, Boolean operator	123
mid(), mid-string	114	norm(), Frobenius norm	124
min(), minimum	115	normal distribution probability, normCdf()	125
minimum, min()	115	normal line, normalLine()	124
minute notation, ' .	228	normalLine()	124
mirr(), modified internal rate of return	115	normCdf()	125
mixed fractions, using propFrac() with	139	normPdf()	125
mod(), modulo	116	not equal, ≠	217
mode settings, getMode()	83	not, Boolean operator	125
modes setting, setMode()	162	nPr(), permutations	126
modified internal rate of return, mirr ()	115	npv(), net present value	126
modulo, mod()	116	nSolve(), numeric solution	127
mRow(), matrix row operation	116	nth root template for	1
mRowAdd(), matrix row multiplication and addition	116	numeric derivative, nDeriv()	121-122
Multiple linear regression t test	118	derivative, nDerivative()	121
multiply, *	211	integral, nInt()	122
MultReg	117	solution, nSolve()	127
MultRegIntervals()	117		
MultRegTests()	118		
N			
nand, Boolean operator	119	O	
natural logarithm, ln()	104	objects create shortcuts to library	96
nCr(), combinations	120	one-variable statistics, OneVar	128
nDerivative(), numeric derivative	121	OneVar, one-variable statistics	128
negation, entering negative numbers	256	operators order of evaluation	255
net present value, npv()	126	or (Boolean), or	129
new list, newList()	121	or, Boolean operator	129
matrix, newMat()	121	ord(), numeric character code	130
newList(), new list	121		
newMat(), new matrix	121		
nfMax(), numeric function maximum	121	P	
nfMin(), numeric function minimum	122	P►Rx(), rectangular x coordinate	130
nInt(), numeric integral	122	P►Ry(), rectangular y coordinate	131
nom), convert effective to nominal rate	123	pass error, PassErr	131
nominal rate, nom()	123	PassErr, pass error	131
		Pdf()	74
		percent, %	216
		permutations, nPr()	126
		piecewise function (2-piece) template for	2

piecewise function (N-piece)			
template for	3		
piecewise()	132		
poissCdf()	132		
poissPdf()	132		
polar			
coordinate, R►Pr()	143		
coordinate, R►Pθ()	142		
vector display, ►Polar	133		
polyCoeff()	133		
polyDegree()	134		
polyEval(), evaluate polynomial	135		
polyGcd()	135-136		
polynomials			
evaluate, polyEval()	135		
random, randPoly()	145		
PolyRoots()	136		
power of ten, 10 [^] ()	231		
power regression,			
PowerReg	136, 149, 151, 187		
power, [^]	213		
PowerReg, power regression	136		
Prgm, define program	137		
prime number test, isPrime()	93		
prime, '	230		
probability densiy, normPdf()	125		
prodSeq()	138		
product(), product	138		
product, Π()	223		
template for	5		
product, product()	138		
programming			
define program, Prgm	137		
display data, Disp	52, 158		
pass error, PassErr	131		
programs			
defining private library	47		
defining public library	47		
programs and programming			
clear error, ClrErr	25		
display I/O screen, Disp	52, 158		
end program, EndPrgm	137		
end try, EndTry	191		
try, Try	191		
proper fraction, propFrac	139		
propFrac, proper fraction	139		
Q			
QR factorization, QR	139		
QR, QR factorization	139		
quadratic regression, QuadReg	140		
QuadReg, quadratic regression	140		
quartic regression, QuartReg	141		
QuartReg, quartic regression	141		
R			
R, radian	227		
R►Pr(), polar coordinate	143		
R►Pθ(), polar coordinate	142		
radian, R	227		
rand(), random number	143		
randBin, random number	144		
randInt(), random integer	144		
randMat(), random matrix	145		
randNorm(), random norm	145		
random			
matrix, randMat()	145		
norm, randNorm()	145		
number seed, RandSeed	146		
polynomial, randPoly()	145		
random sample	145		
randPoly(), random polynomial	145		
randSamp()	145		
RandSeed, random number seed	146		
real(), real	146		
real, real()	146		
reciprocal, ^{^-1}	232		
rectangular-vector display, ►Rect	146		
rectangular x coordinate, P►Rx()	130		
rectangular y coordinate, P►Ry()	131		
reduced row echelon form, rref()	156		
ref(), row echelon form	147		
RefreshProbeVars	148		
regressions			
cubic, CubicReg	40		
exponential, ExpReg	66		

linear regression, LinRegAx	98	sech ⁻¹ (), inverse hyperbolic secant	158
linear regression, LinRegBx	97, 99	sech(), hyperbolic secant	158
logarithmic, LnReg	104	second derivative	
Logistic	108	template for	6
logistic, Logistic	109	second notation, "	228
medium-medium line, MedMed	113	seq(), sequence	159
MultReg	117	seqGen()	159
power regression,		seqn()	160
PowerReg	136, 149, 151, 187	sequence, seq()	159-160
quadratic, QuadReg	140	series(), series	161
quartic, QuartReg	141	series, series()	161
sinusoidal, SinReg	169	set	
remain(), remainder	149	mode, setMode()	162
remainder, remain()	149	setMode(), set mode	162
remove		settings, get current	83
void elements from list	49	shift(), shift	163
Request	149	shift, shift()	163
RequestStr	151	sign(), sign	165
result		sign, sign()	165
display in terms of cosine	29	simult(), simultaneous equations	165
display in terms of e	63	simultaneous equations, simult()	165
display in terms of sine	166	sin ⁻¹ (), arcsine	167
result values, statistics	177	sin(), sine	166
results, statistics	176	sine	
return, Return	152	display expression in terms of	166
Return, return	152	sine, sin()	166
right(), right	152	sinh ⁻¹ (), hyperbolic arcsine	168
right, right()	27, 60, 90, 152	sinh(), hyperbolic sine	168
rk23(), Runge Kutta function	152	SinReg, sinusoidal regression	169
rotate(), rotate	154	sinusoidal regression, SinReg	169
rotate, rotate()	154	solution, deSolve()	49
round(), round	155	solve(), solve	170
round, round()	155	solve, solve()	170
row echelon form, ref()	147	SortA, sort ascending	174
rowAdd(), matrix row addition	155	SortD, sort descending	175
rowDim(), matrix row dimension	156	sorting	
rowNorm(), matrix row norm	156	ascending, SortA	174
rowSwap(), matrix row swap	156	descending, SortD	175
rref(), reduced row echelon form	156	spherical vector display, ►Sphere	175
		sqrt(), square root	176
		square root	
		template for	1
		square root, √()	176, 223
S			
sec ⁻¹ (), inverse secant	157		
sec(), secant	157		

standard deviation, stdDev()	178, 198	student-t distribution probability,	
stat.results	176	tCdf()	185
stat.values	177	student-t probability density, tPdf()	190
statistics		subMat(), submatrix	180-181
combinations, nCr()	120	submatrix, subMat()	180-181
factorial, !	220	substitution with " " operator	232
mean, mean()	112	subtract, -	210
median, median()	112	sum of interest payments	225
one-variable statistics, OneVar	128	sum of principal payments	225
permutations, nPr()	126	sum(), summation	180
random norm, randNorm()	145	sum, Σ ()	224
random number seed,		template for	5
RandSeed	146	sumIf()	180
standard deviation, stdDev()	178, 198	summation, sum()	180
two-variable results, TwoVar	195	sumSeq()	181
variance, variance()	198	system of equations (2-equation)	
stdDevPop(), population standard		template for	3
deviation	178	system of equations (N-equation)	
stdDevSamp(), sample standard		template for	3
deviation	178		
Stop command	179	T	
store variable (→)	233	t test, tTest	192
storing		T, transpose	182
symbol, &	234	tan ⁻¹ (), arctangent	183
string		tan(), tangent	182
dimension, dim()	52	tangent line, tangentLine()	183
length	52	tangent, tan()	182
string(), expression to string	179	tangentLine()	183
strings		tanh ⁻¹ (), hyperbolic arctangent	184
append, &	220	tanh(), hyperbolic tangent	184
character code, ord()	130	Taylor polynomial, taylor()	185
character string, char()	22	taylor(), Taylor polynomial	185
expression to string, string()	179	tCdf(), studentt distribution	
format, format()	73	probability	185
formatting	73	tCollect(), trigonometric collection	186
indirection, #	226	templates	
left, left()	95	absolute value	3-4
mid-string, mid()	114	definite integral	6
right, right()	27, 60, 90, 152	derivative or nth derivative	6
rotate, rotate()	154	e exponent	2
shift, shift()	163	exponent	1
string to expression, expr()	66, 107	first derivative	5
using to create variable names	256	fraction	1
within, InString	89		

indefinite integral	6	tvmI()	193
limit	6	tvmN()	194
Log	2	tvmPmt()	194
matrix (1 × 2)	4	tvmPV()	194
matrix (2 × 1)	4	two-variable results, TwoVar	195
matrix (2 × 2)	4	TwoVar, two-variable results	195
matrix (m × n)	4		
nth root	1	U	
piecewise function (2-piece) ...	2	underscore, _	230
piecewise function (N-piece) ...	3	unit vector, unitV()	197
product, $\prod()$	5	units	
second derivative	6	convert	231
square root	1	unitV(), unit vector	197
sum, $\Sigma()$	5	unLock, unlock variable or variable	
system of equations (2-		group	197
equation)	3	unlocking variables and variable	
system of equations (N-		groups	197
equation)	3	user-defined functions	46
test for void, isVoid()	94	user-defined functions and	
Test_2S, 2-sample F test	75	programs	47
tExpand(), trigonometric expansion	186		
Text command	187	V	
time value of money, Future Value .	193	variable	
time value of money, Interest	193	creating name from a character	
time value of money, number of		string	256
payments	194	variable and functions	
time value of money, payment		copying	29
amount	194	variables	
time value of money, present value .	194	clear all single-letter	25
tInterval, t confidence interval	187	delete, DelVar	48
tInterval_2Samp, twosample t		local, Local	106
confidence interval	188	variables, locking and unlocking 83, 106, 197	
ΔtmpCnv()	189	variance, variance()	198
tmpCnv()	189	varPop()	198
tPdf(), student probability density .	190	varSamp(), sample variance	198
trace()	190	vectors	
transpose, T	182	cross product, crossP()	36
trigonometric collection, tCollect() .	186	cylindrical vector display,	
trigonometric expansion, tExpand()	186	►Cylind	42
Try, error handling command	191	dot product, dotP()	57
tTest, t test	192	unit, unitV()	197
tTest_2Samp, two-sample t test	192	void elements	251
TVM arguments	195	void elements, remove	49
tvmFV()	193	void, test for	94

W

Wait command	199
warnCodes(), Warning codes	200
warning codes and messages	269
when(), when	200
when, when()	200
while, While	201
While, while	201
with, 	232
within string, inString()	89

X

x^2 , square	214
XNOR	220
xor, Boolean exclusive or	201

Z

zeroes(), zeroes	202
zeroes, zeroes()	202
zInterval, z confidence interval	204
zInterval_1Prop, one-proportion z confidence interval	205
zInterval_2Prop, two-proportion z confidence interval	205
zInterval_2Samp, two-sample z confidence interval	206
zTest	206
zTest_1Prop, one-proportion z test ..	207
zTest_2Prop, two-proportion z test	207
zTest_2Samp, two-sample z test	208

X

χ^2 Cdf()	24
χ^2 GOF	24
χ^2 Pdf()	24