TI-Nspire™ CAS Reference Guide

This guidebook applies to TI-Nspire™ software version 4.4. To obtain the latest version of the documentation, go to *education.ti.com/guides*.

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.

License

Please see the complete license installed in **C:\Program Files\TI Education\<TI-Nspire™ Product Name>\license**.

Contents

Expression Templates

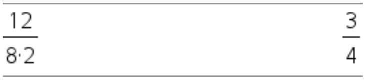
Expression templates give you an easy way to enter math expressions in standard mathematical notation. When you insert a template, it appears on the entry line with small blocks at positions where you can enter elements. A cursor shows which element you can enter.

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

Fraction template — ctrl ÷ keys

$\frac{□}{□}$ **Note:** See also **/ (divide)**, page 210.

Example:



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

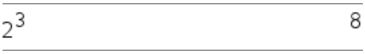
Exponent template — ^ key

$□^{□}$

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 211.

Example:

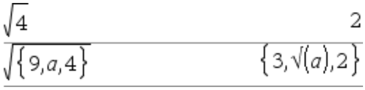


2^3 8

Square root template — ctrl x² keys

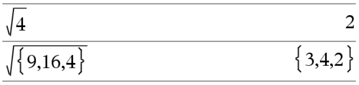
$\sqrt{□}$ **Note:** See also **√() (square root)**, page 221.

Example:



$\sqrt{4}$ 2

$\sqrt{\{9,a,4\}}$ $\{3,\sqrt{(a)},2\}$



$\sqrt{4}$ 2

$\sqrt{\{9,16,4\}}$ $\{3,4,2\}$

Nth root template — ctrl ^ keys

$\sqrt[□]{□}$ **Note:** See also **root()**, page 152.

Example:

Nth root template

ctrl ^ keys

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

e exponent template

ex keys

$e^{\square}$

Natural exponential e raised to a power

Note: See also **e^()**, page 60.

Example:

e^{1}	e
$e^{1.}$	2.71828182846

Log template

ctrl 10x key

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

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

Note: See also **log()**, page 106.

Example:

$\log_{4}(2.)$	0.5

Piecewise template (2-piece)

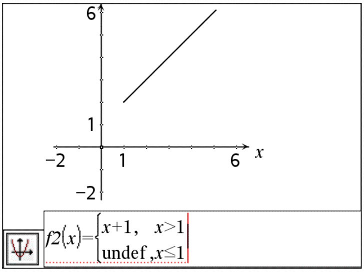
Catalog >

$\begin{cases}\square, & \square\\ \square, & \square\end{cases}$

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 **piecewise()**, page 131.

Example:

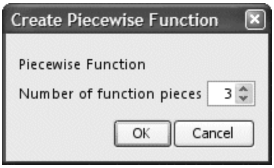


$f2(x)=\begin{cases}x+1, & x>1\\ \text{undef}, & x\le 1\end{cases}$

Piecewise template (N-piece)

Catalog >

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



Note: See also **piecewise()**, page 131.

Example:

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

System of 2 equations template

Catalog >

$\begin{cases} \square \\ \square \end{cases}$

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

Note: See also **system()**, page 180.

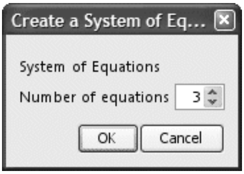
Example:

$\text{solve}\left(\begin{cases} x+y=0 \\ x-y=5 \end{cases}, x, y\right)$	$x=\frac{5}{2} \text{ and } y=\frac{-5}{2}$
$\text{solve}\left(\begin{cases} y=x^2-2 \\ x+2\cdot y=-1 \end{cases}, x, y\right)$	
	$x=\frac{-3}{2} \text{ and } y=\frac{1}{4} \text{ or } x=1 \text{ and } y=-1$

System of N equations template

Catalog >

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



Note: See also **system()**, page 180.

Example:

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

Absolute value template

Catalog >

$|\square|$**Note:** See also **abs()**, page 12.

Example:

Absolute value template

Catalog >

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

dd°mm’ss.ss’’ template

Catalog >

□°□'□"

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.

Example:

30°15'10"	$\frac{10891\cdot\pi}{64800}$

Matrix template (2 x 2)

Catalog >

$\begin{bmatrix} □ & □ \\ □ & □ \end{bmatrix}$

Creates a 2 x 2 matrix.

Example:

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

Matrix template (1 x 2)

Catalog >

$\begin{bmatrix} □ & □ \end{bmatrix}$.

Example:

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

Matrix template (2 x 1)

Catalog >

$\begin{bmatrix} □ \\ □ \end{bmatrix}$

Example:

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

Matrix template (m x n)

Catalog >

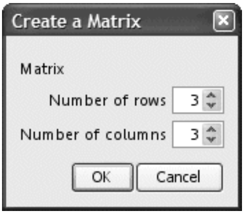
The template appears after you are prompted to specify the number of rows and columns.

Example:

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

Matrix template (m x n)

Catalog >



Create a Matrix

Matrix

Number of rows 3

Number of columns 3

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)

Catalog >

$$\sum_{\square=\square}^{\square}(\square)$$

Note: See also Σ**()** (**sumSeq**), page 222.

Example:

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

Product template (Π)

Catalog >

$$\prod_{\square=\square}^{\square}(\square)$$

Note: See also Π**()** (**prodSeq**), page 221.

Example:

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

First derivative template

Catalog >

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

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

Note: See also **d() (derivative)**, page 219.

Example:

$\frac{d}{dx}\left(x^3\right)$	$3\cdot x^2$
$\frac{d}{dx}\left(x^3\right)\vert x=3$	27

Second derivative template

Catalog >

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

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

Note: See also **d() (derivative)**, page 219.

Example:

$\frac{d^2}{dx^2}\left(x^3\right)$	$6\cdot x$
$\frac{d^2}{dx^2}\left(x^3\right)\vert x=3$	18

Nth derivative template

Catalog >

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

The nth derivative template can be used to calculate the nth derivative.

Note: See also **d() (derivative)**, page 219.

Example:

$\frac{d^3}{dx^3}\left(x^3\right)\vert x=3$	6

Definite integral template

Catalog >

$$\int_{\square}^{\square} \square \, d\square$$

Note: See also∫**() integral()**, page 219.

Example:

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

Indefinite integral template

Catalog >

$$\int \square \, d\square$$

Note: See also ∫**() integral()**, page 219.

Example:

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

Limit template

Catalog >

$$\lim_{\square \to \square^{\square}} (\square)$$

Example:

$\lim_{x \to 5^{\square}} (2\cdot x+3)$	13

Limit template **Catalog >**

Use − or (−) for left hand limit. Use + for right hand limit.

Note: See also **limit()**, page 10.

Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, page 208. 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*

abs(*List1***)** ⇒ *list*
abs(*Matrix1***)** ⇒ *matrix*

Returns the absolute value of the argument.

Note: See also **Absolute value template**, page 7.

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

Note: All undefined variables are treated as real variables.

$\left\|\left\{\frac{\pi}{2},\frac{-\pi}{3}\right\}\right\|$	$\left\{\frac{\pi}{2},\frac{\pi}{3}\right\}$
$\|2-3\cdot i\|$	$\sqrt{13}$
$\|z\|$	$\|z\|$
$\|x+y\cdot i\|$	$\sqrt{x^2+y^2}$

amortTbl() Catalog >

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

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 193.

- 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.

$\text{amortTbl}(12,60,10,5000,,,12,12)$

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

amortTbl() Catalog >

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 Σ**Int()** and Σ**Prn()**, page 223, and **bal()**, page 21.

and Catalog >

BooleanExpr1 **and** *BooleanExpr2* ⇒ *Boolean expression*

BooleanList1 **and** *BooleanList2* ⇒ *Boolean list*

BooleanMatrix1 **and** *BooleanMatrix2* ⇒ *Boolean matrix*

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

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

Integer1 **and***Integer2* ⇒ *integer*

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.

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 Hex base mode:

0h7AC36 and 0h3D5F	0h2C16

Important: Zero, not the letter O.

In Bin base mode:

0b100101 and 0b100	0b100

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:

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

In Gradian angle mode:

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

In Radian angle mode:

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

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

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

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.

$\text{angle}(\{1+2\cdot i,3+0\cdot i,0-4\cdot i\})$

$\left\{\frac{\pi}{2}-\tan^{-1}\left(\frac{1}{2}\right),0,\frac{-\pi}{2}\right\}$

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 175)

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
stat.df	Degrees of freedom of the groups
stat.SS	Sum of squares of the groups
stat.MS	Mean squares for the groups

Output variable	Description
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 175.)

LevRow=0 for Block

LevRow=2,3,...,*Len*-1, for Two Factor, where *Len*=length(*List1*)=length(*List2*) = … = length(*List10*) and *Len* / *LevRow* Î {2,3,…}

`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
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

Output variable	Description
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
stat.SSInteract	Sum of squares of the interaction
stat.MSInteract	Mean squares for interaction

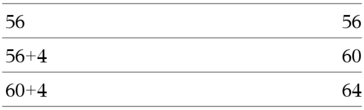
ERROR Outputs

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 ctrl (–) keys

Ans ⇒ *value*

Returns the result of the most recently evaluated expression.



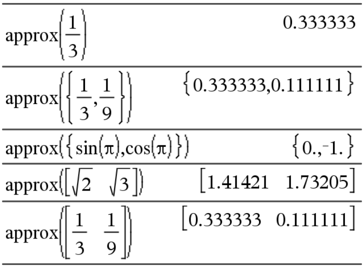
```
56      56
56+4    60
60+4    64
```

approx() Catalog >

approx(*Expr1***)** ⇒ *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 ctrl enter.



$\text{approx}\left(\frac{1}{3}\right)$ 0.333333

$\text{approx}\left(\left\{\frac{1}{3},\frac{1}{9}\right\}\right)$ $\{0.333333,0.111111\}$

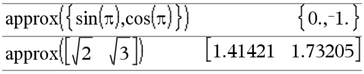
$\text{approx}(\{\sin(\pi),\cos(\pi)\})$ $\{0.,^{-}1.\}$

$\text{approx}\left(\left[\sqrt{2}\quad\sqrt{3}\right]\right)$ $[1.41421\quad 1.73205]$

$\text{approx}\left(\left[\frac{1}{3}\quad\frac{1}{9}\right]\right)$ $[0.333333\quad 0.111111]$

approx(*List1***)** ⇒ *list*
approx(*Matrix1***)** ⇒ *matrix*

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



$\text{approx}(\{\sin(\pi),\cos(\pi)\})$ $\{0.,^{-}1.\}$

$\text{approx}\left(\left[\sqrt{2}\quad\sqrt{3}\right]\right)$ $[1.41421\quad 1.73205]$

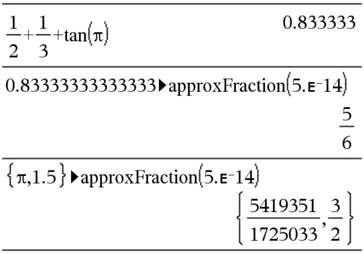
►approxFraction() Catalog >

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

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

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

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



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

0.83333333333333►approxFraction(5.E-14) $\frac{5}{6}$

$\{\pi,1.5\}$►approxFraction(5.E-14) $\left\{\frac{5419351}{1725033},\frac{3}{2}\right\}$

►approxFraction() Catalog >

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

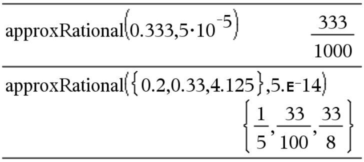
approxRational() Catalog >

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

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

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

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



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

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

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

arccos() See cos⁻¹(), page 35.

arccosh() See cosh⁻¹(), page 36.

arccot() See cot⁻¹(), page 37.

arccoth() See coth⁻¹(), page 38.

arccsc() See csc⁻¹(), page 41.

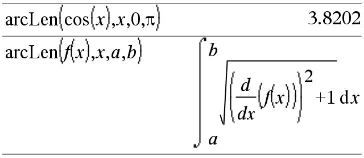
arccsch() See csch⁻¹(), page 41.

arcLen() Catalog >

arcLen(*Expr1***,***Var***,***Start***,***End***)** ⇒ *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.

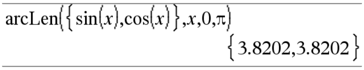


$\text{arcLen}(\cos(x),x,0,\pi)$ 3.8202

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

arcLen(*List1***,***Var***,***Start***,***End***)** ⇒ *list*

Returns a list of the arc lengths of each element of *List1* from *Start* to *End* with respect to *Var*.



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

$\{3.8202,3.8202\}$

arcsec() See sec^{-1}(), page 156.

arcsech() See sech^{-1}(), page 156.

arcsin() See sin^{-1}(), page 166.

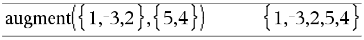
arcsinh() See sinh^{-1}(), page 167.

arctan() See tan^{-1}(), page 181.

arctanh() See tanh^{-1}(), page 183.

augment() Catalog >

augment(*List1, List2***)** ⇒ *list*



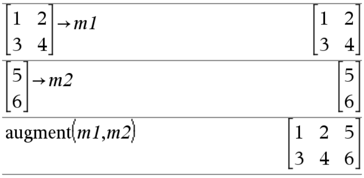
$\text{augment}(\{1,^-3,2\},\{5,4\})$ $\{1,^-3,2,5,4\}$

augment() Catalog >

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

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

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*.



$\begin{bmatrix}1&2\\3&4\end{bmatrix}\to m1$ $\quad\begin{bmatrix}1&2\\3&4\end{bmatrix}$

$\begin{bmatrix}5\\6\end{bmatrix}\to m2$ $\quad\begin{bmatrix}5\\6\end{bmatrix}$

$\text{augment}(m1,m2)$ $\quad\begin{bmatrix}1&2&5\\3&4&6\end{bmatrix}$

avgRC() Catalog >

avgRC(*Expr1***,** *Var* [=*Value*] [**,** *Step*]**)** ⇒ *expression*

avgRC(*Expr1***,** *Var* [=*Value*] [**,** *List1*]**)** ⇒ *list*

avgRC(*List1***,** *Var* [=*Value*] [**,** *Step*]**)** ⇒ *list*

avgRC(*Matrix1***,** *Var* [=*Value*] [**,** *Step*]**)** ⇒ *matrix*

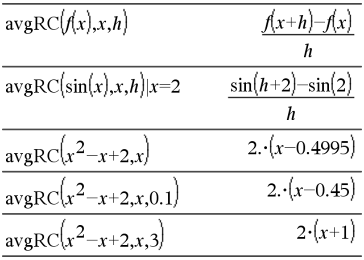
Returns the forward-difference quotient (average rate of change).

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.



$\text{avgRC}(f(x),x,h)$ $\quad\dfrac{f(x+h)-f(x)}{h}$

$\text{avgRC}(\sin(x),x,h)|x=2$ $\quad\dfrac{\sin(h+2)-\sin(2)}{h}$

$\text{avgRC}(x^2-x+2,x)$ $\quad 2.\cdot(x-0.4995)$

$\text{avgRC}(x^2-x+2,x,0.1)$ $\quad 2.\cdot(x-0.45)$

$\text{avgRC}(x^2-x+2,x,3)$ $\quad 2\cdot(x+1)$

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 193.

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 193.

- 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 12.

Note: See also **ΣInt()** and **ΣPrn()**, page 223.

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

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

$$\begin{bmatrix} 0 & 0. & 0. & 5000. \\ 1 & {}^{-}23.35 & {}^{-}825.63 & 4174.37 \\ 2 & {}^{-}19.49 & {}^{-}829.49 & 3344.88 \\ 3 & {}^{-}15.62 & {}^{-}833.36 & 2511.52 \\ 4 & {}^{-}11.73 & {}^{-}837.25 & 1674.27 \\ 5 & {}^{-}7.82 & {}^{-}841.16 & 833.11 \\ 6 & {}^{-}3.89 & {}^{-}845.09 & {}^{-}11.98 \end{bmatrix}$$

bal(4,*tbl*)	1674.27

▶Base2 Catalog >

Integer1 ▶**Base2** ⇒ *integer*

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

256▸Base2	0b100000000
0h1F▸Base2	0b11111

▶Base2

Catalog >

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

$^{-}2^{63}$ 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.

2^{63} becomes $^{-}2^{63}$ and is displayed as
0h8000000000000000 in Hex base mode
0b100...000 (63 zeros) in Binary base mode

2^{64} becomes 0 and is displayed as
0h0 in Hex base mode
0b0 in Binary base mode

$^{-}2^{63} - 1$ becomes $2^{63} - 1$ and is displayed as
0h7FFFFFFFFFFFFFFF in Hex base mode
0b111...111 (64 1’s) in Binary base mode

▶Base10

Catalog >

Integer1 **▶Base10** ⇒ *integer*

0b10011▶Base10	19
0h1F▶Base10	31

▶Base10

Catalog >

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.

▶Base16

Catalog >

Integer1 **▶Base16** ⇒ *integer*

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

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 21.

256▶Base16	0h100
0b111100001111▶Base16	0hF0F

binomCdf() Catalog >

binomCdf(*n*,*p***)** ⇒ *list*

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

binomCdf(*n*,*p*,*upBound***)**for P(0≤X≤*upBound*) ⇒ *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 ≤ *upBound*), set *lowBound*=0

binomPdf() Catalog >

binomPdf(*n*,*p***)** ⇒ *list*

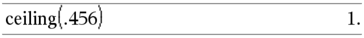
binomPdf(*n*,*p*,*XVal***)** ⇒ *number* if *XVal* is a number, *list* if *XVal* is a list

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

C

Catalog >

ceiling(*Expr1***)** ⇒ *integer*



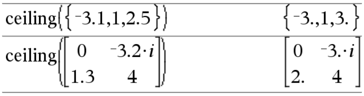
$\text{ceiling}(.456)$ $\quad$ 1.

Returns the nearest integer that is ≥ the argument.

The argument can be a real or a complex number.

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

ceiling(*List1***)** ⇒ *list*
ceiling(*Matrix1***)** ⇒ *matrix*



$\text{ceiling}(\{^-3.1,1,2.5\})$ $\quad$ $\{^-3.,1,3.\}$

$\text{ceiling}\left(\begin{bmatrix} 0 & ^-3.2\cdot i \\ 1.3 & 4 \end{bmatrix}\right)$ $\quad$ $\begin{bmatrix} 0 & ^-3.\cdot i \\ 2. & 4 \end{bmatrix}$

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_

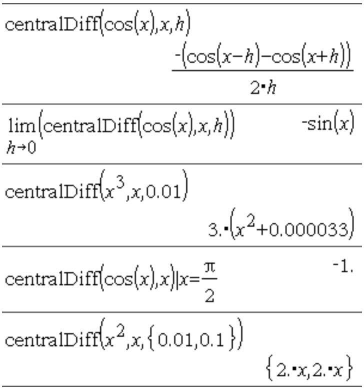
Returns the numerical derivative using the central difference quotient formula.

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*()**.



$\text{centralDiff}(\cos(x),x,h)$

$\frac{^{-}(\cos(x-h)-\cos(x+h))}{2\cdot h}$

$\lim_{h\to 0}(\text{centralDiff}(\cos(x),x,h))$ $\quad {}^{-}\sin(x)$

$\text{centralDiff}(x^3,x,0.01)$

$3.\cdot(x^2+0.000033)$

$\text{centralDiff}(\cos(x),x)|x=\frac{\pi}{2}$ $\quad {}^{-}1.$

$\text{centralDiff}(x^2,x,\{0.01,0.1\})$

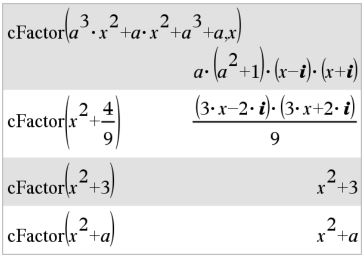
$\{2.\cdot x,2.\cdot x\}$

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.



$\text{cFactor}(a^3\cdot x^2+a\cdot x^2+a^3+a,x)$

$a\cdot(a^2+1)\cdot(x-\boldsymbol{i})\cdot(x+\boldsymbol{i})$

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

$\text{cFactor}(x^2+3)$ $\quad x^2+3$

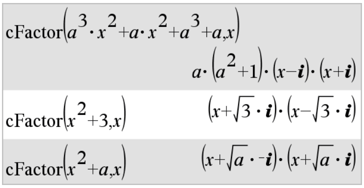
$\text{cFactor}(x^2+a)$ $\quad x^2+a$

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.



$\text{cFactor}\left(a^3\cdot x^2+a\cdot x^2+a^3+a,x\right)$

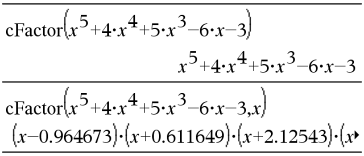
$a\cdot\left(a^2+1\right)\cdot\left(x-\boldsymbol{i}\right)\cdot\left(x+\boldsymbol{i}\right)$

$\text{cFactor}\left(x^2+3,x\right)$ $\left(x+\sqrt{3}\cdot\boldsymbol{i}\right)\cdot\left(x-\sqrt{3}\cdot\boldsymbol{i}\right)$

$\text{cFactor}\left(x^2+a,x\right)$ $\left(x+\sqrt{a}\cdot -\boldsymbol{i}\right)\cdot\left(x+\sqrt{a}\cdot\boldsymbol{i}\right)$

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()**.



$\text{cFactor}\left(x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3\right)$

$x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3$

$\text{cFactor}\left(x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3,x\right)$

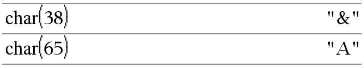
$\left(x-0.964673\right)\cdot\left(x+0.611649\right)\cdot\left(x+2.12543\right)\cdot\left(x\right.$ ▸

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

char() Catalog >

char(*Integer***)** ⇒ *character*

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



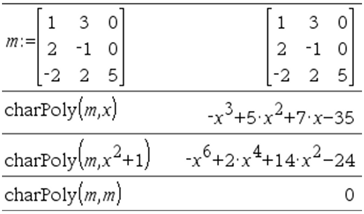
char(38)	"&"
char(65)	"A"

charPoly() Catalog >

charPoly(*squareMatrix,Var***)** ⇒ *polynomial expression*

charPoly(*squareMatrix,Expr***)** ⇒ *polynomial expression*

charPoly(*squareMatrix1,Matrix2***)** ⇒ *polynomial expression*



$m:=\begin{bmatrix}1 & 3 & 0\\ 2 & -1 & 0\\ -2 & 2 & 5\end{bmatrix}$ $\begin{bmatrix}1 & 3 & 0\\ 2 & -1 & 0\\ -2 & 2 & 5\end{bmatrix}$

$\text{charPoly}(m,x)$ $-x^3+5\cdot x^2+7\cdot x-35$

$\text{charPoly}\left(m,x^2+1\right)$ $-x^6+2\cdot x^4+14\cdot x^2-24$

$\text{charPoly}(m,m)$ 0

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

$p_A(\lambda) = \det(\lambda \bullet \mathrm{I} - A)$

where I denotes the $n \times n$ identity matrix.

squareMatrix1 and *squareMatrix2* must have the equal dimensions.

$\chi^2$2way Catalog >

$\chi^2$2way *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 175)

For information on the effect of empty elements in a matrix, see “Empty (Void) Elements,” page 234.

Output variable	Description
stat.χ^2	Chi square stat: sum (observed - expected)^2/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

χ^2Cdf() **Catalog >**

χ^2**Cdf(***lowBound*,*upBound*,*df***)** ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

chi2Cdf(*lowBound*,*upBound*,*df***)** ⇒ *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($X \leq upBound$), set *lowBound* = 0.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

χ^2GOF **Catalog >**

χ^2**GOF** *obsList*,*expList*,*df*

chi2GOF *obsList*,*expList*,*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 175.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	**Description**
stat.χ^2	Chi square stat: sum((observed - expected)^2/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

χ^2Pdf() **Catalog >**

χ^2**Pdf(***XVal*,*df***)** ⇒ *number* if *XVal* is a

χ^2Pdf() Catalog > 📖

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

chi2Pdf(*XVal*,*df***)** ⇒ *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*.

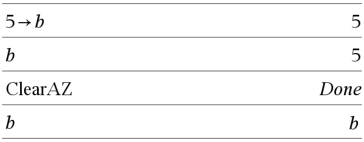
For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

ClearAZ Catalog > 📖

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 195.



```
5→b          5
b            5
ClearAZ   Done
b            b
```

ClrErr Catalog > 📖

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 130, and **Try**, page 189.

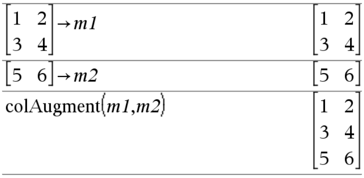
Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

For an example of **ClrErr**, See Example 2 under the **Try** command, page 189.

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*.



$\begin{bmatrix}1&2\\3&4\end{bmatrix}\rightarrow m1$ $\qquad\begin{bmatrix}1&2\\3&4\end{bmatrix}$

$\begin{bmatrix}5&6\end{bmatrix}\rightarrow m2$ $\qquad\begin{bmatrix}5&6\end{bmatrix}$

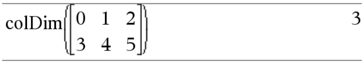
$\text{colAugment}(m1,m2)$ $\qquad\begin{bmatrix}1&2\\3&4\\5&6\end{bmatrix}$

colDim() Catalog >

colDim(*Matrix***)** ⇒ *expression*

Returns the number of columns contained in *Matrix*.

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



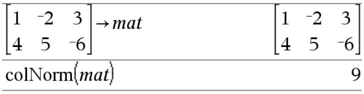
$\text{colDim}\left(\begin{bmatrix}0&1&2\\3&4&5\end{bmatrix}\right)$ $\qquad 3$

colNorm() Catalog >

colNorm(*Matrix***)** ⇒ *expression*

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

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



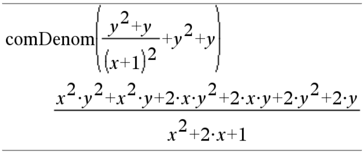
$\begin{bmatrix}1&{}^{-}2&3\\4&5&{}^{-}6\end{bmatrix}\rightarrow mat$ $\qquad\begin{bmatrix}1&{}^{-}2&3\\4&5&{}^{-}6\end{bmatrix}$

$\text{colNorm}(mat)$ $\qquad 9$

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.



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

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

comDenom() Catalog >

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.

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

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

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.

$$\text{Define } \mathit{comden}(\mathit{exprn})=\text{comDenom}(\mathit{exprn},\mathit{abc}) \qquad \mathit{Done}$$

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

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.

$$\mathit{comden}\left(1234\cdot x^2\cdot(y^3-y)+2468\cdot x\cdot(y^2-1)\right)$$
$$1234\cdot x\cdot(x\cdot y+2)\cdot(y^2-1)$$

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

completeSquare () Catalog >

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•x2+b•x+c into the form a•(x-h)2+k

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

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

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

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

completeSquare () Catalog >

- or -

Converts a quadratic equation of the form a•x²+b•x+c=d into the form a•(x-h)²=k

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

The Second argument must be a single univariate term or a single univariate term raised to a rational power, for example x, y², or z^(1/3).

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

$$\text{completeSquare}\left(3\cdot x^2+2\cdot y+7\cdot y^2+4\cdot x=3,\{x,y\}\right)$$
$$3\cdot\left(x+\frac{2}{3}\right)^2+7\cdot\left(y+\frac{1}{7}\right)^2=\frac{94}{21}$$

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

conj() Catalog >

conj(*Expr1***)** ⇒ *expression*

conj(*List1***)** ⇒ *list*

conj(*Matrix1***)** ⇒ *matrix*

Returns the complex conjugate of the argument.

Note: All undefined variables are treated as real variables.

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

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

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

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

constructMat() Catalog >

constructMat (*Expr***,***Var1***,***Var2***,***numRows***,***numCols***)** ⇒ *matrix*

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*.

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

CopyVar Catalog >

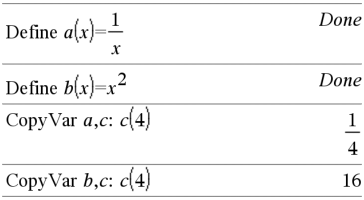
CopyVar *Var1*, *Var2*

CopyVar *Var1***.**, *Var2***.**

CopyVar *Var1*, *Var2* copies the value of variable *Var1* to variable *Var2*, creating *Var2* if necessary. Variable *Var1* must have a value.

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.

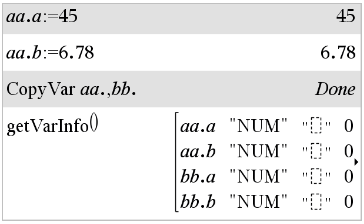
Var1 must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.



Define $a(x)=\frac{1}{x}$	*Done*
Define $b(x)=x^2$	*Done*
CopyVar a,c: $c(4)$	$\frac{1}{4}$
CopyVar b,c: $c(4)$	16

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

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.



$aa.a$:=45	45
$aa.b$:=6.78	6.78
CopyVar $aa.,bb.$	*Done*
getVarInfo()	$\begin{bmatrix} aa.a & \text{"NUM"} & \text{"⬚"} & 0 \\ aa.b & \text{"NUM"} & \text{"⬚"} & 0 \\ bb.a & \text{"NUM"} & \text{"⬚"} & 0 \\ bb.b & \text{"NUM"} & \text{"⬚"} & 0 \end{bmatrix}$

corrMat() Catalog >

corrMat(*List1***,***List2*[**,**...[**,***List20*]]**)**

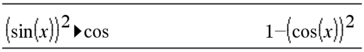
Computes the correlation matrix for the augmented matrix [*List1, List2*, ..., *List20*].

►cos Catalog >

Expr **►cos**

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

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



$(\sin(x))^2$►cos	$1-(\cos(x))^2$

►cos Catalog >

►cos reduces all powers of
sin(...) modulo 1–cos(...)^2
so that any remaining powers of cos(...) have exponents in the range (0, 2). Thus, the result will be free of sin(...) if and only if sin(...) 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() trig key

cos(*Expr1***)** ⇒ *expression*

cos(*List1***)** ⇒ *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*.

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 Degree angle mode:

$$\cos\left(\frac{\pi}{4}^{r}\right) \qquad \frac{\sqrt{2}}{2}$$

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

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

In Gradian angle mode:

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

In Radian angle mode:

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

$$\cos(45°) \qquad \frac{\sqrt{2}}{2}$$

cos(*squareMatrix1***)** ⇒ *squareMatrix*

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

In Radian angle mode:

cos() trig key

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

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

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.

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

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

$\cos^{-1}$() trig key

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

$\cos^{-1}$(*List1***)** ⇒ *list*

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

$\cos^{-1}$(*List1***)** returns a list of the inverse cosines 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 `arccos(...)`.

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

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

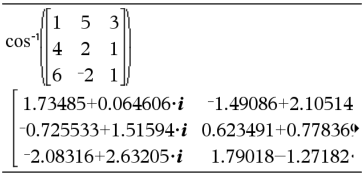
cos⁻¹() trig key

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:



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

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

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

cosh() Catalog >

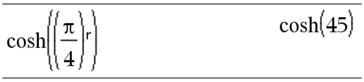
cosh(*Expr1***)** ⇒ *expression*

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*.

In Degree angle mode:



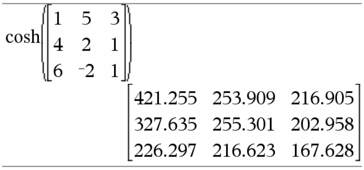
$\cosh\left(\left(\frac{\pi}{4}\right)^{r}\right)$ $\cosh(45)$

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 Radian angle mode:



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

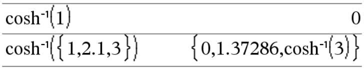
$\begin{bmatrix}421.255 & 253.909 & 216.905\\ 327.635 & 255.301 & 202.958\\ 226.297 & 216.623 & 167.628\end{bmatrix}$

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.



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

$\cosh^{-1}(\{1,2.1,3\})$ $\{0,1.37286,\cosh^{-1}(3)\}$

$\cosh^{-1}()$ Catalog >

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

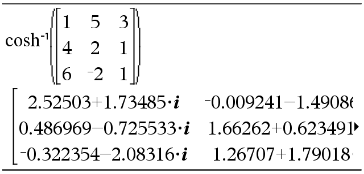
Note: You can insert this function from the keyboard by typing **`arccosh(...)`**.

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:



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

$$\begin{bmatrix}2.52503+1.73485\cdot i & ^{-}0.009241-1.4908\\ 0.486969-0.725533\cdot i & 1.66262+0.623491\\ ^{-}0.322354-2.08316\cdot i & 1.26707+1.79018\end{bmatrix}$$

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

cot() trig key

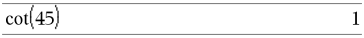
cot(*Expr1***)** ⇒ *expression*

cot(*List1***)** ⇒ *list*

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

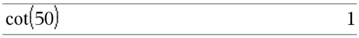
Note: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, ᴳ, or ʳ to override the angle mode temporarily.

In Degree angle mode:



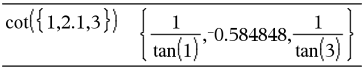
$\cot(45)$ 1

In Gradian angle mode:



$\cot(50)$ 1

In Radian angle mode:



$\cot(\{1,2.1,3\})$ $\left\{\frac{1}{\tan(1)},^{-}0.584848,\frac{1}{\tan(3)}\right\}$

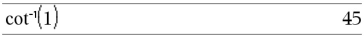
$\cot^{-1}()$ trig key

cot^{-1}**(***Expr1***)** ⇒ *expression*

cot^{-1}**(***List1***)** ⇒ *list*

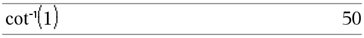
Returns the angle whose cotangent is *Expr1* or returns a list containing the inverse cotangents of each element of *List1*.

In Degree angle mode:



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

In Gradian angle mode:



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

$\mathbf{cot^{-1}()}$ trig key

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 **`arccot(...)`**.

In Radian angle mode:

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

coth() Catalog > 📖

coth(*Expr1***)** ⇒ *expression*

coth(*List1***)** ⇒ *list*

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

$\coth(1.2)$	1.19954
$\coth(\{1,3.2\})$	$\left\{\dfrac{1}{\tanh(1)},1.00333\right\}$

$\mathbf{coth^{-1}()}$ Catalog > 📖

$\mathbf{coth^{-1}(}$*Expr1***)** ⇒ *expression*

$\mathbf{coth^{-1}(}$*List1***)** ⇒ *list*

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(...)`**.

$\coth^{-1}(3.5)$	0.293893
$\coth^{-1}(\{^{-}2,2.1,6\})$	$\left\{\dfrac{^{-}\ln(3)}{2},0.518046,\dfrac{\ln\left(\frac{7}{5}\right)}{2}\right\}$

count() Catalog > 📖

count(*Value1orList1* [,*Value2orList2* [,...]]**)** ⇒ *value*

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

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

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

$\text{count}(2,4,6)$	3
$\text{count}(\{2,4,6\})$	3
$\text{count}\left(2,\{4,6\},\begin{bmatrix}8 & 10\\12 & 14\end{bmatrix}\right)$	7
$\text{count}\left(\dfrac{1}{2},3+4\cdot\boldsymbol{i},\text{undef},\text{"hello"},x+5.,\text{sign}(0)\right)$	2

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

count() Catalog >

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 234.

countif() Catalog >

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 234.

Note: See also **sumIf()**, page 179, and **frequency()**, page 77.

$\text{countIf}\left(\left\{1,3,\text{"abc"},\text{undef},3,1\right\},3\right)$ 2

Counts the number of elements equal to 3.

$\text{countIf}\left(\left\{\text{"abc"},\text{"def"},\text{"abc"},3\right\},\text{"def"}\right)$ 1

Counts the number of elements equal to “def.”

$\text{countIf}\left(\left\{x^{-2},x^{-1},1,x,x^{2}\right\},x\right)$ 1

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

$\text{countIf}\left(\left\{1,3,5,7,9\right\},?<5\right)$ 2

Counts 1 and 3.

$\text{countIf}\left(\left\{1,3,5,7,9\right\},2<?<8\right)$ 3

Counts 3, 5, and 7.

$\text{countIf}\left(\left\{1,3,5,7,9\right\},?<4 \text{ or } ?>6\right)$ 4

Counts 1, 3, 7, and 9.

cPolyRoots() Catalog >

cPolyRoots(*Poly***,***Var***)** ⇒ *list*

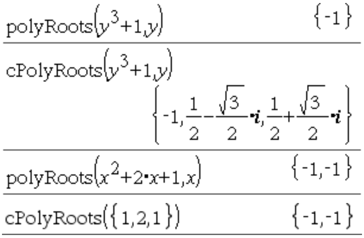
cPolyRoots(*ListOfCoeffs***)** ⇒ *list*

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

Poly must be a polynomial in one variable.

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

Note: See also **polyRoots()**, page 135.



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

$\text{cPolyRoots}\left(y^3+1,y\right)$

$\left\{-1,\frac{1}{2}-\frac{\sqrt{3}}{2}\cdot\boldsymbol{i},\frac{1}{2}+\frac{\sqrt{3}}{2}\cdot\boldsymbol{i}\right\}$

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

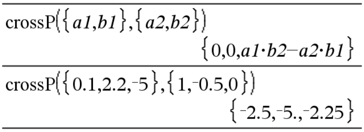
$\text{cPolyRoots}\left(\{1,2,1\}\right) \quad \{-1,-1\}$

crossP() Catalog >

crossP(*List1***,** *List2***)** ⇒ *list*

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

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



$\text{crossP}\left(\{a1,b1\},\{a2,b2\}\right)$

$\{0,0,a1\cdot b2-a2\cdot b1\}$

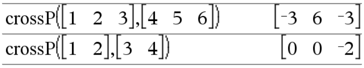
$\text{crossP}\left(\{0.1,2.2,{}^-5\},\{1,{}^-0.5,0\}\right)$

$\{{}^-2.5,{}^-5.,{}^-2.25\}$

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

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.



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

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

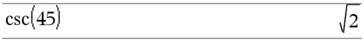
csc() trig key

csc(*Expr1***)** ⇒ *expression*

csc(*List1***)** ⇒ *list*

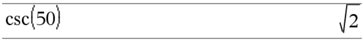
Returns the cosecant of *Expr1* or returns a list containing the cosecants of all elements in *List1*.

In Degree angle mode:



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

In Gradian angle mode:



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

csc() trig key

In Radian angle mode:

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

csc⁻¹() trig key

csc^{-1}(*Expr1***)** ⇒ *expression*

csc^{-1}(*List1***)** ⇒ *list*

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 Degree angle mode:

$\csc^{-1}(1)$	90

In Gradian angle mode:

$\csc^{-1}(1)$	100

In Radian angle mode:

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

csch() Catalog >

csch(*Expr1***)** ⇒ *expression*

csch(*List1***)** ⇒ *list*

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

$\operatorname{csch}(3)$	$\frac{1}{\sinh(3)}$
$\operatorname{csch}(\{1,2.1,4\})$	$\left\{\frac{1}{\sinh(1)},0.248641,\frac{1}{\sinh(4)}\right\}$

csch⁻¹() Catalog >

csch^{-1}(*Expr1***)** ⇒ *expression*

csch^{-1}(*List1***)** ⇒ *list*

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(...)`.

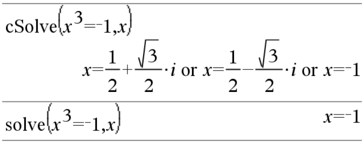
$\operatorname{csch}^{-1}(1)$	$\sinh^{-1}(1)$
$\operatorname{csch}^{-1}(\{1,2.1,3\})$	$\left\{\sinh^{-1}(1),0.459815,\sinh^{-1}\left(\frac{1}{3}\right)\right\}$

cSolve() Catalog >

cSolve(*Equation*, *Var***)** ⇒ *Boolean expression*

cSolve(*Equation*, *Var=Guess***)** ⇒ *Boolean expression*

cSolve(*Inequality*, *Var***)** ⇒ *Boolean expression*



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

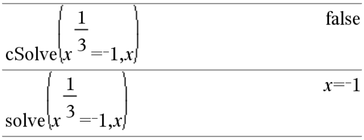
$x=\frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i \text{ or } x=\frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i \text{ or } x=^{-}1$

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

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()**.



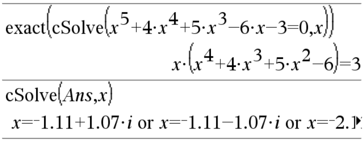
$\text{cSolve}\left(x^{\frac{1}{3}}=^{-}1,x\right)$ false

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

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

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

In Display Digits mode of Fix 2:



$\text{exact}\left(\text{cSolve}\left(x^{5}+4\cdot x^{4}+5\cdot x^{3}-6\cdot x-3=0,x\right)\right)$

$x\cdot\left(x^{4}+4\cdot x^{3}+5\cdot x^{2}-6\right)=3$

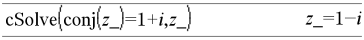
$\text{cSolve}(Ans,x)$

$x=^{-}1.11+1.07\cdot i \text{ or } x=^{-}1.11-1.07\cdot i \text{ or } x=^{-}2.1$ ▸

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

Note: If *Equation* is non-polynomial with functions such as **abs()**, **angle()**, **conj()**, **real ()**, or **imag()**, you should place an underscore (press ctrl ␣) at the end of *Var*. By default, a variable is treated as a real value.

If you use *var_* , the variable is treated as complex.



$\text{cSolve}\left(\text{conj}(z\_)=1+i,z\_\right)$ $z\_=1-i$

cSolve() Catalog >

You should also use *var_* for any other variables in *Equation* that might have unreal values. Otherwise, you may receive unexpected results.

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.

Note: The following examples use an underscore (press ctrl ␣) so that the variables will be treated as complex.

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

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

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

To see the entire result, press ▲ and then use ◀ and ▶ 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.

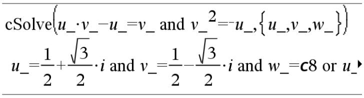
$$\text{cSolve}\left(u\_\cdot v\_-u\_=c\_\cdot v\_ \text{ and } v\_^{2}={}^{-}u\_,\left\{u\_,v\_\right\}\right)$$

$$u\_=\frac{{}^{-}\left(\sqrt{1-4\cdot c\_}+1\right)^{2}}{4} \text{ and } v\_=\frac{\sqrt{1-4\cdot c\_}+1}{2} \text{ or } u\_=\blacktriangleright$$

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

cSolve() Catalog >

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 **c***k*, where *k* is an integer suffix from 1 through 255.



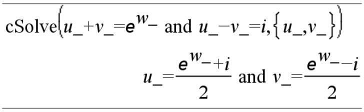
$\text{cSolve}\left(u\_\cdot v\_-u\_=v\_ \text{ and } v\_^2={}^-u\_,\{u\_,v\_,w\_\}\right)$

$u\_=\frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i \text{ and } v\_=\frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i \text{ and } w\_=\mathbf{c}8 \text{ or } u\_$▸

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

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.

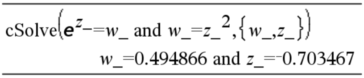
If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in all solution variables, **cSolve()** uses Gaussian elimination to attempt to determine all solutions.



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

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

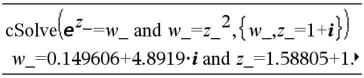
If a system is neither polynomial in all of its variables nor linear in its solution variables, **cSolve()** determines at most one solution using an approximate iterative method. To do so, the number of solution variables must equal the number of equations, and all other variables in the equations must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



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

$w\_=0.494866 \text{ and } z\_={}^-0.703467$

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.



$\text{cSolve}\left(\mathbf{e}^{z\_}=w\_ \text{ and } w\_=z\_^2,\{w\_,z\_=1+\boldsymbol{i}\}\right)$

$w\_=0.149606+4.8919\cdot\boldsymbol{i} \text{ and } z\_=1.58805+1.$▸

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

CubicReg Catalog >

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

Computes the cubic polynomial regression y=a•x3+b•x2+c•x+d on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

CubicReg — Catalog >

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: a•x^3+b•x^2+c•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 *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*

cumulativeSum() — Catalog >

cumulativeSum(*List1***)** ⇒ *list*

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

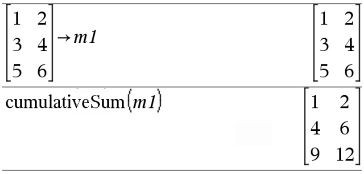
cumulativeSum({1,2,3,4})   {1,3,6,10}

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 234.



$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix} \to m1$ $\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$

$\text{cumulativeSum}(m1)$ $\begin{bmatrix}1 & 2\\4 & 6\\9 & 12\end{bmatrix}$

Cycle Catalog >

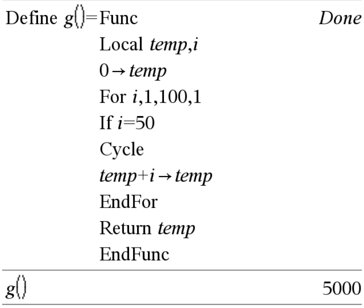
Cycle

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
```

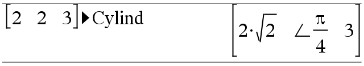
►Cylind Catalog >

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 $\left[2\cdot\sqrt{2}\ \ \angle\frac{\pi}{4}\ \ 3\right]$

cZeros() Catalog >

cZeros(*Expr***,** *Var***)** ⇒ *list*

In Display Digits mode of Fix 3:

cZeros() Catalog >

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()**.

$\text{cZeros}\left(x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3,x\right)$

$\{^-1.1138+1.07314\cdot \boldsymbol{i},^-1.1138-1.07314\cdot \boldsymbol{i},^-2.$▸

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

Note: If *Expr* is non-polynomial with functions such as **abs()**, **angle()**, **conj()**, **real()**, or **imag()**, you should place an underscore (press ctrl ␣) at the end of *Var*. By default, a variable is treated as a real value. If you use *var*_ , the variable is treated as complex.

You should also use *var*_ for any other variables in *Expr* that might have unreal values. Otherwise, you may receive unexpected results.

$\text{cZeros}\left(\text{conj}\left(z\_\right)-1-i,z\_\right)$ $\{1-i\}$

cZeros({*Expr1***,** *Expr2* [**,** …] **},**
{*VarOrGuess1***,***VarOrGuess2* [**,** …] **})** ⇒ *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.

Note: The following examples use an underscore _ (press ctrl ␣) so that the variables will be treated as complex.

cZeros() Catalog >

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*].

$$\mathrm{cZeros}\left(\left\{u\_\cdot v\_-u\_-v\_,v\_^{2}+u\_\right\},\left\{u\_,v\_\right\}\right)$$

$$\begin{bmatrix} 0 & 0 \\ \frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i & \frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i \\ \frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i & \frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i \end{bmatrix}$$

Extract row 2:

$$Ans[2] \qquad \begin{bmatrix} \frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i & \frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i \end{bmatrix}$$

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

$$\mathrm{cZeros}\left(\left\{u\_\cdot v\_-u\_-c\_\cdot v\_,v\_^{2}+u\_\right\},\left\{u\_,v\_\right\}\right)$$

$$\begin{bmatrix} 0 & 0 \\ \frac{-\left(\sqrt{1-4\cdot c\_}-1\right)^{2}}{4} & \frac{-\left(\sqrt{1-4\cdot c\_}-1\right)}{2} \\ \frac{-\left(\sqrt{1-4\cdot c\_}+1\right)^{2}}{4} & \frac{\sqrt{1-4\cdot c\_}+1}{2} \end{bmatrix}$$

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 **c***k*, where *k* is an integer suffix from 1 through 255.

$$\mathrm{cZeros}\left(\left\{u\_\cdot v\_-u\_-v\_,v\_^{2}+u\_\right\},\left\{u\_,v\_,w\_\right\}\right)$$

$$\begin{bmatrix} 0 & 0 & c4 \\ \frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i & \frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i & c4 \\ \frac{1}{2}+\frac{\sqrt{3}}{2}\cdot i & \frac{1}{2}-\frac{\sqrt{3}}{2}\cdot i & c4 \end{bmatrix}$$

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.

$$\mathrm{cZeros}\left(\left\{u\_+v\_-e^{w\_},u\_-v\_-i\right\},\left\{u\_,v\_\right\}\right)$$

$$\begin{bmatrix} \frac{e^{w\_}+i}{2} & \frac{e^{w\_}-i}{2} \end{bmatrix}$$

cZeros() Catalog >

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.

$$\text{cZeros}\left(\left\{\boldsymbol{e}^{z\_}-w\_,w\_-z\_^{2}\right\},\left\{w\_,z\_\right\}\right)$$
$$\begin{bmatrix}0.494866 & ^-0.703467\end{bmatrix}$$

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.

$$\text{cZeros}\left(\left\{\boldsymbol{e}^{z\_}-w\_,w\_-z\_^{2}\right\},\left\{w\_,z\_=1+\boldsymbol{i}\right\}\right)$$
$$\begin{bmatrix}0.149606+4.8919\cdot\boldsymbol{i} & 1.58805+1.54022\cdot\boldsymbol{i}\end{bmatrix}$$

D

dbd() Catalog >

dbd(*date1,date2***)** ⇒ *value*

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

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)

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

►DD Catalog >

Expr1 **►DD** ⇒ *valueList1*
►DD ⇒ *listMatrix1*
►DD ⇒ *matrix*

In Degree angle mode:

►DD — Catalog >

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.

$(1.5°)$►DD	$1.5°$
$(45°22'14.3'')$►DD	$45.3706°$
$(\{45°22'14.3'',60°0'0''\})$►DD	$\{45.3706°,60°\}$

In Gradian angle mode:

1►DD	$\frac{9}{10}°$

In Radian angle mode:

(1.5)►DD	$85.9437°$

►Decimal — Catalog >

Expression1 **►Decimal** ⇒ *expression*

List1 **►Decimal** ⇒ *expression*

Matrix1 **►Decimal** ⇒ *expression*

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

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

$\frac{1}{3}$►Decimal	0.333333

Define — Catalog >

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.

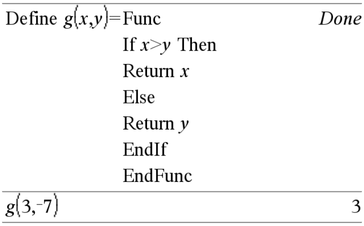
Define $g(x,y)=2\cdot x-3\cdot y$	*Done*
$g(1,2)$	$^{-}4$
$1\rightarrow a$: $2\rightarrow b$: $g(a,b)$	$^{-}4$
Define $h(x)=\text{when}(x<2,2\cdot x-3,^{-}2\cdot x+3)$	*Done*
$h(^{-}3)$	$^{-}9$
$h(4)$	$^{-}5$

Define **Catalog >**

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

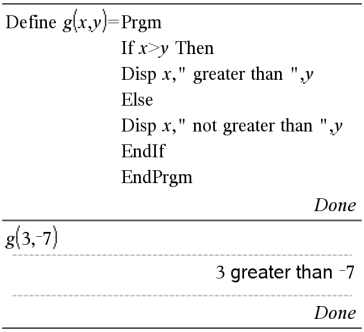


```
Define g(x,y)=Func                    Done
              If x>y Then
              Return x
              Else
              Return y
              EndIf
              EndFunc
g(3,-7)                                  3
```

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)=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
```

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 51, and **Define LibPub**, page 52.

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***,**

Define LibPriv — Catalog > 📖

...) = **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 50, and **Define LibPub**, page 52.

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 50, and **Define LibPriv**, page 51.

deltaList() — See ΔList(), page 102.

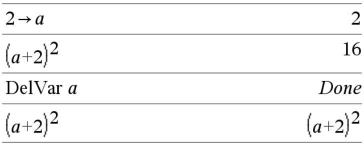
deltaTmpCnv() — See ΔtmpCnv(), page 188.

DelVar Catalog >

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

DelVar *Var***.**

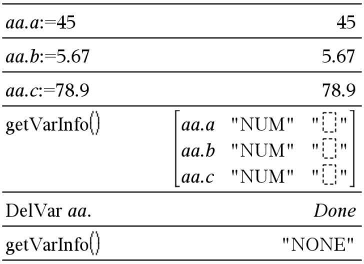
Deletes the specified variable or variable group from memory.



$2 \to a$	2
$(a+2)^2$	16
DelVar *a*	*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 195.

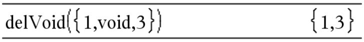
DelVar *Var***.** deletes all members of the *Var***.** variable group (such as the statistics *stat.nn* results or variables created using the **LibShortcut()** function). The dot (**.**) in this form of the **DelVar** command limits it to deleting a variable group; the simple variable *Var* is not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



$aa.a$:=45	45
$aa.b$:=5.67	5.67
$aa.c$:=78.9	78.9
getVarInfo()	$\begin{bmatrix} aa.a & \text{"NUM"} & \text{"[]"} \\ aa.b & \text{"NUM"} & \text{"[]"} \\ aa.c & \text{"NUM"} & \text{"[]"} \end{bmatrix}$
DelVar *aa.*	*Done*
getVarInfo()	"NONE"

delVoid() Catalog >

delVoid(*List1***)** ⇒ *list*



delVoid({1,void,3})	{1,3}

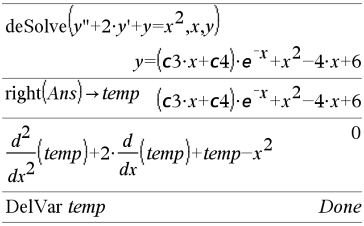
Returns a list that has the contents of *List1* with all empty (void) elements removed.

For more information on empty elements, see page 234.

derivative() See *d*(), page 219.

deSolve() Catalog >

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



deSolve$(y''+2\cdot y'+y=x^2,x,y)$	$y=(\mathbf{c}3\cdot x+\mathbf{c}4)\cdot \mathbf{e}^{-x}+x^2-4\cdot x+6$
right$(Ans) \to temp$	$(\mathbf{c}3\cdot x+\mathbf{c}4)\cdot \mathbf{e}^{-x}+x^2-4\cdot x+6$
$\frac{d^2}{dx^2}(temp)+2\cdot\frac{d}{dx}(temp)+temp-x^2$	0
DelVar *temp*	*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 **c***k*, 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.

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.

$\mathrm{deSolve}\left(y'=(\cos(y))^2\cdot x,x,y\right)$	$\tan(y)=\frac{x^2}{2}+\mathbf{c}4$

$\mathrm{solve}(Ans,y)$	$y=\tan^{-1}\left(\frac{x^2+2\cdot\mathbf{c}4}{2}\right)+\mathbf{n}3\cdot\pi$
$Ans\vert\mathbf{c}4=c-1 \text{ and } \mathbf{n}3=0$	$y=\tan^{-1}\left(\frac{x^2+2\cdot(c-1)}{2}\right)$

deSolve(*1stOrderODE* **and** *initCond***,** *Var***,** *depVar***)** ⇒ *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.

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

deSolve() Catalog >

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

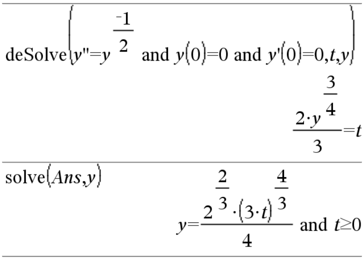
Returns a particular solution that satisfies *2nd Order ODE* and has a specified value of the dependent variable and its first derivative at one point.

For *initCond1*, use the form:

depVar (*initialIndependentValue*) = *initialDependentValue*

For *initCond2*, use the form:

depVar (*initialIndependentValue*) = *initial1stDerivativeValue*



$\text{deSolve}\left(y''=y^{\frac{-1}{2}} \text{ and } y(0)=0 \text{ and } y'(0)=0,t,y\right)$

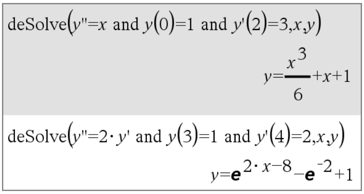
$$\frac{2\cdot y^{\frac{3}{4}}}{3}=t$$

$\text{solve}(Ans,y)$

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

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

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

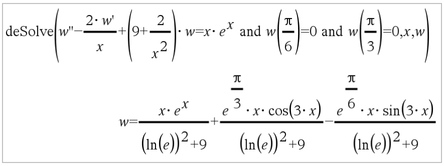


$\text{deSolve}\left(y''=x \text{ and } y(0)=1 \text{ and } y'(2)=3,x,y\right)$

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

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

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



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

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

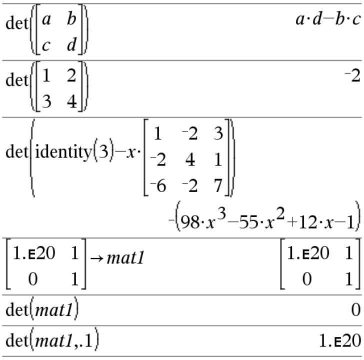
det() Catalog >

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 ctrl enter or set the **Auto or Approximate** mode to Approximate,



$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right)$ $a\cdot d-b\cdot c$

$\det\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$ -2

$\det\left(\text{identity}(3)-x\cdot\begin{bmatrix} 1 & -2 & 3 \\ -2 & 4 & 1 \\ -6 & -2 & 7 \end{bmatrix}\right)$

$$-\left(98\cdot x^3-55\cdot x^2+12\cdot x-1\right)$$

$\begin{bmatrix} 1.\text{E}20 & 1 \\ 0 & 1 \end{bmatrix}\rightarrow mat1$ $\begin{bmatrix} 1.\text{E}20 & 1 \\ 0 & 1 \end{bmatrix}$

$\det(mat1)$ 0

$\det(mat1,.1)$ $1.\text{E}20$

det() Catalog >

computations are done using floating-point arithmetic.

- If *Tolerance* is omitted or not used, the default tolerance is calculated as:
  5E^{-}14 **•max(dim (***squareMatrix***))•rowNorm (***squareMatrix***)**

diag() Catalog >

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.

$\mathrm{diag}\left(\begin{bmatrix}2 & 4 & 6\end{bmatrix}\right)$ $\quad\begin{bmatrix}2 & 0 & 0\\ 0 & 4 & 0\\ 0 & 0 & 6\end{bmatrix}$

diag(*squareMatrix***)** ⇒ *rowMatrix*

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

squareMatrix must be square.

$\begin{bmatrix}4 & 6 & 8\\ 1 & 2 & 3\\ 5 & 7 & 9\end{bmatrix}$ $\quad\begin{bmatrix}4 & 6 & 8\\ 1 & 2 & 3\\ 5 & 7 & 9\end{bmatrix}$

$\mathrm{diag}(Ans)$ $\quad\begin{bmatrix}4 & 2 & 9\end{bmatrix}$

dim() Catalog >

dim(*List***)** ⇒ *integer*

Returns the dimension of *List*.

$\mathrm{dim}(\{0,1,2\})$ $\quad 3$

dim(*Matrix***)** ⇒ *list*

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

$\mathrm{dim}\left(\begin{bmatrix}1 & {}^-1\\ 2 & {}^-2\\ 3 & 5\end{bmatrix}\right)$ $\quad\{3,2\}$

dim(*String***)** ⇒ *integer*

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

dim("Hello") 5

dim("Hello "&"there") 11

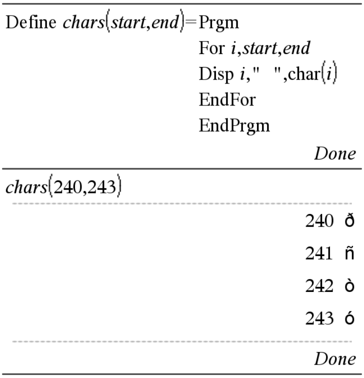
Disp — Catalog >

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  ð
                                      241  ñ
                                      242  ò
                                      243  ó
                                          Done
```

▶DMS — Catalog >

Expr ▶**DMS**

List ▶**DMS**

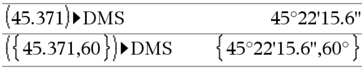
Matrix ▶**DMS**

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 226 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.

In Degree angle mode:



```
(45.371)▶DMS                     45°22'15.6"
({45.371,60})▶DMS       {45°22'15.6",60°}
```

domain() Catalog >

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 ∫() is a disallowed function.

$$\text{domain}\left(\int_{1}^{x} \frac{1}{t}\,dt,x\right) ▸ \text{domain}\left(\int_{1}^{x} \frac{1}{t}\,dt,x\right)$$

$\text{domain}(x^2,x)$	$-\infty<x<\infty$
$\text{domain}\left(\frac{x+1}{x^2+2\cdot x},x\right)$	$x\neq-2$ and $x\neq0$
$\text{domain}\left(\left(\sqrt{x}\right)^2,x\right)$	$0\leq x<\infty$
$\text{domain}\left(\frac{1}{x+y},y\right)$	$y\neq-x$

dominantTerm() Catalog >

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

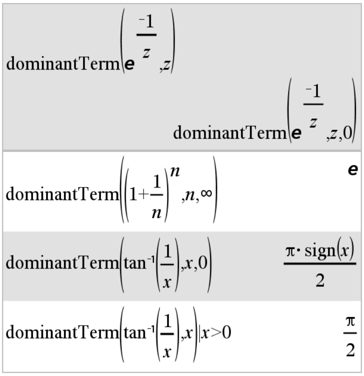
dominantTerm(*Expr1*, *Var* [**,** *Point*]**) |** *Var***>***Point* ⇒ *expression*

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

$\text{dominantTerm}(\tan(\sin(x))-\sin(\tan(x)),x)$	$\frac{x^7}{30}$
$\text{dominantTerm}\left(\frac{1-\cos(x-1)}{(x-1)^3},x,1\right)$	$\frac{1}{2\cdot(x-1)}$
$\text{dominantTerm}\left(x^{-2}\cdot\tan\left(x^{\frac{1}{3}}\right),x\right)$	$\frac{1}{x^{\frac{5}{3}}}$
$\text{dominantTerm}\left(\ln\left(x^x-1\right)\cdot x^{-2},x\right)$	$\frac{\ln(x\cdot\ln(x))}{x^2}$

dominantTerm() Catalog >

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.



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

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

$\text{dominantTerm}\left(\tan^{-1}\left(\frac{1}{x}\right),x,0\right)$ $\frac{\pi\cdot \text{sign}(x)}{2}$

$\text{dominantTerm}\left(\tan^{-1}\left(\frac{1}{x}\right),x\right)|x>0$ $\frac{\pi}{2}$

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 = ∞ or −∞.

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 sign(…) or abs(…) for a real expansion variable or $(-1)^{\text{floor}(\ldots\text{angle}(\ldots)\ldots)}$ 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.

dominantTerm() is useful when you want to know the simplest possible expression that is asymptotic to another expression as *Var*→*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.

dominantTerm() Catalog >

Note: See also **series()**, page 159.

dotP() Catalog >

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.

Both must be row vectors, or both must be column 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

E

e^() $\boxed{e^x}$ key

*e***^(***Expr1***)** ⇒ *expression*

Returns ***e*** raised to the *Expr1* power.

Note: See also ***e* exponent template**, page 6.

Note: Pressing $\boxed{e^x}$ to display ***e*^(** is different from pressing the character $\boxed{\mathbf{E}}$ on the keyboard.

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

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

*e***^(***List1***)** ⇒ *list*

Returns ***e*** raised to the power of each element in *List1*.

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

*e***^(***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()**.

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

e^() **e^x key**

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

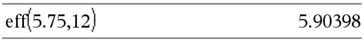
eff() **Catalog >**

eff(*nominalRate,CpY***)** ⇒ *value*

Financial 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 122.



```
eff(5.75,12)                                5.90398
```

eigVc() **Catalog >**

eigVc(*squareMatrix***)** ⇒ *matrix*

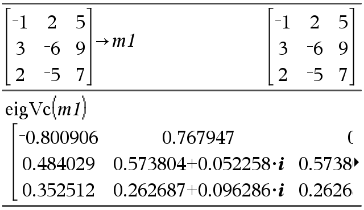
Returns a matrix containing the eigenvectors for a real or complex *squareMatrix*, where each column in the result corresponds to an eigenvalue. Note that an eigenvector is not unique; it may be scaled by any constant factor. The eigenvectors are normalized, meaning that:

if V = [x_1, x_2, … , x_n]

then $x_1^2 + x_2^2 + … + x_n^2 = 1$

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.

In Rectangular Complex Format:



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

eigVc($m1$)

$\begin{bmatrix} ^-0.800906 & 0.767947 & (\\ 0.484029 & 0.573804+0.052258\cdot i & 0.5738▸ \\ 0.352512 & 0.262687+0.096286\cdot i & 0.2626 \end{bmatrix}$

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

eigVl() **Catalog >**

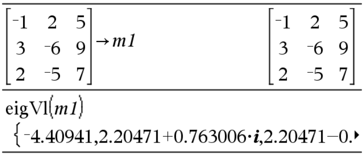
eigVl(*squareMatrix***)** ⇒ *list*

In Rectangular complex format mode:

eigVl() Catalog >

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.



$\begin{bmatrix} ^{-}1 & 2 & 5 \\ 3 & ^{-}6 & 9 \\ 2 & ^{-}5 & 7 \end{bmatrix} \to m1$ $\quad \begin{bmatrix} ^{-}1 & 2 & 5 \\ 3 & ^{-}6 & 9 \\ 2 & ^{-}5 & 7 \end{bmatrix}$

eigVl(*m1*)

{⁻4.40941,2.20471+0.763006·*i*,2.20471−0.▸

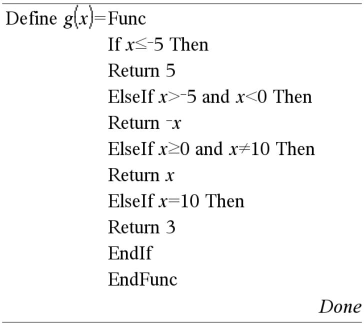
To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

Else See If, page 86.

ElseIf Catalog >

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)=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
                                    Done
```

EndFor See For, page 75.

EndFunc See Func, page 78.

EndIf See If, page 86.

EndLoop — See Loop, page 109.

EndPrgm — See Prgm, page 136.

EndTry — See Try, page 189.

EndWhile — See While, page 199.

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*

Uses the Euler method to solve the system

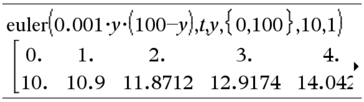
$$\frac{d\ depVar}{d\ Var} = Expr(Var, depVar)$$

with $depVar(Var0)=depVar0$ on the interval $[Var0,VarMax]$. 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*).

Differential equation:
y'=0.001*y*(100-y) and y(0)=10

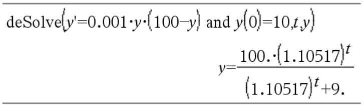


$$\mathrm{euler}\left(0.001\cdot y\cdot(100-y),t,y,\{0,100\},10,1\right)$$

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 10.9 & 11.8712 & 12.9174 & 14.042 \end{bmatrix}$$

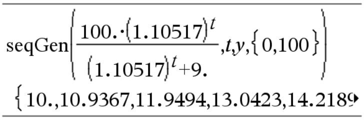
To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

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



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

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



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

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

euler () Catalog >

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.

{*Var0*, *VarMax*} is a two-element list that tells the function to integrate from *Var0* to *VarMax*.

ListOfDepVars0 is a list of initial values for dependent variables.

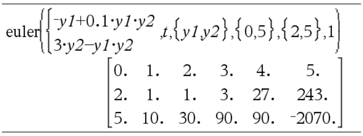
VarStep is a nonzero number such that **sign (***VarStep***)** = **sign(***VarMax*-*Var0***)** and solutions are returned at *Var0*+*i*•*VarStep* for all *i*=0,1,2,… such that *Var0*+*i*•*VarStep* is in [*var0*,*VarMax*] (there may not be a solution value at *VarMax*).

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 *VarStep* / *eulerStep*.

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 $y2(0)=5$



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

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. & 5. \\ 2. & 1. & 1. & 3. & 27. & 243. \\ 5. & 10. & 30. & 90. & 90. & {}^-2070. \end{bmatrix}$$

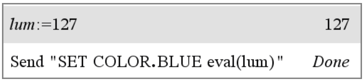
eval () Hub Menu

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.

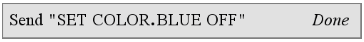
The argument *Expr* must simplify to a real number.

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



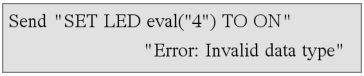
```
lum:=127                                   127
Send "SET COLOR.BLUE eval(lum)"           Done
```

Reset the blue element to OFF.



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

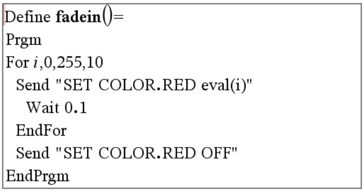
eval() argument must simplify to a real number.



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

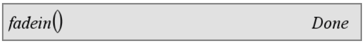
eval () Hub Menu

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.

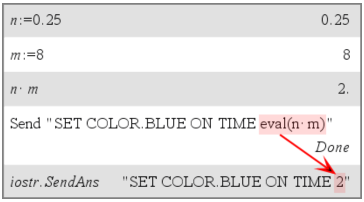


fadein()	*Done*

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 80), **GetStr** (page 83), and **Send** (page 157).



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

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).

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

Exit Catalog >

Exit

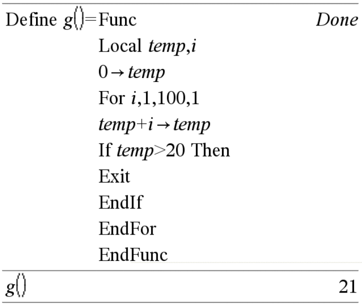
Exits the current **For**, **While**, or **Loop** block.

Function listing:

Exit **Catalog >**

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.



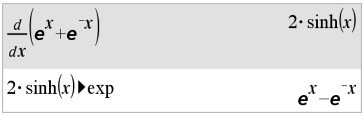
```
Define g()=Func                    Done
           Local temp,i
           0→temp
           For i,1,100,1
           temp+i→temp
           If temp>20 Then
           Exit
           EndIf
           EndFor
           EndFunc
g()                                  21
```

▶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`.



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

$2\cdot\sinh(x)$▶exp $e^{x}-e^{-x}$

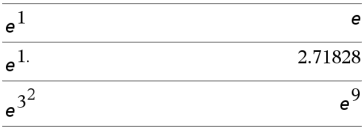
exp() e^x **key**

exp(*Expr1***)** ⇒ *expression*

Returns ***e*** raised to the *Expr1* power.

Note: See also ***e*** exponent template, page 6.

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



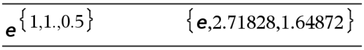
e^{1} e

$e^{1.}$ 2.71828

$e^{3^{2}}$ e^{9}

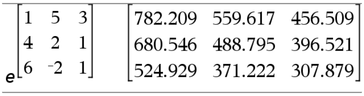
exp(*List1***)** ⇒ *list*

Returns ***e*** raised to the power of each element in *List1*.



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

exp(*squareMatrix1***)** ⇒ *squareMatrix*



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

exp() e^x key

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.

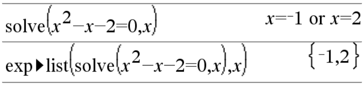
exp▶list() Catalog >

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(...)`**.



$\text{solve}\left(x^2-x-2=0,x\right)$ $x=-1 \text{ or } x=2$

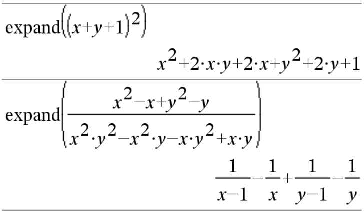
$\text{exp▶list}\left(\text{solve}\left(x^2-x-2=0,x\right),x\right)$ $\{-1,2\}$

expand() Catalog >

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.



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

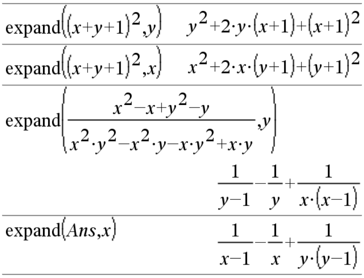
$x^2+2\cdot x\cdot y+2\cdot x+y^2+2\cdot y+1$

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

$\frac{1}{x-1}-\frac{1}{x}+\frac{1}{y-1}-\frac{1}{y}$

exprand() Catalog >

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.



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

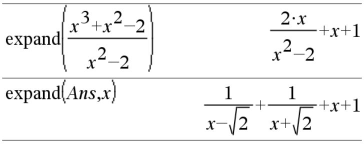
$\text{expand}\left((x+y+1)^2,x\right) \quad x^2+2\cdot x\cdot(y+1)+(y+1)^2$

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

$\frac{1}{y-1}-\frac{1}{y}+\frac{1}{x\cdot(x-1)}$

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

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



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

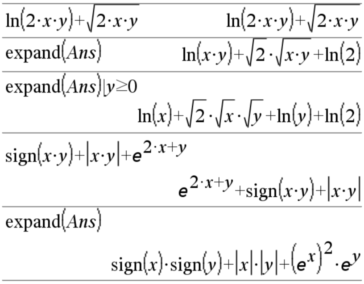
$\text{expand}(\mathit{Ans},x) \quad \frac{1}{x-\sqrt{2}}+\frac{1}{x+\sqrt{2}}+x+1$

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.

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*.



$\ln(2\cdot x\cdot y)+\sqrt{2\cdot x\cdot y} \quad \ln(2\cdot x\cdot y)+\sqrt{2\cdot x\cdot y}$

$\text{expand}(\mathit{Ans}) \quad \ln(x\cdot y)+\sqrt{2}\cdot\sqrt{x\cdot y}+\ln(2)$

$\text{expand}(\mathit{Ans})|y\geq 0$

$\ln(x)+\sqrt{2}\cdot\sqrt{x}\cdot\sqrt{y}+\ln(y)+\ln(2)$

$\text{sign}(x\cdot y)+|x\cdot y|+e^{2\cdot x+y}$

$e^{2\cdot x+y}+\text{sign}(x\cdot y)+|x\cdot y|$

$\text{expand}(\mathit{Ans})$

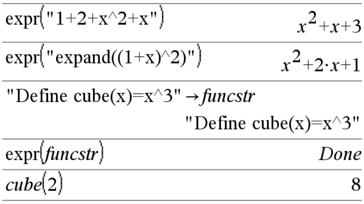
$\text{sign}(x)\cdot\text{sign}(y)+|x|\cdot|y|+\left(e^x\right)^2\cdot e^y$

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

expr() Catalog >

expr(*String***)** ⇒ *expression*

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



$\text{expr}(\text{"1+2+x^2+x"}) \quad x^2+x+3$

$\text{expr}(\text{"expand((1+x)^2)"}) \quad x^2+2\cdot x+1$

"Define cube(x)=x^3" → *funcstr*

"Define cube(x)=x^3"

$\text{expr}(\mathit{funcstr}) \quad \mathit{Done}$

$\mathit{cube}(2) \quad 8$

ExpReg Catalog >

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

Computes the exponential regression $y = a \bullet (b)^x$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: $a \bullet (b)^x$
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 *Freq*, *Category List*, and *Include Categories*

Output variable	Description
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*

F

factor() Catalog >

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

factor(*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 without introducing new non-real subexpressions. This alternative is appropriate if you want factorization with respect to more than one variable.

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

$\text{factor}\left(x^{2}+1\right)$ $\quad x^{2}+1$

$\text{factor}\left(x^{2}-4\right)$ $\quad (x-2)\cdot(x+2)$

$\text{factor}\left(x^{2}-3\right)$ $\quad x^{2}-3$

$\text{factor}\left(x^{2}-a\right)$ $\quad x^{2}-a$

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.

$\text{factor}\left(a^{3}\cdot x^{2}-a\cdot x^{2}-a^{3}+a,x\right)$ $\quad a\cdot\left(a^{2}-1\right)\cdot(x-1)\cdot(x+1)$

$\text{factor}\left(x^{2}-3,x\right)$ $\quad \left(x+\sqrt{3}\right)\cdot\left(x-\sqrt{3}\right)$

$\text{factor}\left(x^{2}-a,x\right)$ $\quad \left(x+\sqrt{a}\right)\cdot\left(x-\sqrt{a}\right)$

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.

factor() Catalog >

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.

$\text{factor}\left(x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3\right)$
$x^5+4\cdot x^4+5\cdot x^3-6\cdot x-3$

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

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.

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 ⌂on key and press enter 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() Catalog >

FCdf (*lowBound***,***upBound***,***dfNumer***,***dfDenom***)** ⇒

FCdf()

Catalog >

number if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

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($X \le upBound$), set *lowBound* = 0.

Fill

Catalog >

Fill *Expr, matrixVar* ⇒ *matrix*

Replaces each element in variable *matrixVar* with *Expr*.

matrixVar must already exist.

$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \to amatrix$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$
Fill 1.01,*amatrix*	*Done*
amatrix	$\begin{bmatrix}1.01 & 1.01\\1.01 & 1.01\end{bmatrix}$

Fill *Expr, listVar* ⇒ *list*

Replaces each element in variable *listVar* with *Expr*.

listVar must already exist.

$\{1,2,3,4,5\} \to alist$	$\{1,2,3,4,5\}$
Fill 1.01,*alist*	*Done*
alist	$\{1.01,1.01,1.01,1.01,1.01\}$

FiveNumSummary

Catalog >

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 175.)

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.

FiveNumSummary Catalog >

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 234.

Output variable	**Description**
stat.MinX	Minimum of x values.
stat.Q_1X	1st Quartile of x.
stat.MedianX	Median of x.
stat.Q_3X	3rd Quartile of x.
stat.MaxX	Maximum of x values.

floor() Catalog >

floor(*Expr1***)** ⇒ *integer*

Returns the greatest integer that is ≤ the argument. This function is identical to **int()**.

The argument can be a real or a complex number.

$\text{floor}\left({}^{-}2.14\right)$ $\quad {}^{-}3.$

floor(*List1***)** ⇒ *list*
floor(*Matrix1***)** ⇒ *matrix*

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

$\text{floor}\left(\left\{\frac{3}{2},0,{}^{-}5.3\right\}\right)$ $\quad \{1,0,{}^{-}6.\}$

$\text{floor}\left(\begin{bmatrix}1.2 & 3.4\\ 2.5 & 4.8\end{bmatrix}\right)$ $\quad \begin{bmatrix}1. & 3.\\ 2. & 4.\end{bmatrix}$

Note: See also **ceiling()** and **int()**.

fMax() Catalog >

fMax(*Expr, Var***)** ⇒ *Boolean expression*
fMax(*Expr, Var,lowBound***)**

fMax(*Expr, Var,lowBound,upBound***)**

$\text{fMax}\left(1-(x-a)^2-(x-b)^2,x\right)$ $\quad x=\frac{a+b}{2}$

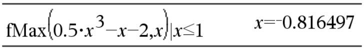
$\text{fMax}\left(.5\cdot x^3-x-2,x\right)$ $\quad x=\infty$

fMax() Catalog >

fMax(*Expr***,** *Var***) |** *lowBound≤Var≤upBound*

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.



$\mathrm{fMax}\left(0.5\cdot x^{3}-x-2,x\right)|x\le 1 \quad x={}^{-}0.816497$

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()**.

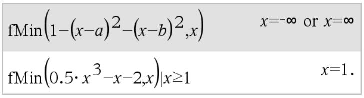
fMin() Catalog >

fMin(*Expr, Var***)** ⇒ *Boolean expression*

fMin(*Expr***,** *Var***,***lowBound***)**

fMin(*Expr***,** *Var***,***lowBound***,***upBound***)**

fMin(*Expr***,** *Var***) |** *lowBound≤Var≤upBound*



$\mathrm{fMin}\left(1-(x-a)^{2}-(x-b)^{2},x\right) \quad x=-\infty \text{ or } x=\infty$

$\mathrm{fMin}\left(0.5\cdot x^{3}-x-2,x\right)|x\ge 1 \quad x=1.$

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()**.

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 tempsum,step,i
           0→tempsum
           1→step
           For i,1,100,step
           tempsum+i→tempsum
           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.

format(1.234567,"f3")	"1.235"
format(1.234567,"s2")	"1.23ᴇ0"
format(1.234567,"e3")	"1.235ᴇ0"
format(1.234567,"g3")	"1.235"
format(1234.567,"g3")	"1,234.567"
format(1.234567,"g3,r:")	"1:235"

format() Catalog >

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.

fPart() Catalog >

fPart(*Expr1***)** ⇒ *expression*
fPart(*List1***)** ⇒ *list*
fPart(*Matrix1***)** ⇒ *matrix*

fPart(-1.234)	-0.234
fPart({1,-2.3,7.003})	{0,-0.3,0.003}

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.

FPdf() Catalog >

FPdf(*XVal***,***dfNumer***,***dfDenom***)** ⇒ *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() Catalog >

freqTable▶list(*List1***,***freqIntegerList***)** ⇒ *list*

freqTable▶list({1,2,3,4},{1,4,3,1})	{1,2,2,2,2,3,3,3,4}
freqTable▶list({1,2,3,4},{1,4,0,1})	{1,2,2,2,2,4}

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.

freqTable►list() Catalog >

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 234.

frequency() Catalog >

frequency(*List1,binsList***)** ⇒ *list*

Returns a list containing counts of the elements in *List1*. The counts are based on ranges (bins) that you define in *binsList*.

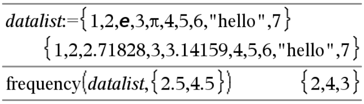
If *binsList* is {b(1), b(2), …, b(n)}, the specified ranges are {**?**≤b(1), b(1)<**?**≤b(2),…,b(n-1)<**?**≤b(n), b(n)>**?**}. The resulting list is one element longer than *binsList*.

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, **?**≤b(1)), countIf(list, b(1)<**?**≤b(2)), …, countIf(list, b(n-1)<**?**≤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 234.

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

Note: See also **countIf()**, page 39.



$datalist:=\{1,2,\mathbf{e},3,\pi,4,5,6,\text{"hello"},7\}$

$\{1,2,2.71828,3,3.14159,4,5,6,\text{"hello"},7\}$

$\text{frequency}(datalist,\{2.5,4.5\})$ $\{2,4,3\}$

Explanation of result:

2 elements from *Datalist* are ≤2.5

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 — Catalog >

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 175.)

For H_a: σ1 > σ2, set *Hypoth*>0
For H_a: σ1 ≠ σ2 (default), set *Hypoth* =0
For H_a: σ1 < σ2, set *Hypoth*<0

For information on the effect of empty elements in a list, see *Empty (Void) Elements*, page 234.

Output variable	Description
stat.F	Calculated F statistic for the data sequence
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 *List 1* and *List 2*
stat.x1_bar stat.x2_bar	Sample means of the data sequences in *List 1* and *List 2*
stat.n1, stat.n2	Size of the samples

Func — Catalog >

Func
    Block
EndFunc

Template for creating a user-defined function.

Define a piecewise function:

Func

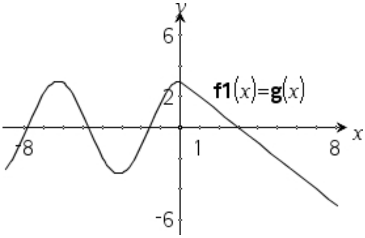
Catalog >

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 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 >

gcd(*Number1, Number2***)** ⇒ *expression*

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.

In Auto or Approximate mode, the **gcd** of fractional floating-point numbers is 1.0.

$\text{gcd}(18,33)$ $\qquad 3$

gcd(*List1, List2***)** ⇒ *list*

Returns the greatest common divisors of the corresponding elements in *List1* and *List2*.

$\text{gcd}(\{12,14,16\},\{9,7,5\})$ $\qquad \{3,7,1\}$

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

Returns the greatest common divisors of the corresponding elements in *Matrix1* and *Matrix2*.

$\text{gcd}\left(\begin{bmatrix}2 & 4\\6 & 8\end{bmatrix},\begin{bmatrix}4 & 8\\12 & 16\end{bmatrix}\right)$ $\qquad \begin{bmatrix}2 & 4\\6 & 8\end{bmatrix}$

geomCdf()

Catalog >

geomCdf(*p,lowBound,upBound***)** ⇒ *number*

geomCdf() Catalog >

if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

geomCdf(*p*,*upBound***)**for P(1≤X≤*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 ≤ *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*]

Programming command: Retrieves a value from a connected TI-Innovator™ Hub and assigns the value to variable *var*.

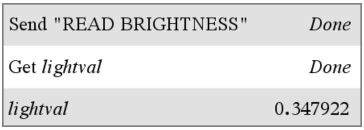
The value must be requested:

- In advance, through a **Send "READ ..."** command.

  — or —

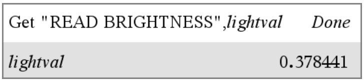
- By embedding a **"READ ..."** request as the optional *promptString* argument. This method lets you use a single command to request the value and retrieve it.

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*.



```
Send "READ BRIGHTNESS"          Done
Get lightval                    Done
lightval                    0.347922
```

Embed the READ request within the **Get** command.



```
Get "READ BRIGHTNESS",lightval   Done
lightval                     0.378441
```

Get **Hub Menu**

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 83 and **Send**, page 157.

getDenom() **Catalog >** 📖

getDenom(*Expr1***)** ⇒ *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
$\text{getDenom}\left(\frac{1}{x}+\frac{y^2+y}{y^2}\right)$	$x\cdot y$

getLangInfo() **Catalog >** 📖

getLangInfo() ⇒ *string*

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.

getLangInfo()	"en"

getLangInfo() Catalog >

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 >

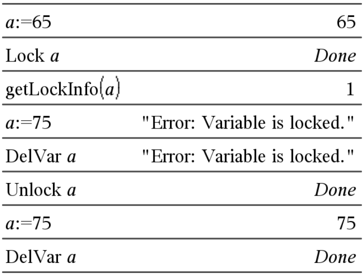
getLockInfo(*Var***)** ⇒ *value*

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 105, and **unLock**, page 195.



```
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
```

getMode() Catalog >

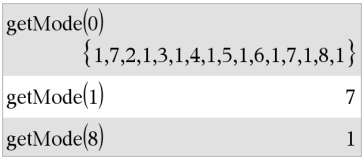
getMode(*ModeNameInteger***)** ⇒ *value*

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.



```
getMode(0)
          {1,7,2,1,3,1,4,1,5,1,6,1,7,1,8,1}
getMode(1)                                7
getMode(8)                                1
```

getMode() Catalog >

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 160.

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

getNum() Catalog >

getNum(*Expr1***)** ⇒ *expression*

Transforms the argument into an expression having a reduced common denominator, and then returns its numerator.

$\mathrm{getNum}\left(\frac{x+2}{y-3}\right)$	$x+2$
$\mathrm{getNum}\left(\frac{2}{7}\right)$	2
$\mathrm{getNum}\left(\frac{1}{x}+\frac{1}{y}\right)$	$x+y$

GetStr Hub Menu

GetStr [*promptString*,] *var*[**,** *statusVar*]

GetStr [*promptString*,] *func***(***arg1***,** *...argn***)**

For examples, see **Get**.

GetStr **Hub Menu**

[**,** *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 ("").

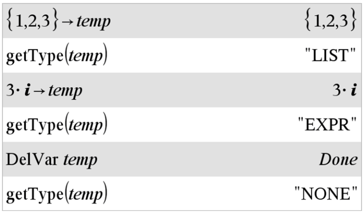
Note: See also **Get**, page 80 and **Send**, page 157.

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}→temp                          {1,2,3}
getType(temp)                          "LIST"
3·i→temp                                  3·i
getType(temp)                          "EXPR"
DelVar temp                              Done
getType(temp)                          "NONE"
```

getVarInfo() **Catalog >** 📖

getVarInfo() ⇒ *matrix* or *string*

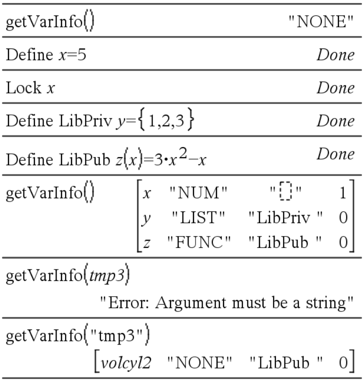
getVarInfo(*LibNameString***)** ⇒ *matrix* or *string*

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.



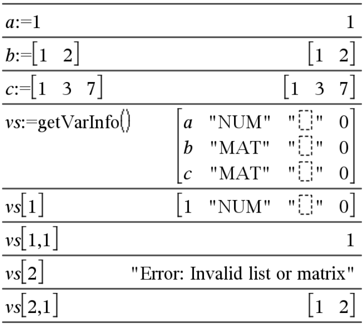
```
getVarInfo()                                        "NONE"
Define x=5                                            Done
Lock x                                                Done
Define LibPriv y={1,2,3}                              Done
Define LibPub z(x)=3·x^2-x                            Done
getVarInfo()        [x  "NUM"    "⬚"         1]
                    [y  "LIST"   "LibPriv "  0]
                    [z  "FUNC"   "LibPub "   0]
getVarInfo(tmp3)
                   "Error: Argument must be a string"
getVarInfo("tmp3")
               [volcyl2  "NONE"  "LibPub "  0]
```

getVarInfo() Catalog >

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) revaluates 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.



$a:=1$ $\quad 1$

$b:=\begin{bmatrix}1 & 2\end{bmatrix}$ $\quad \begin{bmatrix}1 & 2\end{bmatrix}$

$c:=\begin{bmatrix}1 & 3 & 7\end{bmatrix}$ $\quad \begin{bmatrix}1 & 3 & 7\end{bmatrix}$

$vs:=\text{getVarInfo}()$ $\quad \begin{bmatrix}a & \text{"NUM"} & \text{"⬚"} & 0\\ b & \text{"MAT"} & \text{"⬚"} & 0\\ c & \text{"MAT"} & \text{"⬚"} & 0\end{bmatrix}$

$vs[1]$ $\quad \begin{bmatrix}1 & \text{"NUM"} & \text{"⬚"} & 0\end{bmatrix}$

$vs[1,1]$ $\quad 1$

$vs[2]$ $\quad$ "Error: Invalid list or matrix"

$vs[2,1]$ $\quad \begin{bmatrix}1 & 2\end{bmatrix}$

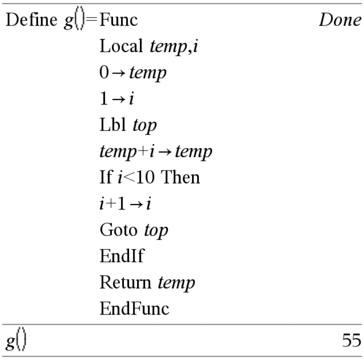
Goto Catalog >

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                    Done
           Local temp,i
           0→temp
           1→i
           Lbl top
           temp+i→temp
           If i<10 Then
           i+1→i
           Goto top
           EndIf
           Return temp
           EndFunc
g()                                  55
```

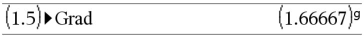
►Grad Catalog >

Expr1►**Grad** ⇒ *expression*

Converts *Expr1* to gradian angle measure.

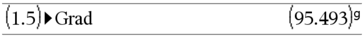
Note: You can insert this operator from the computer keyboard by typing `@>Grad`.

In Degree angle mode:



(1.5)►Grad $\quad (1.66667)^{g}$

In Radian angle mode:



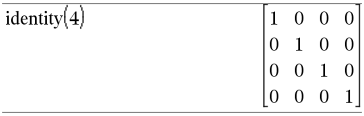
(1.5)►Grad $\quad (95.493)^{g}$

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

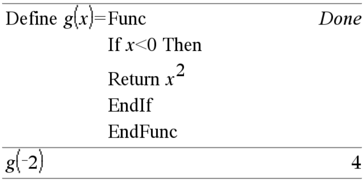
If *BooleanExpr* **Then**
Block
EndIf

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.

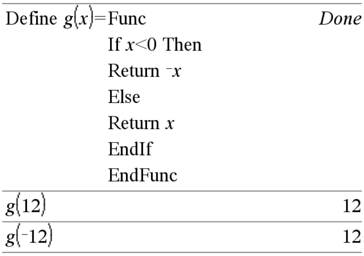


```
Define g(x)=Func                    Done
            If x<0 Then
            Return x^2
            EndIf
            EndFunc
g(⁻2)                                  4
```

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*.



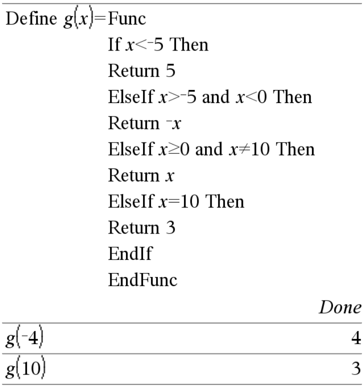
```
Define g(x)=Func                    Done
            If x<0 Then
            Return ⁻x
            Else
            Return x
            EndIf
            EndFunc
g(12)                                 12
g(⁻12)                                12
```

If **Catalog >**

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.



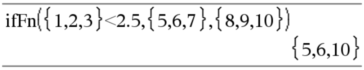
```
Define g(x)=Func
            If x<⁻5 Then
            Return 5
            ElseIf x>⁻5 and x<0 Then
            Return ⁻x
            ElseIf x≥0 and x≠10 Then
            Return x
            ElseIf x=10 Then
            Return 3
            EndIf
            EndFunc
                                        Done
g(⁻4)                                      4
g(10)                                      3
```

ifFn() **Catalog >**

ifFn(*BooleanExpr***,***Value_If_true* [**,***Value_If_false* [**,***Value_If_unknown*]]**)** ⇒ *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*.



```
ifFn({1,2,3}<2.5,{5,6,7},{8,9,10})
                                {5,6,10}
```

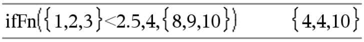
Test value of **1** is less than 2.5, so its corresponding

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.



```
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.

ifFn()

Catalog >

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).

$\text{ifFn}(\{1,2,3\}<2.5,\{5,6,7\}) \qquad \{5,6,\text{undef}\}$

Value_If_false is not specified. Undef is used.

$\text{ifFn}(\{2,\text{"a"}\}<2.5,\{6,7\},\{9,10\},\text{"err"}) \qquad \{6,\text{"err"}\}$

One element selected from *Value_If_true*. One element selected from *Value_If_unknown*.

imag()

Catalog >

imag(*Expr1***)** ⇒ *expression*

Returns the imaginary part of the argument.

Note: All undefined variables are treated as real variables. See also real(), page 145

$\text{imag}(1+2\cdot i) \qquad 2$

$\text{imag}(z) \qquad 0$

$\text{imag}(x+i\cdot y) \qquad y$

imag(*List1***)** ⇒ *list*

Returns a list of the imaginary parts of the elements.

$\text{imag}(\{^{-}3,4-i,i\}) \qquad \{0,^{-}1,1\}$

imag(*Matrix1***)** ⇒ *matrix*

Returns a matrix of the imaginary parts of the elements.

$\text{imag}\left(\begin{bmatrix} a & b \\ i\cdot c & i\cdot d \end{bmatrix}\right) \qquad \begin{bmatrix} 0 & 0 \\ c & d \end{bmatrix}$

impDif()

Catalog >

impDif(*Equation***,** *Var***,** *dependVar*[**,***Ord*]**)** ⇒ *expression*

where the order *Ord* defaults to 1.

Computes the implicit derivative for equations in which one variable is defined implicitly in terms of another.

$\text{impDif}(x^2+y^2=100,x,y) \qquad \frac{^{-}x}{y}$

Indirection

See #(), page 224.

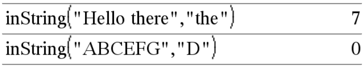
inString() Catalog >

inString(*srcString***,** *subString*[**,** *Start*]**)** ⇒ *integer*

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.



```
inString("Hello there","the")        7
inString("ABCEFG","D")               0
```

int() Catalog >

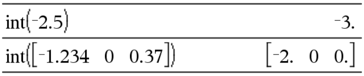
int(*Expr***)** ⇒ *integer*

int(*List1***)** ⇒ *list*
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.



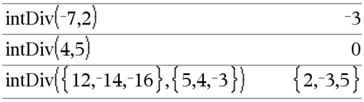
```
int(⁻2.5)                              ⁻3.
int([⁻1.234  0  0.37])          [⁻2.  0  0.]
```

intDiv() Catalog >

intDiv(*Number1***,** *Number2***)** ⇒ *integer*
intDiv(*List1***,** *List2***)** ⇒ *list*
intDiv(*Matrix1***,** *Matrix2***)** ⇒ *matrix*

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.



```
intDiv(⁻7,2)                               ⁻3
intDiv(4,5)                                 0
intDiv({12,⁻14,⁻16},{5,4,⁻3})      {2,⁻3,5}
```

integral

See ∫(), page 219.

interpolate ()

Catalog > 📖

interpolate(*xValue***,** *xList***,** *yList***,** *yPrimeList***)** ⇒ *list*

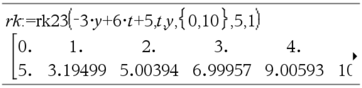
This function does the following:

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.

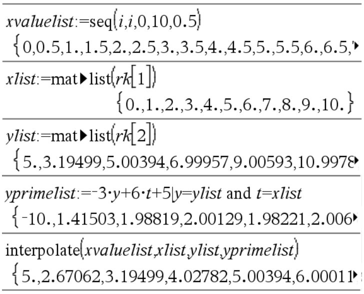
Differential equation:
y'=-3•*y*+6•*t*+5 and *y*(0)=5



```
rk:=rk23(-3·y+6·t+5,t,y,{0,10},5,1)
 [0.   1.        2.        3.        4.        ▸
 [5.   3.19499   5.00394   6.99957   9.00593   1(
```

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

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,▸
xlist:=mat▸list(rk[1])
                    {0.,1.,2.,3.,4.,5.,6.,7.,8.,9.,10.}
ylist:=mat▸list(rk[2])
  {5.,3.19499,5.00394,6.99957,9.00593,10.9978▸
yprimelist:=-3·y+6·t+5|y=ylist and t=xlist
  {-10.,1.41503,1.98819,2.00129,1.98221,2.006▸
interpolate(xvaluelist,xlist,ylist,yprimelist)
  {5.,2.67062,3.19499,4.02782,5.00394,6.00011▸
```

invχ2()

Catalog > 📖

invχ2(*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***)**

invF() Catalog >

computes the Inverse cumulative F distribution function specified by *dfNumer* and *dfDenom* for a given *Area* under the curve.

invBinom() Catalog >

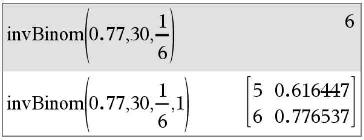
invBinom (*CumulativeProb***,***NumTrials***,***Prob***,** *OutputForm***)**⇒ *scalar* or *matrix*

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 >

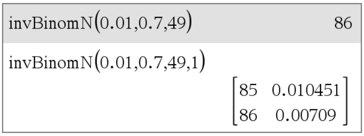
invBinomN(*CumulativeProb***,***Prob***,** *NumSuccess***,***OutputForm***)**⇒ *scalar* or *matrix*

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]]**)**

invNorm() Catalog >

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***)**

Computes the inverse cumulative student-t probability function specified by degree of freedom, *df* for a given *Area* under the curve.

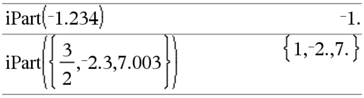
iPart() Catalog >

iPart(*Number***)** ⇒ *integer*
iPart(*List1***)** ⇒ *list*
iPart(*Matrix1***)** ⇒ *matrix*

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.



$\text{iPart}(^{-}1.234)$ $\qquad ^{-}1.$

$\text{iPart}\left(\left\{\frac{3}{2},^{-}2.3,7.003\right\}\right)$ $\qquad \{1,^{-}2.,7.\}$

irr() Catalog >

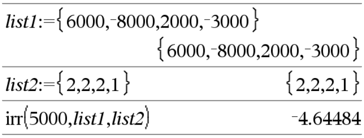
irr(*CF0*,*CFList* [,*CFFreq*]**)** ⇒ *value*

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.



$list1{:=}\{6000,^{-}8000,2000,^{-}3000\}$ $\qquad \{6000,^{-}8000,2000,^{-}3000\}$

$list2{:=}\{2,2,2,1\}$ $\qquad \{2,2,2,1\}$

$\text{irr}(5000,list1,list2)$ $\qquad ^{-}4.64484$

irr() Catalog >

Note: See also **mirr()**, page 114.

isPrime() Catalog >

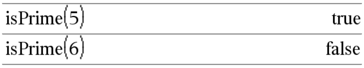
isPrime(*Number***)** ⇒ *Boolean constant expression*

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.

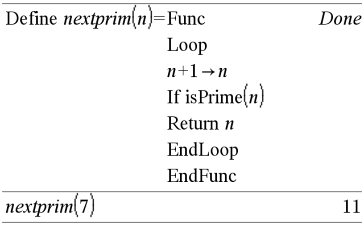
If you merely want to determine if *Number* is prime, use **isPrime()** instead of **factor()**. It is much faster, particularly if *Number* is not prime and has a second-largest factor that exceeds about five digits.

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



```
isPrime(5)                     true
isPrime(6)                     false
```

Function to find the next prime after a specified number:



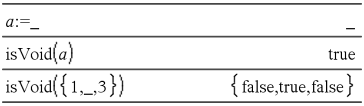
```
Define nextprim(n)=Func               Done
                   Loop
                   n+1→n
                   If isPrime(n)
                   Return n
                   EndLoop
                   EndFunc
nextprim(7)                             11
```

isVoid() Catalog >

isVoid(*Var***)** ⇒ *Boolean constant expression*
isVoid(*Expr***)** ⇒ *Boolean constant expression*
isVoid(*List***)** ⇒ *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 234.



```
a:=_                                          _
isVoid(a)                                  true
isVoid({1,_,3})            {false,true,false}
```

L

Lbl — Catalog >

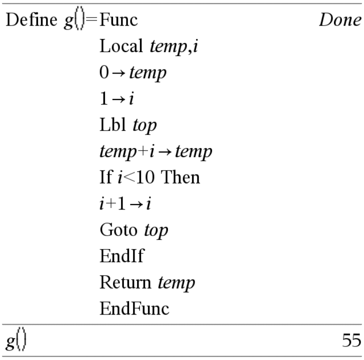
Lbl *labelName*

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.



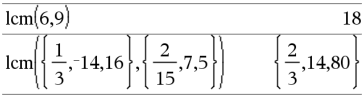
```
Define g()=Func                        Done
           Local temp,i
           0→temp
           1→i
           Lbl top
           temp+i→temp
           If i<10 Then
           i+1→i
           Goto top
           EndIf
           Return temp
           EndFunc
g()                                      55
```

lcm() — Catalog >

lcm(*Number1*, *Number2***)** ⇒ *expression*
lcm(*List1*, *List2***)** ⇒ *list*
lcm(*Matrix1*, *Matrix2***)** ⇒ *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

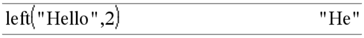
$\text{lcm}\left(\left\{\frac{1}{3},-14,16\right\},\left\{\frac{2}{15},7,5\right\}\right)$ → $\left\{\frac{2}{3},14,80\right\}$

left() — Catalog >

left(*sourceString*[, *Num*]**)** ⇒ *string*

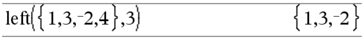
Returns the leftmost *Num* characters contained in character string *sourceString*.

If you omit *Num*, returns all of *sourceString*.



left("Hello",2) → "He"

left(*List1*[, *Num*]**)** ⇒ *list*



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.

$\text{left}(x<3)$	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 33.
To delete a variable group, see **DelVar** on page 53.

This example assumes a properly stored and refreshed library document named **linalg2** that contains objects defined as *clearmat*, *gauss1*, and *gauss2*.

$\text{getVarInfo}(\text{"linalg2"})$

$$\begin{bmatrix} clearmat & \text{"FUNC"} & \text{"LibPub "} \\ gauss1 & \text{"PRGM"} & \text{"LibPriv "} \\ gauss2 & \text{"FUNC"} & \text{"LibPub "} \end{bmatrix}$$

$\text{libShortcut}(\text{"linalg2"},\text{"la"})$

$$\{la.clearmat,la.gauss2\}$$

$\text{libShortcut}(\text{"linalg2"},\text{"la"},1)$

$$\{la.clearmat,la.gauss1,la.gauss2\}$$

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 10.

Direction: negative=from left, positive=from right, otherwise=both. (If omitted, *Direction* defaults to both.)

$\lim\limits_{x\to 5}(2\cdot x+3)$	13
$\lim\limits_{x\to 0^+}\left(\frac{1}{x}\right)$	∞
$\lim\limits_{x\to 0}\left(\frac{\sin(x)}{x}\right)$	1
$\lim\limits_{h\to 0}\left(\frac{\sin(x+h)-\sin(x)}{h}\right)$	$\cos(x)$
$\lim\limits_{n\to\infty}\left(\left(1+\frac{1}{n}\right)^n\right)$	e

limit() or lim()

Catalog >

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.

$\lim_{x \to \infty}\left(a^{x}\right)$	undef
$\lim_{x \to \infty}\left(a^{x}\right) \vert a>1$	∞
$\lim_{x \to \infty}\left(a^{x}\right) \vert a>0 \text{ and } a<1$	0

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.

LinRegBx

Catalog >

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

Computes the linear regression y = a+b•x on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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

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

LinRegBx **Catalog >**

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 234.

Output variable	**Description**
stat.RegEqn	Regression Equation: a+b•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 *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*

LinRegMx **Catalog >**

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

Computes the linear regression y = m•x+b on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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

LinRegMx Catalog >

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 234.

Output variable	**Description**
stat.RegEqn	Regression Equation: y = m•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 Catalog >

LinRegtIntervals *X*,*Y*[,*F*[,**0**[,*CLev*]]]

For Slope. Computes a level C confidence interval for the slope.

LinRegtIntervals *X*,*Y*[,*F*[,**1**,*Xval*[,*CLev*]]]

LinRegtIntervals Catalog >

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 175.)

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 $\geq$ 0.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	**Description**
stat.RegEqn	Regression Equation: a+b•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.ŷ	a + b•XVal

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*=α+βx. It tests the null hypothesis H_0:β=0 (equivalently, ρ=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:β=ρ=0) will be tested.

For H_a: β≠0 and ρ≠0 (default), set *Hypoth*=0
For H_a: β<0 and ρ<0, set *Hypoth*<0
For H_a: β>0 and ρ>0, set *Hypoth*>0

A summary of results is stored in the *stat.results* variable. (See page 175.)

LinRegtTest Catalog >

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	Description
stat.RegEqn	Regression equation: a + b•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() Catalog >

linSolve(*SystemOfLinearEqns*, *Var1*, *Var2*, ...**)** ⇒ *list*

linSolve(*LinearEqn1* **and** *LinearEqn2* **and** ..., *Var1*, *Var2*, ...**)** ⇒ *list*

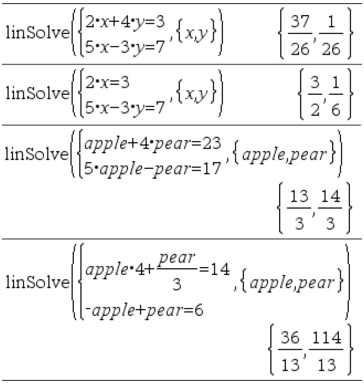
linSolve({*LinearEqn1*, *LinearEqn2*, ...**}**, *Var1*, *Var2*, ...**)** ⇒ *list*

linSolve(*SystemOfLinearEqns*, **{***Var1*, *Var2*, ...**})** ⇒ *list*

linSolve(*LinearEqn1* **and** *LinearEqn2* **and** ..., **{***Var1*, *Var2*, ...**})** ⇒ *list*

linSolve({*LinearEqn1*, *LinearEgn2*, ...**}**, **{***Var1*, *Var2*, ...**})** ⇒ *list*

Returns a list of solutions for the variables *Var1*, *Var2*, ...



$$\text{linSolve}\left(\begin{cases}2\cdot x+4\cdot y=3\\5\cdot x-3\cdot y=7\end{cases},\{x,y\}\right) \quad \left\{\frac{37}{26},\frac{1}{26}\right\}$$

$$\text{linSolve}\left(\begin{cases}2\cdot x=3\\5\cdot x-3\cdot y=7\end{cases},\{x,y\}\right) \quad \left\{\frac{3}{2},\frac{1}{6}\right\}$$

$$\text{linSolve}\left(\begin{cases}apple+4\cdot pear=23\\5\cdot apple-pear=17\end{cases},\{apple,pear\}\right) \quad \left\{\frac{13}{3},\frac{14}{3}\right\}$$

$$\text{linSolve}\left(\begin{cases}apple\cdot 4+\frac{pear}{3}=14\\-apple+pear=6\end{cases},\{apple,pear\}\right) \quad \left\{\frac{36}{13},\frac{114}{13}\right\}$$

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 >

ΔList(*List1***)** ⇒ *list*

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*.

$\Delta\mathrm{List}(\{20,30,45,70\})$ $\{10,15,25\}$

list►mat() Catalog >

list►mat(*List* [**,** *elementsPerRow*]**)** ⇒ *matrix*

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(...)`**.

$\mathrm{list}\blacktriangleright\mathrm{mat}(\{1,2,3\})$ $\begin{bmatrix}1 & 2 & 3\end{bmatrix}$

$\mathrm{list}\blacktriangleright\mathrm{mat}(\{1,2,3,4,5\},2)$ $\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 0\end{bmatrix}$

►ln Catalog >

*Expr***►ln** ⇒ *expression*

Causes the input *Expr* to be converted to an expression containing only natural logs (ln).

$(\log_{10}(x))\blacktriangleright\ln$ $\dfrac{\ln(x)}{\ln(10)}$

►ln Catalog >

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

ln() ctrl eˣ keys

ln(*Expr1***)** ⇒ *expression*

ln(*List1***)** ⇒ *list*

Returns the natural logarithm of the argument.

For a list, returns the natural logarithms of the elements.

$\ln(2.)$ 0.693147

If complex format mode is Real:

$\ln(\{^-3,1.2,5\})$
"Error: Non−real calculation"

If complex format mode is Rectangular:

$\ln(\{^-3,1.2,5\})$ $\{\ln(3)+\pi\cdot \boldsymbol{i},0.182322,\ln(5)\}$

ln(*squareMatrix1***)** ⇒ *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.

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

In Radian angle mode and Rectangular complex format:

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

$$\begin{bmatrix}1.83145+1.73485\cdot \boldsymbol{i} & 0.009193-1.49086\\ 0.448761-0.725533\cdot \boldsymbol{i} & 1.06491+0.623491▸\\ ^-0.266891-2.08316\cdot \boldsymbol{i} & 1.12436+1.79018\cdot\end{bmatrix}$$

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

LnReg Catalog >

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

Computes the logarithmic regression y = a+b•ln(x) on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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

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

LnReg Catalog >

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: a+b•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 *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*

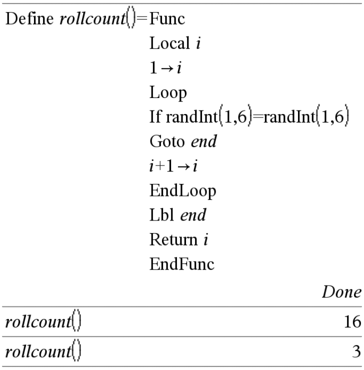
Local **Catalog >**

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()=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
```

Lock **Catalog >**

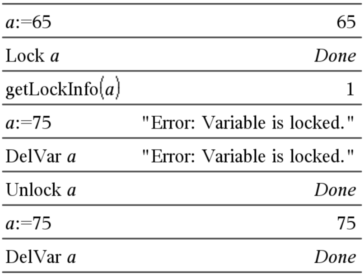
Lock*Var1*[**,** *Var2*] [**,** *Var3*] ...
Lock*Var***.**

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 195, and **getLockInfo()**, page 82.



```
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 10ˣ keys

log(*Expr1*[**,***Expr2*]**)** ⇒ *expression*

log(*List1*[**,***Expr2*]**)** ⇒ *list*

Returns the base-*Expr2* logarithm of the first argument.

Note: See also **Log template**, page 6.

For a list, returns the base-*Expr2* logarithm of the elements.

If the second argument is omitted, 10 is used as the base.

$$\log_{10}(2.) \qquad 0.30103$$

$$\log_{4}(2.) \qquad 0.5$$

$$\log_{3}(10)-\log_{3}(5) \qquad \log_{3}(2)$$

If complex format mode is Real:

$$\log_{10}(\{{}^-3,1.2,5\}) \qquad \text{Error: } \textit{Non-real result}$$

If complex format mode is Rectangular:

$$\log_{10}(\{{}^-3,1.2,5\})$$

$$\{\log_{10}(3)+1.36438\cdot \boldsymbol{i},0.079181,\log_{10}(5)\}$$

log(*squareMatrix1*[**,***Expr*]**)** ⇒ *squareMatrix*

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()**.

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

If the base argument is omitted, 10 is used as base.

In Radian angle mode and Rectangular complex format:

$$\log_{10}\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & {}^-2 & 1\end{bmatrix}\right)$$

$$\begin{bmatrix}0.795387+0.753438\cdot \boldsymbol{i} & 0.003993-0.6474\\ 0.194895-0.315095\cdot \boldsymbol{i} & 0.462485+0.2707\\ {}^-0.115909-0.904706\cdot \boldsymbol{i} & 0.488304+0.7774\end{bmatrix}$$

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

▶logbase Catalog >

Expr▶**logbase(***Expr1***)** ⇒ *expression*

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(...)`.

$$\log_{3}(10)-\log_{5}(5)\blacktriangleright\text{logbase}(5) \qquad \frac{\log_{5}\left(\frac{10}{3}\right)}{\log_{5}(3)}$$

Logistic Catalog >

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

Computes the logistic regression y = (c/(1+a•e^{-bx})) on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: c/(1+a•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 *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*

LogisticD Catalog >

LogisticD *X*, *Y* [, [*Iterations*] , [*Freq*] [, *Category*, *Include*]]

Computes the logistic regression y = (c/(1+a•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 175.)

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: c/(1+a•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 *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*

Loop Catalog >

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 Catalog >

LU *Matrix*, *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•uMatrix = pMatrix•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.

- If you use ctrl enter 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•max(dim(*Matrix*))•rowNorm(*Matrix*)

$$\begin{bmatrix}6&12&18\\5&14&31\\3&8&18\end{bmatrix}\to m1 \qquad \begin{bmatrix}6&12&18\\5&14&31\\3&8&18\end{bmatrix}$$

LU *m1,lower,upper,perm* *Done*

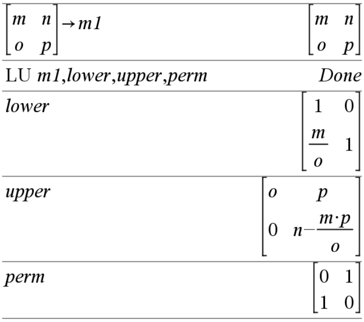
$$lower \qquad \begin{bmatrix}1&0&0\\\frac{5}{6}&1&0\\\frac{1}{2}&\frac{1}{2}&1\end{bmatrix}$$

$$upper \qquad \begin{bmatrix}6&12&18\\0&4&16\\0&0&1\end{bmatrix}$$

$$perm \qquad \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

LU Catalog >

The **LU** factorization algorithm uses partial pivoting with row interchanges.



$\begin{bmatrix} m & n \\ o & p \end{bmatrix} \to m1$ $\qquad$ $\begin{bmatrix} m & n \\ o & p \end{bmatrix}$

LU *m1*,*lower*,*upper*,*perm* $\qquad$ *Done*

lower $\qquad$ $\begin{bmatrix} 1 & 0 \\ \frac{m}{o} & 1 \end{bmatrix}$

upper $\qquad$ $\begin{bmatrix} o & p \\ 0 & n-\frac{m\cdot p}{o} \end{bmatrix}$

perm $\qquad$ $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

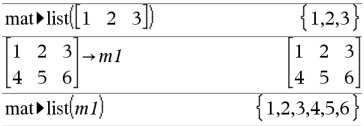
M

mat▶list() Catalog >

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$\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}\right)$ $\qquad$ $\{1,2,3\}$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \to m1$ $\qquad$ $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

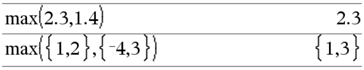
mat▶list(*m1*) $\qquad$ $\{1,2,3,4,5,6\}$

max() Catalog >

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.

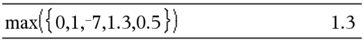


max(2.3,1.4) $\qquad$ 2.3

max$(\{1,2\},\{^-4,3\})$ $\qquad$ $\{1,3\}$

max(*List***)** ⇒ *expression*

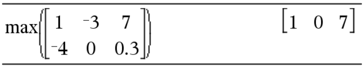
Returns the maximum element in *list*.



max$(\{0,1,^-7,1.3,0.5\})$ $\qquad$ 1.3

max(*Matrix1***)** ⇒ *matrix*

Returns a row vector containing the maximum element of each column in *Matrix1*.



max$\left(\begin{bmatrix} 1 & ^-3 & 7 \\ ^-4 & 0 & 0.3 \end{bmatrix}\right)$ $\qquad$ $\begin{bmatrix} 1 & 0 & 7 \end{bmatrix}$

max() Catalog >

Empty (void) elements are ignored. For more information on empty elements, see page 234.

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*.

$$\text{mean}(\{0.2,0,1,{}^{-}0.3,0.4\}) \qquad 0.26$$

$$\text{mean}(\{1,2,3\},\{3,2,1\}) \qquad \frac{5}{3}$$

mean(*Matrix1*[**,** *freqMatrix*]**)** ⇒ *matrix*

Returns a row vector of the means of all the columns in *Matrix1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

Empty (void) elements are ignored. For more information on empty elements, see page 234.

In Rectangular vector format:

$$\text{mean}\left(\begin{bmatrix} 0.2 & 0 \\ {}^{-}1 & 3 \\ 0.4 & {}^{-}0.5 \end{bmatrix}\right) \qquad \begin{bmatrix} {}^{-}0.133333 & 0.833333 \end{bmatrix}$$

$$\text{mean}\left(\begin{bmatrix} \frac{1}{5} & 0 \\ {}^{-}1 & 3 \\ \frac{2}{5} & \frac{{}^{-}1}{2} \end{bmatrix}\right) \qquad \begin{bmatrix} \frac{{}^{-}2}{15} & \frac{5}{6} \end{bmatrix}$$

$$\text{mean}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix},\begin{bmatrix} 5 & 3 \\ 4 & 1 \\ 6 & 2 \end{bmatrix}\right) \qquad \begin{bmatrix} \frac{47}{15} & \frac{11}{3} \end{bmatrix}$$

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*.

$$\text{median}(\{0.2,0,1,{}^{-}0.3,0.4\}) \qquad 0.2$$

median(*Matrix1*[**,** *freqMatrix*]**)** ⇒ *matrix*

Returns a row vector containing the medians of the columns in *Matrix1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

$$\text{median}\left(\begin{bmatrix} 0.2 & 0 \\ 1 & {}^{-}0.3 \\ 0.4 & {}^{-}0.5 \end{bmatrix}\right) \qquad \begin{bmatrix} 0.4 & {}^{-}0.3 \end{bmatrix}$$

median() Catalog >

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 234.

MedMed Catalog >

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

Computes the median-median line y = (m•x+b) on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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 234.

Output variable	Description
stat.RegEqn	Median-median line equation: m•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 *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*

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("Hello there",2)	"ello there"
mid("Hello there",7,3)	"the"
mid("Hello there",1,5)	"Hello"
mid("Hello there",1,0)	"⬚"

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({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(*sourceStringList***,** *Start*[**,** *Count*]**)** ⇒ *list*

Returns *Count* strings from the list of strings *sourceStringList*, beginning with element number *Start*.

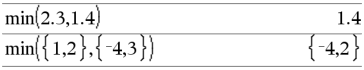
mid({"A","B","C","D"},2,2)	{"B","C"}

min() Catalog >

min(*Expr1***,** *Expr2***)** ⇒ *expression*

min(*List1, List2***)** ⇒ *list*
min(*Matrix1, Matrix2***)** ⇒ *matrix*

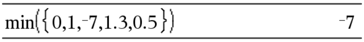
Returns the minimum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the minimum value of each pair of corresponding elements.



```
min(2.3,1.4)                          1.4
min({1,2},{-4,3})                  {-4,2}
```

min(*List***)** ⇒ *expression*

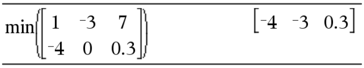
Returns the minimum element of *List*.



```
min({0,1,-7,1.3,0.5})                  -7
```

min(*Matrix1***)** ⇒ *matrix*

Returns a row vector containing the minimum element of each column in *Matrix1*.



$$\min\left(\begin{bmatrix}1 & -3 & 7\\ -4 & 0 & 0.3\end{bmatrix}\right) \qquad \begin{bmatrix}-4 & -3 & 0.3\end{bmatrix}$$

Note: See also **fMin()** and **max().**

mirr() Catalog >

mirr (*financeRate***,***reinvestRate***,***CF0***,***CFList* [**,***CFFreq*]**)**

Financial function that returns the modified internal rate of return of an investment.

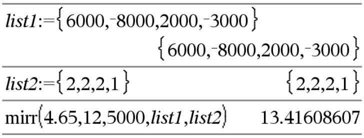
financeRate is the interest rate that you pay on the cash flow amounts.

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.



```
list1:={6000,-8000,2000,-3000}
                    {6000,-8000,2000,-3000}
list2:={2,2,2,1}                  {2,2,2,1}
mirr(4.65,12,5000,list1,list2)  13.41608607
```

mirr() Catalog >

Note: See also **irr()**, page 92.

mod() Catalog >

mod(*Expr1*, *Expr2***)** ⇒ *expression*

mod(*List1*, *List2***)** ⇒ *list*
mod(*Matrix1*, *Matrix2***)** ⇒ *matrix*

Returns the first argument modulo the second argument as defined by the identities:

mod(x,0) = x
mod(x,y) = x − y floor(x/y)

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 148

$\text{mod}(7,0)$	7
$\text{mod}(7,3)$	1
$\text{mod}(^{-}7,3)$	2
$\text{mod}(7,^{-}3)$	$^{-}2$
$\text{mod}(^{-}7,^{-}3)$	$^{-}1$
$\text{mod}(\{12,^{-}14,16\},\{9,7,^{-}5\})$	$\{3,0,^{-}4\}$

mRow() Catalog >

mRow(*Expr*, *Matrix1*, *Index***)** ⇒ *matrix*

Returns a copy of *Matrix1* with each element in row *Index* of *Matrix1* multiplied by *Expr*.

$\text{mRow}\left(\frac{^{-}1}{3},\begin{bmatrix}1&2\\3&4\end{bmatrix},2\right)$	$\begin{bmatrix}1&2\\^{-}1&\frac{^{-}4}{3}\end{bmatrix}$

mRowAdd() Catalog >

mRowAdd(*Expr*, *Matrix1*, *Index1*, *Index2***)** ⇒ *matrix*

Returns a copy of *Matrix1* with each element in row *Index2* of *Matrix1* replaced with:

Expr • row *Index1* + row *Index2*

$\text{mRowAdd}\left(^{-}3,\begin{bmatrix}1&2\\3&4\end{bmatrix},1,2\right)$	$\begin{bmatrix}1&2\\0&^{-}2\end{bmatrix}$
$\text{mRowAdd}\left(n,\begin{bmatrix}a&b\\c&d\end{bmatrix},1,2\right)$	$\begin{bmatrix}a&b\\a\cdot n+c&b\cdot n+d\end{bmatrix}$

MultReg Catalog >

MultReg *Y*, *X1*[,*X2*[,*X3*,…[,*X10*]]]

Calculates multiple linear regression of list *Y* on lists *X1*, *X2*, …, *X10*. A summary of results is stored in the *stat.results* variable. (See page 175.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	Description
stat.RegEqn	Regression Equation: b0+b1•x1+b2•x2+ …
stat.b0, stat.b1, …	Regression coefficients
stat.R^2	Coefficient of multiple determination
stat.ŷList	ŷList = b0+b1•x1+ …
stat.Resid	Residuals from the regression

MultRegIntervals Catalog >

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 175.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	Description
stat.RegEqn	Regression Equation: b0+b1•x1+b2•x2+ …
stat.ŷ	A point estimate: ŷ = b0 + b1 • xl + … for *XValList*
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*,...[, *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 175.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Outputs

Output variable	Description
stat.RegEqn	Regression Equation: b0+b1•x1+b2•x2+ ...
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.ŷList	ŷList = b0+b1•x1+ . . .
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

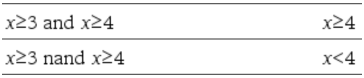
N

nand ctrl = keys

BooleanExpr1 **nand** *BooleanExpr2* returns *Boolean expression*
BooleanList1 **nand** *BooleanList2* returns *Boolean list*
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.

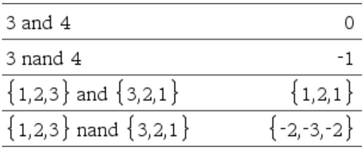
For lists and matrices, returns comparisons element by element.



```
x≥3 and x≥4                         x≥4
x≥3 nand x≥4                        x<4
```

nand ctrl = keys

Integer1 **nand** *Integer2* ⇒ *integer*



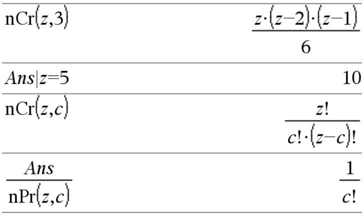
3 and 4	0
3 nand 4	-1
$\{1,2,3\}$ and $\{3,2,1\}$	$\{1,2,1\}$
$\{1,2,3\}$ nand $\{3,2,1\}$	$\{-2,-3,-2\}$

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 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).

nCr() Catalog >

nCr(*Expr1***,** *Expr2***)** ⇒ *expression*



$\text{nCr}(z,3)$	$\dfrac{z\cdot(z-2)\cdot(z-1)}{6}$
$Ans\vert z=5$	10
$\text{nCr}(z,c)$	$\dfrac{z!}{c!\cdot(z-c)!}$
$\dfrac{Ans}{\text{nPr}(z,c)}$	$\dfrac{1}{c!}$

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.

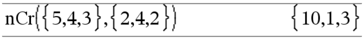
nCr(*Expr***, 0)** ⇒ **1**

nCr(*Expr***,** *negInteger***)** ⇒ **0**

nCr(*Expr***,** *posInteger***)** ⇒ *Expr***•(***Expr*–1**) ... (***Expr*–*posInteger***+1) /** *posInteger***!**

nCr(*Expr***,** *nonInteger***)** ⇒ *expression***! / ((***Expr*–*nonInteger***)!•***nonInteger***!)**

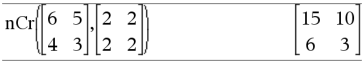
nCr(*List1***,** *List2***)** ⇒ *list*



$\text{nCr}(\{5,4,3\},\{2,4,2\})$	$\{10,1,3\}$

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

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



$\text{nCr}\left(\begin{bmatrix}6 & 5\\ 4 & 3\end{bmatrix},\begin{bmatrix}2 & 2\\ 2 & 2\end{bmatrix}\right)$	$\begin{bmatrix}15 & 10\\ 6 & 3\end{bmatrix}$

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 >

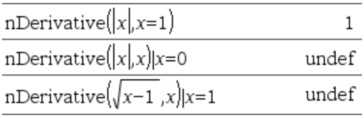
nDerivative(*Expr1***,***Var*=*Value*[**,***Order*]**)** ⇒ *value*

nDerivative(*Expr1***,***Var*[**,***Order*]**)** **|***Var*=*Value* ⇒ *value*

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**.

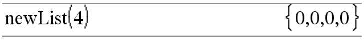


$\text{nDerivative}(\lvert x\rvert, x=1)$	1
$\text{nDerivative}(\lvert x\rvert, x)\mid x=0$	undef
$\text{nDerivative}(\sqrt{x-1}, x)\mid x=1$	undef

newList() Catalog >

newList(*numElements***)** ⇒ *list*

Returns a list with a dimension of *numElements*. Each element is zero.

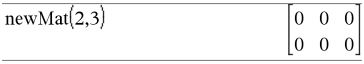


$\text{newList}(4)$	$\{0,0,0,0\}$

newMat() Catalog >

newMat(*numRows***,** *numColumns***)** ⇒ *matrix*

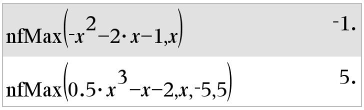
Returns a matrix of zeros with the dimension *numRows* by *numColumns*.



$\text{newMat}(2,3)$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

nfMax() Catalog >

nfMax(*Expr***,** *Var***)** ⇒ *value*
nfMax(*Expr***,** *Var*, *lowBound***)** ⇒ *value*
nfMax(*Expr***,** *Var*, *lowBound***,** *upBound***)** ⇒ *value*
nfMax(*Expr*, *Var***)** **|** *lowBound*≤*Var*≤*upBound* ⇒ *value*



$\text{nfMax}(^{-}x^{2}-2\cdot x-1,x)$	$^{-}1.$
$\text{nfMax}(0.5\cdot x^{3}-x-2,x,^{-}5,5)$	$5.$

nfMax() Catalog >

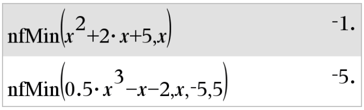
Returns a candidate numerical value of variable *Var* where the local maximum of *Expr* occurs.

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*



$\text{nfMin}\left(x^{2}+2\cdot x+5,x\right)$ -1.

$\text{nfMin}\left(0.5\cdot x^{3}-x-2,x,-5,5\right)$ -5.

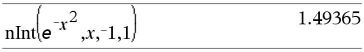
Returns a candidate numerical value of variable *Var* where the local minimum of *Expr* occurs.

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()**.

nInt() Catalog >

nInt(*Expr1, Var, Lower, Upper***)** ⇒ *expression*

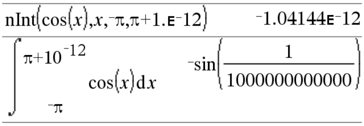


$\text{nInt}\left(e^{-x^{2}},x,-1,1\right)$ 1.49365

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 ∫ **(***Expr1*, *Var*, *Lower*, *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.

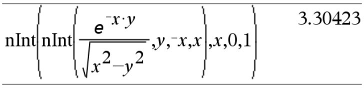


$\text{nInt}(\cos(x),x,{}^{-}\pi,\pi+1.\text{E}{}^{-}12)$ ${}^{-}1.04144\text{E}{}^{-}12$

$\int_{{}^{-}\pi}^{\pi+10^{-12}} \cos(x)\,dx$ ${}^{-}\sin\left(\frac{1}{1000000000000}\right)$

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.

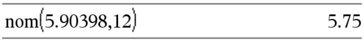


$\text{nInt}\left(\text{nInt}\left(\frac{e^{-x\cdot y}}{\sqrt{x^2-y^2}},y,{}^{-}x,x\right),x,0,1\right)$ 3.30423

Note: See also ∫**()**, page 219.

nom() Catalog >

nom(*effectiveRate,CpY***)** ⇒ *value*



nom(5.90398,12) 5.75

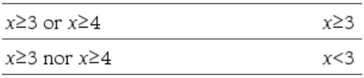
Financial function that converts the annual effective interest rate *effectiveRate* to a nominal rate, given *CpY* as the number of compounding periods per year.

effectiveRate must be a real number, and *CpY* must be a real number > 0.

Note: See also **eff()**, page 61.

nor ctrl = keys

BooleanExpr1 **nor** *BooleanExpr2* returns *Boolean expression*
BooleanList1 **nor** *BooleanList2* returns *Boolean list*
BooleanMatrix1 **nor** *BooleanMatrix2* returns *Boolean matrix*



$x\geq3$ or $x\geq4$ $x\geq3$

$x\geq3$ nor $x\geq4$ $x<3$

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.

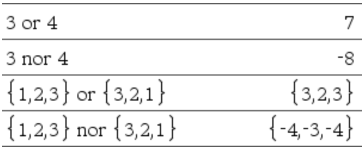
nor

ctrl = **keys**

Integer1 **nor** *Integer2* ⇒ *integer*

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).



3 or 4 → 7

3 nor 4 → -8

$\{1,2,3\}$ or $\{3,2,1\}$ → $\{3,2,3\}$

$\{1,2,3\}$ nor $\{3,2,1\}$ → $\{-4,-3,-4\}$

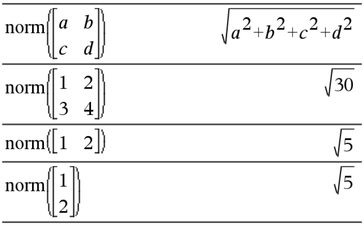
norm()

Catalog >

norm(*Matrix***)** ⇒ *expression*

norm(*Vector***)** ⇒ *expression*

Returns the Frobenius norm.



$\mathrm{norm}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right)$ → $\sqrt{a^2+b^2+c^2+d^2}$

$\mathrm{norm}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$ → $\sqrt{30}$

$\mathrm{norm}\left(\begin{bmatrix} 1 & 2 \end{bmatrix}\right)$ → $\sqrt{5}$

$\mathrm{norm}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$ → $\sqrt{5}$

normalLine()

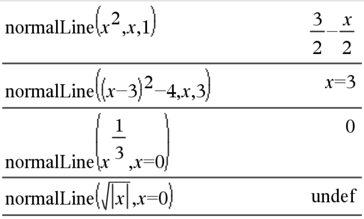
Catalog >

normalLine(*Expr1***,***Var***,***Point***)** ⇒ *expression*

normalLine(*Expr1***,***Var***=***Point***)** ⇒ *expression*

Returns the normal line to the curve represented by *Expr1* at the point specified in *Var=Point*.

Make sure that the independent variable is not defined. For example, If f1(x):=5 and x:=3, then **normalLine(**f1(x),x,2**)** returns “false.”



$\mathrm{normalLine}\left(x^2,x,1\right)$ → $\frac{3}{2}-\frac{x}{2}$

$\mathrm{normalLine}\left((x-3)^2-4,x,3\right)$ → $x=3$

$\mathrm{normalLine}\left(x^{\frac{1}{3}},x=0\right)$ → 0

$\mathrm{normalLine}\left(\sqrt{|x|},x=0\right)$ → undef

normCdf() Catalog >

normCdf(*lowBound***,***upBound*[,μ[,σ]]**)** ⇒ *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 ≤ *upBound*), set *lowBound* = $^{-}\infty$.

normPdf() Catalog >

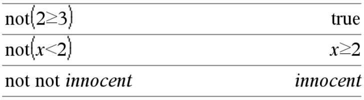
normPdf(*XVal*[,*μ*[,*σ*]]**)** ⇒ *number* if *XVal* is a number, *list* if *XVal* is a list

Computes the probability density function for the normal distribution at a specified *XVal* value for the specified μ and σ.

not Catalog >

not *BooleanExpr* ⇒ *Boolean expression*

Returns true, false, or a simplified form of the argument.



not(2≥3)	true
not(*x*<2)	*x*≥2
not not *innocent*	*innocent*

not *Integer1* ⇒ *integer*

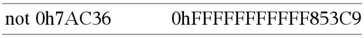
Returns the one's complement of a real integer. Internally, *Integer1* is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one's complement. Results are displayed according to the Base mode.

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 21.

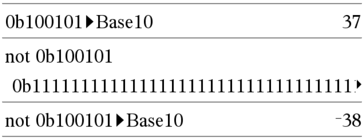
In Hex base mode:

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(*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***)!**

$\mathrm{nPr}(z,3)$	$z\cdot(z-2)\cdot(z-1)$
$Ans\vert z=5$	60
$\mathrm{nPr}(z,^-3)$	$\dfrac{1}{(z+1)\cdot(z+2)\cdot(z+3)}$
$\mathrm{nPr}(z,c)$	$\dfrac{z!}{(z-c)!}$
$Ans\cdot\mathrm{nPr}(z-c,^-c)$	1

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.

$\mathrm{nPr}(\{5,4,3\},\{2,4,2\})$	$\{20,24,6\}$

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.

$\mathrm{nPr}\left(\begin{bmatrix}6&5\\4&3\end{bmatrix},\begin{bmatrix}2&2\\2&2\end{bmatrix}\right)$	$\begin{bmatrix}30&20\\12&6\end{bmatrix}$

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.

InterestRate is the rate by which to discount the cash flows (the cost of money) over one period.

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*.

$list1:=\{6000,^-8000,2000,^-3000\}$	$\{6000,^-8000,2000,^-3000\}$
$list2:=\{2,2,2,1\}$	$\{2,2,2,1\}$
$\mathrm{npv}(10,5000,list1,list2)$	4769.91

npv() Catalog >

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() Catalog >

nSolve(*Equation***,***Var***[=***Guess***])** ⇒ *number or error_string*

nSolve(*Equation***,***Var***[=***Guess***],***lowBound***)** ⇒ *number or error_string*

nSolve(*Equation***,***Var* **[=***Guess***],***lowBound***,***upBound***)** ⇒ *number or error_string*

nSolve(*Equation***,***Var***[=***Guess***]) |** *lowBound*≤*Var*≤*upBound* ⇒ *number or error_string*

$\text{nSolve}\left(x^2+5\cdot x-25=9,x\right)$	3.84429
$\text{nSolve}\left(x^2=4,x=^-1\right)$	$^-2.$
$\text{nSolve}\left(x^2=4,x=1\right)$	2.

Note: If there are multiple solutions, you can use a guess to help find a particular solution.

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.

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."

$\text{nSolve}\left(x^2+5\cdot x-25=9,x\right)\mid x<0$	$^-8.84429$
$\text{nSolve}\left(\frac{(1+r)^{24}-1}{r}=26,r\right)\mid r>0 \text{ and } r<0.25$	0.006886
$\text{nSolve}\left(x^2=^-1,x\right)$	"No solution found"

nSolve() Catalog >

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

O

OneVar Catalog >

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 175.)

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 $\geq$ 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 234.

Output variable	**Description**
stat.$\bar{x}$	Mean of x values
stat.Σx	Sum of x values
stat.Σx^2	Sum of x^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_1X	1st Quartile of x
stat.MedianX	Median of x
stat.Q_3X	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.

$x \geq 3$ or $x \geq 4$	$x \geq 3$

Define $g(x)$=Func If $x \leq 0$ or $x \geq 5$ Goto *end* Return $x \cdot 3$ Lbl *end* EndFunc	*Done*
$g(3)$	9
$g(0)$	*A function did* not return *a value*

Integer1 **or** *Integer2* ⇒ *integer*

In Hex base mode:

0h7AC36 or 0h3D5F	0h7BD7F

Important: Zero, not the letter O.

In Bin base mode:

0b100101 or 0b100	0b100101

or Catalog >

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 21.

Note: See **xor**.

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

ord() Catalog >

ord(*String***)** ⇒ *integer*
ord(*List1***)** ⇒ *list*

Returns the numeric code of the first character in character string *String*, or a list of the first characters of each list element.

ord("hello")	104
char(104)	"h"
ord(char(24))	24
ord({"alpha","beta"})	{97,98}

P

P▶Rx() Catalog >

P▶Rx(*rExpr***,** θ*Expr***)** ⇒ *expression*
P▶Rx(*rList***,** θ*List***)** ⇒ *list*
P▶Rx(*rMatrix***,** θ*Matrix***)** ⇒ *matrix*

Returns the equivalent x-coordinate of the (r, θ) pair.

In Radian angle mode:

P▶Rx(r,θ)	$\cos(\theta)\cdot r$
P▶Rx$(4,60°)$	2
P▶Rx$\left(\{^-3,10,1.3\},\left\{\frac{\pi}{3},\frac{^-\pi}{4},0\right\}\right)$	
	$\left\{\frac{^-3}{2},5\cdot\sqrt{2},1.3\right\}$

P▶Rx() Catalog >

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 °, G, or r to override the angle mode setting temporarily.

Note: You can insert this function from the computer keyboard by typing `P@>Rx(...)`.

P▶Ry() Catalog >

P▶Ry(*rExpr*, θ*Expr***)** ⇒ *expression*

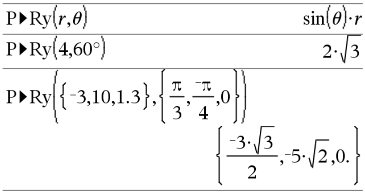
P▶Ry(*rList*, θ*List***)** ⇒ *list*
P▶Ry(*rMatrix*, θ*Matrix***)** ⇒ *matrix*

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 °, G, or r to override the angle mode setting temporarily.

Note: You can insert this function from the computer keyboard by typing `P@>Ry(...)`.

In Radian angle mode:



$\text{P▶Ry}(r,\theta)$	$\sin(\theta)\cdot r$
$\text{P▶Ry}(4,60°)$	$2\cdot\sqrt{3}$
$\text{P▶Ry}\left(\{^{-}3,10,1.3\},\left\{\frac{\pi}{3},\frac{^{-}\pi}{4},0\right\}\right)$	
	$\left\{\frac{^{-}3\cdot\sqrt{3}}{2},{}^{-}5\cdot\sqrt{2},0.\right\}$

PassErr Catalog >

PassErr

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.

For an example of **PassErr**, See Example 2 under the **Try** command, page 189.

PassErr Catalog >

Note: See also **ClrErr**, page 29, and **Try**, page 189.

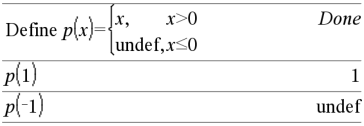
Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

piecewise() Catalog >

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 7.



```
Define p(x)= { x,      x>0                Done
             { undef,x≤0
p(1)                                         1
p(⁻1)                                    undef
```

poissCdf() Catalog >

poissCdf(λ**,***lowBound***,***upBound***)** ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

poissCdf(λ**,***upBound***)**for P(0≤X≤*upBound*) ⇒ *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 ≤ *upBound*), set *lowBound*=0

poissPdf() Catalog >

poissPdf(λ**,***XVal***)** ⇒ *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.

►Polar

Catalog >

Vector **►Polar**

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

Displays *vector* in polar form [r∠ θ]. 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 145.

$[1\ \ 3.]\blacktriangleright\text{Polar} \qquad [3.16228\ \ \angle 1.24905]$

$[x\ \ y]\blacktriangleright\text{Polar}$

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

complexValue **►Polar**

Displays *complexVector* in polar form.

- Degree angle mode returns (r∠ θ).
- 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∠ θ) polar entry.

In Radian angle mode:

$(3+4\cdot i)\blacktriangleright\text{Polar} \qquad e^{i\cdot\left(\frac{\pi}{2}-\tan^{-1}\left(\frac{3}{4}\right)\right)}\cdot 5$

$\left(\left(4\ \angle\frac{\pi}{3}\right)\right)\blacktriangleright\text{Polar} \qquad e^{\frac{i\cdot\pi}{3}}\cdot 4$

In Gradian angle mode:

$(4\cdot i)\blacktriangleright\text{Polar} \qquad (4\ \angle\ 100)$

In Degree angle mode:

$(3+4\cdot i)\blacktriangleright\text{Polar} \qquad \left(5\ \angle\ 90-\tan^{-1}\left(\frac{3}{4}\right)\right)$

polyCoeffs()

Catalog >

polyCoeffs(*Poly* [,*Var*]**)** ⇒ *list*

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.

$\text{polyCoeffs}(4\cdot x^2-3\cdot x+2,x) \qquad \{4,^{-}3,2\}$

$\text{polyCoeffs}((x-1)^2\cdot(x+2)^3)$

$\{1,4,1,^{-}10,^{-}4,8\}$

Expands the polynomial and selects *x* for the omitted *Var*.

polyCoeffs() Catalog >

$\text{polyCoeffs}\left((x+y+z)^2,x\right)$	$\left\{1,2\cdot(y+z),(y+z)^2\right\}$
$\text{polyCoeffs}\left((x+y+z)^2,y\right)$	$\left\{1,2\cdot(x+z),(x+z)^2\right\}$
$\text{polyCoeffs}\left((x+y+z)^2,z\right)$	$\left\{1,2\cdot(x+y),(x+y)^2\right\}$

polyDegree() Catalog >

polyDegree($Poly$ [,Var]**)** ⇒ *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.

$\text{polyDegree}(5)$	0
$\text{polyDegree}(\ln(2)+\pi,x)$	0

Constant polynomials

$\text{polyDegree}\left(4\cdot x^2-3\cdot x+2,x\right)$	2
$\text{polyDegree}\left((x-1)^2\cdot(x+2)^3\right)$	5

$\text{polyDegree}\left(\left(x+y^2+z^3\right)^2,x\right)$	2
$\text{polyDegree}\left(\left(x+y^2+z^3\right)^2,y\right)$	4

$\text{polyDegree}\left((x-1)^{10000},x\right)$	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($List1$**,** $Expr1$**)** ⇒ *expression*
polyEval($List1$**,** $List2$**)** ⇒ *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***)** ⇒ *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*]**)** ⇒ *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 >

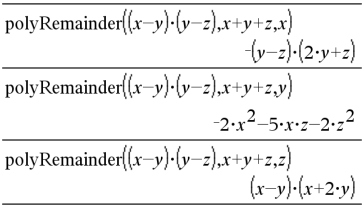
polyRemainder(*Poly1*,*Poly2* [,*Var*]**)** ⇒ *expression*

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

polyRemainder() Catalog >



$\text{polyRemainder}((x-y)\cdot(y-z),x+y+z,x)$ $\quad {}^{-}(y-z)\cdot(2\cdot y+z)$

$\text{polyRemainder}((x-y)\cdot(y-z),x+y+z,y)$ $\quad {}^{-}2\cdot x^2-5\cdot x\cdot z-2\cdot z^2$

$\text{polyRemainder}((x-y)\cdot(y-z),x+y+z,z)$ $\quad (x-y)\cdot(x+2\cdot y)$

polyRoots() Catalog >

polyRoots(*Poly***,***Var***)** ⇒ *list*

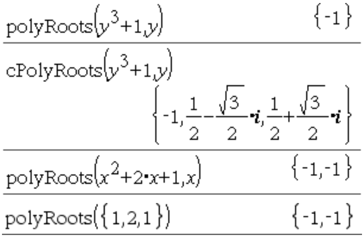
polyRoots(*ListOfCoeffs***)** ⇒ *list*

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 40.



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

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

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

$\text{polyRoots}(\{1,2,1\})$ $\quad \{{}^{-}1,{}^{-}1\}$

PowerReg Catalog >

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

Computes the power regressiony = (a•(x)^{b})on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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.

PowerReg Catalog >

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: $a\bullet(x)^b$
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 *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*

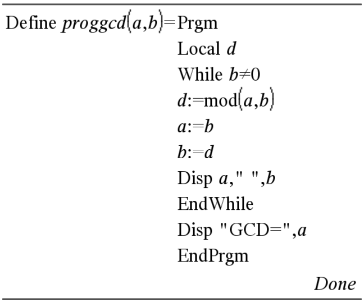
Prgm Catalog >

Prgm
    Block
EndPrgm

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.

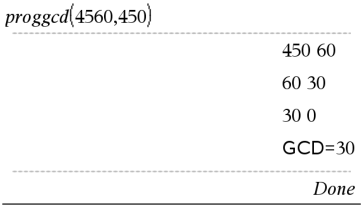
Calculate GCD and display intermediate results.



```
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
```

Prgm 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.



```
proggcd(4560,450)
                                        450 60
                                        60 30
                                        30 0
                                        GCD=30
                                          Done
```

prodSeq() See Π (), page 221.

Product (PI) See Π (), page 221.

product() Catalog >

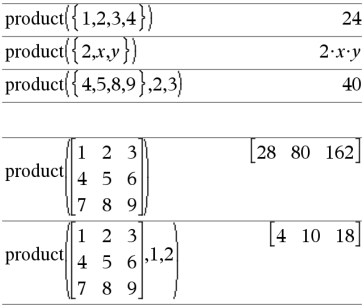
product(*List*[**,** *Start*[**,** *End*]]**)** ⇒ *expression*

Returns the product of the elements contained in *List*. *Start* and *End* are optional. They specify a range of elements.

product(*Matrix1*[**,** *Start*[**,** *End*]]**)** ⇒ *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 234.



$\text{product}(\{1,2,3,4\})$ 24

$\text{product}(\{2,x,y\})$ $2\cdot x\cdot y$

$\text{product}(\{4,5,8,9\},2,3)$ 40

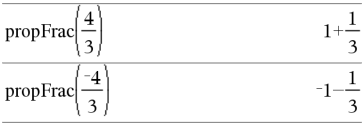
$\text{product}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\right)$ $\begin{bmatrix}28&80&162\end{bmatrix}$

$\text{product}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix},1,2\right)$ $\begin{bmatrix}4&10&18\end{bmatrix}$

propFrac() Catalog >

propFrac(*Expr1*[**,** *Var*]**)** ⇒ *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.



$\text{propFrac}\left(\frac{4}{3}\right)$ $1+\frac{1}{3}$

$\text{propFrac}\left(\frac{^-4}{3}\right)$ $^-1-\frac{1}{3}$

propFrac() Catalog >

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.

$\text{propFrac}\left(\frac{x^2+x+1}{x+1}+\frac{y^2+y+1}{y+1},x\right)$	$\frac{1}{x+1}+x+\frac{y^2+y+1}{y+1}$
$\text{propFrac}(Ans)$	$\frac{1}{x+1}+x+\frac{1}{y+1}+y$

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.

$\text{propFrac}\left(\frac{11}{7}\right)$	$1+\frac{4}{7}$
$\text{propFrac}\left(3+\frac{1}{11}+5+\frac{3}{4}\right)$	$8+\frac{37}{44}$
$\text{propFrac}\left(3+\frac{1}{11}-\left(5+\frac{3}{4}\right)\right)$	${}^{-}2-\frac{29}{44}$

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.

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 ctrl enter or set the **Auto or Approximate** mode to Approximate,

The floating-point number (9.) in m1 causes results to be calculated in floating-point form.

$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9.\end{bmatrix}\to m1$	$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9.\end{bmatrix}$
QR *m1*,*qm*,*rm*	*Done*
qm	$\begin{bmatrix}0.123091&0.904534&0.408248\\0.492366&0.301511&{}^{-}0.816497\\0.86164&{}^{-}0.301511&0.408248\end{bmatrix}$
rm	$\begin{bmatrix}8.12404&9.60114&11.0782\\0.&0.904534&1.80907\\0.&0.&0.\end{bmatrix}$

QR Catalog >

computations are done using floating-point arithmetic.

- If *Tol* is omitted or not used, the default tolerance is calculated as:
  5E−14 •max(dim(*Matrix*)) •rowNorm (*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*.

$\begin{bmatrix} m & n \\ o & p \end{bmatrix} \to m1$	$\begin{bmatrix} m & n \\ o & p \end{bmatrix}$
QR $m1,qm,rm$	*Done*
qm	$\begin{bmatrix} \frac{m}{\sqrt{m^2+o^2}} & \frac{^{-}\mathrm{sign}(m\cdot p-n\cdot o)\cdot o}{\sqrt{m^2+o^2}} \\ \frac{o}{\sqrt{m^2+o^2}} & \frac{m\cdot \mathrm{sign}(m\cdot p-n\cdot o)}{\sqrt{m^2+o^2}} \end{bmatrix}$
rm	$\begin{bmatrix} \sqrt{m^2+o^2} & \frac{m\cdot n+o\cdot p}{\sqrt{m^2+o^2}} \\ 0 & \frac{\lvert m\cdot p-n\cdot o\rvert}{\sqrt{m^2+o^2}} \end{bmatrix}$

QuadReg Catalog >

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

Computes the quadratic polynomial regression $y=a\bullet x^2+b\bullet x+c$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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.

QuadReg Catalog > 📖

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 234.

Output variable	**Description**
stat.RegEqn	Regression equation: $a\bullet x^{2}+b\bullet x+c$
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 *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*

QuartReg Catalog > 📖

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

Computes the quartic polynomial regression $y = a\bullet x^{4}+b\bullet x^{3}+c\bullet x^{2}+d\bullet x+e$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 175.)

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.

QuartReg **Catalog >**

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 234.

Output variable	Description
stat.RegEqn	Regression equation: a•x^4+b•x^3+c• x^2+d•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() **Catalog >**

R►Pθ (*xExpr***,** *yExpr***)** ⇒ *expression*

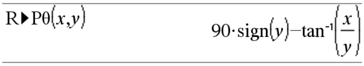
R►Pθ (*xList***,** *yList***)** ⇒ *list*
R►Pθ (*xMatrix***,** *yMatrix***)** ⇒ *matrix*

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.

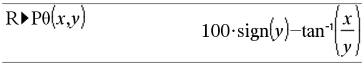
Note: You can insert this function from the computer keyboard by typing **`R@>Ptheta (...)`**.

In Degree angle mode:



$\mathrm{R}\blacktriangleright\mathrm{P}\theta(x,y)$ $90\cdot\mathrm{sign}(y)-\tan^{-1}\left(\frac{x}{y}\right)$

In Gradian angle mode:



$\mathrm{R}\blacktriangleright\mathrm{P}\theta(x,y)$ $100\cdot\mathrm{sign}(y)-\tan^{-1}\left(\frac{x}{y}\right)$

In Radian angle mode:

R▶Pθ() Catalog >

$\mathrm{R▸P\theta}(3,2)$	$\tan^{-1}\left(\frac{2}{3}\right)$
$\mathrm{R▸P\theta}\left(\begin{bmatrix}3 & ^-4 & 2\end{bmatrix},\begin{bmatrix}0 & \frac{\pi}{4} & 1.5\end{bmatrix}\right)$	$\begin{bmatrix}0 & \tan^{-1}\left(\frac{16}{\pi}\right)+\frac{\pi}{2} & 0.643501\end{bmatrix}$

R▶Pr() Catalog >

R▶Pr (*xExpr***,** *yExpr***)** ⇒ *expression*

R▶Pr (*xList***,** *yList***)** ⇒ *list*
R▶Pr (*xMatrix***,** *yMatrix***)** ⇒ *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:

$\mathrm{R▸Pr}(3,2)$	$\sqrt{13}$
$\mathrm{R▸Pr}(x,y)$	$\sqrt{x^2+y^2}$
$\mathrm{R▸Pr}\left(\begin{bmatrix}3 & ^-4 & 2\end{bmatrix},\begin{bmatrix}0 & \frac{\pi}{4} & 1.5\end{bmatrix}\right)$	$\begin{bmatrix}3 & \frac{\sqrt{\pi^2+256}}{4} & 2.5\end{bmatrix}$

▶Rad Catalog >

Expr1▶*Rad* ⇒ *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:

$(1.5)\mathrm{▸Rad}$	$(0.02618)^r$

In Gradian angle mode:

$(1.5)\mathrm{▸Rad}$	$(0.023562)^r$

rand() Catalog >

rand() ⇒ *expression*
rand(*#Trials***)** ⇒ *list*

rand() returns a random value between 0 and 1.

rand(*#Trials***)** returns a list containing *#Trials* random values between 0 and 1.

Set the random-number seed.

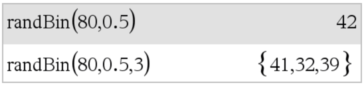
RandSeed 1147	*Done*
rand(2)	{0.158206,0.717917}

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.



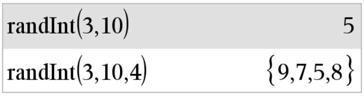
randBin(80,0.5)	42
randBin(80,0.5,3)	{41,32,39}

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.



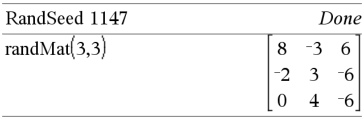
randInt(3,10)	5
randInt(3,10,4)	{9,7,5,8}

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)	$\begin{bmatrix} 8 & ^-3 & 6 \\ ^-2 & 3 & ^-6 \\ 0 & 4 & ^-6 \end{bmatrix}$

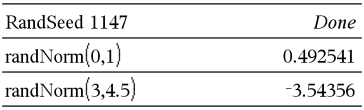
Note: The values in this matrix will change each time you press enter.

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 [μ−3•σ, μ+3•σ].

randNorm(μ, σ, *#Trials***)** returns a list containing *#Trials* decimal numbers from the specified normal distribution.



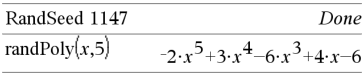
RandSeed 1147 *Done*
randNorm(0,1) 0.492541
randNorm(3,4.5) -3.54356

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.

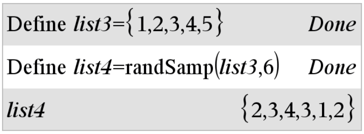


RandSeed 1147 *Done*
randPoly(*x*,5) $^{-}2\cdot x^5+3\cdot x^4-6\cdot x^3+4\cdot x-6$

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.

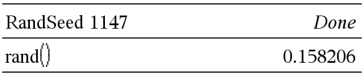


Define *list3*={1,2,3,4,5} *Done*
Define *list4*=randSamp(*list3*,6) *Done*
list4 {2,3,4,3,1,2}

RandSeed **Catalog >**

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() Catalog >

real(*Expr1***)** ⇒ *expression*

Returns the real part of the argument.

Note: All undefined variables are treated as real variables. See also **imag()**, page 88.

$\text{real}(2+3\cdot i)$	2
$\text{real}(z)$	z
$\text{real}(x+i\cdot y)$	x

real(*List1***)** ⇒ *list*

Returns the real parts of all elements.

$\text{real}(\{a+i\cdot b,3,i\})$	$\{a,3,0\}$

real(*Matrix1***)** ⇒ *matrix*

Returns the real parts of all elements.

$\text{real}\left(\begin{bmatrix} a+i\cdot b & 3 \\ c & i \end{bmatrix}\right)$	$\begin{bmatrix} a & 3 \\ c & 0 \end{bmatrix}$

►Rect Catalog >

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 132.

$$\left(\begin{bmatrix} 3 & \angle\frac{\pi}{4} & \angle\frac{\pi}{6} \end{bmatrix}\right)\blacktriangleright\text{Rect}$$

$$\begin{bmatrix} \frac{3\cdot\sqrt{2}}{4} & \frac{3\cdot\sqrt{2}}{4} & \frac{3\cdot\sqrt{3}}{2} \end{bmatrix}$$

$$\begin{bmatrix} a & \angle b & \angle c \end{bmatrix}$$

$$\begin{bmatrix} a\cdot\cos(b)\cdot\sin(c) & a\cdot\sin(b)\cdot\sin(c) & a\cdot\cos(c) \end{bmatrix}$$

complexValue **►Rect**

Displays *complexValue* in rectangular form a+bi. The *complexValue* can have any complex form. However, an re$^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use parentheses for an (r∠ θ) polar entry.

In Radian angle mode:

$\left(4\cdot e^{\frac{\pi}{3}}\right)\blacktriangleright\text{Rect}$	$4\cdot e^{\frac{\pi}{3}}$
$\left(\left(4\ \angle\frac{\pi}{3}\right)\right)\blacktriangleright\text{Rect}$	$2+2\cdot\sqrt{3}\cdot i$

In Gradian angle mode:

$((1\ \angle\ 100))\blacktriangleright\text{Rect}$	i

In Degree angle mode:

$((4\ \angle\ 60))\blacktriangleright\text{Rect}$	$2+2\cdot\sqrt{3}\cdot i$

Note: To type ∠, select it from the symbol list in the Catalog.

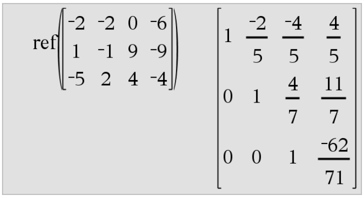
ref() Catalog >

ref(*Matrix1*[**,** *Tol*]**)** ⇒ *matrix*

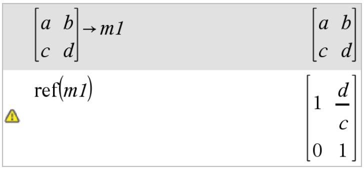
Returns the row echelon form of *Matrix1*.

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 ctrl enter 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 •max(dim(*Matrix1*)) •rowNorm (*Matrix1*)



$\text{ref}\left(\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix}\right) \quad \begin{bmatrix} 1 & \frac{-2}{5} & \frac{-4}{5} & \frac{4}{5} \\ 0 & 1 & \frac{4}{7} & \frac{11}{7} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$



$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow m1 \quad \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

$\text{ref}(m1) \quad \begin{bmatrix} 1 & \frac{d}{c} \\ 0 & 1 \end{bmatrix}$

Avoid undefined elements in *Matrix1*. 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:

$$\text{ref}\left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) \qquad \begin{bmatrix} 1 & \frac{1}{a} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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.

$$\text{ref}\left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) | a=0 \qquad \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Note: See also **rref()**, page 155.

RefreshProbeVars Catalog >

RefreshProbeVars

Allows you to access sensor data from all connected sensor probes in your TI-Basic program.

StatusVar Value	Status
statusVar =0	Normal (continue with the program)
statusVar =1	The Vernier DataQuest™ application is in data collection mode. **Note:** The Vernier DataQuest™ application must be in meter mode for this command to work.
statusVar =2	The Vernier DataQuest™ application is not launched.
statusVar =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() Catalog >

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:

remain(x,0) x
remain(x,y) x−y•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 115.

remain(7,0)	7
remain(7,3)	1
remain($^-7$,3)	$^-1$
remain(7,$^-3$)	1
remain($^-7$,$^-3$)	$^-1$
remain({12,$^-14$,16},{9,7,$^-5$})	{3,0,1}

$$\text{remain}\left(\begin{bmatrix}9 & ^-7\\ 6 & 4\end{bmatrix},\begin{bmatrix}4 & 3\\ 4 & ^-3\end{bmatrix}\right) \qquad \begin{bmatrix}1 & ^-1\\ 2 & 1\end{bmatrix}$$

Request Catalog >

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*.

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.

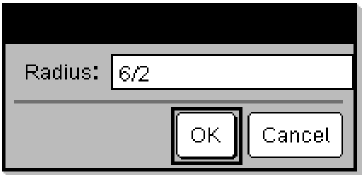
- If *DispFlag* is omitted or evaluates to **1**, the prompt message and user’s response are displayed in the Calculator history.
- If *DispFlag* evaluates to **0**, the prompt and response are not displayed in the history.

Define a program:

```
Define request_demo()=Prgm
    Request “Radius: ”,r
    Disp “Area = “,pi*r2
EndPrgm
```

Run the program and type a response:

```
request_demo()
```



Radius: 6/2

Result after selecting **OK**:

```
Radius: 6/2
Area= 28.2743
```

Request — Catalog >

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 ⌂on key and press enter 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.

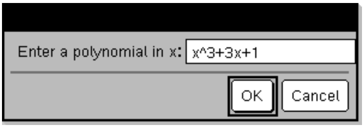
Note: See also **RequestStr**, page 149.

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()
```



Enter a polynomial in x: x^3+3x+1

Result after entering x^3+3x+1 and selecting **OK**:

```
Real roots are: {-0.322185}
```

RequestStr — Catalog >

RequestStr *promptString*, *var*[, *DispFlag*]

Define a program:

RequestStr

Catalog >

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:

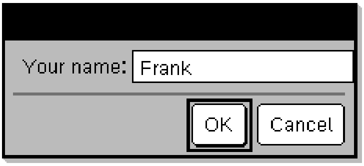
- **Handheld:** Hold down the ⌂ on key and press enter repeatedly.
- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **iPad®:** The app displays a prompt. You can continue waiting or cancel.

Note: See also **Request**, page 148.

```
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()
```



Your name: Frank

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

Catalog >

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()

Catalog >

right(*List1*[**,** *Num*]**)** ⇒ *list*

right({1,3,⁻2,4},3)	{3,⁻2,4}

right() Catalog >

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("Hello",2) "lo"

$\text{right}(x<3)$ 3

rk23 () Catalog >

rk23(*Expr*, *Var*, *depVar*, {*Var0*, *VarMax*}, *depVar0*, *VarStep* [, *diftol*]**)** ⇒ *matrix*

rk23(*SystemOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep*[, *diftol*]**)** ⇒ *matrix*

rk23(*ListOfExpr*, *Var*, *ListOfDepVars*, {*Var0*, *VarMax*}, *ListOfDepVars0*, *VarStep*[, *diftol*]**)** ⇒ *matrix*

Uses the Runge-Kutta method to solve the system

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

with *depVar*(*Var0*)=*depVar0* on the interval [*Var0*,*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*).

Differential equation:

y'=0.001*y*(100-y) and y(0)=10

$\text{rk23}(0.001\cdot y\cdot(100-y),t,y,\{0,100\},10,1)$

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4 \\ 10. & 10.9367 & 11.9493 & 13.042 & 14.2 \end{bmatrix} ▸$$

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

Same equation with *diftol* set to 1.E–6

$\text{rk23}(0.001\cdot y\cdot(100-y),t,y,\{0,100\},10,1,1.\text{E-}6)$

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 10.9367 & 11.9495 & 13.0423 & 14.2189 \end{bmatrix} ▸$$

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

rk23 () Catalog >

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.

{*Var0*, *VarMax*} is a two-element list that tells the function to integrate from *Var0* to *VarMax*.

ListOfDepVars0 is a list of initial values for dependent variables.

If *VarStep* evaluates to a nonzero number: sign(*VarStep*) = sign(*VarMax*-*Var0*) and solutions are returned at *Var0*+i**VarStep* for all i=0,1,2,… such that *Var0*+i**VarStep* is in [*var0*,*VarMax*] (may not get a solution value at *VarMax*).

if *VarStep* evaluates to zero, solutions are returned at the "Runge-Kutta" *Var* values.

diftol is the error tolerance (defaults to 0.001).

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

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

$$\text{rk23}\left(\begin{Bmatrix} ^-y1+0.1\cdot y1\cdot y2 \\ 3\cdot y2-y1\cdot y2 \end{Bmatrix},t,\{y1,y2\},\{0,5\},\{2,5\},1\right)$$

$$\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 2. & 1.94103 & 4.78694 & 3.25253 & 1.82848 \\ 5. & 16.8311 & 12.3133 & 3.51112 & 6.27245 \end{bmatrix} ▸$$

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 5.

$\sqrt[3]{8}$ 2

$\sqrt[3]{3}$ $3^{\frac{1}{3}}$

$\sqrt[3]{3.}$ 1.44225

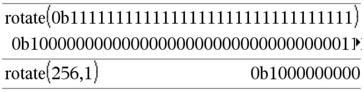
rotate() Catalog >

rotate(*Integer1*[**,***#ofRotations*]**)** ⇒ *integer*

In Bin base mode:

rotate() Catalog > 📖

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 21.

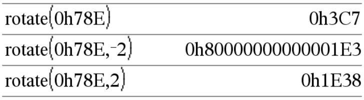


rotate(0b1111111111111111111111111111111)	
	0b10000000000000000000000000000000011▸
rotate(256,1)	0b1000000000

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

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).

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).

For example, in a right rotation:

Each bit rotates right.

0b0000000000000011110101100000110101

Rightmost bit rotates to leftmost.

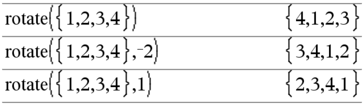
produces:

0b1000000000000000111101011000001101 0

The result is displayed according to the Base mode.

rotate(*List1*[**,***#ofRotations*]**)** ⇒ *list*

In Dec base mode:

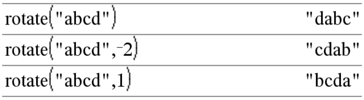


rotate({1,2,3,4})	{4,1,2,3}
rotate({1,2,3,4},⁻2)	{3,4,1,2}
rotate({1,2,3,4},1)	{2,3,4,1}

Returns a copy of *List1* rotated right or left by *#of Rotations* elements. Does not alter *List1*.

If *#ofRotations* is positive, the rotation is to the left. If *#of Rotations* is negative, the rotation is to the right. The default is −1 (rotate right one element).

rotate(*String1*[**,***#ofRotations*]**)** ⇒ *string*



rotate("abcd")	"dabc"
rotate("abcd",⁻2)	"cdab"
rotate("abcd",1)	"bcda"

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).

round() Catalog >

round(*Expr1*[**,** *digits*]**)** ⇒ *expression*

$$\text{round}(1.234567,3) \qquad 1.235$$

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*]**)** ⇒ *list*

$$\text{round}(\{\pi,\sqrt{2},\ln(2)\},4) \qquad \{3.1416,1.4142,0.6931\}$$

Returns a list of the elements rounded to the specified number of digits.

round(*Matrix1*[**,** *digits*]**)** ⇒ *matrix*

$$\text{round}\left(\begin{bmatrix}\ln(5) & \ln(3)\\ \pi & e^{1}\end{bmatrix},1\right) \qquad \begin{bmatrix}1.6 & 1.1\\ 3.1 & 2.7\end{bmatrix}$$

Returns a matrix of the elements rounded to the specified number of digits.

rowAdd() Catalog >

rowAdd(*Matrix1***,** *rIndex1***,** *rIndex2***)** ⇒ *matrix*

$$\text{rowAdd}\left(\begin{bmatrix}3 & 4\\ ^-3 & ^-2\end{bmatrix},1,2\right) \qquad \begin{bmatrix}3 & 4\\ 0 & 2\end{bmatrix}$$

$$\text{rowAdd}\left(\begin{bmatrix}a & b\\ c & d\end{bmatrix},1,2\right) \qquad \begin{bmatrix}a & b\\ a+c & b+d\end{bmatrix}$$

Returns a copy of *Matrix1* with row *rIndex2* replaced by the sum of rows *rIndex1* and *rIndex2*.

rowDim() Catalog >

rowDim(*Matrix***)** ⇒ *expression*

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix} \to m1 \qquad \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

$$\text{rowDim}(m1) \qquad 3$$

Returns the number of rows in *Matrix*.

Note: See also **colDim()**, page 30.

rowNorm() Catalog >

rowNorm(*Matrix***)** ⇒ *expression*

$$\text{rowNorm}\left(\begin{bmatrix}^-5 & 6 & ^-7\\ 3 & 4 & 9\\ 9 & ^-9 & ^-7\end{bmatrix}\right) \qquad 25$$

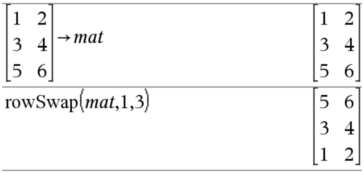
Returns the maximum of the sums of the absolute values of the elements in the rows in *Matrix*.

Note: All matrix elements must simplify to numbers. See also **colNorm()**, page 30.

rowSwap() Catalog >

rowSwap(*Matrix1*, *rIndex1*, *rIndex2***)** ⇒ *matrix*

Returns *Matrix1* with rows *rIndex1* and *rIndex2* exchanged.



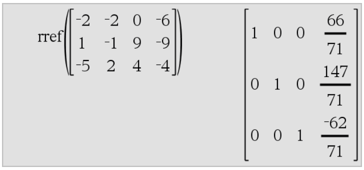
$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix} \rightarrow mat$ $\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$

$\text{rowSwap}(mat,1,3)$ $\begin{bmatrix}5 & 6\\3 & 4\\1 & 2\end{bmatrix}$

rref() Catalog >

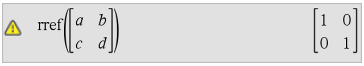
rref(*Matrix1*[, *Tol*]**)** ⇒ *matrix*

Returns the reduced row echelon form of *Matrix1*.



$\text{rref}\left(\begin{bmatrix}-2 & -2 & 0 & -6\\1 & -1 & 9 & -9\\-5 & 2 & 4 & -4\end{bmatrix}\right)$ $\begin{bmatrix}1 & 0 & 0 & \frac{66}{71}\\0 & 1 & 0 & \frac{147}{71}\\0 & 0 & 1 & \frac{-62}{71}\end{bmatrix}$

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.



⚠ $\text{rref}\left(\begin{bmatrix}a & b\\c & d\end{bmatrix}\right)$ $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$

- If you use ctrl enter 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 •max(dim(*Matrix1*)) •rowNorm (*Matrix1*)

Note: See also **ref()**, page 146.

S

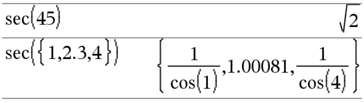
sec() trig key

sec(*Expr1***)** ⇒ *expression*

sec(*List1***)** ⇒ *list*

Returns the secant of *Expr1* or returns a list containing the secants of all elements in *List1*.

In Degree angle mode:



$\sec(45)$ $\sqrt{2}$

$\sec(\{1,2.3,4\})$ $\left\{\frac{1}{\cos(1)},1.00081,\frac{1}{\cos(4)}\right\}$

sec() trig key

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.

sec $^{-1}$() trig key

sec $^{-1}$**(***Expr1***)** ⇒ *expression*

sec $^{-1}$**(***List1***)** ⇒ *list*

Returns the angle whose secant is *Expr1* or returns a list containing the inverse secants 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 `arcsec(...)`.

In Degree angle mode:

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

In Gradian angle mode:

$\sec^{-1}(\sqrt{2})$ $\quad 50$

In Radian angle mode:

$\sec^{-1}(\{1,2,5\})$ $\quad \left\{0,\frac{\pi}{3},\cos^{-1}\left(\frac{1}{5}\right)\right\}$

sech() Catalog >

sech(*Expr1***)** ⇒ *expression*

sech(*List1***)** ⇒ *list*

Returns the hyperbolic secant of *Expr1* or returns a list containing the hyperbolic secants of the *List1* elements.

$\text{sech}(3)$ $\quad \frac{1}{\cosh(3)}$

$\text{sech}(\{1,2.3,4\})$ $\quad \left\{\frac{1}{\cosh(1)},0.198522,\frac{1}{\cosh(4)}\right\}$

sech $^{-1}$() Catalog >

sech $^{-1}$**(***Expr1***)** ⇒ *expression*

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*.

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

In Radian angle and Rectangular complex mode:

$\text{sech}^{-1}(1)$ $\quad 0$

$\text{sech}^{-1}(\{1,^{-}2,2.1\})$ $\quad \left\{0,\frac{2\cdot\pi}{3}\cdot i,8.\text{E}^{-}15+1.07448\cdot i\right\}$

Send — Hub Menu

Send *exprOrString1* [**,** *exprOrString2*] ...

Programming command: Sends one or more TI-Innovator™ Hub commands to a connected hub.

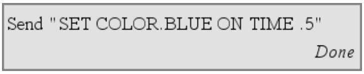
exprOrString must be a valid TI-Innovator™ Hub Command. Typically, *exprOrString* contains a **"SET ..."** command to control a device or a **"READ ..."** command to request data.

The arguments are sent to the hub in succession.

Note: You can use the **Send** command within a user-defined program but not within a function.

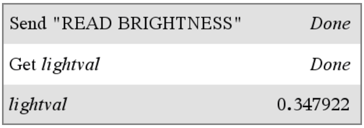
Note: See also **Get** (page 80), **GetStr** (page 83), and **eval()** (page 64).

Example: Turn on the blue element of the built-in RGB LED for 0.5 seconds.



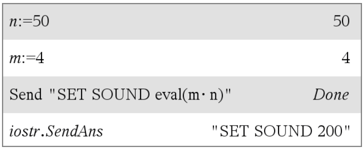
```
Send "SET COLOR.BLUE ON TIME .5"
                            Done
```

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*.



Send "READ BRIGHTNESS"	*Done*
Get *lightval*	*Done*
lightval	0.347922

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.



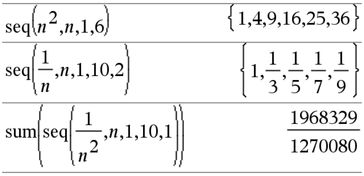
n:=50	50
m:=4	4
Send "SET SOUND eval(m· n)"	*Done*
iostr.SendAns	"SET SOUND 200"

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.

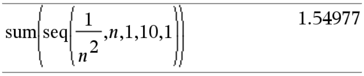
The default value for *Step* = 1.



$\text{seq}\left(n^2,n,1,6\right)$	$\{1,4,9,16,25,36\}$
$\text{seq}\left(\frac{1}{n},n,1,10,2\right)$	$\left\{1,\frac{1}{3},\frac{1}{5},\frac{1}{7},\frac{1}{9}\right\}$
$\text{sum}\left(\text{seq}\left(\frac{1}{n^2},n,1,10,1\right)\right)$	$\frac{1968329}{1270080}$

Note: To force an approximate result,

Handheld: Press ctrl enter.
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 *depVar*(*Var*)=*Expr* as follows: Increments independent variable *Var* from *Var0* through *VarMax* by *VarStep*, evaluates *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*

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**.

Generate the first 5 terms of the sequence $u(n) = u(n\text{-}1)^2/2$, 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:

$$\text{seqGen}\left(\frac{u(n-1)+1}{n},n,u,\{2,5\},\{3\}\right)$$

$$\left\{3,\frac{4}{3},\frac{7}{12},\frac{19}{60}\right\}$$

Example in which initial term is symbolic:

$$\text{seqGen}(u(n-1)+2,n,u,\{1,5\},\{a\})$$

$$\{a,a+2,a+4,a+6,a+8\}$$

System of two sequences:

$$\text{seqGen}\left(\left\{\frac{1}{n},\frac{u2(n-1)}{2}+u1(n-1)\right\},n,\{u1,u2\},\{1,5\},\begin{bmatrix}\_\\2\end{bmatrix}\right)$$

$$\begin{bmatrix}1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5}\\ 2 & 2 & \frac{3}{2} & \frac{13}{12} & \frac{19}{24}\end{bmatrix}$$

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() Catalog >

seqn(*Expr***(***u***,** *n*[**,** *ListOfInitTerms*[**,** *nMax*[**,** *CeilingValue*]]]**)** ⇒ *list*

Generate the first 6 terms of the sequence $u(n) = u(n\text{-}1)/2$, with $u(1)$=**2**.

$$\text{seqn}\left(\frac{u(n-1)}{n},\{2\},6\right)$$

$$\left\{2,1,\frac{1}{3},\frac{1}{12},\frac{1}{60},\frac{1}{360}\right\}$$

seqn() Catalog >

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.

$$\text{seqn}\left(\frac{1}{n^2},6\right) \qquad \left\{1,\frac{1}{4},\frac{1}{9},\frac{1}{16},\frac{1}{25},\frac{1}{36}\right\}$$

seqn(*Expr***(***n*[**,** *nMax*[**,** *CeilingValue*]]**)** ⇒ *list*

Generates a list of terms for a nonrecursive 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**

series() Catalog >

series(*Expr1***,** *Var***,** *Order*[**,** *Point*]**)** ⇒ *expression*

series(*Expr1***,** *Var***,** *Order*[**,** *Point*]**) |** *Var***>***Point* ⇒ *expression*

series(*Expr1***,** *Var***,** *Order*[**,** *Point*]**) |** *Var***<***Point* ⇒ *expression*

$$\text{series}\left(\frac{1-\cos(x-1)}{(x-1)^2},x,4,1\right) \qquad \frac{1}{2}-\frac{(x-1)^2}{24}+\frac{(x-1)^4}{720}$$

$$\text{series}\left(\frac{^-1}{\mathbf{e}^{z\_}},z\_,1\right) \qquad z\_-1$$

$$\text{series}\left(\left(1+\frac{1}{n}\right)^n,n,2,\infty\right) \qquad \mathbf{e}-\frac{\mathbf{e}}{2\cdot n}+\frac{11\cdot \mathbf{e}}{24\cdot n^2}$$

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.

$$\text{series}\left(\tan^{-1}\left(\frac{1}{x}\right),x,5\right)|x>0 \qquad \frac{\pi}{2}-x+\frac{x^3}{3}-\frac{x^5}{5}$$

$$\text{series}\left(\int \frac{\sin(x)}{x}\,\mathrm{d}x,x,6\right) \qquad x-\frac{x^3}{18}+\frac{x^5}{600}$$

$$\text{series}\left(\int_0^x \sin(x\cdot \sin(t))\,\mathrm{d}t,x,7\right) \qquad \frac{x^3}{2}-\frac{x^5}{24}-\frac{29\cdot x^7}{720}$$

$$\text{series}\left(\left(1+\mathbf{e}^x\right)^2,x,2,1\right)$$

$$(\mathbf{e}+1)^2+2\cdot \mathbf{e}\cdot (\mathbf{e}+1)\cdot (x-1)+\mathbf{e}\cdot (2\cdot \mathbf{e}+1)\cdot (x-1)^2$$

series() Catalog >

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 = ∞ 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 sign(…) or abs(…) for a real expansion variable or $(-1)^{\text{floor(…angle(…)…)}}$ 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.

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 58.

setMode() Catalog >

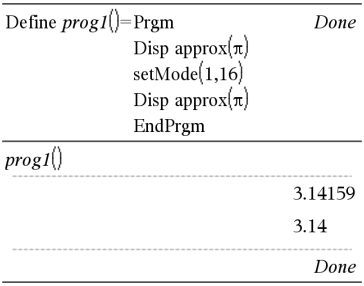
setMode(*modeNameInteger*, *settingInteger***)** ⇒ *integer*
setMode(*list***)** ⇒ *integer list*

Valid only within a function or program.

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.

setMode() Catalog >

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.



```
Define prog1()=Prgm                       Done
               Disp approx(π)
               setMode(1,16)
               Disp approx(π)
               EndPrgm
prog1()
                                       3.14159
                                          3.14
                                          Done
```

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 82.

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

Mode Name	Mode Integer	Setting Integers
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()

Catalog >

shift(*Integer1*[**,***#ofShifts*]**)** ⇒ *integer*

Shifts 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 21.

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).

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.

For example, in a right shift:

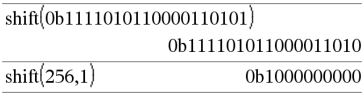
Each bit shifts right.

0b0000000000000111101011000011010

Inserts 0 if leftmost bit is 0,
or 1 if leftmost bit is 1.

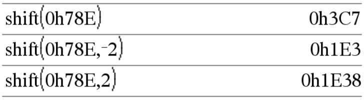
produces:

In Bin base mode:



```
shift(0b1111010110000110101)
                    0b111101011000011010
shift(256,1)                0b1000000000
```

In Hex base mode:



```
shift(0h78E)                       0h3C7
shift(0h78E,⁻2)                    0h1E3
shift(0h78E,2)                    0h1E38
```

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

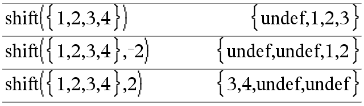
shift() Catalog >

0b00000000000000111101011000011010

The result is displayed according to the Base mode. Leading zeros are not shown.

shift(*List1*[**,***#ofShifts*]**)** ⇒ *list*

In Dec base mode:



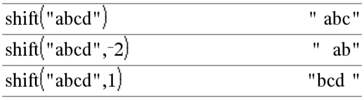
shift({1,2,3,4})	{undef,1,2,3}
shift({1,2,3,4},-2)	{undef,undef,1,2}
shift({1,2,3,4},2)	{3,4,undef,undef}

Returns a copy of *List1* shifted right or left by *#ofShifts* elements. Does not alter *List1*.

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(*String1*[**,***#ofShifts*]**)** ⇒ *string*



shift("abcd")	" abc"
shift("abcd",-2)	" ab"
shift("abcd",1)	"bcd "

Returns a copy of *String1* shifted right or left by *#ofShifts* characters. Does not alter *String1*.

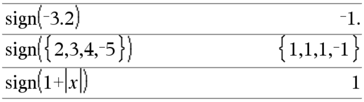
If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is −1 (shift right one character).

Characters introduced at the beginning or end of *string* by the shift are set to a space.

sign() Catalog >

sign(*Expr1***)** ⇒ *expression*

sign(*List1***)** ⇒ *list*
sign(*Matrix1***)** ⇒ *matrix*



sign(-3.2)	-1.
sign({2,3,4,-5})	{1,1,1,-1}
sign(1+|x|)	1

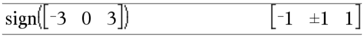
For real and complex *Expr1*, returns *Expr1*/**abs(***Expr1***)** when *Expr1*≠ 0.

Returns 1 if *Expr1* is positive. Returns −1 if *Expr1*is negative.

sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

If complex format mode is Real:



sign([-3 0 3])	[-1 ±1 1]

simult() Catalog >

simult(*coeffMatrix***,** *constVector*[**,** *Tol*]**)** ⇒ *matrix*

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also **linSolve()**, page 101.

coeffMatrix must be a square matrix that contains the coefficients of the equations.

constVector must have the same number of rows (same dimension) as *coeffMatrix* and contain the constants.

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 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 •max(dim(*coeffMatrix*)) •rowNorm(*coeffMatrix*)

Solve for x and y:
x + 2y = 1
3x + 4y = −1

$$\text{simult}\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\begin{bmatrix}1\\ -1\end{bmatrix}\right) \qquad \begin{bmatrix}-3\\ 2\end{bmatrix}$$

The solution is x=−3 and y=2.

Solve:
ax + by = 1
cx + dy = 2

$$\begin{bmatrix}a & b\\ c & d\end{bmatrix}\to matx1 \qquad \begin{bmatrix}a & b\\ c & d\end{bmatrix}$$

$$\text{simult}\left(matx1,\begin{bmatrix}1\\ 2\end{bmatrix}\right) \qquad \begin{bmatrix}\frac{-(2\cdot b-d)}{a\cdot d-b\cdot c}\\ \frac{2\cdot a-c}{a\cdot d-b\cdot c}\end{bmatrix}$$

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

$$\text{simult}\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\begin{bmatrix}1 & 2\\ -1 & -3\end{bmatrix}\right) \qquad \begin{bmatrix}-3 & -7\\ 2 & \frac{9}{2}\end{bmatrix}$$

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`.

$$(\cos(x))^2\blacktriangleright\text{sin} \qquad 1-(\sin(x))^2$$

►sin

Catalog >

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(...) modulo 1–sin(...)^2
so that any remaining powers of sin(...) have exponents in the range (0, 2). Thus, the result will be free of cos(...) if and only if cos(...) 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.

sin()

trig key

sin(*Expr1***)** ⇒ *expression*

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.

In Degree angle mode:

$\sin\left(\frac{\pi}{4}^{r}\right)$	$\frac{\sqrt{2}}{2}$
$\sin(45)$	$\frac{\sqrt{2}}{2}$
$\sin(\{0,60,90\})$	$\left\{0,\frac{\sqrt{3}}{2},1\right\}$

In Gradian angle mode:

$\sin(50)$	$\frac{\sqrt{2}}{2}$

In Radian angle mode:

$\sin\left(\frac{\pi}{4}\right)$	$\frac{\sqrt{2}}{2}$
$\sin(45°)$	$\frac{\sqrt{2}}{2}$

sin(*squareMatrix1***)** ⇒ *squareMatrix*

In Radian angle mode:

sin() trig key

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.

$$\sin\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & -2 & 1\end{bmatrix}\right)$$
$$\begin{bmatrix}0.9424 & -0.04542 & -0.031999\\ -0.045492 & 0.949254 & -0.020274\\ -0.048739 & -0.00523 & 0.961051\end{bmatrix}$$

sin⁻¹() trig 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.

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

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(*squareMatrix1***)** ⇒ *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:

$$\sin^{-1}\left(\begin{bmatrix}1 & 5\\ 4 & 2\end{bmatrix}\right)$$
$$\begin{bmatrix}-0.174533-0.12198\cdot \boldsymbol{i} & 1.74533-2.35591\cdot \boldsymbol{i}\\ 1.39626-1.88473\cdot \boldsymbol{i} & 0.174533-0.593162\cdot \boldsymbol{i}\end{bmatrix}$$

sinh() Catalog >

sinh(*Expr1***)** ⇒ *expression*

sinh(*List1***)** ⇒ *list*

sinh (*Expr1***)** returns the hyperbolic sine of the argument as an expression.

$\sinh(1.2)$ 1.50946

$\sinh(\{0,1.2,3.\})$ $\{0,1.50946,10.0179\}$

sinh() Catalog >

sinh (*List1***)** returns a list of the hyperbolic sines of each element of *List1*.

sinh(*squareMatrix1***)** ⇒ *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()**.

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

In Radian angle mode:

$$\sinh\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & {}^{-}2 & 1\end{bmatrix}\right) \quad \begin{bmatrix}360.954 & 305.708 & 239.604\\ 352.912 & 233.495 & 193.564\\ 298.632 & 154.599 & 140.251\end{bmatrix}$$

sinh⁻¹() Catalog >

sinh^{-1}**(***Expr1***)** ⇒ *expression*

sinh^{-1}**(***List1***)** ⇒ *list*

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(...)`.

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

$$\sinh^{-1}(\{0,2.1,3\}) \qquad \{0,1.48748,\sinh^{-1}(3)\}$$

sinh^{-1}**(***squareMatrix1***)** ⇒ *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:

$$\sinh^{-1}\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & {}^{-}2 & 1\end{bmatrix}\right) \quad \begin{bmatrix}0.041751 & 2.15557 & 1.1582\\ 1.46382 & 0.926568 & 0.112557\\ 2.75079 & {}^{-}1.5283 & 0.57268\end{bmatrix}$$

SinReg Catalog >

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 175.)

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 234.

Output variable	**Description**
stat.RegEqn	Regression Equation: a•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 *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*

solve() Catalog >

solve(*Equation*, *Var***)** ⇒ *Boolean expression*
solve(*Equation*, *Var*=*Guess***)** ⇒ *Boolean expression*
solve(*Inequality*, *Var***)** ⇒ *Boolean expression*

$$\text{solve}\left(a\cdot x^{2}+b\cdot x+c=0,x\right)$$
$$x=\frac{\sqrt{b^{2}-4\cdot a\cdot c}-b}{2\cdot a}\text{ or }x=\frac{-\left(\sqrt{b^{2}-4\cdot a\cdot c}+b\right)}{2\cdot a}$$

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.

$$Ans|a=1\text{ and }b=1\text{ and }c=1$$
$$x=\frac{-1}{2}+\frac{\sqrt{3}}{2}\cdot i\text{ or }x=\frac{-1}{2}-\frac{\sqrt{3}}{2}\cdot i$$

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.

$$\text{solve}\left((x-a)\cdot \boldsymbol{e}^{x}=-x\cdot(x-a),x\right)$$
$$x=a\text{ or }x=-0.567143$$

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.

$$(x+1)\cdot\frac{x-1}{x-1}+x-3 \qquad 2\cdot x-2$$

For inequalities of types ≥, ≤, <, or >, explicit solutions are unlikely unless the inequality is linear and contains only *Var*.

$$\text{solve}(5\cdot x-2\geq 2\cdot x,x) \qquad x\geq\frac{2}{3}$$

For the Exact mode, portions that cannot be solved are returned as an implicit equation or inequality.

$$\text{exact}\left(\text{solve}\left((x-a)\cdot e^{x}=-x\cdot(x-a),x\right)\right)$$
$$e^{x}+x=0\text{ or }x=a$$

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.

In Radian angle mode:

$$\text{solve}\left(\tan(x)=\frac{1}{x},x\right)|x>0\text{ and }x<1$$
$$x=0.860334$$

solve() Catalog >

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.

$\text{solve}(x=x+1,x)$	false
$\text{solve}(x=x,x)$	true

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.

$2\cdot x-1\le 1$ and $\text{solve}(x^2\ne 9,x)$	$x\ne{}^-3$ and $x\le 1$

Solutions might contain a unique new undefined constant of the form **n***j* with *j* being an integer in the interval 1–255. Such variables designate an arbitrary integer.

In Radian angle mode:

$\text{solve}(\sin(x)=0,x)$	$x=\boldsymbol{n}1\cdot\pi$

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.

$\text{solve}\left(x^{\frac{1}{3}}={}^-1,x\right)$	$x={}^-1$
$\text{solve}(\sqrt{x}={}^-2,x)$	false
$\text{solve}({}^-\sqrt{x}={}^-2,x)$	$x=4$

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.

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

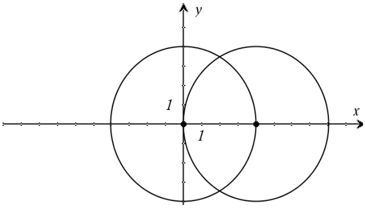
solve() Catalog >

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.

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.

$$\text{solve}\left(x^2+y^2=r^2 \text{ and} (x-r)^2+y^2=r^2,\{x,y\}\right)$$
$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2} \text{ or } x=\frac{r}{2} \text{ and } y=\frac{^-\sqrt{3}\cdot r}{2}$$

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.

$$\text{solve}\left(x^2+y^2=r^2 \text{ and} (x-r)^2+y^2=r^2,\{x,y,z\}\right)$$
$$x=\frac{r}{2} \text{ and } y=\frac{\sqrt{3}\cdot r}{2} \text{ and } z=\mathbf{c}1 \text{ or } x=\frac{r}{2} \text{ and } y▸$$

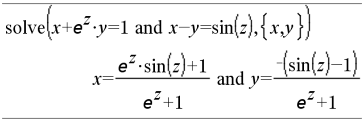
To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

The cylinder solutions illustrate how families of solutions might contain arbitrary constants of the form **c***k*, where *k* is an integer suffix from 1 through 255.

solve() Catalog >

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.

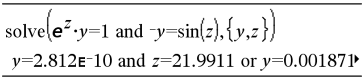
If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in the solution variables, **solve()** uses Gaussian elimination to attempt to determine all real solutions.



$\text{solve}(x+e^{z}\cdot y=1 \text{ and } x-y=\sin(z),\{x,y\})$

$x=\dfrac{e^{z}\cdot\sin(z)+1}{e^{z}+1} \text{ and } y=\dfrac{^-(\sin(z)-1)}{e^{z}+1}$

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.

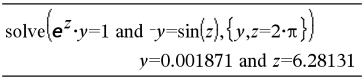


$\text{solve}(e^{z}\cdot y=1 \text{ and } ^-y=\sin(z),\{y,z\})$

$y=2.812\text{E}^-10 \text{ and } z=21.9911 \text{ or } y=0.001871$▸

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

Each solution variable starts at its guessed value if there is one; otherwise, it starts at 0.0.



$\text{solve}(e^{z}\cdot y=1 \text{ and } ^-y=\sin(z),\{y,z=2\cdot\pi\})$

$y=0.001871 \text{ and } z=6.28131$

Use guesses to seek additional solutions one by one. For convergence, a guess may have to be rather close to a solution.

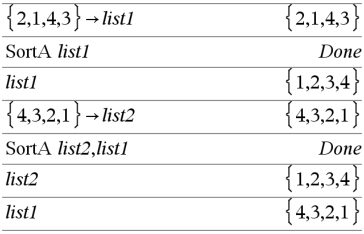
SortA Catalog >

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.



{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}

SortA **Catalog >**

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 234.

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 234.

$\{2,1,4,3\} \rightarrow list1$	$\{2,1,4,3\}$
$\{1,2,3,4\} \rightarrow list2$	$\{1,2,3,4\}$
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 [ρ∠ θ∠ φ].

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,

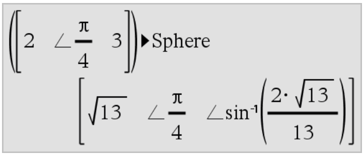
Handheld: Press ctrl enter.
Windows®: Press **Ctrl+Enter**.
Macintosh®: Press ⌘+**Enter**.
iPad®: Hold **enter**, and select ≈.

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} \blacktriangleright \text{Sphere}$$
$$\begin{bmatrix}3.74166 & \angle 1.10715 & \angle 0.640522\end{bmatrix}$$

$$\left(\begin{bmatrix}2 & \angle \frac{\pi}{4} & 3\end{bmatrix}\right) \blacktriangleright \text{Sphere}$$
$$\begin{bmatrix}3.60555 & \angle 0.785398 & \angle 0.588003\end{bmatrix}$$

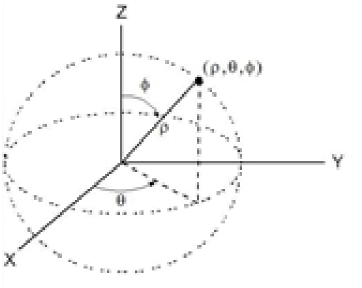
Press enter

►Sphere Catalog >



$\left(\begin{bmatrix}2 & \angle\frac{\pi}{4} & 3\end{bmatrix}\right)$►Sphere

$\begin{bmatrix}\sqrt{13} & \angle\frac{\pi}{4} & \angle\sin^{-1}\left(\frac{2\cdot\sqrt{13}}{13}\right)\end{bmatrix}$



sqrt() Catalog >

sqrt(*Expr1***)** ⇒ *expression*

sqrt(*List1***)** ⇒ *list*

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 5.

$\sqrt{4}$	2
$\sqrt{\{9,a,4\}}$	$\{3,\sqrt{a},2\}$

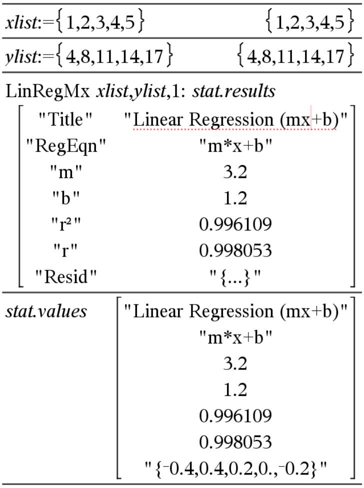
stat.results Catalog >

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.



```
xlist:={1,2,3,4,5}                        {1,2,3,4,5}
ylist:={4,8,11,14,17}                  {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             ]
 [  "r²"               0.996109          ]
 [  "r"                0.998053          ]
 [ "Resid"             "{...}"           ]
stat.values       [ "Linear Regression (mx+b)" ]
                  [          "m*x+b"           ]
                  [            3.2             ]
                  [            1.2             ]
                  [          0.996109          ]
                  [          0.998053          ]
                  [    "{-0.4,0.4,0.2,0.,-0.2}"  ]
```

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.

stat.a
stat.AdjR^2
stat.b
stat.b0
stat.b1
stat.b2
stat.b3
stat.b4
stat.b5
stat.b6
stat.b7
stat.b8
stat.b9
stat.b10
stat.bList
stat.χ^2
stat.c
stat.CLower
stat.CLowerList
stat.CompList
stat.CompMatrix
stat.CookDist
stat.CUpper

stat.dfDenom
stat.dfBlock
stat.dfCol
stat.dfError
stat.dfInteract
stat.dfReg
stat.dfNumer
stat.dfRow
stat.DW
stat.e
stat.ExpMatrix
stat.F
stat.FBlock
stat.Fcol
stat.FInteract
stat.FreqReg
stat.Frow
stat.Leverage
stat.LowerPred
stat.LowerVal
stat.m
stat.MaxX
stat.MaxY

stat.MedianY
stat.MEPred
stat.MinX
stat.MinY
stat.MS
stat.MSBlock
stat.MSCol
stat.MSError
stat.MSInteract
stat.MSReg
stat.MSRow
stat.n
Stat.$\hat{p}$
stat.$\hat{p}$1
stat.$\hat{p}$2
stat.$\hat{p}$Diff
stat.PList
stat.PVal
stat.PValBlock
stat.PValCol
stat.PValInteract
stat.PValRow
stat.Q1X

stat.Q3X
stat.Q3Y
stat.r
stat.r^2
stat.RegEqn
stat.Resid
stat.ResidTrans
stat.σx
stat.σy
stat.σx1
stat.σx2
stat.Σx
stat.Σx^2
stat.Σxy
stat.Σy
stat.Σy^2
stat.s
stat.SE
stat.SEList
stat.SEPred
stat.sResid
stat.SEslope
stat.sp

stat.SSBlock
stat.SSCol
stat.SSX
stat.SSY
stat.SSError
stat.SSInteract
stat.SSReg
stat.SSRow
stat.tList
stat.UpperPred
stat.UpperVal
stat.$\bar{x}$
stat.$\bar{x}$1
stat.$\bar{x}$2
stat.$\bar{x}$Diff
stat.$\bar{x}$List
stat.XReg
stat.XVal
stat.XValList
stat.$\bar{y}$
stat.$\hat{y}$
stat.$\hat{y}$List
stat.YReg

stat.CUpperList
stat.d
stat.ME
stat.MedianX
stat.Q1Y
stat.SS

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

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.

See the **stat.results** example.

stDevPop() **Catalog >**

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 234.

In Radian angle and auto modes:

$$\text{stDevPop}(\{a,b,c\}) \quad \frac{\sqrt{2\cdot(a^2-a\cdot(b+c)+b^2-b\cdot c+c^2)}}{3}$$

$$\text{stDevPop}(\{1,2,5,{}^{-}6,3,{}^{-}2\}) \quad \frac{\sqrt{465}}{6}$$

$$\text{stDevPop}(\{1.3,2.5,{}^{-}6.4\},\{3,2,5\}) \quad 4.11107$$

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*.

$$\text{stDevPop}\left(\begin{bmatrix}1 & 2 & 5\\ {}^{-}3 & 0 & 1\\ 5 & 7 & 3\end{bmatrix}\right) \quad \begin{bmatrix}\frac{4\cdot\sqrt{6}}{3} & \frac{\sqrt{78}}{3} & \frac{2\cdot\sqrt{6}}{3}\end{bmatrix}$$

$$\text{stDevPop}\left(\begin{bmatrix}{}^{-}1.2 & 5.3\\ 2.5 & 7.3\\ 6 & {}^{-}4\end{bmatrix},\begin{bmatrix}4 & 2\\ 3 & 3\\ 1 & 7\end{bmatrix}\right) \quad \begin{bmatrix}2.52608 & 5.21506\end{bmatrix}$$

stDevPop() Catalog >

Note: *Matrix1* must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 234.

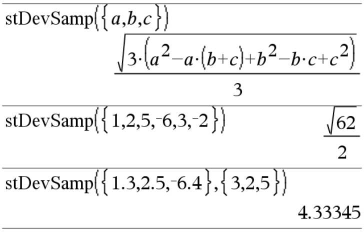
stDevSamp() Catalog >

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 234.



stDevSamp$(\{a,b,c\})$ $\frac{\sqrt{3\cdot(a^2-a\cdot(b+c)+b^2-b\cdot c+c^2)}}{3}$

stDevSamp$(\{1,2,5,^-6,3,^-2\})$ $\frac{\sqrt{62}}{2}$

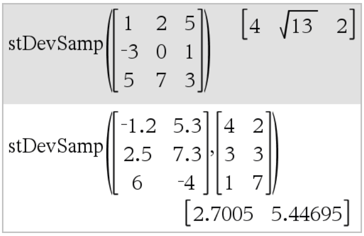
stDevSamp$(\{1.3,2.5,^-6.4\},\{3,2,5\})$ 4.33345

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 234.



stDevSamp$\left(\begin{bmatrix}1 & 2 & 5\\ -3 & 0 & 1\\ 5 & 7 & 3\end{bmatrix}\right)$ $\begin{bmatrix}4 & \sqrt{13} & 2\end{bmatrix}$

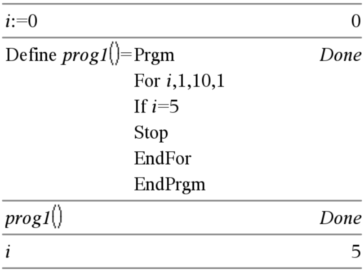
stDevSamp$\left(\begin{bmatrix}-1.2 & 5.3\\ 2.5 & 7.3\\ 6 & -4\end{bmatrix},\begin{bmatrix}4 & 2\\ 3 & 3\\ 1 & 7\end{bmatrix}\right)$ $\begin{bmatrix}2.7005 & 5.44695\end{bmatrix}$

Stop Catalog >

Stop

Programming command: Terminates the program.

Stop is not allowed in functions.



```
i:=0                                  0
Define prog1()=Prgm                Done
               For i,1,10,1
               If i=5
               Stop
               EndFor
               EndPrgm
prog1()                            Done
i                                     5
```

Stop 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.

Store See →(store), page 231.

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(*Matrix1***[,** *startRow***][,** *startCol***][,** *endRow***][,** *endCol***])** ⇒ *matrix*

Returns the specified submatrix of *Matrix1*.

Defaults: *startRow*=1, *startCol*=1, *endRow*=last row, *endCol*=last column.

$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\to m1$	$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$
$\text{subMat}(m1,2,1,3,2)$	$\begin{bmatrix}4&5\\7&8\end{bmatrix}$
$\text{subMat}(m1,2,2)$	$\begin{bmatrix}5&6\\8&9\end{bmatrix}$

Sum (Sigma) See Σ(), page 222.

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 234.

$\text{sum}(\{1,2,3,4,5\})$	15
$\text{sum}(\{a,2\cdot a,3\cdot a\})$	$6\cdot a$
$\text{sum}(\text{seq}(n,n,1,10))$	55
$\text{sum}(\{1,3,5,7,9\},3)$	21

sum() Catalog >

sum(*Matrix1*[**,** *Start*[**,** *End*]]**)** ⇒ *matrix*

Returns a row vector containing the sums of all elements in the columns in *Matrix1*.

Start and *End* are optional. They specify a range of rows.

Any void argument produces a void result. Empty (void) elements in *Matrix1* are ignored. For more information on empty elements, see page 234.

$\mathrm{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}\right)$ $\begin{bmatrix}5&7&9\end{bmatrix}$

$\mathrm{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}\right)$ $\begin{bmatrix}12&15&18\end{bmatrix}$

$\mathrm{sum}\left(\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix},2,3\right)$ $\begin{bmatrix}11&13&15\end{bmatrix}$

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.

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*.

$\mathrm{sumIf}(\{1,2,e,3,\pi,4,5,6\},2.5<?<4.5)$

$e+\pi+7$

$\mathrm{sumIf}(\{1,2,3,4\},2<?<5,\{10,20,30,40\})$

70

sumIf() Catalog >

Empty (void) elements are ignored. For more information on empty elements, see page 234.

Note: See also **countIf()**, page 39.

sumSeq() See Σ(), page 222.

system() Catalog >

system(*Eqn1*[**,** *Eqn2*[**,** *Eqn3*[**,** ...]]]**)**

system(*Expr1*[**,** *Expr2*[**,** *Expr3*[**,** ...]]]**)**

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 7.

$\text{solve}\left(\begin{cases} x+y=0 \\ x-y=8 \end{cases},x,y\right)$	$x=4 \text{ and } y=^{-}4$

T

T (transpose) Catalog >

*Matrix1***T** ⇒ *matrix*

Returns the complex conjugate transpose of *Matrix1*.

Note: You can insert this operator from the computer keyboard by typing `@t`.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^{\mathsf{T}}$	$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{\mathsf{T}}$	$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$
$\begin{bmatrix} 1+i & 2+i \\ 3+i & 4+i \end{bmatrix}^{\mathsf{T}}$	$\begin{bmatrix} 1-i & 3-i \\ 2-i & 4-i \end{bmatrix}$

tan() trig key

tan(*Expr1***)** ⇒ *expression*

tan(*List1***)** ⇒ *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*.

In Degree angle mode:

$\tan\left(\left(\frac{\pi}{4}\right)^{r}\right)$	1
$\tan(45)$	1
$\tan(\{0,60,90\})$	$\{0,\sqrt{3},\text{undef}\}$

tan() trig key

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 Gradian angle mode:

$\tan\left(\frac{\pi}{4}{}^{r}\right)$	1
$\tan(50)$	1
$\tan(\{0,50,100\})$	$\{0,1,\text{undef}\}$

In Radian angle mode:

$\tan\left(\frac{\pi}{4}\right)$	1
$\tan(45°)$	1
$\tan\left(\left\{\pi,\frac{\pi}{3},{}^{-}\pi,\frac{\pi}{4}\right\}\right)$	$\{0,\sqrt{3},0,1\}$

tan(*squareMatrix1***)** ⇒ *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()**.

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

In Radian angle mode:

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

$$\begin{bmatrix}{}^{-}28.2912 & 26.0887 & 11.1142\\ 12.1171 & {}^{-}7.83536 & {}^{-}5.48138\\ 36.8181 & {}^{-}32.8063 & {}^{-}10.4594\end{bmatrix}$$

$\tan^{-1}$() trig key

$\tan^{-1}$(*Expr1***)** ⇒ *expression*

$\tan^{-1}$(*List1***)** ⇒ *list*

$\tan^{-1}$(*Expr1***)** returns the angle whose tangent is *Expr1* as an expression.

$\tan^{-1}$(*List1***)** returns a list of the inverse tangents 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 `arctan(...)`.

In Degree angle mode:

$\tan^{-1}(1)$	45

In Gradian angle mode:

$\tan^{-1}(1)$	50

In Radian angle mode:

$\tan^{-1}(\{0,0.2,0.5\})$	$\{0,0.197396,0.463648\}$

$\tan^{-1}$(*squareMatrix1***)** ⇒ *squareMatrix*

In Radian angle mode:

$\tan^{-1}()$ trig key

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()**.

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

$$\tan^{-1}\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & {}^{-}2 & 1\end{bmatrix}\right)$$
$$\begin{bmatrix}{}^{-}0.083658 & 1.26629 & 0.62263\\ 0.748539 & 0.630015 & {}^{-}0.070012\\ 1.68608 & {}^{-}1.18244 & 0.455126\end{bmatrix}$$

tangentLine() Catalog >

tangentLine(*Expr1***,***Var***,***Point***)** ⇒ *expression*

tangentLine(*Expr1***,***Var***=***Point***)** ⇒ *expression*

Returns the tangent line to the curve represented by *Expr1* at the point specified in *Var=Point*.

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.”

$\text{tangentLine}(x^{2},x,1)$	$2\cdot x-1$
$\text{tangentLine}((x-3)^{2}-4,x=3)$	${}^{-}4$
$\text{tangentLine}\left(x^{\frac{1}{3}},x=0\right)$	$x=0$
$\text{tangentLine}\left(\sqrt{x^{2}-4},x=2\right)$	undef
$x:=3\text{: tangentLine}(x^{2},x,1)$	5

tanh() Catalog >

tanh(*Expr1***)** ⇒ *expression*

tanh(*List1***)** ⇒ *list*

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*.

$\tanh(1.2)$	0.833655
$\tanh(\{0,1\})$	$\{0,\tanh(1)\}$

tanh(*squareMatrix1***)** ⇒ *squareMatrix*

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()**.

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

In Radian angle mode:

$$\tanh\left(\begin{bmatrix}1 & 5 & 3\\ 4 & 2 & 1\\ 6 & {}^{-}2 & 1\end{bmatrix}\right)$$
$$\begin{bmatrix}{}^{-}0.097966 & 0.933436 & 0.425972\\ 0.488147 & 0.538881 & {}^{-}0.129382\\ 1.28295 & {}^{-}1.03425 & 0.428817\end{bmatrix}$$

tanh^{-1}() **Catalog >**

tanh^{-1}(*Expr1***)** ⇒ *expression*

tanh^{-1}(*List1***)** ⇒ *list*

tanh^{-1}(*Expr1***)** returns the inverse hyperbolic tangent of the argument as an expression.

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(...)`.

In Rectangular complex format:

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

$\tanh^{-1}(\{1,2.1,3\})$

$$\left\{\text{undef},0.518046-1.5708\cdot \boldsymbol{i},\frac{\ln(2)}{2}-\frac{\pi}{2}\cdot \boldsymbol{i}\right\}$$

tanh^{-1}(*squareMatrix1***)** ⇒ *squareMatrix*

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.

In Radian angle mode and Rectangular complex format:

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

$$\begin{bmatrix} {}^{-}0.099353+0.164058\cdot \boldsymbol{i} & 0.267834-1.4908 \\ {}^{-}0.087596-0.725533\cdot \boldsymbol{i} & 0.479679-0.94730 \blacktriangleright \\ 0.511463-2.08316\cdot \boldsymbol{i} & {}^{-}0.878563+1.7901 \end{bmatrix}$$

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

taylor() **Catalog >**

taylor(*Expr1***,** *Var***,** *Order*[**,** *Point*]**)** ⇒ *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.

$\text{taylor}\left(e^{\sqrt{x}},x,2\right)$ $\quad \text{taylor}\left(e^{\sqrt{x}},x,2,0\right)$

$\text{taylor}\left(e^{t},t,4\right)|t=\sqrt{x}$

$$\frac{x^2}{24}+\frac{x^{\frac{3}{2}}}{6}+\frac{x}{2}+\sqrt{x}+1$$

$\text{taylor}\left(\frac{1}{x\cdot(x-1)},x,3\right)$ $\quad \text{taylor}\left(\frac{1}{x\cdot(x-1)},x,3,0\right)$

$$\text{expand}\left(\frac{\text{taylor}\left(\frac{x}{x\cdot(x-1)},x,4\right)}{x},x\right)$$

$${}^{-}x^3-x^2-x-\frac{1}{x}-1$$

tCdf() Catalog >

tCdf(*lowBound***,***upBound***,***df***)** ⇒ *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 ≤ *upBound*), set *lowBound* = $^{-}\infty$.

tCollect() Catalog >

tCollect(*Expr1***)** ⇒ *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.

$\mathrm{tCollect}\left((\cos(\alpha))^2\right)$ $\quad \dfrac{\cos(2\cdot\alpha)+1}{2}$

$\mathrm{tCollect}\left(\sin(\alpha)\cdot\cos(\beta)\right)$ $\quad \dfrac{\sin(\alpha-\beta)+\sin(\alpha+\beta)}{2}$

tExpand() Catalog >

tExpand(*Expr1***)** ⇒ *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.

$\mathrm{tExpand}\left(\sin(3\cdot\varphi)\right)$ $\quad 4\cdot\sin(\varphi)\cdot(\cos(\varphi))^2-\sin(\varphi)$

$\mathrm{tExpand}\left(\cos(\alpha-\beta)\right)$

$\cos(\alpha)\cdot\cos(\beta)+\sin(\alpha)\cdot\sin(\beta)$

tExpand() Catalog >

Note: Degree-mode scaling by π/180 interferes with the ability of **tExpand()** to recognize expandable forms. For best results, **tExpand()** should be used in Radian mode.

Text Catalog >

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 148, or **RequestStr**, page 149.

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.

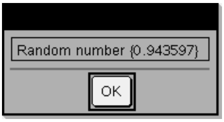
Within the Prgm...EndPrgm template, complete each line by pressing ↵ instead of enter. On the computer keyboard, hold down **Alt** and press **Enter**.

```
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:



Random number {0.943597}

Then See If, page 86.

tInterval Catalog >

tInterval *List*[**,** *Freq*[**,** *CLevel*]]

(Data list input)

tInterval Catalog >

tInterval $\bar{x}$**,** *sx*, *n***[,** *CLevel***]**

(Summary stats input)

Computes a *t* confidence interval. A summary of results is stored in the *stat.results* variable. (See page 175.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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.σx	Sample standard deviation
stat.n	Length of the data sequence with sample mean

tInterval_2Samp Catalog >

tInterval_2Samp *List1***,***List2*[**,***Freq1*[**,***Freq2* [**,***CLevel*[**,***Pooled*]]]]

(Data list input)

tInterval_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 175.)

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 234.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat.$\bar{x}$1-$\bar{x}$2	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom
stat.$\bar{x}$1, stat.$\bar{x}$2	Sample means of the data sequences from the normal random distribution
stat.σx1, stat.σx2	Sample standard deviations for *List 1* and *List 2*
stat.n1, stat.n2	Number of samples in data sequences
stat.sp	The pooled standard deviation. Calculated when *Pooled* = YES

tmpCnv() Catalog >

tmpCnv(*Expr*_°*tempUnit***,** _°*tempUnit2***)** ⇒ *expression* _°*tempUnit2*

Converts a temperature value specified by *Expr* from one unit to another. Valid temperature units are:

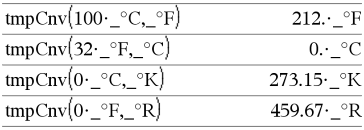
_°C Celsius
_°F Fahrenheit
_°K Kelvin
_°R Rankine

To type °, select it from the Catalog symbols.

to type _ , press ctrl ⌴.

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***)** ⇒ *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

To enter °, select it from the Symbol Palette or type `@d`.

To type _ , press ctrl ␣.

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()**.

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.

tPdf() Catalog >

tPdf(*XVal*,*df***)** ⇒ *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***)** ⇒ *expression*

Returns the trace (sum of all the elements on the main diagonal) of *squareMatrix*.

$\mathrm{trace}\left(\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}\right)$	15
$\mathrm{trace}\left(\begin{bmatrix}a & 0\\1 & a\end{bmatrix}\right)$	$2\cdot a$

Try Catalog >

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 248.

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.

```
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
```

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.

$$eigenvals\left(\begin{bmatrix} {}^{-}3 \\ {}^{-}41 \\ 5 \end{bmatrix},\begin{bmatrix} {}^{-}1 & 2 & {}^{-}3.1 \end{bmatrix}\right)$$

$$eigenvals\left(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix},\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

Note: See also **ClrErr**, page 29, and **PassErr**, page 130.

```
Define eigenvals(a,b)=Prgm
© Program eigenvals(A,B) displays
eigenvalues of A•B

Try
  Disp "A= ",a
  Disp "B= ",b
  Disp " "

  Disp "Eigenvalues of A•B are:",eigVl(a*b)

Else
  If errCode=230 Then
    Disp "Error: Product of A•B must be a
square matrix"
    ClrErr
  Else
    PassErr
  EndIf
EndTry

EndPrgm
```

tTest Catalog >

tTest μ*0*,*List*[,*Freq*[,*Hypoth*]]

(Data list input)

tTest μ*0*,x̄,*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 175.)

Test H_0: μ = μ0, against one of the following:

For H_a: μ < μ0, set *Hypoth*<0
For H_a: μ ≠ μ0 (default), set *Hypoth*=0
For H_a: μ > μ0, set *Hypoth*>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

Output variable	Description
stat.t	(x̄ − μ0) / (stdev / sqrt(n))
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom
stat.x̄	Sample mean of the data sequence in *List*
stat.sx	Sample standard deviation of the data sequence
stat.n	Size of the sample

tTest_2Samp Catalog >

tTest_2Samp *List1*,*List2*[,*Freq1*[,*Freq2*
[,*Hypoth*[,*Pooled*]]]]

(Data list input)

tTest_2Samp x̄*1*,*sx1*,*n1*,x̄*2*,*sx2*,*n2*[,*Hypoth*
[,*Pooled*]]

(Summary stats input)

tTest_2Samp **Catalog >**

Computes a two-sample *t* test. A summary of results is stored in the *stat.results* variable. (See page 175.)

Test H_0: μ1 = μ2, against one of the following:

For H_a: μ1< μ2, set *Hypoth*<0
For H_a: μ1≠ μ2 (default), set *Hypoth*=0
For H_a: μ1> μ2, set *Hypoth*>0

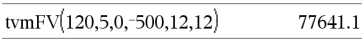
Pooled=**1** pools variances
Pooled=**0** does not pool variances

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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.$\bar{x}$1, stat.$\bar{x}$2	Sample means of the data sequences in *List 1* and *List 2*
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in *List 1* and *List 2*
stat.n1, stat.n2	Size of the samples
stat.sp	The pooled standard deviation. Calculated when *Pooled*=1.

tvmFV() **Catalog >**

tvmFV(*N*,*I*,*PV*,*Pmt*,[*PpY*],[*CpY*],[*PmtAt*]**)** ⇒ *value*

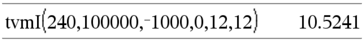

tvmFV(120,5,0,⁻500,12,12)   77641.1

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 193. See also **amortTbl()**, page 12.

tvmI() **Catalog >**

tvmI(*N*,*PV*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)** ⇒ *value*


tvmI(240,100000,⁻1000,0,12,12)   10.5241

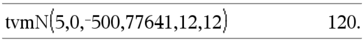
tvmI() Catalog >

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 193. See also **amortTbl()**, page 12.

tvmN() Catalog >

tvmN(*I*,*PV*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)** ⇒ *value*



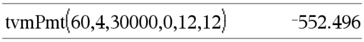
```
tvmN(5,0,⁻500,77641,12,12)          120.
```

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 193. See also **amortTbl()**, page 12.

tvmPmt() Catalog >

tvmPmt(*N*,*I*,*PV*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)** ⇒ *value*



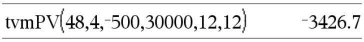
```
tvmPmt(60,4,30000,0,12,12)      ⁻552.496
```

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 193. See also **amortTbl()**, page 12.

tvmPV() Catalog >

tvmPV(*N*,*I*,*Pmt*,*FV*,[*PpY*],[*CpY*],[*PmtAt*]**)** ⇒ *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 193. See also **amortTbl()**, page 12.

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 175.)

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.

TwoVar Catalog >

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 234.

Output variable	Description
stat.$\bar{x}$	Mean of x values
stat.Σx	Sum of x values
stat.Σx2	Sum of x2 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 y2 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_1X	1st Quartile of x
stat.MedianX	Median of x
stat.Q_3X	3rd Quartile of x
stat.MaxX	Maximum of x values
stat.MinY	Minimum of y values
stat.Q_1Y	1st Quartile of y
stat.MedY	Median of y
stat.Q_3Y	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

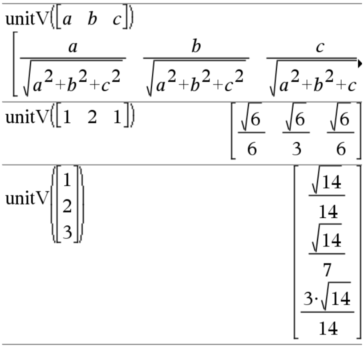
U

unitV() Catalog >

unitV(*Vector1***)** ⇒ *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]) $\left[\frac{a}{\sqrt{a^2+b^2+c^2}} \quad \frac{b}{\sqrt{a^2+b^2+c^2}} \quad \frac{c}{\sqrt{a^2+b^2+c}}\right.$ ▸

unitV([1 2 1]) $\begin{bmatrix}\frac{\sqrt{6}}{6} & \frac{\sqrt{6}}{3} & \frac{\sqrt{6}}{6}\end{bmatrix}$

unitV$\left(\begin{bmatrix}1\\2\\3\end{bmatrix}\right)$ $\begin{bmatrix}\frac{\sqrt{14}}{14}\\ \frac{\sqrt{14}}{7}\\ \frac{3\cdot\sqrt{14}}{14}\end{bmatrix}$

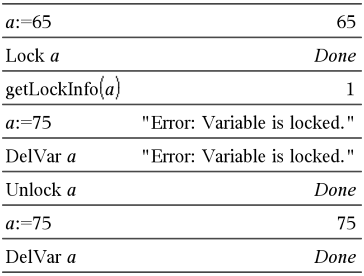
To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

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 105, and **getLockInfo()**, page 82.



```
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*

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 234.

$\mathrm{varPop}(\{5,10,15,20,25,30\})$ $\dfrac{875}{12}$

$Ans\cdot 1.$ 72.9167

varSamp() Catalog >

varSamp(*List*[**,***freqList*]**)** ⇒ *expression*

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 234.

$\mathrm{varSamp}(\{a,b,c\})$

$\dfrac{a^2-a\cdot(b+c)+b^2-b\cdot c+c^2}{3}$

$\mathrm{varSamp}(\{1,2,5,^{-}6,3,^{-}2\})$ $\dfrac{31}{2}$

$\mathrm{varSamp}(\{1,3,5\},\{4,6,2\})$ $\dfrac{68}{33}$

varSamp(*Matrix1*[**,***freqMatrix*]**)** ⇒ *matrix*

Returns a row vector containing the sample variance of each column in *Matrix1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

$\mathrm{varSamp}\left(\begin{bmatrix}1 & 2 & 5\\ ^{-}3 & 0 & 1\\ .5 & .7 & 3\end{bmatrix}\right)$ $\begin{bmatrix}4.75 & 1.03 & 4\end{bmatrix}$

$\mathrm{varSamp}\left(\begin{bmatrix}^{-}1.1 & 2.2\\ 3.4 & 5.1\\ ^{-}2.3 & 4.3\end{bmatrix},\begin{bmatrix}6 & 3\\ 2 & 4\\ 5 & 1\end{bmatrix}\right)$

$\begin{bmatrix}3.91731 & 2.08411\end{bmatrix}$

varSamp() Catalog >

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 234.

Note: *Matrix1* must contain at least two rows.

W

Wait Catalog >

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 ⌂on key and press enter 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 >

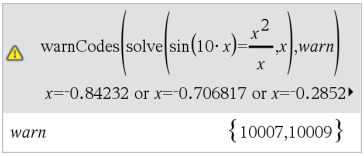
warnCodes(*Expr1*, *StatusVar***)** ⇒ *expression*

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 248.



⚠ $\text{warnCodes}\left(\text{solve}\left(\sin(10\cdot x)=\frac{x^2}{x},x\right),warn\right)$

$x=^-0.84232$ or $x=^-0.706817$ or $x=^-0.2852$▸

warn	{10007,10009}

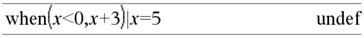
To see the entire result, press ▲ and then use ◀ and ▶ to move the cursor.

when() Catalog >

when(*Condition*, *trueResult* [**,** *falseResult*] [**,** *unknownResult*]**)** ⇒ *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.

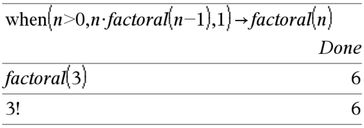
Omit both *falseResult* and *unknownResult* to make an expression defined only in the region where *Condition* is true.



$\text{when}(x<0,x+3)\mid x=5$	undef

Use an **undef** *falseResult* to define an expression that graphs only on an interval.

when() is helpful for defining recursive functions.



$\text{when}(n>0,n\cdot factoral(n-1),1)\to factoral(n)$

Done

$factoral(3)$	6
3!	6

While Catalog >

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
                       $tempsum+\frac{1}{i}\rightarrow tempsum$
                       i+1→i
                       EndWhile
                       Return tempsum
                       EndFunc
                                                    Done
```

sum_of_recip(3)	$\frac{11}{6}$

X

xor Catalog >

BooleanExpr1 **xor** *BooleanExpr2* returns *Boolean expressionBooleanList1* **xor** *BooleanList2* returns *Boolean listBooleanMatrix1* **xor** *BooleanMatrix2* returns *Boolean matrix*

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 128.**

true xor true	false
5>3 xor 3>5	true

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

xor Catalog >

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 21.

Note: See **or**, page 128.

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

Z

zeros() Catalog >

zeros(*Expr*, *Var***)** ⇒ *list*

zeros(*Expr*, *Var=Guess***)** ⇒ *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***)**.

$$\mathrm{zeros}\left(a\cdot x^2+b\cdot x+c,x\right)$$

$$\left\{\frac{\sqrt{b^2-4\cdot a\cdot c}-b}{2\cdot a},\frac{-\left(\sqrt{b^2-4\cdot a\cdot c}+b\right)}{2\cdot a}\right\}$$

$$a\cdot x^2+b\cdot x+c|x=Ans[2] \qquad 0$$

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*.

$$\mathrm{exact}\left(\mathrm{zeros}\left(a\cdot\left(e^x+x\right)\cdot\left(\mathrm{sign}(x)-1\right),x\right)\right) \qquad \{\square\}$$

$$\mathrm{exact}\left(\mathrm{solve}\left(a\cdot\left(e^x+x\right)\cdot\left(\mathrm{sign}(x)-1\right)=0,x\right)\right)$$

$$e^x+x=0 \text{ or } x>0 \text{ or } a=0$$

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

zeros({*Expr1*, *Expr2***}, {***VarOrGuess1*, *VarOrGuess2* [**,** …]**})** ⇒ *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:

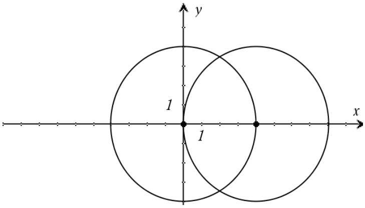
zeros() Catalog >

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.

$$\text{zeros}\left(\left\{x^2+y^2-r^2,(x-r)^2+y^2-r^2\right\},\{x,y\}\right)$$
$$\begin{bmatrix}\frac{r}{2} & \frac{^{-}\sqrt{3}\cdot r}{2}\\ \frac{r}{2} & \frac{\sqrt{3}\cdot r}{2}\end{bmatrix}$$

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*].

Extract row 2:

$$Ans[2] \qquad \begin{bmatrix}\frac{r}{2} & \frac{\sqrt{3}\cdot r}{2}\end{bmatrix}$$

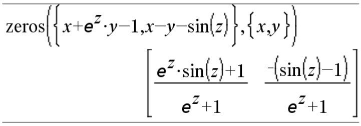
You can also (or instead) include unknowns that do not appear in the expressions. For example, you can include z as an unknown to extend the previous example to two parallel intersecting cylinders of radius r. The cylinder zeros illustrate how families of zeros might contain arbitrary constants in the form ck, where k is an integer suffix from 1 through 255.

$$\text{zeros}\left(\left\{x^2+y^2-r^2,(x-r)^2+y^2-r^2\right\},\{x,y,z\}\right)$$
$$\begin{bmatrix}\frac{r}{2} & \frac{^{-}\sqrt{3}\cdot r}{2} & \boldsymbol{c}1\\ \frac{r}{2} & \frac{\sqrt{3}\cdot r}{2} & \boldsymbol{c}1\end{bmatrix}$$

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.

zeros() Catalog >

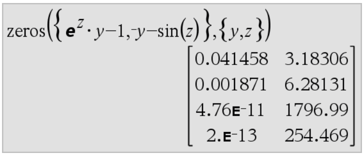
If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in the unknowns, **zeros()** uses Gaussian elimination to attempt to determine all real zeros.



$$\text{zeros}\left(\left\{x+e^{z}\cdot y-1,x-y-\sin(z)\right\},\left\{x,y\right\}\right)$$

$$\begin{bmatrix}\frac{e^{z}\cdot\sin(z)+1}{e^{z}+1} & \frac{-(\sin(z)-1)}{e^{z}+1}\end{bmatrix}$$

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.

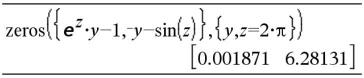


$$\text{zeros}\left(\left\{e^{z}\cdot y-1,^{-}y-\sin(z)\right\},\left\{y,z\right\}\right)$$

$$\begin{bmatrix}0.041458 & 3.18306\\ 0.001871 & 6.28131\\ 4.76\text{E-}11 & 1796.99\\ 2.\text{E-}13 & 254.469\end{bmatrix}$$

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.



$$\text{zeros}\left(\left\{e^{z}\cdot y-1,^{-}y-\sin(z)\right\},\left\{y,z=2\cdot\pi\right\}\right)$$

$$\begin{bmatrix}0.001871 & 6.28131\end{bmatrix}$$

zInterval Catalog >

zInterval σ,*List*[,*Freq*[,*CLevel*]]

(Data list input)

zInterval σ,$\bar{x}$,*n* [,*CLevel*]

(Summary stats input)

Computes a *z* confidence interval. A summary of results is stored in the *stat.results* variable. (See page 175.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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 *List*

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 175.)

x is a non-negative integer.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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 175.)

x1 and *x2* are non-negative integers.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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 σ_1**,**σ_2 **,***List1***,***List2*[**,***Freq1*
[**,***Freq2***,**[*CLevel*]]]

(Data list input)

zInterval_2Samp σ_1**,**σ_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 175.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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.σx1, stat.σx2	Sample standard deviations for *List 1* and *List 2*
stat.n1, stat.n2	Number of samples in data sequences
stat.r1, stat.r2	Known population standard deviations for data sequence *List 1* and *List 2*

zTest

Catalog >

zTest $\mu 0$**,**σ**,***List***,**[*Freq*[**,***Hypoth*]]

zTest — Catalog > 📖

(Data list input)

zTest $\mu 0$,σ,$\bar{x}$,*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 175.)

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 234.

Output variable	**Description**
stat.z	$(\bar{x} - \mu 0) / (\sigma / \text{sqrt}(n))$
stat.P Value	Least probability at which the null hypothesis can be rejected
stat.$\bar{x}$	Sample mean of the data sequence in *List*
stat.sx	Sample standard deviation of the data sequence. Only returned for *Data* input.
stat.n	Size of the sample

zTest_1Prop — Catalog > 📖

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 — Catalog > 📖

zTest_2Prop *x1*,*n1*,*x2*,*n2*[,*Hypoth*]

zTest_2Prop Catalog >

Computes a two-proportion *z* test. A summary of results is stored in the *stat.results* variable. (See page 175.)

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* ≠ *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 234.

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 Catalog >

zTest_2Samp σ_1**,**σ_2 **,***List1***,***List2*[**,***Freq1* [**,***Freq2*[**,***Hypoth*]]]

(Data list input)

zTest_2Samp σ_1**,**σ_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 175.)

Test H_0: μ1 = μ2, against one of the following:

For H_a: μ1 < μ2, set *Hypoth*<0
For H_a: μ1 ≠ μ2 (default), set *Hypoth*=0
For H_a: μ1 > μ2, *Hypoth*>0

zTest_2Samp Catalog >

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 234.

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 *List1* and *List2*
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in *List1* and *List2*
stat.n1, stat.n2	Size of the samples

Symbols

+ (add) $\boxed{+}$ key

Expr1 **+** *Expr2* ⇒ *expression*

Returns the sum of the two arguments.

56	56
56+4	60
60+4	64
64+4	68
68+4	72

List1 **+** *List2* ⇒ *list*

Matrix1 **+** *Matrix2* ⇒ *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.

$\left\{22,\pi,\frac{\pi}{2}\right\}\to l1$	$\left\{22,\pi,\frac{\pi}{2}\right\}$
$\left\{10,5,\frac{\pi}{2}\right\}\to l2$	$\left\{10,5,\frac{\pi}{2}\right\}$
$l1+l2$	$\{32,\pi+5,\pi\}$
$Ans+\{\pi,^{-}5,^{-}\pi\}$	$\{\pi+32,\pi,0\}$
$\begin{bmatrix}a & b\\ c & d\end{bmatrix}+\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}a+1 & b\\ c & d+1\end{bmatrix}$

Expr **+** *List1* ⇒ *list*

List1 **+** *Expr* ⇒ *list*

Returns a list containing the sums of *Expr* and each element in *List1*.

$15+\{10,15,20\}$	$\{25,30,35\}$
$\{10,15,20\}+15$	$\{25,30,35\}$

Expr **+** *Matrix1* ⇒ *matrix*

Matrix1 **+** *Expr* ⇒ *matrix*

Returns a matrix with *Expr* added to each element on the diagonal of *Matrix1*. *Matrix1* must be square.

Note: Use **.+** (dot plus) to add an expression to each element.

$20+\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$	$\begin{bmatrix}21 & 2\\ 3 & 24\end{bmatrix}$

− (subtract) $\boxed{-}$ key

Expr1 − *Expr2* ⇒ *expression*

Returns *Expr1* minus *Expr2*.

6−2	4
$\pi-\frac{\pi}{6}$	$\frac{5\cdot\pi}{6}$

List1 −*List2*⇒ *list*

Matrix1 −*Matrix2* ⇒ *matrix*

$\left\{22,\pi,\frac{\pi}{2}\right\}-\left\{10,5,\frac{\pi}{2}\right\}$	$\{12,\pi-5,0\}$
$\begin{bmatrix}3 & 4\end{bmatrix}-\begin{bmatrix}1 & 2\end{bmatrix}$	$\begin{bmatrix}2 & 2\end{bmatrix}$

− (subtract) — $\boxed{-}$ key

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.

Expr − *List1* ⇒ *list*

List1 − *Expr* ⇒ *list*

$15-\{10,15,20\}$	$\{5,0,^-5\}$
$\{10,15,20\}-15$	$\{^-5,0,5\}$

Subtracts each *List1* element from *Expr* or subtracts *Expr* from each *List1* element, and returns a list of the results.

Expr − *Matrix1* ⇒ *matrix*

Matrix1 − *Expr* ⇒ *matrix*

$20-\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$	$\begin{bmatrix}19 & ^-2\\ ^-3 & 16\end{bmatrix}$

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) — $\boxed{\times}$ key

Expr1•*Expr2* ⇒ *expression*

Returns the product of the two arguments.

$2\cdot 3.45$	6.9
$x\cdot y\cdot x$	$x^2\cdot y$

List1•*List2* ⇒ *list*

Returns a list containing the products of the corresponding elements in *List1* and *List2*.

Dimensions of the lists must be equal.

$\{1.,2,3\}\cdot\{4,5,6\}$	$\{4.,10,18\}$
$\left\{\frac{2}{a},\frac{3}{2}\right\}\cdot\left\{a^2,\frac{b}{3}\right\}$	$\left\{2\cdot a,\frac{b}{2}\right\}$

Matrix1•*Matrix2* ⇒ *matrix*

Returns the matrix product of *Matrix1* and *Matrix2*.

The number of columns in *Matrix1* must equal the number of rows in *Matrix2*.

$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\cdot\begin{bmatrix}a & d\\ b & e\\ c & f\end{bmatrix}$	
	$\begin{bmatrix}a+2\cdot b+3\cdot c & d+2\cdot e+3\cdot f\\ 4\cdot a+5\cdot b+6\cdot c & 4\cdot d+5\cdot e+6\cdot f\end{bmatrix}$

• (multiply) ☒ key

Expr •List1 ⇒ *list*

List1•Expr ⇒ *list*

Returns a list containing the products of *Expr* and each element in *List1*.

$\pi\cdot\{4,5,6\}$	$\{4\cdot\pi,5\cdot\pi,6\cdot\pi\}$

Expr •Matrix1 ⇒ *matrix*

Matrix1•Expr ⇒ *matrix*

Returns a matrix containing the products of *Expr* and each element in *Matrix1*.

$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\cdot 0.01$	$\begin{bmatrix}0.01 & 0.02\\0.03 & 0.04\end{bmatrix}$
$\lambda\cdot\text{identity}(3)$	$\begin{bmatrix}\lambda & 0 & 0\\0 & \lambda & 0\\0 & 0 & \lambda\end{bmatrix}$

Note: Use **.•**(dot multiply) to multiply an expression by each element.

⁄(divide) ÷ key

Expr1 ⁄ Expr2 ⇒ *expression*

Returns the quotient of *Expr1* divided by *Expr2*.

Note: See also **Fraction template**, page 5.

$\frac{2}{3.45}$	$.57971$
$\frac{x^3}{x}$	x^2

List1 ⁄ List2 ⇒ *list*

Returns a list containing the quotients of *List1* divided by *List2*.

Dimensions of the lists must be equal.

$\frac{\{1.,2,3\}}{\{4,5,6\}}$	$\left\{0.25,\frac{2}{5},\frac{1}{2}\right\}$

Expr ⁄ List1 ⇒ *list*

List1 ⁄ Expr ⇒ *list*

Returns a list containing the quotients of *Expr* divided by *List1* or*List1* divided by *Expr*.

$\frac{a}{\{3,a,\sqrt{a}\}}$	$\left\{\frac{a}{3},1,\sqrt{a}\right\}$
$\frac{\{a,b,c\}}{a\cdot b\cdot c}$	$\left\{\frac{1}{b\cdot c},\frac{1}{a\cdot c},\frac{1}{a\cdot b}\right\}$

Matrix1 ⁄ Expr ⇒ *matrix*

Returns a matrix containing the quotients of *Matrix1 ⁄ Expr*.

$\frac{\begin{bmatrix}a & b & c\end{bmatrix}}{a\cdot b\cdot c}$	$\begin{bmatrix}\frac{1}{b\cdot c} & \frac{1}{a\cdot c} & \frac{1}{a\cdot b}\end{bmatrix}$

Matrix1 ⁄ Value ⇒ *matrix*

∕(divide) ÷ key

Note: Use **.**∕ (dot divide) to divide an expression by each element.

^ (power) ^ key

Expr1 **^** *Expr2*⇒ *expression*

List1 **^** *List2* ⇒ *list*

4^2	16
$\{a,2,c\}^{\{1,b,3\}}$	$\{a,2^b,c^3\}$

Returns the first argument raised to the power of the second argument.

Note: See also **Exponent template**, page 5.

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 **^** *List1* ⇒ *list*

$p^{\{a,2,{}^{-}3\}}$	$\left\{p^a,p^2,\frac{1}{p^3}\right\}$

Returns *Expr* raised to the power of the elements in *List1*.

List1 **^** *Expr* ⇒ *list*

$\{1,2,3,4\}^{{}^{-}2}$	$\left\{1,\frac{1}{4},\frac{1}{9},\frac{1}{16}\right\}$

Returns the elements in *List1* raised to the power of *Expr*.

squareMatrix1 **^** *integer* ⇒ *matrix*

$\begin{bmatrix}1&2\\3&4\end{bmatrix}^2$	$\begin{bmatrix}7&10\\15&22\end{bmatrix}$
$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{{}^{-}1}$	$\begin{bmatrix}{}^{-}2&1\\\frac{3}{2}&\frac{{}^{-}1}{2}\end{bmatrix}$
$\begin{bmatrix}1&2\\3&4\end{bmatrix}^{{}^{-}2}$	$\begin{bmatrix}\frac{11}{2}&\frac{{}^{-}5}{2}\\\frac{{}^{-}15}{4}&\frac{7}{4}\end{bmatrix}$

Returns *squareMatrix1* raised to the *integer* power.

squareMatrix1 must be a square matrix.

If *integer* = −1, computes the inverse matrix.
If *integer* < −1, computes the inverse matrix to an appropriate positive power.

x^2 (square) $\boxed{x^2}$ key

Expr1^2⇒ *expression*

Returns the square of the argument.

List1^2 ⇒ *list*

Returns a list containing the squares of the elements in *List1*.

squareMatrix1^2 ⇒ *matrix*

Returns the matrix square of *squareMatrix1*. This is not the same as calculating the square of each element. Use .^2 to calculate the square of each element.

4^2	16
$\{2,4,6\}^2$	$\{4,16,36\}$
$\begin{bmatrix}2&4&6\\3&5&7\\4&6&8\end{bmatrix}^2$	$\begin{bmatrix}40&64&88\\49&79&109\\58&94&130\end{bmatrix}$
$\begin{bmatrix}2&4&6\\3&5&7\\4&6&8\end{bmatrix}.\hat{}\,2$	$\begin{bmatrix}4&16&36\\9&25&49\\16&36&64\end{bmatrix}$

.+ (dot add) $\boxed{.}\boxed{+}$ keys

Matrix1 **.+** *Matrix2* ⇒ *matrix*

Expr **.+** *Matrix1*⇒ *matrix*

*Matrix1***.+***Matrix2* returns a matrix that is the sum of each pair of corresponding elements in *Matrix1* and *Matrix2*.

Expr **.+** *Matrix1* returns a matrix that is the sum of *Expr* and each element in *Matrix1*.

$\begin{bmatrix}a&2\\b&3\end{bmatrix}.+\begin{bmatrix}c&4\\5&d\end{bmatrix}$	$\begin{bmatrix}a+c&6\\b+5&d+3\end{bmatrix}$
$x\ .+\begin{bmatrix}c&4\\5&d\end{bmatrix}$	$\begin{bmatrix}x+c&x+4\\x+5&x+d\end{bmatrix}$

. ⁻(dot subt.) $\boxed{.}\boxed{-}$ keys

Matrix1 **.–** *Matrix2*⇒ *matrix*

Expr **.–** *Matrix1* ⇒ *matrix*

*Matrix1***.–** *Matrix2* returns a matrix that is the difference between each pair of corresponding elements in *Matrix1* and *Matrix2*.

Expr **.–** *Matrix1* returns a matrix that is the difference of *Expr* and each element in *Matrix1*.

$\begin{bmatrix}a&2\\b&3\end{bmatrix}.-\begin{bmatrix}c&4\\d&5\end{bmatrix}$	$\begin{bmatrix}a-c&{}^{-}2\\b-d&{}^{-}2\end{bmatrix}$
$x\ .-\begin{bmatrix}c&4\\d&5\end{bmatrix}$	$\begin{bmatrix}x-c&x-4\\x-d&x-5\end{bmatrix}$

.

.•(dot mult.) . × keys

Matrix1 **.•** *Matrix2* ⇒ *matrix*

Expr **.•** *Matrix1* ⇒ *matrix*

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*.

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix} .\cdot \begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} a\cdot c & 8 \\ 5\cdot b & 3\cdot d \end{bmatrix}$
$x .\cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	$\begin{bmatrix} a\cdot x & b\cdot x \\ c\cdot x & d\cdot x \end{bmatrix}$

./(dot divide) . ÷ keys

Matrix1 **./** *Matrix2* ⇒ *matrix*

Expr **./** *Matrix1* ⇒ *matrix*

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*.

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix} ./\left(\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}\right)$	$\begin{bmatrix} \frac{a}{c} & \frac{1}{2} \\ \frac{b}{5} & \frac{3}{d} \end{bmatrix}$
$x ./\left(\begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}\right)$	$\begin{bmatrix} \frac{x}{c} & \frac{x}{4} \\ \frac{x}{5} & \frac{x}{d} \end{bmatrix}$

.^ (dot power) . ^ keys

Matrix1 **.^** *Matrix2* ⇒ *matrix*

Expr **. ^** *Matrix1* ⇒ *matrix*

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*.

$\begin{bmatrix} a & 2 \\ b & 3 \end{bmatrix} .\hat{} \begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} a^{c} & 16 \\ b^{5} & 3^{d} \end{bmatrix}$
$x .\hat{} \begin{bmatrix} c & 4 \\ 5 & d \end{bmatrix}$	$\begin{bmatrix} x^{c} & x^{4} \\ x^{5} & x^{d} \end{bmatrix}$

− (negate) (−) key

−*Expr1* ⇒ *expression*

−*List1* ⇒ *list*

−*Matrix1* ⇒ *matrix*

⁻2.43	⁻2.43
⁻{⁻1,0.4,1.2E19}	{1,⁻0.4,⁻1.2E19}
$^{-}a\cdot{}^{-}b$	$a\cdot b$

− (negate) (−) key

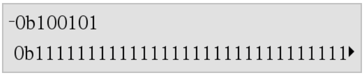
Returns the negation of the argument.

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
0b11111111111111111111111111▸

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

% (percent) ctrl 📖 keys

Expr1% ⇒ *expression*

List1% ⇒ *list*

Matrix1% ⇒ *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 ctrl enter.
Windows®: Press **Ctrl+Enter**.
Macintosh®: Press **⌘+Enter**.
iPad®: Hold **enter**, and select ≈.

13%	0.13

({1,10,100})%	{0.01,0.1,1.}

= (equal) = key

Expr1=*Expr2* ⇒ *Boolean expression*

List1=*List2* ⇒ *Boolean list*

Matrix1=*Matrix2* ⇒ *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.

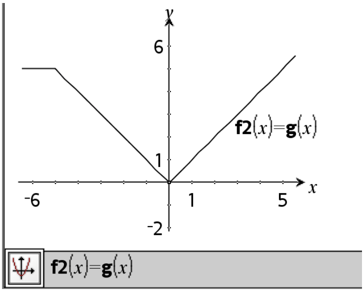
Example function that uses math test symbols: =, ≠, <, ≤, >, ≥

```
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
                                    Done
```

Result of graphing g(x)

= (equal) = key

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



≠ (not equal) ctrl = keys

Expr1≠Expr2 ⇒ *Boolean expression*

List1≠List2 ⇒ *Boolean list*

Matrix1≠Matrix2 ⇒ *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 **/=**

See "=" (equal) example.

< (less than) ctrl = keys

*Expr1***<***Expr2* ⇒ *Boolean expression*

*List1***<***List2* ⇒ *Boolean list*

*Matrix1***<***Matrix2* ⇒ *Boolean matrix*

Returns true if *Expr1* is determined to be less than *Expr2*.

Returns false if *Expr1* is determined to be greater than or equal to *Expr2*.

See "=" (equal) example.

< (less than) ctrl = keys

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≤ (less or equal) ctrl = keys

Expr1≤Expr2 ⇒ *Boolean expression*

List1≤List2 ⇒ *Boolean list*

Matrix1 ≤Matrix2 ⇒ *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 **<=**

See “=” (equal) example.

> (greater than) ctrl = keys

Expr1>*Expr2* ⇒ *Boolean expression*

List1>*List2* ⇒ *Boolean list*

Matrix1>*Matrix2* ⇒ *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.

For lists and matrices, returns comparisons element by element.

See “=” (equal) example.

≥ (greater or equal) ctrl = keys

Expr1≥Expr2 ⇒ *Boolean expression*

List1≥List2 ⇒ *Boolean list*

Matrix1 ≥Matrix2 ⇒ *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 **>=**

See “=” (equal) example.

⇒ (logical implication) ctrl = keys

BooleanExpr1 ⇒ *BooleanExpr2* returns *Boolean expression*

BooleanList1 ⇒ *BooleanList2* returns *Boolean list*

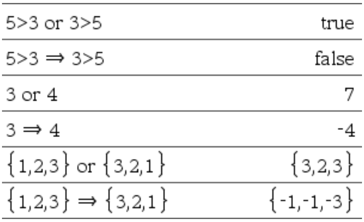
BooleanMatrix1 ⇒ *BooleanMatrix2* returns *Boolean matrix*

Integer1 ⇒ *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 **=>**



5>3 or 3>5	true
5>3 ⇒ 3>5	false
3 or 4	7
3 ⇒ 4	-4
{1,2,3} or {3,2,1}	{3,2,3}
{1,2,3} ⇒ {3,2,1}	{-1,-1,-3}

⇔ (logical double implication, XNOR) ctrl = keys

BooleanExpr1 ⇔ *BooleanExpr2* returns *Boolean expression*

BooleanList1 ⇔ *BooleanList2* returns *Boolean list*

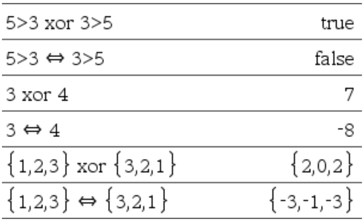
BooleanMatrix1 ⇔ *BooleanMatrix2* returns *Boolean matrix*

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 **<=>**



```
5>3 xor 3>5                      true
5>3 ⇔ 3>5                       false
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}
```

! (factorial) ?!▸ key

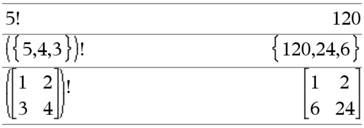
*Expr1***!** ⇒ *expression*

*List1***!** ⇒ *list*

*Matrix1***!** ⇒ *matrix*

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.



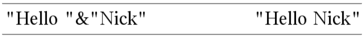
```
5!                                120
({5,4,3})!                 {120,24,6}
```

$$\left(\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\right)! \qquad \begin{bmatrix}1 & 2\\ 6 & 24\end{bmatrix}$$

& (append) ctrl 🕮 keys

String1 **&** *String2* ⇒ *string*

Returns a text string that is *String2* appended to *String1*.



```
"Hello "&"Nick"         "Hello Nick"
```

d() (derivative) Catalog >

d(*Expr1*, *Var*[, *Order*]) ⇒ *expression*

d(*List1*, *Var*[, *Order*]) ⇒ *list*

d(*Matrix1*,*Var*[, *Order*]) ⇒ *matrix*

$\frac{d}{dx}(f(x)\cdot g(x))$	$\frac{d}{dx}(f(x))\cdot g(x)+\frac{d}{dx}(g(x))\cdot f(x)$
$\frac{d}{dy}\left(\frac{d}{dx}(x^2\cdot y^3)\right)$	$6\cdot y^2\cdot x$
$\frac{d}{dx}(\{x^2,x^3,x^4\})$	$\{2\cdot x,3\cdot x^2,4\cdot x^3\}$

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 9; **Second derivative**, page 10; or **Nth derivative**, page 10.

∫() (integral) Catalog >

∫(*Expr1*, *Var*[,*Lower*,*Upper*]) ⇒ *expression*

∫(*Expr1*,*Var*[,*Constant*]) ⇒ *expression*

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

∫() (integral) Catalog >

Returns the integral of *Expr1* with respect to the variable *Var* from *Lower* to *Upper*.

Note: See also **Definite** or **Indefinite integral template**, page 10.

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.

$\int x^2 \mathrm{d}x$	$\frac{x^3}{3}$
$\int(a\cdot x^2,x,c)$	$\frac{a\cdot x^3}{3}+c$

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.

∫() returns itself for pieces of *Expr1* that it cannot determine as an explicit finite combination of its built-in functions and operators.

$\int b\cdot e^{-x^2}+\frac{a}{x^2+a^2}\mathrm{d}x$	$b\cdot\int e^{-x^2}\mathrm{d}x+\tan^{-1}\left(\frac{x}{a}\right)$

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 ≈.

∫() (integral) Catalog >

$$\int_{-1}^{1} e^{-x^2}\,dx \qquad 1.49365$$

∫() can be nested to do multiple integrals. Integration limits can depend on integration variables outside them.

Note: See also **nInt()**, page 121.

$$\int_{0}^{a}\int_{0}^{x} \ln(x+y)\,dy\,dx$$

$$\frac{a^2\cdot\ln(a)}{2}+\frac{a^2\cdot(4\cdot\ln(2)-3)}{4}$$

√() (square root) ctrl x² keys

√(*Expr1***)** ⇒ *expression*

√(*List1***)** ⇒ *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 5.

$$\sqrt{4} \qquad 2$$

$$\sqrt{\{9,a,4\}} \qquad \{3,\sqrt{a},2\}$$

Π() (prodSeq) Catalog >

Π(*Expr1***,** *Var***,** *Low***,** *High***)** ⇒ *expression*

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.

Note: See also **Product template (Π)**, page 9.

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

$$\prod_{k=1}^{n}\left(k^2\right) \qquad (n!)^2$$

$$\prod_{n=1}^{5}\left(\left\{\frac{1}{n},n,2\right\}\right) \qquad \left\{\frac{1}{120},120,32\right\}$$

Π() (prodSeq) Catalog >

Π(*Expr1***,** *Var***,** *Low***,** *Low*−1**)** ⇒ 1

Π(*Expr1***,** *Var***,** *Low***,** *High***)** ⇒ **1/Π(***Expr1***,** *Var***,** *High+1***,** *Low−1***)** if *High* < *Low−1*

$$\prod_{k=4}^{3}(k) \qquad 1$$

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.

$$\prod_{k=4}^{1}\left(\frac{1}{k}\right) \qquad 6$$

$$\prod_{k=4}^{1}\left(\frac{1}{k}\right)\cdot\prod_{k=2}^{4}\left(\frac{1}{k}\right) \qquad \frac{1}{4}$$

Σ() (sumSeq) Catalog >

Σ(*Expr1***,** *Var***,** *Low***,** *High***)** ⇒ *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 9.

$$\sum_{n=1}^{5}\left(\frac{1}{n}\right) \qquad \frac{137}{60}$$

$$\sum_{k=1}^{n}\left(k^{2}\right) \qquad \frac{n\cdot(n+1)\cdot(2\cdot n+1)}{6}$$

$$\sum_{n=1}^{\infty}\left(\frac{1}{n^{2}}\right) \qquad \frac{\pi^{2}}{6}$$

Σ(*Expr1***,** *Var***,** *Low***,** *Low−1***)** ⇒ 0

Σ(*Expr1***,** *Var***,** *Low***,** *High***)** ⇒ μ

Σ(*Expr1***,** *Var***,** *High+1***,** *Low*−1**)** if *High* < *Low−1*

$$\sum_{k=4}^{3}(k) \qquad 0$$

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.

$$\sum_{k=4}^{1}(k) \qquad {}^{-}5$$

$$\sum_{k=4}^{1}(k)+\sum_{k=2}^{4}(k) \qquad 4$$

ΣInt() Catalog >

ΣInt(*NPmt1*, *NPmt2*, *N*, *I*, *PV* ,[*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [*roundValue*]**)** ⇒ *value*

ΣInt(1,3,12,4.75,20000,,12,12)   ⁻213.48

ΣInt(*NPmt1*,*NPmt2*,*amortTable***)** ⇒ *value*

tbl:=amortTbl(12,12,4.75,20000,,12,12)

$$\begin{bmatrix} 0 & 0. & 0. & 20000. \\ 1 & -77.49 & -1632.43 & 18367.6 \\ 2 & -71.17 & -1638.75 & 16728.8 \\ 3 & -64.82 & -1645.1 & 15083.7 \\ 4 & -58.44 & -1651.48 & 13432.2 \\ 5 & -52.05 & -1657.87 & 11774.4 \\ 6 & -45.62 & -1664.3 & 10110.1 \\ 7 & -39.17 & -1670.75 & 8439.32 \\ 8 & -32.7 & -1677.22 & 6762.1 \\ 9 & -26.2 & -1683.72 & 5078.38 \\ 10 & -19.68 & -1690.24 & 3388.14 \\ 11 & -13.13 & -1696.79 & 1691.35 \\ 12 & -6.55 & -1703.37 & -12.02 \end{bmatrix}$$

ΣInt(1,3,*tbl*)   ⁻213.48

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 193.

- 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.

ΣInt(*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 12.

Note: See also ΣPrn(), below, and **Bal()**, page 21.

ΣPrn() Catalog >

ΣPrn(*NPmt1*, *NPmt2*, *N*, *I*, *PV*, [*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [*roundValue*]**)** ⇒ *value*

ΣPrn(1,3,12,4.75,20000,,12,12)   ⁻4916.28

ΣPrn(*NPmt1*, *NPmt2*, *amortTable***)** ⇒ *value*

Amortization function that calculates the sum of the principal during a specified range of payments.

ΣPrn() Catalog > 🕮

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 193.

- 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.

ΣPrn(*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 **amortTbl()**, page 12.

Note: See also ΣInt(), above, and **Bal()**, page 21.

```
tbl:=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 ⎦
ΣPrn(1,3,tbl)                                ⁻4916.28
```

(indirection) ctrl 🕮 keys

*varNameString*

Refers to the variable whose name is *varNameString*. This lets you use strings to create variable names from within a function.

```
#("x"&"y"&"z")                               xyz
```

Creates or refers to the variable xyz .

```
10→r                                          10
"r"→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*

Enters a number in scientific notation. The number is interpreted as $mantissa \times 10^{exponent}$.

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.

23000.	23000.
2300000000.+4.1E15	4.1E15
$3\cdot 10^4$	30000

g (gradian) — π▸ key

*Expr1*g ⇒ *expression*

*List1*g ⇒ *list*

*Matrix1*g ⇒ *matrix*

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 π/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`.

In Degree, Gradian or Radian mode:

$\cos(50^g)$	$\frac{\sqrt{2}}{2}$
$\cos(\{0,100^g,200^g\})$	$\{1,0,^{-}1\}$

r(radian) — π▸ key

*Expr1*r ⇒ *expression*

*List1*r ⇒ *list*

*Matrix1*r ⇒ *matrix*

In Degree, Gradian or Radian angle mode:

$\cos\left(\frac{\pi}{4^r}\right)$	$\frac{\sqrt{2}}{2}$
$\cos\left(\left\{0^r,\frac{\pi}{12}^r,^{-}(\pi)^r\right\}\right)$	$\left\{1,\frac{(\sqrt{3}+1)\cdot\sqrt{2}}{4},^{-}1\right\}$

^{r}(radian) π▸ 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/π.

In Radian angle mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by 200/π.

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) π▸ key

Expr1° ⇒ *expression*

List1° ⇒ *list*

Matrix1° ⇒ *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 π/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:

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

In Radian angle mode:

Note: To force an approximate result,

Handheld: Press ctrl enter.
Windows®: Press **Ctrl+Enter**.
Macintosh®: Press **⌘+Enter**.
iPad®: Hold **enter**, and select ≈.

$\cos\left(\left\{0,\frac{\pi}{4},90°,30.12°\right\}\right)$
$\{1.,0.707107,0.,0.864976\}$

°, ', '' (degree/minute/second) ctrl 📖 keys

dd°*mm*'*ss.ss*'' ⇒ *expression*

In Degree angle mode:

°, ', '' (degree/minute/second) ctrl 📖 keys

dd A positive or negative number
mm A non-negative number
ss.ss A non-negative number

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 (").

25°13'17.5"	25.2215
25°30'	$\frac{51}{2}$

∠ (angle) ctrl 📖 keys

[*Radius*,∠ θ_*Angle*] ⇒ *vector*
(polar input)

[*Radius*,∠ θ_*Angle*,*Z_Coordinate*] ⇒ *vector*
(cylindrical input)

[*Radius*,∠ θ_*Angle*,∠ θ_*Angle*] ⇒ *vector*
(spherical input)

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

Note: You can insert this symbol from the computer keyboard by typing `@<`.

In Radian mode and vector format set to: rectangular

$[5\ \angle 60°\ \ \angle 45°]$	$\begin{bmatrix}\frac{5\cdot\sqrt{2}}{4} & \frac{5\cdot\sqrt{6}}{4} & \frac{5\cdot\sqrt{2}}{2}\end{bmatrix}$

cylindrical

$[5\ \angle 60°\ \ \angle 45°]$	$\begin{bmatrix}\frac{5\cdot\sqrt{2}}{2} & \angle\frac{\pi}{3} & \frac{5\cdot\sqrt{2}}{2}\end{bmatrix}$

spherical

$[5\ \angle 60°\ \ \angle 45°]$	$\begin{bmatrix}5 & \angle\frac{\pi}{3} & \angle\frac{\pi}{4}\end{bmatrix}$

(*Magnitude* ∠ *Angle*) ⇒ *complexValue*
(polar input)

Enters a complex value in (r∠ θ) polar form. The *Angle* is interpreted according to the current Angle mode setting.

In Radian angle mode and Rectangular complex format:

$5+3\cdot i-\left(10\ \angle\frac{\pi}{4}\right)$	$5-5\cdot\sqrt{2}+\left(3-5\cdot\sqrt{2}\right)\cdot i$

Note: To force an approximate result,

Handheld: Press ctrl enter.
Windows®: Press **Ctrl+Enter**.
Macintosh®: Press ⌘+**Enter**.
iPad®: Hold **enter**, and select ≈.

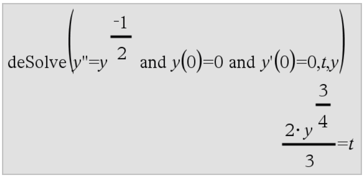
∠ (angle) ctrl 📖 keys

$5+3\cdot i-\left(10\ \angle -\frac{\pi}{4}\right)$	$^{-}2.07107-4.07107\cdot i$

' (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.



$$\text{deSolve}\left(y''=y^{\frac{-1}{2}}\ \text{and}\ y(0)=0\ \text{and}\ y'(0)=0,t,y\right)$$

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

_ (underscore as an empty element) See “Empty (Void) Elements,” page 234.

_ (underscore as unit designator) ctrl ␣ keys

*Expr*_*Unit*

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.

3·_m▸_ft	9.84252·_ft

Note: You can find the conversion symbol, ►, in the Catalog. Click ∫Σ, and then click **Math Operators**.

*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.

Assuming z is undefined:

real(z)	z
real(z_)	real(z_)
imag(z)	0
imag(z_)	imag(z_)

► (convert) ctrl 📖 keys

Expr_Unit1► *_Unit2* ⇒ *Expr_Unit2*

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, ►, from the top of the list.

You can also type unit names manually. To type “_” when typing unit names on the handheld, press ctrl ␣.

Note: To convert temperature units, use **tmpCnv()** and Δ**tmpCnv()**. The ► conversion operator does not handle temperature units.

3·_m►_ft	9.84252·_ft

10^() Catalog > 📖

10^ (*Expr1***)** ⇒ *expression*

10^ (*List1***)** ⇒ *list*

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in *List1*.

$10^{1.5}$	31.6228
$10^{\{0,^{-}2,2,a\}}$	$\left\{1,\frac{1}{100},100,10^{a}\right\}$

10^(*squareMatrix1***)** ⇒ *squareMatrix*

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()**.

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

$$10^{\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & ^{-}2 & 1 \end{bmatrix}}$$

$$\begin{bmatrix} 1.14336\text{E}7 & 8.17155\text{E}6 & 6.67589\text{E}6 \\ 9.95651\text{E}6 & 7.11587\text{E}6 & 5.81342\text{E}6 \\ 7.65298\text{E}6 & 5.46952\text{E}6 & 4.46845\text{E}6 \end{bmatrix}$$

^⁻¹ (reciprocal) Catalog > 📖

Expr1 **^⁻¹** ⇒ *expression*

List1 **^⁻¹** ⇒ *list*

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in *List1*.

$(3.1)^{-1}$	0.322581
$\{a,4,^{-}0.1,x,^{-}2\}^{-1}$	$\left\{\frac{1}{a},\frac{1}{4},^{-}10.,\frac{1}{x},\frac{^{-}1}{2}\right\}$

squareMatrix1 **^⁻¹** ⇒ *squareMatrix*

Returns the inverse of *squareMatrix1*.

squareMatrix1 must be a non-singular square matrix.

$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}^{-1}$	$\begin{bmatrix}^{-}2 & 1\\ \frac{3}{2} & \frac{^{-}1}{2}\end{bmatrix}$
$\begin{bmatrix}1 & 2\\ a & 4\end{bmatrix}^{-1}$	$\begin{bmatrix}\frac{^{-}2}{a-2} & \frac{1}{a-2}\\ \frac{a}{2\cdot(a-2)} & \frac{^{-}1}{2\cdot(a-2)}\end{bmatrix}$

| (constraint operator) ctrl 📖 keys

Expr **|** *BooleanExpr1*[**and** *BooleanExpr2*]...

Expr **|** *BooleanExpr1*[**or***BooleanExpr2*]...

$x+1\vert x=3$	4
$x+y\vert x=\sin(y)$	$\sin(y)+y$
$x+y\vert\sin(y)=x$	$x+y$

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

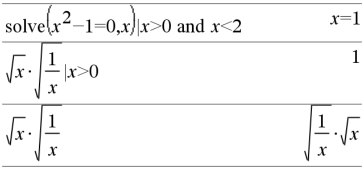
Substitutions are in the form of an equality, such as x=3 or y=sin(x). To be most effective, the left side should be a simple variable. *Expr* | *Variable* = *value* will substitute *value* for every occurrence of *Variable* in *Expr*.

$x^3-2\cdot x+7\rightarrow f(x)$	*Done*
$f(x)\vert x=\sqrt{3}$	$\sqrt{3}+7$
$(\sin(x))^2+2\cdot\sin(x)-6\vert\sin(x)=d$	$d^2+2\cdot d-6$

| (constraint operator)

ctrl 📖 keys

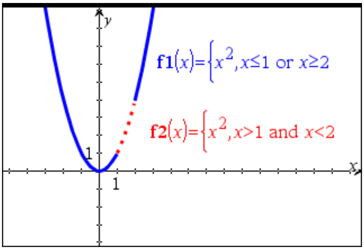
Interval constraints take the form of one or more inequalities joined by logical "**and**" or "**or**" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.



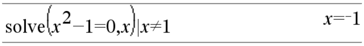
$\text{solve}(x^2-1=0,x)|x>0 \text{ and } x<2$ $\quad x=1$

$\sqrt{x}\cdot\sqrt{\frac{1}{x}}|x>0$ $\quad 1$

$\sqrt{x}\cdot\sqrt{\frac{1}{x}}$ $\quad \sqrt{\frac{1}{x}}\cdot\sqrt{x}$



Exclusions use the "not equals" (/= or ≠) 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.



$\text{solve}(x^2-1=0,x)|x\neq 1$ $\quad x=-1$

→ (store)

ctrl var key

Expr → *Var*

List → *Var*

Matrix → *Var*

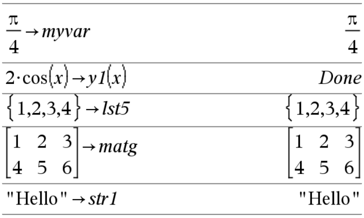
Expr → *Function*(*Param1*,...)

List → *Function*(*Param1*,...)

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*.



$\frac{\pi}{4}\rightarrow myvar$ $\quad \frac{\pi}{4}$

$2\cdot\cos(x)\rightarrow y1(x)$ $\quad$ *Done*

$\{1,2,3,4\}\rightarrow lst5$ $\quad \{1,2,3,4\}$

$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}\rightarrow matg$ $\quad \begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$

"Hello"→*str1* $\quad$ "Hello"

→ (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*

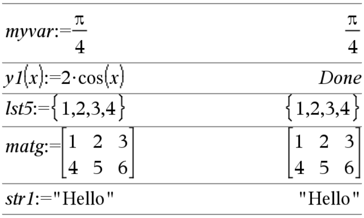
Var **:=** *List*

Var **:=** *Matrix*

*Function***(***Param1***,**...**)** **:=** *Expr*

*Function***(***Param1***,**...**)** **:=** *List*

*Function***(***Param1***,**...**)** **:=** *Matrix*



$myvar:=\frac{\pi}{4}$ $\frac{\pi}{4}$

$y1(x):=2\cdot\cos(x)$ *Done*

$lst5:=\{1,2,3,4\}$ $\{1,2,3,4\}$

$matg:=\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$ $\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$

$str1:=$"Hello" "Hello"

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.

© (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)=Func
            © Declare variables
            Local i,result
            result:=0
            For i,1,n,1 ©Loop n times
            result:=result+i^2
            EndFor
            Return result
            EndFunc
                                        Done
g(3)                                      14
```

0b, 0h 0 B keys, 0 H 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
```

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.

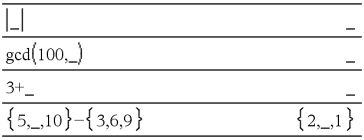
You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “*Graphing spreadsheet data*.”

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 53, and **isVoid()**, page 93.

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.

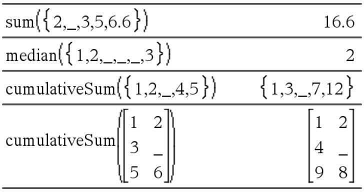


|_|	_
gcd(100,_)	_
3+_	_
{5,_,10}−{3,6,9}	{2,_,1}

List arguments containing void elements

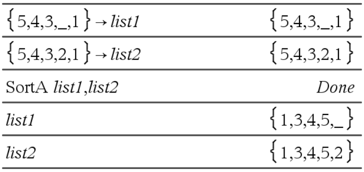
The following functions and commands ignore (skip) void elements found in list arguments.

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



sum({2,_,3,5,6.6})	16.6
median({1,2,_,_,_,3})	2
cumulativeSum({1,2,_,4,5})	{1,3,_,7,12}
cumulativeSum$\left(\begin{bmatrix}1 & 2\\ 3 & _\\ 5 & 6\end{bmatrix}\right)$	$\begin{bmatrix}1 & 2\\ 4 & \_\\ 9 & 8\end{bmatrix}$

SortA and **SortD** move all void elements within the first argument to the bottom.



{5,4,3,_,1}→*list1*	{5,4,3,_,1}
{5,4,3,2,1}→*list2*	{5,4,3,2,1}
SortA *list1*,*list2*	*Done*
list1	{1,3,4,5,_}
list2	{1,3,4,5,2}

List arguments containing void elements

{1,2,3,_,5}→*list1*	{1,2,3,_,5}
{1,2,3,4,5}→*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.}

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.}
stat.XReg	{_,_,4.,5.}
stat.YReg	{_,_,5.,6.6}
stat.FreqReg	{_,_,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.}

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 √6, you can type **sqrt (6)** on the entry line. When you press enter, the expression **sqrt(6)** is changed to √6. Some shortrcuts 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	**pi**
θ	**theta**
∞	**infinity**
$\le$	**<=**
$\ge$	**>=**
$\ne$	**/=**
⇒ (logical implication)	=>
⇔ (logical double implication, XNOR)	<=>
→ (store operator)	**=:**
| | (absolute value)	**abs(...)**
√()	**sqrt(...)**
d()	**derivative(...)**
∫()	**integral(...)**
Σ() (Sum template)	**sumSeq(...)**
Π() (Product template)	**prodSeq(...)**
sin^{-1}(), cos^{-1}(), ...	**arcsin(...), arccos(...), ...**
ΔList()	**deltaList(...)**
ΔtmpCnv()	**deltaTmpCnv(...)**

From the Computer Keyboard

To enter this:	Type this shortcut:
c1, ***c2***, ... (constants)	@**c1**, @**c2**, ...

To enter this:	**Type this shortcut:**
n1, ***n2***, ... (integer constants)	`@n1`, `@n2`, ...
i (imaginary constant)	`@i`
e (natural log base e)	`@e`
E (scientific notation)	`@E`
T (transpose)	`@t`
r (radians)	`@r`
° (degrees)	`@d`
g (gradians)	`@g`
∠ (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 (≠ or /=), less than (<), less than or equal (≤ or <=), greater than (>), greater than or equal (≥ or >=)
11	Logical **not**
12	Logical **and**
13	Logical **or**
14	**xor**, **nor**, **nand**
15	Logical implication (⇒)
16	Logical double implication, XNOR (⇔)
17	Constraint operator (“|”)
18	Store (→)

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• (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→r and “r”→s1, then #s1=10.

Post Operators

Post operators are operators that come directly after an argument, such as 5!, 25%, or 60°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 (−) 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.

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 189.

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 If a<b will cause this error if either a or b is undefined when the If 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: • 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 esc or on 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->a, where a is an undefined variable, will cause this error.
200	Constraint expression invalid For example, solve(3x^2-4=0,x) | x<0 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, **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, 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, \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, √(-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: • 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	Function is not defined. Use one of the following commands: • **Define** • **:=** • sto → to define a function.
1100	Non-real calculation For example, if the software is in the Real setting, √(-1) is invalid. To allow complex results, change the “Real or Complex” Mode Setting to RECTANGULAR or POLAR.
1110	Invalid bounds
1120	No sign change
1130	Argument cannot be a list or matrix
1140	Argument error 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.
1150	Argument error 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.
1160	Invalid library pathname

Error code	Description
	A pathname must be in the form $xxx\backslash yyy$, where: • The xxx part can have 1 to 16 characters. • The yyy part can have 1 to 15 characters. See the Library section in the documentation for more details.
1170	Invalid use of library pathname • A value cannot be assigned to a pathname using **Define**, **:=**, or sto →. • A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition.
1180	Invalid library variable name. Make sure that the name: • Does not contain a period • Does not begin with an underscore • Does not exceed 15 characters See the Library section in the documentation for more details.
1190	Library document not found: • Verify library is in the MyLib folder. • Refresh Libraries. See the Library section in the documentation for more details.
1200	Library variable not found: • Verify library variable exists in the first problem in the library. • Make sure library variable has been defined as LibPub or LibPriv. • Refresh Libraries. See the Library section in the documentation for more details.
1210	Invalid library shortcut name. Make sure that the name: • Does not contain a period • Does not begin with an underscore • Does not exceed 16 characters • Is not a reserved name See the Library section in the documentation for more details.
1220	Domain error: The tangentLine and normalLine functions support real-valued functions only.
1230	Domain error.

Error code	**Description**
	Trigonometric conversion operators are not supported in Degree or Gradian angle modes.
1250	Argument Error Use a system of linear equations. Example of a system of two linear equations with variables x and y: 3x+7y=5 2y-5x=-1
1260	Argument Error: 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.
1270	Argument Error Order of the derivative must be equal to 1 or 2.
1280	Argument Error Use a polynomial in expanded form in one variable.
1290	Argument Error Use a polynomial in one variable.
1300	Argument Error The coefficients of the polynomial must evaluate to numeric values.
1310	Argument error: A function could not be evaluated for one or more of its arguments.
1380	Argument error: Nested calls to domain() function are not allowed.

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 198.

Warning code	Message
10000	Operation might introduce false solutions.
10001	Differentiating an equation may produce a false equation.
10002	Questionable solution
10003	Questionable accuracy
10004	Operation might lose solutions.
10005	cSolve might specify more zeros.
10006	Solve may specify more zeros.
10007	More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess. Examples using solve(): • solve(Equation, Var=Guess)|lowBound<Var<upBound • solve(Equation, Var)|lowBound<Var<upBound • solve(Equation, Var=Guess)
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 undef^0 replaced by 1
10014	undef^0 replaced by 1
10015	1^∞ or 1^undef replaced by 1
10016	1^undef replaced by 1
10017	Overflow replaced by ∞ or −∞
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.

Warning code	Message
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 "\" 'Variable MathTestSymbol Constant' or a conjunct of these forms, for example 'x<3 and x>-12'

Support and Service

Texas Instruments Support and Service

General Information: North and South America

Home Page:	education.ti.com
KnowledgeBase and e-mail inquiries:	education.ti.com/support
Phone:	(800) TI-CARES / (800) 842-2737 For North and South America and U.S. Territories
International contact information:	https://education.ti.com/en/us/customer-support/support_worldwide

For Technical Support

Knowledge Base and support by e-mail:	education.ti.com/support or ti-cares@ti.com
Phone (not toll-free):	(972) 917-8324

For Product (Hardware) Service

Customers in the U.S., Canada, Mexico, and U.S. territories: Always contact Texas Instruments Customer Support before returning a product for service.

For All Other Countries:

For general information

For more information about TI products and services, contact TI by e-mail or visit the TI Internet address.

E-mail inquiries:	ti-cares@ti.com
Home Page:	education.ti.com

Service and Warranty Information

For information about the length and terms of the warranty or about product service, refer to the warranty statement enclosed with this product or contact your local Texas Instruments retailer/distributor.

Index