

TI-Nspire™ CX Reference Guide

Important Information

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

© 2021 Texas Instruments Incorporated

Actual products may vary slightly from provided images.

Contents

Expression Templates	
Alphabetical Listing	7
A	
В	
C	
D	
E	
F	
G	
l	
L	
M	
N	
0	
Р	
Q	
R	
s	
Т	
U	
V	
W	
X	
Z	166
Symbols	172
TI-Nspire™ CX II - Draw Commands	195
Graphics Programming	195
Graphics Screen	
Default View and Settings	
Graphics Screen Errors Messages	
Invalid Commands While in Graphics Mode	
C	
D	
F	
G	
P	
S	
U	

Empty (Void) Elements	210
Shortcuts for Entering Maths Expressions	212
EOS™ (Equation Operating System) Hierarchy	214
TI-Nspire CX II - TI-Basic Programming Features	216
Auto-indentation in Programming Editor	216
Improved Error Messages for TI-Basic	216
Constants and Values	219
Error Codes and Messages	220
Warning Codes and Messages	228
General Information	230
Online Help	230
Contact TI Support	
Service and Warranty Information	230
Index	221

Expression Templates

Expression templates give you an easy way to enter maths 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.

Use the arrow keys or press tab to move the cursor to each element's position, and type a value or expression for the element. Press enter or ctri enter to evaluate the expression.

Fraction template		ctrl 🗦 keys
<u> </u>	Example:	
Note: See also / (divide), page 174.	<u>12</u> 8·2	$\frac{3}{4}$

Exponent template		^ key
m0	Example:	
Note: Time the first value areas 🕡 and	2 ³	8
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 175.

Square root template		ctri x² keys
Note: See also $\sqrt{()}$ (square root), page	Example:	2
184.	$\sqrt{\frac{4}{\left\{9,a,4\right\}}}$	$\frac{2}{\left\{3,\sqrt{(a)},2\right\}}$
	$ \frac{\sqrt{4}}{\sqrt{\left\{9,16,4\right\}}} $	2 {3,4,2}

Nth root template





Note: See also root(), page 128.

Example:

3√8	2
3[{8.27.15}	{2,3,2.46621}

e exponent template

ex kevs



Natural exponential *e* raised to a power

Note: See also e^(), page 43.

Example:

2.71828182846 e^1

Log template

ctri 10X kev



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

Note: See also log(), page 85.

Example:

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

Piecewise template (2-piece)

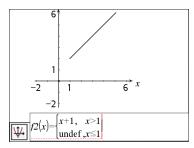
Catalogue >



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 109.

Example:



Piecewise template (N-piece)

Catalogue >



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

Example:

Piecewise template (N-piece)





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

Note: See also piecewise(), page 109.

System of 2 equations template

Catalogue >

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

Note: See also system(), page 149.

Example:

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

$$x=\frac{-3}{2} \text{ and } y=\frac{1}{4} \text{ or } x=1 \text{ and } y=-1$$

System of N equations template

Catalogue >

Lets you create a system of *M*inear equations. Prompts for N.

Example:



Create a System of Eq... 🗷 System of Equations Number of equations Cancel

Note: See also system(), page 149.

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

Absolute value template

Catalogue >



Note: See also abs(), page 7.

Example:

2,-3,4,-43 {2,3,4,64}

dd°mm'ss.ss" template





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"	0.528011
50 15 10	0.520011

Matrix template (2 x 2)





Creates a 2 x 2 matrix.

Example:

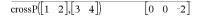
1	2 .5 4	5	10
3	4	15	20]

Matrix template (1 x 2)

Catalogue >



Example:



Matrix template (2 x 1)





Example:

[5].0.01	[0.05]
[8]	[0.08]

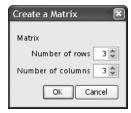
Matrix template (m x n)

Example:



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

	$\sqrt{4}$	2	6		[4	2	9
diag	1	2	3				
١	[5	7	9∬				



Matrix template (m x n)



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

Sum template (Σ)





Example:

7	25
\rightarrow (n)	
\(n)	
n=3	

Note: See also Σ () (sumSeq), page 185.

Product template (Π)

Catalogue >



Example:



Note: See also Π () (prodSeq), page 184.

First derivative template

Catalogue >





Example:



The first derivative template can be used to calculate first derivative at a point numerically, using auto differentiation methods.

Note: See also d() (derivative), page 183.

Second derivative template

Catalogue >



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

Example:

Second derivative template



The second derivative template can be used to calculate second derivative at a point numerically, using auto differentiation methods.

18

Note: See also d() (derivative), page 183.

Definite integral template		Catalogue >
	Example:	
	$\int_{0}^{10} x^2 dx$	333.333
The definite integral template can be used to calculate the definite integral	Jo	

().
Note: See also nint(), page 100.

numerically, using the same method as nint

Alphabetical Listing

Items whose names are not alphabetic (such as +, ! and >) are listed at the end of this section, starting page 172. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

Δ

abs()		Catalogue > 🔃
abs(Value1)⇒value	 ∫ π _π \	{1.5708,1.0472}
$abs(List1) \Rightarrow list$	$\left \left\{ \frac{\pi}{2}, \frac{-\pi}{3} \right\} \right $	
abs(Matrix I)⇒matrix	$\frac{ 2-3\cdot i }{}$	3.60555

Returns the absolute value of the argument.

Note: See also Absolute value template, page 3.

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

amortTbl() Catalogue > 🗐

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

Amortisation function that returns a matrix as an amortisation 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 PmtAtare described in the table of TVM arguments, page 159.

- 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 PmtAtare the same as for the TVM functions.

mortTbl(12,60,10,5000,,,12,12)					
	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	

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 amortisation functions Σ **Int()** and Σ Prn(), page 185, and bal(), page 15.

and Catalogue > 🗐

BooleanExpr1 and BooleanExpr2⇒Boolean expression

BooleanList1 and $BooleanList2 \Rightarrow Boolean$ list

BooleanMatrix L and BooleanMatrix2⇒Boolean matrix

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

*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
ODIOUIUI and ODIOO	ODIOO

In Dec base mode:

37 and 0	b100	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()

Catalogue > [3]

angle(Value1)⇒value

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

In Degree angle mode:

$$angle(0+2\cdot i) 90$$

In Gradian angle mode:

$$angle(0+3\cdot i)$$
 100

In Radian angle mode:

$$\frac{\text{angle}(1+i)}{\text{angle}(\left\{1+2\cdot i, 3+0\cdot i, 0-4\cdot i\right\})} \\ \left\{1.10715, 0., -1.5708\right\}$$

 $angle(List1) \Rightarrow list$

 $angle(Matrix 1) \Rightarrow matrix$

Returns a list or matrix of angles of the elements in *List1* or *Matrix1*, interpreting each element as a complex number that represents a two-dimensional rectangular coordinate point.

ANOVA

Catalogue > 🗐

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 144).

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

Catalogue > 🗐 ANOVA2way

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 (page 144).

LevRow=0 for Block

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

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
stat.MSBlock	Mean squares for factor
stat.dfError	Degrees of freedom of the errors

Output variable	Description
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.FcoI	F statistic of the column factor
stat.PValCol	Probability value of the column factor
stat.dfCoI	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	${\sf F}$ 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	56	56
Returns the result of the most recently	56+4	60
evaluated expression.	60+4	64

approx()

$approx(Value 1) \Rightarrow number$

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.

 $approx(List1) \Rightarrow list$

 $approx(Matrix 1) \Rightarrow matrix$

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

	Catalogue > 📳
$approx\left(\frac{1}{3}\right)$	0.333333
$\operatorname{approx}\left\{\left\{\frac{1}{3},\frac{1}{9}\right\}\right\}$	{0.333333,0.111111}
$\operatorname{approx}(\{\sin(\pi),\cos(\pi)$	{ 0.,-1. }
$approx([\sqrt{2} \sqrt{3}])$	[1.41421 1.73205]
$\operatorname{approx}\left[\left[\frac{1}{3} \frac{1}{9}\right]\right]$	[0.333333 0.111111]
10.13.13	
$approx(\{sin(\pi),cos(\pi)\})$	
$approx([\sqrt{2} \sqrt{3}])$	[1.41421 1.73205]

Catalogue > [32]

▶approxFraction()

Catalogue > 😰

Value ▶approxFraction([Tol])⇒value

List \triangleright approxFraction([Tol]) \Rightarrow list

 $Matrix \rightarrow approxFraction([Tol]) \Rightarrow matrix$

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

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

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

 $\{\pi,1.5\}$ ▶approxFraction(5.ε-14) $\begin{cases} \frac{5419351}{1725033}, \frac{3}{2} \end{cases}$

approxRational()

approxRational(Value[, Tol])⇒value

 $approxRational(List[, Tol]) \Rightarrow list$

 $approxRational(Matrix[, Tol]) \Rightarrow matrix$

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

Catalogue > 🗊

approxRational (0.333,5·10⁻⁵) 333 1000 approxRational (4.0.2.0.33.4.125) 5 = 14)

approxRational($\{0.2,0.33,4.125\}$,5. ϵ -14) $\left\{\frac{1}{5},\frac{33}{100},\frac{33}{8}\right\}$

arccos()

See cos⁻¹(), page 26.

arccosh()

See cosh⁻¹(), page 27.

arccot()

See cot⁻¹(), page 28.

arccoth()

See coth⁻¹(), page 29.

arccsc()

See csc⁻¹(), page 31.

arccsch()

See csch⁻¹(), page 32.

arcsec()

See sec⁻¹(), page 131.

arcsech()

See sech⁻¹(), page 132.

arcsin()

See sin⁻¹(), page 139.

arcsinh()

See sinh⁻¹(), page 140.

arctan()

See tan-1(), page 150.

arctanh()

See tanh-1(), page 151.

augment()

Catalogue > 23

{1,-3,2,5,4}

 $augment(List1, List2) \Rightarrow list$

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

 $augment(Matrix1, Matrix2) \Rightarrow matrix$

Returns a new matrix that is *Matrix2* appended to Matrix 1. When the "," character is used, the matrices must have equal row dimensions, and *Matrix2* is appended to *Matrix1* as new columns. Does not alter Matrix 1 or Matrix 2.

$\begin{bmatrix} 1 & 2 \end{bmatrix} \rightarrow m1$		1	2
[3 4]		3	$_{4}$
$\begin{bmatrix} 5 \end{bmatrix}_{\rightarrow m2}$			5
[6]			[6]
augment $(m1, m2)$	1	2	5
	3	4	6

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

avgRC() Catalogue > 2 avgRC(Expr1, Var [=Value] [, Step]) $\Rightarrow expression$ $\frac{x:=2}{avgRC(x^2-x+2,x)}$ 3.001 avgRC(Expr1, Var [=Value] [, $avgRC(x^2-x+2,x,1)$ 3.1 avgRC($x^2-x+2,x,1$) 6

 $avgRC(List1, Var [=Value] [, Step]) \Rightarrow list$

 $avgRC(Matrix 1, Var [=Value] [, Step]) \Rightarrow matrix$

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.

В

bal() Catalogue > [2]

bal(NPmt,N,I,PV,[Pmt],[FV],[PpY],[CpY],[PmtAt],[roundValue]) $\Rightarrow value$

 $bal(NPmt,amortTable) \Rightarrow value$

Amortisation 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 159.

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 159.

• If you omit *Pmt*, it defaults to

bal(5,6,5.75,500	00,	,12,12)		833.11
tbl:=amortTbl(6	,6	,5.75,50	00,,12,12)	
[0	0.	0.	5000.
	1	-23.35	-825.63	4174.37
	2	-19.49	-829.49	3344.88
	3	-15.62	-833.36	2511.52
	4	-11.73	-837.25	1674.27
	5		-841.16	833.11
Į	6	-3.89	-845.09	-11.98
bal(4,tbl)				1674.27

Pmt=tvmPmt (N,I,PV,FV,PpY,CpY,PmtAt).

- If you omit FV, it defaults to FV=0.
- The defaults for PpY, CpY and PmtAtare 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 amortisation table *amortTable*. The amort Table argument must be a matrix in the form described under amortTbl(), page 7.

Note: See also Σ **Int()** and Σ **Prn()**, page 186.

▶Base2		Catalogue > 🗐	
Integer1 ▶Base2⇒integer	256▶Base2	0b100000000	
Note: You can insert this operator from the	0h1F▶Base2	0b11111	

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

Converts *Integer 1* 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

Oh hexadecimalNumber

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

Without a prefix, *Integer 1* 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 Ohfffffffffffff in Hex base mode 0b111...111 (64 1's) in Binary base mode



-263 is displayed as 0h8000000000000000 in Hex base mode 0b100...000 (63 zeroes) in Binary base mode

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

263 becomes -263 and is displayed as 0h8000000000000000 in Hex base mode 0b100...000 (63 zeroes) in Binary base mode

264 becomes 0 and is displayed as

0h0 in Hex base mode

0b0 in Binary base mode

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

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

Base 10 Catalogue > 🗐 *Integer1* ▶Base10⇒*integer* 0b10011 ▶ Base10 19 Note: You can insert this operator from the 0h1F▶Base10 31 computer keyboard by typing @>Base10. Converts *Integer 1* to a decimal (base 10)

0b binaryNumber

respectively.

Oh hexadecimal Number

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

number. A binary or hexadecimal entry must always have a 0b or 0h prefix,

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

Without a prefix, *Integer 1* is treated as decimal. The result is displayed in decimal, regardless of the Base mode.

▶Base16 Catalogue > [3]

Integer1 ▶Base16⇒*integer*

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

Converts *Integer 1* to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

0b binaryNumber

Oh hexadecimal Number

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 16.

256▶Base16	0h100
0b111100001111▶Base16	0hF0F

binomCdf() Catalogue > [1]

 $binomCdf(n,p) \Rightarrow list$

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

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

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

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

binomPdf()

Catalogue > 🗐

binomPdf $(n,p) \Rightarrow list$

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

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

С

ceiling()	Catalogue > 🖫

 $ceiling(Value 1) \Rightarrow value$

ceiling(.456) 1.

Returns the nearest integer that is \geq the argument.

The argument can be a real or a complex number.

Note: See also floor().

ceiling(List1) $\Rightarrow list$ ceiling(Matrix1) $\Rightarrow matrix$

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

ceiling({-3.1,1,2.5})	{-3.,1,3.}
ceiling $\begin{bmatrix} 0 & -3.2 \cdot i \end{bmatrix}$	0 -3.·i
1.3 4	[2. 4]

centralDiff()

Catalogue > 🕮

centralDiff(Expr1,Var = Value**]**[,Step**])** \Rightarrow expression

centralDiff(
$$\cos(x), x$$
)| $x = \frac{\pi}{2}$

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

centralDiff(Expr1,Var [=Value][,List]) \Rightarrow list

centralDiff(List1,Var [=Value][,Step]) \Rightarrow list

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().

char()		Catalogue > 📳
$char(Integer) \Rightarrow character$	char(38)	"&"
Returns a character string containing the	char(65)	"A"

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

χ22way Catalogue > 👰

χ**22way** 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 144)

For information on the effect of empty elements in a matrix, see "Empty (Void) Elements," page 210.

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^2 Cdf(lowBound,upBound,df) \Rightarrow number if lowBound and upBound are numbers, list if lowBound and upBound are lists

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

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

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

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 210.

χ2GOF Catalogue > 23

γ**2GOF** 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 144.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 210.

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^2 Pdf() Catalogue > 🗐

 χ^2 Pdf(XVal,df) \Rightarrow number if XVal is a number, list if XVal is a list

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

list if XVal is a list

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

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 210.

ClearAZ		Catalogue > 🗐
ClearAZ	5 → b	5
Clears all single-character variables in the	b	5
current problem space.	ClearAZ	Done
If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See	b	"Error: Variable is not defined"

ClrErr Catalogue > 🗐

ClrErr

unLock, page 161.

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 dialogue box will be displayed as normal.

Note: See also PassErr, page 108, and Try, page 155.

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

For an example of **CIrErr**, See Example 2 under the **Try** command, page 155.

colAugment()

 $colAugment(Matrix1, Matrix2) \Rightarrow matrix$

Returns a new matrix that is *Matrix2* appended to *Matrix1*. The matrices must have equal column dimensions, and Matrix 2 is appended to Matrix 1 as new rows. Does not alter Matrix 1 or Matrix 2.

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

colDim()

 $colDim(Matrix) \Rightarrow expression$

Returns the number of columns contained in Matrix.

Note: See also rowDim().

Catalogue > 🗐

5 6

Catalogue > 🕮

colDim 0 1 2

colNorm() Catalogue > 🗐

 $colNorm(Matrix) \Rightarrow expression$

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().

 $3 \mid_{\rightarrow mat}$ 4 5 -6 colNorm(mat)

coni()

 $conj(Value 1) \Rightarrow value$ $conj(List1) \Rightarrow list$ $conj(Matrix 1) \Rightarrow matrix$

Returns the complex conjugate of the argument.

Catalogue > 🕮

 $conj(1+2\cdot i)$ $1-2 \cdot i$ $\begin{bmatrix} 2 & 1+3 \cdot i \end{bmatrix}$

constructMat()

constructMat

 $(Expr, Var1, Var2, numRows, numCols) \Rightarrow$ matrix

Returns a matrix based on the arguments.

Catalogue > 🕮

constructMat

constructMat()

Catalogue > 2

Catalogue > 🕮

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

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

CopyVar

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 userdefined function, copies the definition of that function to function Var2. Function Var1 must be defined.

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

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

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.

	_	_
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

aa.a:=45				4 5
<i>aa.b</i> :=6.78			6.	78
CopyVar aa.,bb.			Do	ne
getVarInfo()	aa.a	"NUM" "NUM" "NUM" "NUM"	"[]"	0
	aa.b	"NUM"	"[]"	0
	bb.a	"NUM"	"[]"	0
	bb.b	"NUM"	"[]"	0

corrMat()

Catalogue > 🗐

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



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

cos()

trig key

 $cos(Value 1) \Rightarrow value$

 $cos(List1) \Rightarrow list$

cos(*Value1***)** returns the cosine of the argument as a value.

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.

$\cos(squareMatrix 1) \Rightarrow squareMatrix$

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

When a scalar function f(A) operates on squareMatrix I (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:

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

In Degree angle mode:

$\cos\left(\left(\frac{\pi}{r}\right)_r\right)$	0.707107
$\frac{\cos(4)}{4}$	
cos(45)	0.707107
cos({0,60,90})	{1.,0.5,0.}

In Gradian angle mode:

$\cos(\{0,50,100\})$ {1.,0.707107,0.}

In Radian angle mode:

$\cos\left(\frac{\pi}{4}\right)$	0.707107
cos(45°)	0.707107

In Radian angle mode:

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

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

cos()

trig key

Then A = X B X-1 and f(A) = X f(B) X-1. For example, cos(A) = X cos(B) X-1 where:

$$cos(B) =$$

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

All computations are performed using floating-point arithmetic.

cos-1()

trig kev

$$cos-1(Value 1) \Rightarrow value$$

 $cos-1(List 1) \Rightarrow list$

cos-1(*Value1*) returns the angle whose cosine is *Value1*.

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** (...).

 $cos-1(squareMatrix 1) \Rightarrow squareMatrix$

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().

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

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}) {1.5708,1.36944,1.0472}

In Radian angle mode and Rectangular Complex Format:

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

$$\begin{bmatrix} 1.73485 + 0.064606 \cdot i & -1.49086 + 2.10514 \\ -0.725533 + 1.51594 \cdot i & 0.623491 + 0.778369 \\ -2.08316 + 2.63205 \cdot i & 1.79018 - 1.27182 \end{bmatrix}$$

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

cosh()

Catalogue > 23

In Degree angle mode:

cosh()

Catalogue > 🕄

 $cosh(Value1) \Rightarrow value$ $cosh(List1) \Rightarrow list$

1.74671E19

cosh(Value 1) returns the hyperbolic cosine of the argument.

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

 $cosh(squareMatrix1) \Rightarrow squareMatrix$

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-1(1)

cosh-'({1,2.1,3})

cosh-1()

Catalogue > 🗐

{0,1.37286,1.76275}

 $cosh-1(Value1) \Rightarrow value$ $cosh-1(List1) \Rightarrow list$

cosh-1(*Value1*) returns the inverse hyperbolic cosine of the argument.

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 (...).

 $cosh-1(squareMatrix 1) \Rightarrow squareMatrix$

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:

$$\begin{bmatrix}
\cosh^{-1} & 4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}$$

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

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

cot()



$$cot(Value 1) \Rightarrow value$$

 $cot(List 1) \Rightarrow list$

cot(45) 1.

Returns the cotangent of *Value1* or returns a list of the cotangents of all elements in List1.

In Gradian angle mode:

In Degree angle mode:

cot(50) 1.

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

In Radian angle mode:

cot⁻¹()



 $cot-1(Value 1) \Rightarrow value$ $\cot -1(List 1) \Rightarrow list$

In Degree angle mode:

cot-1(1) 45.

Returns the angle whose cotangent is Value 1 or returns a list containing the inverse cotangents of each element of List1.

In Gradian angle mode:

cot-1(1) 50.

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

In Radian angle mode:

cot-1(1) 0.785398

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

coth()

Catalogue > 🗐

 $coth(Value1) \Rightarrow value$ $coth(List1) \Rightarrow list$

coth(1.2) 1.19954 $coth(\{1,3.2\})$ {1.31304,1.00333}

Returns the hyperbolic cotangent of *Value1* or returns a list of the hyperbolic cotangents of all elements of List1.

coth-1()

Catalogue > 23

 $coth-1(Value 1) \Rightarrow value$ $coth-1(List 1) \Rightarrow list$ coth⁻¹(3.5) 0.293893 coth⁻¹({-2,2.1,6}) {-0.549306,0.518046,0.168236}

Returns the inverse hyperbolic cotangent of *Value1* or returns a list containing the inverse hyperbolic cotangents of each element of *List1*.

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

count(Value lor List 1 [,Value 2 or List 2 [,...]]**)** $\Rightarrow value$

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.

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 210.

count(2,4,6)		3
count({2,4,6})		3
$\overline{\operatorname{count}\left(2,\left\{4,6\right\},\left[8\atop12\right]}$	10 14	7

countif() Catalogue > [3]

 $countif(List,Criteria) \Rightarrow value$

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

Criteria can be:

· A value, expression, or string. For

 $countIf(\{1,3,"abc",undef,3,1\},3)$

Counts the number of elements equal to 3.

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

example, **3** counts only those elements in *List* that simplify to the value 3.

 A Boolean expression containing the symbol ? as a place holder 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 210.

Note: See also sumif(), page 148, and frequency(), page 55.

Counts the number of elements equal to "def."

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

Counts 1 and 3.

$$\overline{\text{countIf}(\{1,3,5,7,9\},2<8)}</math$$

Counts 3, 5, and 7.

Counts 1, 3, 7, and 9.

cPolyRoots()

 $cPolyRoots(Poly,Var) \Rightarrow list$

 $cPolyRoots(ListOfCoeffs) \Rightarrow list$

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 expanded form in one variable. Do not use unexpanded forms such as $y^2 \cdot y + I$ or $x \cdot x + 2 \cdot x + I$

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

Note: See also polyRoots(), page 111.

Catalogue > 🕮

$polyRoots(y^3+1,y)$	{-1}
cPolyRoots(y ³ +1,y)	
{-1,0.5-0.866025 -i ,0.5+	0.866025 -i }
$polyRoots(x^2+2\cdot x+1,x)$	{-1,-1}
cPolyRoots({1,2,1})	{-1,-1}

crossP()

 $crossP(List1, List2) \Rightarrow list$

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

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

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

 $crossP(Vector1, Vector2) \Rightarrow vector$

Returns a row or column vector (depending on the arguments) that is the cross product of *Vector I* and *Vector 2*.

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.

crossP([1	2 3],[4 5 6])	[-3 6 -3
crossP[1	2][3 4])	0 0 -2

csc()		trig key
$W \mapsto 1$	In Degree angle mode:	

 $csc(Value 1) \Rightarrow value$ $csc(List 1) \Rightarrow list$

Returns the cosecant of *Value1* or returns a list containing the cosecants of all elements in *List1*.

csc(45) 1.41421

In Gradian angle mode:

csc(50) 1.41421

In Radian angle mode:

 $\csc\left\{\left\{1, \frac{\pi}{2}, \frac{\pi}{3}\right\}\right\} \qquad \left\{1.1884, 1., 1.1547\right\}$

csc-1() trig key

 $csc-1(Value 1) \Rightarrow value$ $csc-1(List 1) \Rightarrow list$

Returns the angle whose cosecant is Value 1 or returns a list containing the inverse cosecants of each element of List 1.

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 arcsc(...).

In Degree angle mode:

csc-1(1) 90.

In Gradian angle mode:

csc⁻¹(1) 100.

In Radian angle mode:

 $csc^{-1}(\{1,4,6\}) = \{1.5708, 0.25268, 0.167448\}$

csch()

Catalogue > 💱

 $csch(Value 1) \Rightarrow value$

 $csch(List1) \Rightarrow list$

Returns the hyperbolic cosecant of *Value1* or returns a list of the hyperbolic cosecants of all elements of *List1*.

csch(3)	0.099822
csch({1,2.1,4})	
{0.850918,	0.248641,0.036644

csch-1() Catalogue > 🗐

 $\operatorname{csch-1}(Value) \Rightarrow value$ $\operatorname{csch-1}(List1) \Rightarrow list$

Returns the inverse hyperbolic cosecant of Value 1 or returns a list containing the inverse hyperbolic cosecants of each element of List 1.

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

 csch³(1)
 0.881374

 csch³({1,2.1,3})
 {0.881374,0.459815,0.32745}

CubicReg

Catalogue > 🗐

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

Computes the cubic polynomial regression $y=a•x^3+b•x^2+c•x+d$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable. (See page 144.)

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 or string 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 210.

Output variable	Description	
stat.RegEqn	Regression equation: a•x³+b•x²+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()

Catalogue > 🗐

 $cumulativeSum(List1) \Rightarrow list$

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

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

 $cumulativeSum(Matrix1) \Rightarrow matrix$

Returns a matrix of the cumulative sums of the elements in *Matrix I*. 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 210.

1 2		1	2
3 4 -	→ m1	3	4
5 6		5	6
cumula	tiveSum $(m1)$	1	2
		4	6
		9	12

Cycle

Catalogue > 🗐

Cycle

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

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

Cycle

Catalogue > 23

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

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

Define g	()=Func	Done
	Local temp,i	
	$0 \rightarrow temp$	
	For <i>i</i> ,1,100,1	
	If <i>i</i> =50	
	Cycle	
	$temp+i \rightarrow temp$	
	EndFor	
	Return temp	
	EndFunc	
g()		5000

► Cylind

Catalogue > 🗐

Vector ▶ Cylind

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

Displays the row or column vector in cylindrical form $[r, \angle \theta, z]$.

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

[2 2 3]▶Cylind 2.82843 ∠0.785398 3.

D

dbd()		Catalogue > 🚉
$dbd(date1, date2) \Rightarrow value$	dbd(12.3103,1.0104)	1
Returns the number of days between date l	the actual-day-count $\frac{dod(12.3103,1.0104)}{dbd(1.0107,6.0107)}$	151
and <i>date2</i> using the actual-day-count	dbd(3112.03,101.04)	1
method.	dbd(101.07,106.07)	151
date I and date 2 can be numbers or lists of numbers within the range of the dates on		

the standard calendar. If both date 1 and date2 are lists, they must be the same length.

date 1 and date 2 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.

dbd()

Catalogue > 🗐

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

DDMM.YY (format use commonly in Europe)

▶DD Catalogue > []3

Expr1 **▶DD**⇒value

List1 ▶DD⇒list

Matrix1 ▶**DD**⇒*matrix*

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

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

In Degree angle mode:

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

In Gradian angle mode:

1▶DD	9
	10

In Radian angle mode:

1/₃ ▶Decimal

_		
- 1	1.5)▶DD	85.9437°

▶Decimal Catalogue > [1]3

Number1 ▶Decimal⇒value

List1 ▶Decimal⇒value

Matrix1 ▶Decimal⇒*value*

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.

0.333333

Define Catalogue > [1]

Define Var = Expression

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

Expression

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

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

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

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

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

EndFunc

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

EndPrgm

In this form, the user-defined function or programme 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**).

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

Note: See also **Define LibPriv**, page 37, and **Define LibPub**, page 37.

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 $h(x)$ =when $(x<2,2\cdot x-3,-2\cdot x+3)$	Done
h(-3)	-9
h(4)	-5

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

Define g(x,y)=Prgm

Disp
$$x$$
," greater than ", y Else
Disp x ," not greater than ", y EndIf
EndPrgm

Done

 $g(3,-7)$

3 greater than -7

Done

If x>y Then

Define LibPriv

Catalogue > 🕮

Define LibPriv Var = Expression

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

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

EndFunc

EndPrgm

Define LibPriv Program(Param1, Param2, ...) = Block

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

Note: See also Define, page 35, and Define LibPub, page 37.

Define LibPub

Catalogue > 🗐

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 programme. Public functions and programs appear in the Catalogue after the library has been saved and refreshed.

Note: See also Define, page 35, and Define LibPriv, page 37.

deltaList()

See Δ List(), page 81.

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

DelVar Var.

DelVar

Deletes the specified variable or variable group from memory.

If one or more of the variables are locked. this command displays an error message and deletes only the unlocked variables. See unLock, page 161.

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.

$2 \rightarrow a$	2
$(a+2)^2$	16
DelVar a	Done
$(a+2)^2$	"Error: Variable is not defined"

Catalogue > 🕮

aa.a:=45			45
aa.b:=5.67			5.67
aa.c:=78.9			78.9
getVarInfo()	aa.a	"NUM"	"[]"]
	aa.b	"NUM"	"[]"
	aa.c	"NUM"	"[]"]
DelVar aa.			Done
getVarInfo()	•	"N	ONE"

delVoid()

$delVoid(List1) \Rightarrow list$

Returns a list that has the contents of List1with all empty (void) elements removed.

For more information on empty elements, see page 210.

Catalogue > 23

Catalogue > [3]

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

det() det(squareMatrix[, *Tolerance*])⇒*expression* Returns the determinant of *squareMatrix*.

$\det\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	-2
$ \begin{bmatrix} 1.E20 & 1 \\ 0 & 1 \end{bmatrix} \rightarrow mat1 $	1. E 20 1 0 1
det(mat1)	0
det(mat1,.1)	1. E 20

det()

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, computations are done using floatingpoint arithmetic.
- If *Tolerance* is omitted or not used, the default tolerance is calculated as:

5E-14 ·max(dim(squareMatrix)) · rowNorm(squareMatrix)

diag()		Catalogue > 📳
$diag(List) \Rightarrow matrix$	diag([2 4 6])	2 0 0
diag(rowMatrix)⇒matrix		$\begin{bmatrix} 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$

 $diag(columnMatrix) \Rightarrow matrix$

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

diag(squareMatrix)⇒rowMatrix

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

squareMatrix must be square.

dim()

4 6 8	4 6	8
1 2 3	1 2	3
[5 7 9]	5 7	9]
diag(Ans)	[4 2	9]

4()	J	araioBac - 🔩
dim(List)⇒integer	$\overline{\dim\bigl(\bigl\{0,1,2\bigr\}\bigr)}$	3
Returns the dimension of $List$.		
$dim(Matrix) \Rightarrow list$, [1 -1]	{3,2}
Returns the dimensions of matrix as a two- element list {rows, columns}.	$\begin{bmatrix} 2 & -2 \\ 3 & 5 \end{bmatrix}$	

Catalogue > [1]

dim() Catalogue > ② dim(String)⇒integer dim("Hello") 5

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

dim("Hello")	5
dim("Hello "&"there")	11

Disp

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 programme 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	

DispAt

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

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

The line number can be specified as an expression.

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

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

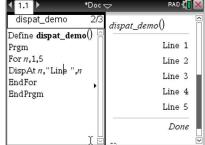
DispAtand Disp can be used within the same programme.

Catalogue > 🕎

Done

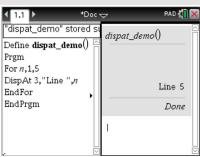
Catalogue > 🗐

DispAt Example 1.1 *Doc RAD (1) History dama 38



DispAt Catalogue > [2]

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



Illustrative examples:

_	<u>-</u>		
D	efine z()=	Outp	ut
Pr	gm	z()	
Fc	or n,1,3		Iteration 1:
Di	spAt 1,"N: ",n		Line 1: N:1
Di	sp "Hello"		Line 2: Hello
Er	ndFor		
Er	ndPrgm		Iteration 2:
			Line 1: N:2
			Line 2: Hello
			Line 3: Hello
			Iteration 3:
			Line 1: N:3
			Line 2: Hello
			Line 3: Hello
			Line 4: Hello
D	efine z1()=	z1()	
Pr	gm		Line 1: N:3
Fc	or n,1,3		Line 2: Hello
Di	spAt 1,"N: ",n		Line 3: Hello
Er	ndFor		Line 4: Hello
			Line 5: Hello
Fo	or n,1,4		
Di	isp "Hello"		
Er	ndFor		
Er	ndPrgm		

Error conditions:

Error Message	Description
DispAt line number must be between 1 and 8	Expression evaluates the line number outside the range 1-8 (inclusive)
Too few arguments	The function or command is missing one or more arguments.
No arguments	Same as current 'syntax error' dialogue
Too many arguments	Limit argument. Same error as Disp.
Invalid data type	First argument must be a number.
Void: DispAt void	"Hello World" Datatype error is thrown for the void (if the callback is defined)
	1

DMS	Catalogue > 🔯

Value ▶DMS

List DMS

Matrix ▶DMS

(-- ---\, ----

 $\begin{array}{lll} (45.371) \bullet {\rm DMS} & 45^{\circ} 22' 15.6" \\ \hline (\{45.371,60\}) \bullet {\rm DMS} & \{45^{\circ} 22' 15.6",60^{\circ}\} \end{array}$

In Degree angle mode:

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 °, ', '' (page 189) 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.

dotP()	Catalo	gue > 🗊
$dotP(List1, List2) \Rightarrow expression$	$dotP(\{1,2\},\{5,6\})$	17
Returns the "dot" product of two lists.		
$dotP(Vector1, Vector2) \Rightarrow expression$	dotP([1 2 3],[4 5 6])	32
Returns the "dot" product of two vectors.		

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

Ε

 $e^{(Value 1)} \Rightarrow value$

Returns **e** raised to the *Value1* power.

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

Note: Pressing e^x to display e^n (is different from pressing the character E on the keyboard.

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

$$e^{(List1)} \Rightarrow list$$

Returns e raised to the power of each element in List 1.

 $e^{(squareMatrix l)} \Rightarrow squareMatrix$

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

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

e^1	2.71828
e^{3^2}	8103.08

{1,1.,0.5}	{2.71828,2.71828,1.64872}

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

eff() Catalogue > [3]

eff(nominalRate,CpY**)** \Rightarrow 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.

nominal Rate must be a real number, and CpY must be a real number > 0.

Note: See also nom(), page 101.

eigVc()

Catalogue > [1]

 $eigVc(squareMatrix) \Rightarrow matrix$

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:

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

eigVc(m1)

To see the entire result, press \triangle and then use \triangleleft and \triangleright to move the cursor.

eigVI()

Catalogue > 😰

 $eigVI(squareMatrix) \Rightarrow list$

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

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

In Rectangular complex format mode:

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

eigVl(m1)

To see the entire result, press \triangle and then use \triangleleft and \triangleright to move the cursor.

Else

See If, page 67.

Catalogue > 🔯 ElseIf If BooleanExpr1 Then Define g(x)=Func Block1 If $x \le -5$ Then Elself BooleanExpr2 Then Return 5 Block2 ElseIf x > -5 and x < 0 Then Return -x Elself Boolean ExprN Then ElseIf $x \ge 0$ and $x \ne 10$ Then BlockNReturn x EndIf ElseIf x=10 Then Return 3 EndIf Note for entering the example: For EndFunc instructions on entering multi-line Done programme and function definitions, refer to the Calculator section of your product guidebook. **EndFor** See For, page 53. **EndFunc** See Func, page 56. **EndIf** See If, page 67. EndLoop See Loop, page 87. **EndPrgm** See Prgm, page 112. See Try, page 155. EndTry

EndWhile

See While, page 165.

euler ()

Catalogue > 🕮

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

euler(SystemOfExpr, Var, ListOfDepVars, $\{Var0, VarMax\},$ ListOfDepVars0, $VarStep[, eulerStep]) \Rightarrow matrix$

euler(ListOfExpr, Var, ListOfDepVars, {Var0, VarMax}, ListOfDepVars0, $VarStep[, eulerStep]) \Rightarrow matrix$

Uses the Euler method to solve the system $\frac{d \ depVar}{} = Expr(Var, depVar)$

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*).

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.

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

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

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

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

System of equations:

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

with v1(0)=2 and v2(0)=5

euler
$$\begin{cases} yI+0.1 \cdot yI \cdot y2 \\ 3 \cdot y2 - yI \cdot y2 \end{cases}$$
, $\{yI,y2\}, \{0,5\}, \{2,5\}, 1 \}$
 $\begin{bmatrix} 0. & 1. & 2. & 3. & 4. & 5. \\ 2. & 1. & 1. & 3. & 27. & 243. \\ 5. & 10. & 30. & 90. & 90. & -2070. \end{bmatrix}$

Catalogue > 23

euler ()

VarStep is a nonzero number such that sign (VarStep) = sign(VarMax-Var0) and solutions are returned at $Var0+i \cdot VarStep$ for all i=0,1,2,... such that $Var0+i \cdot VarStep$ 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.

eval () Hub Menu

 $eval(Expr) \Rightarrow string$

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"

Programme 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 programme.

1	Done	,
1		Done

eval () Hub Menu

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 58), GetStr (page 64), and Send (page 132).



Exit Catalogue > [1]

Exit

Exits the current For, While, or Loop block.

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

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

Function listing:

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

exp()		e ^x key
$exp(Value1) \Rightarrow value$	e^1	2.71828
Returns ${\it e}$ raised to the $Value 1$ power.	e ^{3²}	8103.08
Note: See also \boldsymbol{e} exponent template, page 2.		
Vou can enter a complex number in reiA		

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

 $\exp(List1) \Rightarrow list$

Returns e raised to the power of each element in List 1.

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

exp()

ex kev

 $exp(squareMatrix 1) \Rightarrow squareMatrix$

Returns the matrix exponential of square Matrix 1. This is not the same as calculating e raised to the power of each element. For information about the calculation method, refer to cos().

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

[1	l	5	3	782.209	559.617	456.509
4	Į.	2	1	680.546	488.795	396.521
ام ا	5	-2	1	524.929	371.222	307.879

expr() Catalogue > 💓

 $expr(String) \Rightarrow expression$

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

"Define cube(x)= x^3 " $\rightarrow funcstr$			
	"Define cube(x)=x^3"		
expr(funcstr)	Done		
cube(2)	8		

ExpReg Catalogue > 🕎

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

Computes the exponential regression $y = a \cdot (b)^x$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable. (See page 144.)

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 or string 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 210.

Output variable	Description
stat.RegEqn	Regression equation: a•(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 $Include$ $Categories$
stat.YReg	List of data points in the modified $YList$ 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

Catalogue > 23 factor()

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 100digit number could take more than a century.

To stop a calculation manually,

- Handheld: Hold down the Gion 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.

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

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() Catalogue > 23

FCdf

(lowBound,upBound,dfNumer,dfDenom)⇒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	Catalogue	> Q2
Fill Value, matrixVar⇒matrix	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow amatrix$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Replaces each element in variable matrix Var with Value.	[3 4] Fill 1.01,amatrix	[3 4] Done
matrixVar must already exist.	amatrix 1.01 1.01	1.01 1.01
Fill Value, listVar⇒list	$ \overline{\left\{1,2,3,4,5\right\} \rightarrow alist} \qquad \left\{1,2,3,4,5\right\} $,3,4,5}
Replaces each element in variable $listVar$ with $Value$.	Fill 1.01, <i>alist alist</i> {1.01,1.01,1.01,1.0	$\frac{Done}{1,1.01}$

listVar must already exist.

FiveNumSummary

Catalogue > 🕮

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 (page 144).

X represents a list containing the data.

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

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

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

An empty (void) element in any of the lists X, Freq, or Category results in a void for the corresponding element of all those lists. For more information on empty elements, see page 210.

Output variable	Description
stat.MinX	Minimum of x values.
stat.Q ₁ X	1st Quartile of x.
stat.MedianX	Median of x.
stat.Q ₃ X	3rd Quartile of x.
stat.MaxX	Maximum of x values.

floor()	Catalogue > 🗐

 $floor(Value 1) \Rightarrow integer$

floor(-2.14)

Returns the greatest integer that is \leq the argument. This function is identical to int().

The argument can be a real or a complex number.

 $floor(List1) \Rightarrow list$

 $floor(Matrix 1) \Rightarrow matrix$

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

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

floor $\left\{\frac{3}{2},0,-5.3\right\}$	{1,0,-6.}
floor $\begin{bmatrix} 1.2 & 3.4 \\ 2.5 & 4.8 \end{bmatrix}$	1. 3.
[2.5 4.8]	2. 4.

For

Catalogue > 🗐

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Define $g()$ =Func	Done
Local tempsum, step, i	
$0 \rightarrow tempsum$	
$1 \rightarrow step$	
For $i, 1, 100, step$	
$tempsum+i \rightarrow tempsum$	
EndFor	
EndFunc	
g()	5050

format() Catalogue > 🗊

format(Value[, formatString]**)**⇒string

Returns *Value* as a character string based on the format template.

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()

Catalogue > 23

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() Catalogue > [3]

 $fPart(Expr1) \Rightarrow expression$

 $\frac{\text{fPart}(-1.234)}{\text{fPart}(\{1,-2.3,7.003\})} \frac{-0.234}{\{0,-0.3,0.003\}}$

 $fPart(List1) \Rightarrow list$

 $fPart(Matrix 1) \Rightarrow matrix$

Returns the fractional part of the argument.

For a list or matrix, returns the fractional parts of the elements.

The argument can be a real or a complex number.

FPdf() Catalogue > [3]

 $\mathsf{FPdf}(XVal, dfNumer, dfDenom) \Rightarrow number \text{ if } XVal \text{ is a number, } list \text{ if } XVal \text{ is a list}$

Computes the F distribution probability at XVal for the specified dfNumer (degrees of freedom) and dfDenom.

freqTable list()

Catalogue > 🗐

freqTable | $list(List1, freqIntegerList) \Rightarrow list$

Returns a list containing the elements from List1 expanded according to the frequencies in freqIntegerList. This function can be used for building a frequency table for the Data & Statistics application.

freqTable
$$\blacktriangleright$$
 list($\{1,2,3,4\},\{1,4,3,1\}$) $\{1,2,2,2,2,3,3,3,4\}$ freqTable \blacktriangleright list($\{1,2,3,4\},\{1,4,0,1\}$) $\{1,2,2,2,2,4\}$

List1 can be any valid list.

freqIntegerList must have the same dimension as List1 and must contain nonnegative 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 210.

frequency()

Catalogue > 🕮

 $frequency(List1,binsList) \Rightarrow list$

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

If binsList is $\{b(1), b(2), ..., b(n)\}$, the specified ranges are $\{? \le b(1), b(1) < ? \le b(2), ..., b(n-1) < ? \le b(n), b(n) > ?\}$. 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, ? \leq b(1)), countif(list, b(1)<? \leq b(2)), ..., countif(list, b(n-1)<? \leq b(n)), countif (list, b(n)>?)}.

Elements of List1 that cannot be "placed in a bin" are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 210.

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

Note: See also countif(), page 29.

Explanation of result:

- **2** elements from Datalist are ≤ 2.5
- **4** elements from Datalist are >2.5 and \leq 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 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 (page 144).

For H_a : $\sigma 1 > \sigma 2$, set Hypoth > 0

For H_a : $\sigma 1 \neq \sigma 2$ (default), set Hypoth = 0

For H_a : $\sigma 1 < \sigma 2$, set Hypoth < 0

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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	Sample means of the data sequences in List 1 and List 2
stat.x2_bar	
stat.n1, stat.n2	Size of the samples

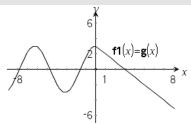
Func	Catalog	ue > 眞길
Func Block	Define a piecewise function:	
EndFunc	Define $g(x)$ =Func	Done
	If $x < 0$ Then	
Template for creating a user-defined	Return $3 \cdot \cos(x)$	
function.	Else	
	Return 3–x	
	EndIf	
	EndFunc	
	Result of graphing g(x)	

Func

Catalogue > 🕡

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 programme and function definitions, refer to the Calculator section of your product guidebook.



G

gcd() Catalogue > [3]

gcd(Number1, Number2**)**⇒expression

Returns the highest common factor 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.

$$gcd(List1, List2) \Rightarrow list$$

Returns the highest common factors of the corresponding elements in *List1* and *List2*.

 $gcd(Matrix1, Matrix2) \Rightarrow matrix$

Returns the highest common factors of the corresponding elements in *Matrix1* and *Matrix2*.

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

$$\gcd\begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}, \begin{bmatrix} 4 & 8 \\ 12 & 16 \end{bmatrix} \qquad \qquad \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

geomCdf() Catalogue > [3]

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

geomCdf(p,upBound**)**for P($1 \le X \le upBound$) $\Rightarrow number$ 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 $P(X \le upBound)$, set lowBound = 1.

Catalogue > 23 geomPdf()

geomPdf $(p,XVal) \Rightarrow number$ 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) [, status Var]

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

The value must be requested:

- In advance, through a Send "READ ..." command.
 - or —
- By embedding a "READ ..." request as the optional *promptString* argument. This method lets you use a single command to request the value and retrieve it.

Implicit simplification takes place. For example, a received string of "123" is interpreted as a numeric value. To preserve the string, use GetStr instead of Get.

If you include the optional argument status Var, it is assigned a value based on the success of the operation. A value of zero means that no data was received.

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 BRIGHTNE	SS",lightval	Done
lightval	0.37	78441

Get Hub Menu

In the second syntax, the *func()* argument allows a programme to store the received string as a function definition. This syntax operates as if the programme executed the command:

Define func(arg1, ...argn) = received string

The programme can then use the defined function *func*().

Note: You can use the **Get** command within a user-defined programme but not within a function.

Note: See also **GetStr**, page 64 and **Send**, page 132.

getDenom()

$getDenom(Fraction1) \Rightarrow value$

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

	Catalogue > 🕎
<i>x</i> :=5: <i>y</i> :=6	6
$getDenom \left(\frac{x+2}{y-3} \right)$	3
$getDenom\left(\frac{2}{7}\right)$	7
$ getDenom \left(\frac{1}{x} + \frac{y^2 + y}{y^2} \right) $	30

getKey()

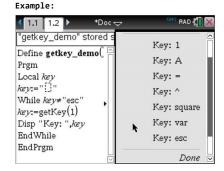
getKey([0|1]) ⇒ returnString

Description:getKey() - allows a TI-Basic programme to get keyboard input - handheld, desktop and emulator on desktop.

Example:

- keypressed := getKey() will return a key or an empty string if no key has been pressed. This call will return immediately.
- keypressed := getKey(1) will wait till a key is pressed. This call will pause

getKey()



Catalogue > 23

Catalogue > 📳

execution of the programme till a key is pressed.

Handling of key presses:

Handheld Device/Emulator Key	Desktop	Return Value
Esc	Esc	"esc"
Touchpad - Top click	n/a	"up"
On	n/a	"home"
Countainage	- /-	Usanskahus adli
Scratchapps Tavalanda Laft aliah	n/a	"scratchpad"
Touchpad - Left click	n/a	"left"
Touchpad - Centre click	n/a	"centre"
Touchpad - Right click	n/a	"right"
Doc	n/a	"doc"
Tab	Tab	"tab"
Touchpad - Bottom click	Down Arrow	"down"
Menu	n/a	"menu"
Ctrl	Ctrl	no return
Shift	Shift	no return
Var	n/a	"var"
Del	n/a	"del"
		"="
trig	= n/a	= "trig"
0 to 9	0-9	"0" "9"
Templates	n/a	"template"
Catalogue	n/a	"cat"
	.,,	Cut
٨	۸	пУп
X^2	n/a	"square"
/ (division key)	/	"/"

Handheld Device/Emulator Key	Desktop	Return Value
* (multiply key)	*	п*п
e^x	n/a	"exp"
10^x	n/a	"10power"
+	+	"+"
-	-	u_u
(("("
))	")"
		" "
(-)	n/a	"-" (negate sign)
Enter	Enter	"enter"
ee	n/a	"E" (scientific notation E)
a - z	a-z	alpha = letter pressed (lower case) ("a" - "z")
shift a-z	shift a-z	alpha = letter pressed "A" - "Z"
		Note: ctrl-shift works to lock caps
?!	n/a	"?!"
pi	n/a	"pi"
Flag	n/a	no return
,	,	" "
Return	n/a	"return"
Space	Space	" " (space)
Inaccessible	Special Character Keys like @,!,^, etc.	The character is returned
n/a	Function Keys	No returned character
n/a	Special desktop control keys	No returned character
Inaccessible	Other desktop keys that are	Same character you get in

Handheld Device/Emulator Key	Desktop	Return Value	
	not available on the calculator while getkey() is waiting for a keystroke. ({, },;, :,)	Notes (not in a maths box)	

Note: It is important to note that the presence of getKey() in a programme changes how certain events are handled by the system. Some of these are described below.

Terminate programme and Handle event - Exactly as if the user were to break out of programme by pressing the **ON** key

"Support" below means - System works as expected - programme continues to run.

Event	Device	Desktop - TI-Nspire™ Student Software
Quick Poll	Terminate programme, handle event	Same as the handheld (TI- Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)
Remote file mgmt	Terminate programme, handle event	Same as the handheld. (TI-Nspire™ Student
(Incl. sending 'Exit Press 2 Test' file from another handheld or desktop- handheld)		Software, TI-Nspire™ Navigator™ NC Teacher Software-only)
End Class	Terminate programme,	Support
	handle event	(TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)

Event	Device	Desktop - TI-Nspire™ All Versions
TI-Innovator™ Hub connect/disconnect	Support - Can successfully issue commands to the TI-Innovator™ Hub. After you exit the programme the TI-Innovator™ Hub is still working with the handheld.	Same as the handheld

getLangInfo()	Catalogue > 🎚	
$getLangInfo() \Rightarrow string$	getLangInfo()	"en"

getLangInfo()

Catalogue > 🗐

Returns a string that corresponds to the short name of the currently active language. You can, for example, use it in a programme or function to determine the current language.

English = "en"
Danish = "da"
German = "de"
Finnish = "fi"
French = "fr"
Italian = "it"
Dutch = "nl"
Belgian Dutch = "nl_BE"
Norwegian = "no"
Portuguese = "pt"
Spanish = "es"
Swedish = "sv"

getLockInfo()

Catalogue > 🗐

$getLockInfo(Var) \Rightarrow value$

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 84, and unLock, page 161.

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*

 $getMode(0) \Rightarrow list$

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.

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

For a listing of the modes and their settings, refer to the table below.

If you save the settings with $getMode(0) \rightarrow var$, you can use setMode(var) in a function or programme to temporarily restore the settings within the execution of the function or programme only. See setMode(), page 135.

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
Vector Format	6	1=Rectangular, 2=Cylindrical, 3=Spherical
Base	7	1=Decimal, 2=Hex, 3=Binary

getNum()		Catalogue > 🔃
$getNum(Fraction1) \Rightarrow value$	x:=5: y:=6	6
Transforms the argument into an expression having a reduced common denominator, and then returns its numerator.	$\operatorname{getNum}\left(\frac{x+2}{y-3}\right)$	7
	$\operatorname{getNum}\left(\frac{2}{7}\right)$	2
	$getNum\left(\frac{1}{x} + \frac{1}{y}\right)$	11

GetStr	Hub Menu

GetStr[promptString,] var[, statusVar]

For examples, see **Get**.

GetStr Hub Menu

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

Programming command: Operates identically to the **Get** command, except that the retrieved value is always interpreted as a string. By contrast, the **Get** command interprets the response as an expression unless it is enclosed in quotation marks ("").

Note: See also Get, page 58 and Send, page 132.

getType()	Catalogue > 🗓	
getType(var)⇒string	$\{1,2,3\} \rightarrow temp$	{1,2,3}
Returns a string that indicates the data type of variable var .	getType(temp)	"LIST"
	$3 \cdot i \rightarrow temp$	3· i
If \emph{var} has not been defined, returns the string "NONE".	${\tt getType}(temp)$	"EXPR"
	DelVar temp	Done
	getType(temp)	"NONE"

getVarInfo() Catalogue > 23

 $getVarInfo() \Rightarrow matrix \text{ or } string$

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			L	one	
Lock x			L	one	
Define LibPriv $y = \{1,2,3\}$			Done		
Define LibPub $z(x)=3\cdot x^2-x$			L	Oone	
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()

a matrix.

Note the example to the left, 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

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 generalised matrix structure where an element of a matrix can be either a matrix or a list.

a:=1				1
$b := \begin{bmatrix} 1 & 2 \end{bmatrix}$			[1	2]
c:=[1 3 7]			[1 3	7]
vs:=getVarInfo()	a	"NUM"	"[]"	0
	b	"MAT"	"[]"	0
	$\lfloor c$	"MAT"	"[]"	0
vs[1]	[1	"NUM"	"[]"	0]
vs[1,1]				1
vs[2] "Er	ror: Ir	ıvalid list	or matr	ix"
$\overline{vs[2,1]}$			[1	2]

Catalogue > 🕮

Catalogue > 🕮

Goto

Goto label Name

Transfers control to the label labelName.

lahelName must be defined in the same function using a LbI instruction.

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

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

Grad

Catalogue > 23

55

 $Expr1
ightharpoonup Grad \Rightarrow expression$

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 (1.66667)⁹

In Radian angle mode:

(1.5)▶Grad (95.493)⁹

identity()		Catalogue > 🕡
$identity(Integer) \Rightarrow matrix$	identity(4)	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$
Returns the identity matrix with a dimension of <i>Integer</i> .		$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Integer must be a positive integer		[]

$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
[0 0 0 1]
Catalogue > 👰
Define $g(x)$ =Func Done If $x < 0$ Then
Return x^2 EndIf EndFunc
g(-2) 4
Define $g(x)$ =Func Done If $x < 0$ Then
Return ⁻x
Else Return <i>x</i>
EndIf EndFunc
$\frac{g(12)}{g(-12)}$ 12

Block1 and *Block2* can be a single statement.

If BooleanExpr1 Then
Block1

ElseIf BooleanExpr2 Then
Block2

Elself 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 $x \ge 0$ and $x \ne 10$ Then	
Return x	
ElseIf $x=10$ Then	
Return 3	
EndIf	
EndFunc	
Γ)one

	Done
g(-4)	4
g(10)	3

ifFn()

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}

Catalogue > 🕮

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 $\bf 3$ is not less than 2.5, so its corresponding $Value_If_False$ element of $\bf 10$ 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()

Catalogue > 🗐

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).

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

Value_If_false is not specified. Undef is used.

One element selected from $Value_If_true$.
One element selected from $Value_If_unknown$.

imag()

Catalogue > 🗐

 $imag(Value 1) \Rightarrow value$

 $imag(1+2\cdot i)$

2

Returns the imaginary part of the argument.

 $imag(\{-3,4-i,i\})$

{0,-1,1}

 $imag(List1) \Rightarrow list$

Returns a list of the imaginary parts of the elements.

 $imag(Matrix 1) \Rightarrow matrix$

string *subString* begins.

Returns a matrix of the imaginary parts of the elements.

$\operatorname{imag} \begin{bmatrix} 1 & 2 \\ i \cdot 3 & i \cdot 4 \end{bmatrix}$

$$\begin{bmatrix} 0 & 0 \\ 3 & 4 \end{bmatrix}$$

Indirection

See #(), page 187.

inString()

Catalogue > 😰

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

Returns the character position in string srcString at which the first occurrence of

Start, if included, specifies the character position within *srcString* where the search begins. Default = 1 (the first character of *srcString*).

If srcString does not contain subString or Start is > the length of srcString, returns zero.

int() Catalogue > 🗐

 $int(Value) \Rightarrow integer$ $int(List1) \Rightarrow list$ $int(Matrix1) \Rightarrow matrix$

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

Returns the greatest integer that is less than or equal to the argument. This function is identical to floor().

The argument can be a real or a complex number.

For a list or matrix, returns the greatest integer of each of the elements.

intDiv() Catalogue > [3]

intDiv(Number1, Number2) \Rightarrow integer intDiv(List1, List2) \Rightarrow list intDiv(Matrix1, Matrix2) \Rightarrow matrix

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

Returns the signed integer part of $(Number 1 \div Number 2)$.

For lists and matrices, returns the signed integer part of (argument 1 ÷ argument 2) for each element pair.

interpolate ()

Catalogue >

interpolate(xValue, xList, yList, yPrimeList) $\Rightarrow list$

This function does the following:

Differential equation: $y'=-3 \cdot y + 6 \cdot t + 5$ and y(0)=5

To see the entire result, press \triangle and then use \triangleleft and \triangleright to move the cursor.

interpolate ()

Catalogue > 23

Given xList, yList= $\mathbf{f}(xList)$, and yPrimeList= $\mathbf{f}'(xList)$ for some unknown function \mathbf{f} , a cubic interpolant is used to approximate the function \mathbf{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 $\mathbf{f}(xValue)$; otherwise, it returns \mathbf{undef} .

xList, *yList*, and *yPrimeList* must be of equal dimension ≥ 2 and contain expressions that simplify to numbers.

xValue can be a number or a list of numbers.

Use the interpolate() function to calculate the function values for the xvaluelist:

xvaluelist = seq(i,i,0,10,0.5)

{0,0.5,1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,}

 $x list := mat \triangleright list (rk[1])$

{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

 $yprimelist:=-3\cdot y+6\cdot t+5|y=y|$ and t=xlist

interpolate(xvaluelist,xlist,ylist,yprimelist)

{-10.,1.41503,1.98819,2.00129,1.98221,2.006•

{5,2.67062,3.19499,4.02782,5.00394,6.00011

invχ2()

Catalogue > 📳

inv_χ²(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()

Catalogue > 🕄

invF(Area,dfNumer,dfDenom)

invF(Area,dfNumer,dfDenom)

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

invBinom

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

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 cumulative probability of k successes 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%?

invBinom $\left(0.77,30,\frac{1}{6}\right)$	6
invBinom $\left(0.77,30,\frac{1}{6},1\right)$	5 0.616447 6 0.776537

invBinomN()

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

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 cumulative probability of x successes 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.

Catalogue > 23

Example: Monique is practising goal shots for netball. She knows from experience that her chance of making any one shot is 70%. She plans to practise 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?

invNorm()

Catalogue > 🗐

 $invNorm(Area[,\mu[,\sigma]])$

Computes the inverse cumulative normal distribution function for a given Area under the normal distribution curve specified by μ and σ .

invt()

Catalogue > 🗐

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() Catalogue > [2]

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

 $\frac{\text{iPart}(-1.234)}{\text{iPart}\left(\frac{3}{2}, -2.3, 7.003\right)} -1.$

Returns the integer part of the argument.

For lists and matrices, returns the integer part of each element.

The argument can be a real or a complex number.

irr() Catalogue > [3]

 $irr(CF0,CFList [,CFFreq]) \Rightarrow value$

Financial function that calculates internal rate of return of an investment.

CF0 is the initial cash flow at time 0; it must be a real number.

CFList is a list of cash flow amounts after the initial cash flow CFO.

CFFreq is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of CFList. The default is 1; if you enter values, they must be positive integers < 10,000.

Note: See also mirr(), page 93.

$\frac{list1:=\{6000, -8000, 2000, -3000\}}{\{6000, -8000, 2000, -3000\}}$ $\frac{list2:=\{2, 2, 2, 1\}}{iir(5000, list1, list2)} \qquad \frac{2, 2, 2, 1}{4, 64484}$

isPrime() Catalogue > [1]

isPrime(Number**)** \Rightarrow 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.

isPrime(5) true isPrime(6) false



If Number exceeds about 306 digits and has no factors \leq 1021, **isPrime**(Number) displays an error message.

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

Function to find the next prime after a specified number:

Define nextprim(n)=Func		Done
	Loop	
	$n+1 \rightarrow n$	
	If $isPrime(n)$	
	Return n	
	EndLoop	
	EndFunc	
nextprim(7)		11

 $a := _{_}$

isVoid(a)

isVoid({1, .3})

Define g = Func

isVoid()

 $isVoid(Var) \Rightarrow Boolean \ constant \ expression$

 $isVoid(Expr) \Rightarrow Boolean \ constant \ expression$

 $isVoid(List) \Rightarrow list of Boolean constant expressions$

Returns true or false to indicate if the argument is a void data type.

For more information on void elements, see page 210.

– true

Catalogue > 🕮

{ false,true,false }

L

Lbl

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Catalogue > 🗐

Done

Define Sy	1 dile	Done
	Local temp,i	
	$0 \rightarrow temp$	
	$1 \rightarrow i$	
	Lbl top	
	$temp+i \rightarrow temp$	
	If $i < 10$ Then	
	$i+1 \rightarrow i$	
	Goto top	
	EndIf	
	Return temp	
	EndFunc	
g()		55

lcm()

Catalogue > 🕮

"He'

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

 $lcm(List1, List2) \Rightarrow list$

lcm(6,9) $\frac{2}{-14,80}$

 $lcm(Matrix1, Matrix2) \Rightarrow matrix$

Returns the least common multiple of the two arguments. The Icm of two fractions is the Icm of their numerators divided by the gcd of their denominators. The Icm of fractional floating-point numbers is their product.

For two lists or matrices, returns the least common multiples of the corresponding elements.

Catalogue > 23 left()

left("Hello",2)

left(sourceString[, Num])⇒string

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

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

$$left(List1[, Num]) \Rightarrow list$$

left({1,3,-2,4},3) {1,3,-2} Returns the leftmost Num elements

contained in List1. If you omit *Num*, returns all of *List1*.

 $left(Comparison) \Rightarrow expression$

Returns the left-hand side of an equation or inequality.

libShortcut() Catalogue > 🕮

libShortcut(LibNameString, ShortcutNameString [, LibPrivFlag]) $\Rightarrow list of variables$

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

libShortcut()

Catalogue > 23

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, page

To delete a variable group, see DelVar, page 38.

```
getVarInfo("linalg2")
          clearmat "FUNC"
                                "LibPub "
                     "PRGM"
                                "LibPriv "
           gauss1
                     "FUNC"
                                "LibPub "
           gauss2
libShortcut("linalg2","la")
                    { la.clearmat,la.gauss2 ]
libShortcut("linalg2","la",1)
          \{la.clearmat, la.gauss 1, la.gauss 2\}
```

LinRegBx

Catalogue > 🕮

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

Computes the linear regressiony = $a+b \cdot xon$ lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

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 or string 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.

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

Catalogue > 🗐

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

Computes the linear regression $y = m \cdot x + b$ on lists X and Y with frequency Freq. A summary of results is stored in the *stat.results* variable (page 144).

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 or string 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.

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

Catalogue > 🗐

LinRegtIntervals X,Y[F,O[Cev]]

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

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

For Response. Computes a predicted y-value, a level C prediction interval for a single observation and a level C confidence interval for the mean response.

A summary of results is stored in the *stat.results* variable (page 144).

All the lists must have equal dimension.

X and Y are lists of independent and dependent variables.

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

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,	Prediction interval for a single observation	
stat.UpperPred]		
stat.MEPred	Prediction interval margin of error	
stat.SEPred	Standard error for prediction	
stat.ŷ	a + b ·XVal	

LinRegtTest Catalogue > 😰

LinRegtTest X, Y[,Freq[,Hypoth]]

Computes a linear regression on the X and Y lists and a t test on the value of slope β and the correlation coefficient ρ for the equation $y=\alpha+\beta x$. It tests the null hypothesis H_0 : 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 ($H_0:\beta=\rho=0$) will be tested.

For H_a : $\beta \neq 0$ and $\rho \neq 0$ (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 (page 144).

Output variable	Description
stat.RegEqn	Regression equation: $a + b \cdot x$
stat.t	t-Statistic for significance test
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom
stat.a, stat.b	Regression coefficients
stat.s	Standard error about the line
stat.SESlope	Standard error of slope
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression

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

 $linSolve(LinearEqn1 \text{ and } LinearEqn2 \text{ and } ..., Var1, Var2, ...) \Rightarrow list$

linSolve({LinearEqn1, LinearEqn2, ...}, Var1, Var2, ...) $\Rightarrow list$

linSolve(SystemOfLinearEqns, {Var1, Var2, ...}) $\Rightarrow list$

linSolve(LinearEqn1 and LinearEqn2 and ..., {Var1, Var2, ...}) $\Rightarrow list$

linSolve({LinearEqn1, LinearEgn2, ...}, {Var1, Var2, ...}) $\Rightarrow list$

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

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.

linSolve $\left\{ \begin{cases} 2 \cdot x + 4 \cdot y = 3 \\ 5 \cdot x - 3 \cdot y = 7 \end{cases}, \left\{ x_{i,y} \right\} \right\}$	$\left\{\frac{37}{26}, \frac{1}{26}\right\}$
linSolve $\left\{ \begin{cases} 2 \cdot x = 3 \\ 5 \cdot x - 3 \cdot y = 7 \end{cases}, \left\{ x_i y \right\} \right\}$	$\left\{\frac{3}{2},\frac{1}{6}\right\}$
linSolve $\begin{cases} apple+4 \cdot pear=23 \\ 5 \cdot apple-pear=17 \end{cases}$	
	$\left\{\frac{13}{3}, \frac{14}{3}\right\}$
linSolve $\begin{cases} apple \cdot 4 + \frac{pear}{3} = 14, \\ -apple + pear = 6 \end{cases}$	apple,pear}
	$\left\{ \frac{36}{13}, \frac{114}{13} \right\}$

Δ List()

 Δ List(List1) $\Rightarrow list$

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.

Catalogue > 🗐

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

list>mat()

Catalogue > [3]

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, zeroes are added.

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

list▶mat({1,2,3})	[1 2 3]
list ▶ mat({1,2,3,4,5},2)	1 2
	3 4 5 0
	5 0

In()

In(Value1)⇒value

 $In(List1) \Rightarrow list$

Returns the natural logarithm of the argument.

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

ln(2.)

ex kevs 0.693147

If complex format mode is Real:

$$ln(\{-3,1.2,5\})$$

"Error: Non-real calculation"

 $In(squareMatrix 1) \Rightarrow squareMatrix$

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 diagonalisable. The result always contains floating-point numbers.

If complex format mode is Rectangular:

$$\frac{\ln(\{-3,1.2,5\})}{\{1.09861+3.14159 \cdot \mathbf{i}, 0.182322, 1.60944\}}$$

In Radian angle mode and Rectangular complex format:

$$\ln \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$$

1.83145+1.73485·*i* 0.009193-1.49086 0.448761-0.725533·i 1.06491+0.623491 -0.266891-2.08316·*i* 1.12436+1.79018·

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

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

Computes the logarithmic regression $y = a+b \cdot ln(x)$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

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 \geq 0.

Category is a list of numeric or string 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.

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

Catalogue > 🕮

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Define rollcount	()=Func
	Local i
	$1 \rightarrow i$
	Loop
	If $randInt(1,6)=randInt(1,6)$
	Goto end
	$i+1 \rightarrow i$
	EndLoop
	Lbl end
	Return i
	EndFunc
	Done
rollcount()	16
rollcount()	3

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

Lock Var.

Locks the specified variables or variable group. Locked variables cannot be modified or deleted.

You cannot lock or unlock the system variable Ans, and you cannot lock the system variable groups stat. or tvm.

Note: The Lock command clears the Undo/Redo history when applied to unlocked variables.

See unLock, page 161, andgetLockinfo(), page 63.

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()

log(Value1[,Value2])⇒value

 $log(List1[,Value2]) \Rightarrow list$

Returns the base-Value 2 logarithm of the first argument.

Note: See also Log template, page 2.

For a list, returns the base-Value 2 logarithm of the elements.

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

$\log_{10}(2.)$	0.30103
$\frac{\log_4(2.)}{}$	0.5
$\frac{\log_{3}(10) - \log_{3}(5)}{\log_{3}(10) - \log_{3}(5)}$	0.63093

If complex format mode is Real:

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

"Error: Non-real calculation"

If complex format mode is Rectangular:

$$\frac{\log_{10}(\{-3,1.2,5\})}{\{0.477121+1.36438\cdot \mathbf{i},0.079181,0.69897\}}$$

In Radian angle mode and Rectangular complex format:

$$\log_{10} \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$$

0.795387+0.753438·i 0.003993-0.64747 0.194895-0.315095·i 0.462485+0.2707? $-0.115909 - 0.904706 \cdot i \quad 0.488304 + 0.77746$

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

log(squareMatrix1 [,Value])⇒squareMatrix

Returns the matrix base-Value logarithm of squareMatrix1. This is not the same as calculating the base-Value logarithm of each element. For information about the calculation method, refer to cos().

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

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

Logistic

Catalogue > 🕮 **Logistic** X, Y[, [Freq] [, Category, Include]]

Computes the logistic regressiony = $(c/(1+a \cdot e^{-bx}))$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

All the lists must have equal dimension except for Include.

Logistic

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 or string 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 210.

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

Catalogue > 🕄 LogisticD

LogisticD X, Y [, [Iterations], [Freq] [, Category, Includel 1

Computes the logistic regression $y = (c/(1+a \cdot e^{-bx})+d)$ on lists X and Y with frequency Freq, using a specified number of *Iterations*. A summary of results is stored in the *stat.results* variable (page 144).

All the lists must have equal dimension except for Include.

X and Y are lists of independent and dependent variables.

LogisticD

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 or string 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.

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	Catalogue > ৠু
Loop	Define rollcount()=Func
D1 1	Local i
Block	$1 \rightarrow i$
Fudi con	Loop
EndLoop	If randInt $(1,6)$ =randInt $(1,6)$
Repeatedly executes the statements in	Goto end
Block. Note that the loop will be executed	$i+1 \rightarrow i$
endlessly, unless a Goto or Exit instruction	EndLoop
is executed within $Block$.	Lbl end
is executed within block.	Return i
<i>Block</i> is a sequence of statements	EndFunc
separated with the ":" character.	Done
	rollcount() 16
	rollcount() 3

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

LU Catalogue > 23

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*.

 $lMatrix \cdot uMatrix = pMatrix \cdot matrix$

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

- If you use ctri enter or set the Auto or Approximate mode to Approximate, computations are done using floatingpoint arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as: $5E-14 \cdot max(dim(Matrix)) \cdot rowNorm$ (Matrix)

The **LU** factorization algorithm uses partial pivoting with row interchanges.

6	12	18		6	12	18
5	14	31	→ m1	5	14	31
3	8	18		3	8	18]
LU	m1	low	er,upper,perm		L	one
lon	er			1	0	0
				<u>5</u>	1	0
				$\frac{1}{2}$	$\frac{1}{2}$	1
ирр	er			6	12	18
				0	4	16
				0	0	1
per	m				1 0	0
					0 1	0
				L	0 0	1]

(...).

page 210.

Note: See also min().

matilist()		Catalogue > 📳
mat list($Matrix$) $\Rightarrow list$	mat▶list([1 2 3])	{1,2,3}
Returns a list filled with the elements in Matrix. The elements are copied from	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow m1$	[1 2 3] [4 5 6]
Matrix row by row. Note: You can insert this function from the computer keyboard by typing mat@>list.	mat≯list(m1)	{1,2,3,4,5,6}

max()		Catalogue > 🕎
$max(Value1, Value2) \Rightarrow expression$	max(2.3,1.4)	2.3
max(List1, List2)⇒list	$\max(\{1,2\},\{-4,3\})$	${1,3}$
max(Matrix1, Matrix2)⇒matrix		
Returns the maximum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the maximum value of each pair of corresponding elements.		
$max(List) \Rightarrow expression$	$\max(\{0,1,-7,1.3,0.5\})$	1.3
Returns the maximum element in $list.$		
max(Matrix1)⇒matrix	max[[1 -3 7]]	[1 0 7]
Returns a row vector containing the maximum element of each column in <i>Matrix1</i> .	-4 0 0.3	
Empty (void) elements are ignored. For more information on empty elements, see		

mean()	Catalo	ogue > 🗊
$mean(List[,freqList]) \Rightarrow expression$	mean({0.2,0,1,-0.3,0.4})	0.26
Returns the mean of the elements in $List.$	mean($\{1,2,3\},\{3,2,1\}$)	<u>5</u> 3

Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.

 $mean(Matrix 1[, freqMatrix]) \Rightarrow matrix$

Returns a row vector of the means of all the columns in *Matrix I*.

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 210.

In Rectangular vector format:

[-0.133333
$\begin{bmatrix} \frac{-2}{15} & \frac{5}{6} \end{bmatrix}$
$\begin{bmatrix} \frac{47}{15} & \frac{11}{3} \end{bmatrix}$

median()

 $median(List[,freqList]) \Rightarrow expression$

Returns the median of the elements in *List*.

Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.

 $median(Matrix 1[, freqMatrix]) \Rightarrow matrix$

Returns a row vector containing the medians of the columns in *Matrix 1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

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 210.

Catalogue > 🔃

 $median({0.2,0,1,-0.3,0.4})$ 0.2

$$\operatorname{median} \begin{bmatrix} 0.2 & 0 \\ 1 & -0.3 \\ 0.4 & -0.5 \end{bmatrix}$$

$$\begin{bmatrix} 0.4 & -0.3 \end{bmatrix}$$

MedMed

Catalogue > 🕎

MedMed X,Y [, Freq] [, Category, Include]]

MedMed

Computes the median-median liney = $(m \cdot x+b)$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

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 or string 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 210.

Output variable	Description
stat.RegEqn	Median-median line equation: m ·x+b
stat.m, stat.b	Model coefficients
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() Catalogue > 23 mid(sourceString, Start[, Count])⇒string mid("Hello there",2) "ello there" mid("Hello there",7,3) "the" Returns Count characters from character mid("Hello there",1,5) "Hello" string *sourceString*, beginning with character number Start. "[]" mid("Hello there",1,0)

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 \geq 0. If Count = 0, returns an empty string.

 $mid(sourceList, Start [, Count]) \Rightarrow list$

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 \geq 0. If Count = 0, returns an empty list.

mid(sourceStringList, Start[, Count])⇒list

Returns *Count* strings from the list of strings *sourceStringList*, beginning with element number *Start*.

mid({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)	{□}

min() Catalogue > []3

min(Value1, Value2)⇒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.

 $min(List) \Rightarrow expression$

Returns the minimum element of List.

 $min(Matrix 1) \Rightarrow matrix$

Returns a row vector containing the minimum element of each column in *Matrix I*.

Note: See also max().

min(2.3,1.4)	1.4
$\min(\left\{1,2\right\},\left\{-4,3\right\})$	{-4,2}

$$\min(\{0,1,-7,1.3,0.5\})$$
 -7

$$\min \begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix} \qquad \begin{bmatrix} -4 & -3 & 0.3 \end{bmatrix}$$

mirr() Catalogue > 🗐

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 CFO.

CFFreq is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of CFList. The default is 1; if you enter values, they must be positive integers < 10,000.

Note: See also irr(), page 73.

identities: mod(x,0) = x

mod(x,y) = x - y floor(x/y)

list1:={6000,-8000,2000,-3000}	
{6000,-800	00,2000,-3000}
list2:={2,2,2,1}	{2,2,2,1}
mirr(4.65,12,5000,list1,list2)	13.41608607

mod()	Catalogue >
$mod(Value1, Value2) \Rightarrow expression$	mod(7,0)
$mod(List1, List2) \Rightarrow list$	mod(7,3) mod(-7,3)
mod(Matrix1, Matrix2)⇒matrix	mod(7,-3)
Returns the first argument modulo the second argument as defined by the	$ \frac{\text{mod}(-7,-3)}{\text{mod}(\{12,-14,16\},\{9,7,-5\})} $

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

When the second argument is non-zero, the

Ŋ

Catalogue > 23

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 123

mRow() Catalogue > [3]

 $mRow(Value, Matrix 1, Index) \Rightarrow matrix$

Returns a copy of Matrix I with each element in row Index of Matrix I multiplied by Value.

${\text{mRow}}\begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$	2	1	2
$\begin{bmatrix} 3 & 1 \\ 3 & 4 \end{bmatrix}$,2	-1	$\frac{-4}{3}$

mRowAdd()

 \Rightarrow matrix

mRowAdd(Value, Matrix1, Index1, Index2)

Returns a copy of Matrix 1 with each element in row Index 2 of Matrix 1 replaced with:

Value · row Index1 + row Index2

mRowAdd
$$\begin{pmatrix} -3, \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, 1, 2 \end{pmatrix}$$
 $\begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix}$

MultReg Catalogue > [2]

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 (page 144).

All the lists must have equal dimension.

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.ŷ List	ŷ List = b0+b1 ·x1+

Output variable	Description
stat.Resid	Residuals from the regression

MultRegIntervals

Catalogue > 🗐

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 (page 144).

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

Output variable	Description
stat.RegEqn	Regression Equation: b0+b1 ·x1+b2 ·x2+
stat.ŷ	A point estimate: $\hat{y} = b0 + b1 \cdot xl +$ for $XValList$
stat.dfError	Error degrees of freedom
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,	Prediction interval for a single observation
stat.UpperrPred	
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

Catalogue > 🗐

MultRegTests Y, X1[X2[X3,...[X10]]]

Catalogue > 🕄

MultRegTests

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 (page 144).

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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 ${\cal F}$ 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
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	ŷ List = b0+b1 ·x1+
stat.Resid	Residuals from the regression
stat.sResid	Standardized residuals; obtained by dividing a residual by its standard deviation

Output variable	Description
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

BooleanExpr1nandBooleanExpr2 returns Boolean expression

BooleanList1nandBooleanList2 returns
Boolean list

BooleanMatrix1nandBooleanMatrix2 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.

Integer1 nand*Integer2* ⇒ *integer*

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

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

3 and 4	0
3 nand 4	-1
${1,2,3}$ and ${3,2,1}$	{1,2,1}
{1,2,3} nand {3,2,1}	{-2,-3,-2}

nCr()

Catalogue > 🕮

$nCr(Value1, Value2) \Rightarrow expression$

For integer Value 1 and Value 2 with $Value 1 \ge Value 2 \ge 0$, nCr() is the number of combinations of Value 1 things taken Value 2 at a time. (This is also known as a binomial coefficient.)

nCr(z,3) z=5	10
nCr(z,3) z=6	20

$nCr(Value, 0) \Rightarrow 1$

 $nCr(Value, negInteger) \Rightarrow 0$

 $nCr(Value, posInteger) \Rightarrow Value \cdot$ (Value-1)... (Value-posInteger+1)/ posInteger!

 $nCr(Value, nonInteger) \Rightarrow expression!/$ ((Value-nonInteger)! ·nonInteger!)

$$nCr(List1, List2) \Rightarrow list$$

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

 $nCr(Matrix 1, Matrix 2) \Rightarrow matrix$

Returns a matrix of combinations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

$$\frac{\text{nCr}(z,3)|z=5}{\text{nCr}(z,3)|z=6}$$
 10

$$nCr({5,4,3},{2,4,2})$$
 {10,1,3}

nCr[6	5][2	2	15	10
\[4	3][2	2	6	3

nDerivative()

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.

If the variable *Var* does not contain a numeric value, you must provide Value.

Order of the derivative must be 1 or 2.

Catalogue > 🕮

nDerivative($ x ,x=1$)	1
nDerivative $(x ,x) x=0$	undef
nDerivative $(\sqrt{x-1}, x) x=1$	undef

nDerivative()

Catalogue > 23

Note: The nDerivative() algorithm has a limitiation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of $x \cdot (x^2+x)^1(1/3)$ at x=0 is equal to 0. However, because the first derivative of the subexpression $(x^2+x)^1(1/3)$ is undefined at x=0, and this value is used to calculate the derivative of the total expression, x=0 in x

If you encounter this limitation, verify the solution graphically. You can also try using **centralDiff()**.

nDerivative $\langle x \cdot (x^2 + x)^{\frac{1}{3}}, x, 1 \rangle x = 0$ undef $\frac{1}{\sqrt{x \cdot (x^2 + x)^{\frac{1}{3}}}, x / x = 0}$ centralDiff $\langle x \cdot (x^2 + x)^{\frac{1}{3}}, x / x = 0$ 0.000033		
$\frac{1}{\text{centralDiff}(x \cdot (x^2 + x)^{\frac{1}{3}}, x) _{x=0}}$	$\left(\begin{array}{cc} \frac{1}{2} \end{array}\right)$	
	nDerivative $\langle x \cdot (x^2 + x)^3, x, 1 x =$	0
	centralDiff $\left(x \cdot \left(x^2 + x\right)^{\frac{1}{3}}, x\right) x = 0$	
		0.000033

newList() Catalogue > [2]

 $newList(numElements) \Rightarrow list$

Returns a list with a dimension of *numElements*. Each element is zero.

 $newList(4) {0,0,0,0}$

newMat() Catalogue > [3]

 $\textbf{newMat}(mumRows, numColumns) \Rightarrow matrix$

Returns a matrix of zeroes with the dimension *numRows* by *numColumns*.

newMat(2,3)	0	0	0
	0	0	0

nfMax() Catalogue > [3]

nfMax(Expr, Var)⇒value

 $nfMax(Expr, Var, lowBound) \Rightarrow value$

nfMax(*Expr***,** *Var*, *lowBound***,** *upBound***)**⇒*value*

nfMax(*Expr*, *Var***)** | *lowBound*≤*Var* ≤*upBound*⇒*value*

nfMax
$$\left(-x^2 - 2 \cdot x - 1, x\right)$$
 -1.
nfMax $\left(0.5 \cdot x^3 - x - 2, x, -5, 5\right)$ 5.

-5.

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.

nfMin() Catalogue > 💓

 $nfMin(Expr, Var) \Rightarrow value$

nfMin(*Expr***,** *Var***,** *lowBound***)**⇒*value*

nfMin(*Expr*, *Var*, *lowBound*, *upBound*)⇒*value*

nfMin(*Expr*, *Var***)** | *lowBound*≤*Var* ≤*upBound*⇒*value*

Returns a candidate numerical value of variable ${\it Var}$ where the local minimum of ${\it Expr}$ occurs.

If you supply lowBound and upBound, the function looks in the closed interval [lowBound,upBound] for the local minimum.

$$\operatorname{nfMin}(x^2+2\cdot x+5,x)$$

 $nfMin(0.5 \cdot r^3 - r - 2 \cdot r - 5.5)$

nInt() Catalogue > 🗊

nInt(*Expr1*, *Var*, *Lower*, *Upper***)**⇒*expression*

If the integrand Expr1 contains no variable other than Var, and if Lower and Upper are constants, positive ∞ , or negative ∞ , then $\operatorname{nInt}()$ returns an approximation of $\int (Expr1, Var, Lower, Upper)$. This approximation is a weighted average of some sample values of the integrand in the interval Lower < Var < Upper.

 $\frac{1.49365}{\text{nInt}(e^{-x^2}, x, -1.1)}$

nInt()

Catalogue > 23

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 vield a worthwhile improvement.

 $nInt(cos(x), x, -\pi, \pi+1.E-12)$ -1.04144e-12

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.

$$\operatorname{nInt}\left(\operatorname{nInt}\left(\frac{e^{-x \cdot y}}{\sqrt{x^2 - y^2}}, y, -x, x\right), x, 0, 1\right)$$
 3.30423

nom(5.90398,12)

nom()

Catalogue > 🕮

 $nom(effectiveRate, CpY) \Rightarrow value$

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 43.

5.75

nor

ctri = kevs

BooleanExpr1norBooleanExpr2 returns Boolean expression

BooleanList Inor Boolean List 2 returns Boolean list

BooleanMatrix InorBooleanMatrix 2 returns Boolean matrix

Returns the negation of a logical or operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

nor

*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()		Catalogue > 🗊
norm(<i>Matrix</i>) ⇒ <i>expression</i>	$norm \begin{bmatrix} 1 & 2 \end{bmatrix}$	5.47723
$norm(Vector) \Rightarrow expression$	$\frac{(3 \ 4)}{\operatorname{norm}([1 \ 2])}$	2.23607
Returns the Frobenius norm.	$ \overline{\operatorname{norm}\begin{bmatrix}1\\2\end{bmatrix}} $	2.23607

normCdf() Catalogue > 🕮

normCdf($lowBound,upBound[,\mu[,\sigma]]$) $\Rightarrow number$ if lowBound and upBound are numbers, list if lowBound and upBound are lists

Computes the normal distribution probability between lowBound and upBound for the specified µ (default=0) and σ (default=1).

For $P(X \le upBound)$, set lowBound = -9E999.

normPdf() Catalogue > 🕮

normPdf($XVal[,\mu[,\sigma]]$ **)** \Rightarrow *number* if XVal is a number, *list* if XVal is a list

Computes the probability density function for the normal distribution at a specified XVal value for the specified μ and σ .

not BooleanExpr⇒Boolean expression

Returns true, false, or a simplified form of the argument.

not *Integer1*⇒*integer*

Returns the one's complement of a real integer. Internally, *Integer 1* 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 16.

not (2≥3)	true
not 0hB0▶Base16	0hFFFFFFFFFFFF4F
not not 2	2

In Hex base mode:

Important: Zero, not the letter O.

not 0h7AC36	OhFFFFFFFFFFF853C9

In Bin base mode:

0b100101▶Base10	37
not 0b100101	
0b111111111111111111111111111111111111	11111111111
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.

Catalogue > 🕮 nPr()

 $nPr(Value1, Value2) \Rightarrow expression$

For integer Value 1 and Value 2 with $Value 1 \ge Value 2 \ge 0$, nPr() is the number of permutations of Value 1 things taken Value 2 at a time.

 $nPr(Value, 0) \Rightarrow 1$

 $nPr(Value, negInteger) \Rightarrow 1/((Value+1) \cdot$ (Value+2)... (Value-negInteger))

 $nPr(Value, posInteger) \Rightarrow Value \cdot$ (Value-1)... (Value-posInteger+1)

 $nPr(Value, nonInteger) \Rightarrow Value! /$ (Value-nonInteger)!

 $nPr(List1, List2) \Rightarrow list$

$$nPr(\{5,4,3\},\{2,4,2\})$$
 {20,24,6}

Returns a list of permutations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

 $nPr(Matrix 1, Matrix 2) \Rightarrow matrix$

Returns a matrix of permutations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

nPr∏6	5][2	2	30	20
4	3 2	2	12	6

npv()

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*.

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.

Catalogue > 🗐

list1:={ 6000,-8000,2000,-3000 }		
	{6000,-8000,2000,-3000}	
list2:={2,2,2,1}	{2,2,2,1}	
npv(10,5000,list1,li	st2) 4769.91	

nSolve()

nSolve(Equation,Var[=Guess])⇒number or error_string

nSolve(*Equation,Var*[=*Guess*],*lowBound***)** ⇒ *number or error string*

nSolve(Equation, Var

Catalogue > 🗐

$nSolve(x^2+5\cdot x-25=9,x)$	3.84429
$nSolve(x^2=4,x=-1)$	-2.
n Solve $(x^2=4,x=1)$	2.

Note: If there are multiple solutions, you can use a guess to help find a particular solution.

[=Guess], lowBound, upBound) $\Rightarrow number$ or error string

nSolve(Equation, Var[=Guess]) | lowBound $\leq Var \leq upBound \Rightarrow number or error string$

Iteratively searches for one approximate real numeric solution to Equation for its one variable. Specify the variable as:

variable – or – variable = real number

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

nSolve() 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."

$$\frac{\text{nSolve}(x^2 + 5 \cdot x - 25 = 9, x)|_{x < 0}}{\text{nSolve}\left(\frac{(1 + r)^{24} - 1}{r} = 26, r\right)|_{r > 0} \text{ and } r < 0.25}{0.006886}$$

$$\frac{0.006886}{\text{nSolve}(x^2 = -1, x)}$$
"No solution found"

0

OneVar Catalogue > 🗐

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 (page 144).

All the lists must have equal dimension except for Include.

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

Category is a list of numeric category codes for the corresponding X values.

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

An empty (void) element in any of the lists X, Freq or Category results in a void for the corresponding element of all those lists. An empty element in any of the lists XI through X20 results in a void for the corresponding element of all those lists. For more information on empty elements, see page 210.

Output variable	Description	
stat.x	Mean of x values	
stat.Σx	Sum of x values	
stat. Σx^2	Sum of x ² values	
stat.sx	Sample standard deviation of x	
stat. x	Population standard deviation of x	
stat.n	Number of data points	
stat.MinX	Minimum of x values	
stat.Q ₁ X	1st Quartile of x	
stat. MedianX	Median of x	
stat.Q ₃ X	3rd Quartile of x	
stat.MaxX	Maximum of x values	
stat.SSX	Sum of squares of deviations from the mean of x	

	or	Catalogue > 📳
--	----	---------------

BooleanExpr1orBooleanExpr2 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.

Define $g(x)$	r)=Func	Done
	If $x \le 0$ or $x \ge 5$	
	Goto end	
	Return $x \cdot 3$	
	Lbl end	
	EndFunc	
g(3)		9
g(0)	A function did not re	turn a value

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Integer1 **or** *Integer2*⇒*integer*

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 16.

of the first characters of each list element.

Note: See xor.

In Hex base mode:

0h7AC36 or 0h3D5F 0h7BD7F

Important: Zero, not the letter O.

In Bin base mode:

0b100101 or 0b100 0b100101

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()	(Catalogue > 😰
ord(String)⇒integer	ord("hello")	104
$ord(List l) \Rightarrow list$	char(104)	"h"
Old(Elist1)—rist	ord(char(24))	24
Returns the numeric code of the first character in character string <i>String</i> , or a list	$\operatorname{ord}(\{\text{"alpha","beta"}\})$	{97,98}

P▶Rx() Catalogue > [3]

 $P \to Rx(rExpr, \theta Expr) \Rightarrow expression$

 $P \rightarrow Rx(rList, \theta List) \Rightarrow list$

 $P \to Rx(rMatrix, \theta Matrix) \Rightarrow matrix$

Returns the equivalent x-coordinate of the (r, θ) pair.

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

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

In Radian angle mode:

$$\frac{P \blacktriangleright Rx(4,60^{\circ})}{P \blacktriangleright Rx\left\{\left\{-3,10,1.3\right\},\left\{\frac{\pi}{3},\frac{-\pi}{4},0\right\}\right\}} \\
\left\{-1.5,7.07107,1.3\right\}$$

P▶Ry() Catalogue > 🗐

P▶Ry(rValue, θValue)⇒value

 $P \rightarrow Ry(rList, \theta List) \Rightarrow list$

 $P Ry(rMatrix, \theta Matrix) \Rightarrow matrix$

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. or

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

In Radian angle mode:

P▶Ry(4,60°) 3.4641
P▶Ry
$$\left\{ -3,10,1.3 \right\}, \left\{ \frac{\pi}{3}, \frac{-\pi}{4}, 0 \right\}$$
 $\left\{ -2.59808, -7.07107, 0 \right\}$

PassErr

Catalogue > 📳

PassErr

Passes an error to the next level.

If system variable errCode is zero, PassErr does not do anything.

For an example of **PassErr**, See Example 2 under the **Try** command, page 155.

PassFrr

The **Else** clause of the **Try...Else...EndTry** block should use Cirerr or Passerr. If the error is to be processed or ignored, use CIrErr. 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 dialogue box will be displayed as normal.

Note: See also CirErr, page 22, and Try, page 155.

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

piecewise()	Catalogue > 🗐	
piecewise(<i>Expr1</i> [, <i>Cond1</i> [, <i>Expr2</i> [, <i>Cond2</i> [,]]])	Define $p(x) = \begin{cases} x, & x > 0 \\ \text{undef}, x \le 0 \end{cases}$	Done
Returns definitions for a piecewise function	p(1)	1
in the form of a list. You can also create piecewise definitions by using a template.	<i>p</i> (-1)	undef

Note: See also Piecewise template, page 2.

poissCdf() Catalogue > 🗐

poissCdf(λ , lowBound, upBound) \Rightarrow number if lowBound and upBound are numbers, list if lowBound and upBound are lists

poissCdf(λ **,**upBound**)**for P($0 \le X \le upBound$) $\Rightarrow number$ if upBound is a number, list if upBound is a list

Computes a cumulative probability for the discrete Poisson distribution with specified mean λ .

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

Catalogue > 🕮 poissPdf()

poissPdf(λ , XVal) \Rightarrow number if XVal is a number, list if XVal is a list

Computes a probability for the discrete Poisson distribution with the specified mean λ .

Vector Polar

[1 3.]▶Polar [3.16228 ∠71.5651]

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

Displays *vector* in polar form $[r \angle \theta]$. The vector must be of dimension 2 and can be a row or a column.

Note: Polar is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update *ans*.

Note: See also ▶Rect, page 120.

complex Value ▶Polar

Displays complex Vector in polar form.

- Degree angle mode returns ($r\angle\theta$).
- Radian angle mode returns $re^{i\theta}$.

complex Value can have any complex form. However, an $re^{i\theta}$ entry causes an error in Degree angle mode.

Note: You must use the parentheses for an $(r \angle \theta)$ polar entry.

In Radian angle mode:

(3+4· <i>i</i>)▶Polar	e ^{0.927295·i} ·5
$(4 \angle \frac{\pi}{3})$ Polar	e ^{1.0472⋅i} ⋅4.

In Gradian angle mode:

In Degree angle mode:

$$(3+4·i)$$
 ▶ Polar $(5 ∠ 53.1301)$

polyEval()

Catalogue > 😰

 $polyEval(List1, Expr1) \Rightarrow expression$

 $polyEval(List1, List2) \Rightarrow expression$

Interprets the first argument as the coefficient of a descending-degree polynomial and returns the polynomial evaluated for the value of the second argument.

$\overline{\text{polyEval}(\{1,2,3,4\},2)}$	26
polyEval({1,2,3,4},{2,-7})	{26,-262}

polyRoots()

Catalogue > 🕮

 $polyRoots(Polv,Var) \Rightarrow list$

 $polyRoots(ListOfCoeffs) \Rightarrow list$

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 expanded form in one variable. Do not use unexpanded forms such as y2·y+1 or $x \cdot x + 2 \cdot x + 1$

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

Note: See also cPolyRoots(), page 30.

$polyRoots(y^3+1,y)$	{-1}
cPolyRoots(y ³ +1,y)	
{-1,0.5-0.866025 <i>i</i> ,0.5+0).866025 -i }
$polyRoots(x^2+2\cdot x+1,x)$	{-1,-1}
$polyRoots({1,2,1})$	{-1,-1}

PowerReg

Catalogue > 🕮

PowerReg X,Y [, Freq] [, Category, Include]]

Computes the power regressiony = $(a \cdot (x)b)$ on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

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 or string 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 210.

Output variable	Description	
stat.RegEqn	Regression equation: a ·(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	

Catalogue > 🗐 **Prgm**

Prgm Block

EndPrgm

Template for creating a user-defined programme. Must be used with the Define, Define LibPub or Define LibPriv command.

Block can be a single statement, a series of statements separated with the ":" character or a series of statements on separate lines.

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

Calculate GCD and display intermediate results.

Define $proggcd(a,b)$ =	Prgm	
	Local d	
	While $b\neq 0$	
	d:=mod(a,b)	
	a := b	
	b := d	
	Disp a," ",b	
	EndWhile	
	Disp "GCD=", a	
	EndPrgm	
		D

Done proggcd(4560,450) 450 60 60 30 30 0 GCD=30 Done

Product (PI)

See Π (), page 184.

product()

$product(List[, Start[, End]]) \Rightarrow expression$

Returns the product of the elements contained in List. Start and End are optional. They specify a range of elements.

 $product(Matrix 1[, Start[, End]]) \Rightarrow matrix$

Returns a row vector containing the products of the elements in the columns of Matrix 1. Start and end are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 210.

Catalogue > 🕮

product({1,2,3,4})	24
product({4,5,8,9},2,3)	40

. /[1	2	3 6 9	[28 80 162]
product 4	5	6	
_7	8	9∬	
$\frac{\sqrt{7}}{\text{product} \begin{bmatrix} 1\\4\\7 \end{bmatrix}}$	2	3	[4 10 18]
product 4	5	6 ,1,2	
\[7	8	9]	

propFrac()

$propFrac(Value 1[, Var]) \Rightarrow value$

propFrac(rational number) returns rational number as the sum of an integer and a fraction having the same sign and a greater denominator magnitude than numerator magnitude.

propFrac(rational expression,Var) returns the sum of proper ratios and a polynomial with respect to Var. The degree of Var in the denominator exceeds the degree of Var in the numerator in each proper ratio. Similar powers of Var are collected. The terms and their factors are sorted with Var as the main variable.

Catalogue > 🗐

$\operatorname{propFrac}\left(\frac{4}{3}\right)$	$1+\frac{1}{3}$
$\operatorname{propFrac}\left(\frac{-4}{3}\right)$	$-1-\frac{1}{3}$

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.

You can use the **propFrac()** function to represent mixed fractions and demonstrate addition and subtraction of mixed fractions.

$\overline{\operatorname{propFrac}\!\left(\frac{11}{7}\right)}$	$1 + \frac{4}{7}$
$\frac{1}{\text{propFrac}\left(3+\frac{1}{11}+5+\frac{3}{4}\right)}$	$8 + \frac{37}{44}$
$ \overline{\text{propFrac}\left(3 + \frac{1}{11} - \left(5 + \frac{3}{4}\right)\right)} $	$-2-\frac{29}{44}$

Q

QR Catalogue > 🗐

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, computations are done using floatingpoint arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as: 5E-14 ·max(dim(Matrix)) ·rowNorm (Matrix)

The floating-point number (9.) in m1 causes results to be calculated in floating-point form.

_								_
		2	3		1	2	3	
	4	5	6	→ m1	4	5	6	
l	7	8	9.		7	8	9.	

 QR m1,qm,rm
 Done

 qm
 0.123091
 0.904534
 0.408248

 0.492366
 0.301511
 -0.816497

 0.86164
 -0.301511
 0.408248

 rm
 8.12404
 9.60114
 11.0782

 0.
 0.904534
 1.80907

0.

0.

0.

Catalogue > 23

OR

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.

QuadReg Catalogue > 🗐

QuadReg X,Y [, Freq] [, Category, Include]]

Computes the quadratic polynomial regressiony = a $\cdot x^2+b \cdot x+con$ lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable (page 144).

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 or string 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 210.

Output variable	Description
stat.RegEqn	Regression equation: a $\cdot x^2 + b \cdot x + c$
stat.a, stat.b, stat.c	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression

stat.XReg	List of data points in the modified 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 Catalogue > 🗐

QuartReg X,Y [, Freq] [, Category, Include]]

Computes the quartic polynomial regressiony = a $\cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + eon lists X$ and Y with frequency *Freq.* A summary of results is stored in the stat.results variable (page 144).

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 or string 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 210.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$
stat.a, stat.b, stat.c, stat.d, stat.e	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression

Output variable	Description
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▶**P**θ() Catalogue > 🗐

 $R \triangleright P\theta$ (xValue, yValue) \Rightarrow value

 $R \triangleright P\theta (xList, yList) \Rightarrow list$

 $R \triangleright P\theta (xMatrix, yMatrix) \Rightarrow matrix$

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:

R▶Pθ(2,2)	45.
-----------	-----

In Gradian angle mode:

In Radian angle mode:

R►Pr() Catalogue > 🗐

 $R \triangleright Pr(xValue, yValue) \Rightarrow value$

 $R \triangleright Pr(xList, yList) \Rightarrow list$

 $R \triangleright Pr(xMatrix, yMatrix) \Rightarrow matrix$

Returns the equivalent r-coordinate of the (x,y) pair arguments.

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

In Radian angle mode:

R Pr(3,2) 3.60555

R Pr(3 -4 2],
$$\begin{bmatrix} 0 & \frac{\pi}{4} & 1.5 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4.07638 & \frac{5}{2} \end{bmatrix}$$

▶ Rad Catalogue > 🗐

 $Value1 \triangleright Rad \Rightarrow value$ In Degree angle mode:

▶ Rad

Catalogue > 🗐

Converts the argument to radian angle measure.

(1.5)▶Rad (0.02618)

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

In Gradian angle mode:

(1.5)▶Rad (0.023562)¹

rand()

Catalogue > 📳

 $rand() \Rightarrow expression$ $rand(\#Trials) \Rightarrow list$

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()

Catalogue > 🗐

randBin(n, p) \Rightarrow expression randBin(n, p, #Trials) \Rightarrow 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.

randInt(3,10,4)

46.

{43.,39.,41.}

Catalogue > 🗐

randInt()

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(3,10) 3.

{9.,3.,4.,7.}

randBin(80,0.5)

randBin(80,0.5,3)

randint

(lowBound,upBound ,#Trials) returns a list containing #Trials random integers within the specified range.

randMat()

 $randMat(numRows, numColumns) \Rightarrow$ matrix

Returns a matrix of integers between -9 and 9 of the specified dimension.

Both arguments must simplify to integers.

Catalogue > 🗐

RandSeed 1147	Done		
randMat(3,3)	8	-3	6
	-2	3	-6
	[0	4	-6]

Note: The values in this matrix will change each time you press enter.

randNorm()

 $randNorm(\mu, \sigma) \Rightarrow expression$ $randNorm(\mu, \sigma, \#Trials) \Rightarrow list$

randNorm(μ, σ) returns a decimal number from the specified normal distribution. It could be any real number but will be heavily concentrated in the interval $[\mu-3\bullet\sigma, \mu+3\bullet\sigma]$.

 $randNorm(\mu, \sigma, \#Trials)$ returns a list containing #Trials decimal numbers from the specified normal distribution.

Catalogue > 🔯

RandSeed 1147	Done
randNorm(0,1)	0.492541
randNorm(3,4.5)	-3.54356

randPoly()

 $randPoly(Var, Order) \Rightarrow expression$

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.

Catalogue > 23

RandSeed 1147 Done randPoly
$$(x,5)$$
 $-2 \cdot x^5 + 3 \cdot x^4 - 6 \cdot x^3 + 4 \cdot x - 6$

randSamp()

Catalogue > 🕮

 $randSamp(List, \#Trials[, noRepl]) \Rightarrow list$

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 $list3 = \{1,2,3\}$	3,4,5}	Done
Define list4=randS	amp(<i>list3</i> ,6)	Done
list4	{1.,3.,3.,1.	,3.,1.}

RandSeed Catalogue > 2

RandSeed Number

If *Number* = 0, sets the seeds to the factory defaults for the random-number generator. If $Number \neq 0$, it is used to generate two seeds, which are stored in system variables seed1 and seed2.

RandSeed 1147	Done
rand()	0.158206

real() Catalogue > 🕮 $real(Value1) \Rightarrow value$ $real(2+3\cdot i)$ Returns the real part of the argument. $real(List1) \Rightarrow list$ $real(\{1+3\cdot i, 3, i\})$ {1,3,0} Returns the real parts of all elements. $real(Matrix 1) \Rightarrow matrix$ $_{\text{real}}|_{1+3\cdot i}$ 3 1 3 2 0

▶ Rect

 $\left(3 \angle \frac{\pi}{4} \angle \frac{\pi}{6} \right)$ Rect

Vector ▶ Rect

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

Returns the real parts of all elements.

Displays Vector in rectangular form [x, y, zl. 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 110.

1.06066 1.06066 2.59808

complex Value ► 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 $(r \angle \theta)$ polar entry.

In Radian angle mode:

$\left(\frac{\pi}{4 \cdot e^{3}}\right)$ Rect	11.3986
$\frac{\left(4 \angle \frac{\pi}{3}\right) \triangleright \text{Rect}}{\left(4 \angle \frac{\pi}{3}\right) \triangleright \text{Rect}}$	2.+3.4641· <i>i</i>

In Gradian angle mode:

In Degree angle mode:

$$((4 ∠ 60))$$
 Rect 2.+3.4641·*i*

Note: To type \angle , select it from the symbol list in the Catalogue.

ref()

 $ref(Matrix 1[, Tol]) \Rightarrow matrix$

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 ctri enter or set the Auto or Approximate mode to Approximate, computations are done using floatingpoint arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as: $5E-14 \cdot max(dim(Matrix I)) \cdot rowNorm$ (Matrix 1)

Avoid undefined elements in *Matrix 1*. 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:

Catalogue > 🗐

$$\operatorname{ref} \begin{bmatrix}
-2 & -2 & 0 & -6 \\
1 & -1 & 9 & -9 \\
-5 & 2 & 4 & -4
\end{bmatrix} \qquad
\begin{bmatrix}
1 & \frac{-2}{5} & \frac{-4}{5} & \frac{4}{5} \\
0 & 1 & \frac{4}{7} & \frac{11}{7} \\
0 & 0 & 1 & \frac{-62}{71}
\end{bmatrix}$$

$ ref \begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} $	$\left[1 \frac{1}{a}\right]$	0
∐0 0 1∬	0 1	0
	0 0	1

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.

	a	1	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} a=0$	o	1	0
ref	0	1	0 a=0	0	0	1
\	0	0	1∬	0	0	0]

Note: See also rref(), page 130.

RefreshProbeVars

Catalogue > 📳

RefreshProbeVars

Allows you to access sensor data from all connected sensor probes in your TI-Basic program.

StatusVar Value	Status
statusVar	Normal (continue with the
=0	program)
	The Vernier DataQuest™ application is in data collection mode.
statusVar =1	Note: The Vernier DataQuest™ application must be in meter mode for this command to work.

statusVar The Vernier DataQuest™
=2 application is not launched.
statusVar The Vernier DataQuest™
=3 application is launched, but you

Example

Define temp()=

Prqm

© Check if system is ready

RefreshProbeVars status

If status=0 Then

Disp "ready"

For n, 1, 50

RefreshProbeVars status

temperature:=meter.temperature

Disp "Temperature:

If temperature>30 Then

Disp "Too hot"

", temperature

EndIf

RefreshProbeVars

Catalogue > 23

StatusVar Value

Status have not connected any probes.

© Wait for 1 second between samples

Wait 1

EndFor

Else

Disp "Not ready. Try again

later" EndIf

EndPrqm

Note: This can also be used with TI-

InnovatorTM Hub.

remain()

Catalogue > 🕮

 $remain(Value1, Value2) \Rightarrow value$ $remain(List1, List2) \Rightarrow list$ remain(Matrix 1. Matrix 2) $\Rightarrow matrix$

Returns the remainder of the first argument with respect to the second argument as defined by the identities:

remain(x,0) xremain(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 93.

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}

emain 9	-7],[4	3	1	-1
\[6	$4 \rfloor \lfloor 4$	-3]]	[2	1]

Request

Catalogue > 🗐

Request promptString, var[, DispFlag [, status Var]]

Request promptString, func(arg1, ...argn) [, DispFlag [, statusVar]]

Define a program:

Define request demo()=Prgm Request "Radius: ",r Disp "Area = ",pi*r2 EndPrgm

Run the program and type a response:

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.

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 status Var is set to a value of 1.
- Otherwise, variable status Var 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.

request_demo()



Result after selecting OK:

Radius: 6/2 Area= 28.2743

Define a program:

Define polynomial()=Prgm
 Request "Enter a polynomial in
x:",p(x)
 Disp "Real roots are:",polyRoots
(p(x),x)
EndPrgm

Run the program and type a response:

polynomial()



Result after entering x^3+3x+1 and selecting **OK**:

Real roots are: {-0.322185}

Request

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 file 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 125.

RequestStr

RequestStr promptString, var[, DispFlag]

Programming command: Operates identically to the first syntax of the Request command, except that the user's response is always interpreted as a string. By contrast, the **Request** command interprets the response as an expression unless the user encloses it in quotation marks ("").

Note: You can use the RequestStr command within a user-defined program but not within a function.

To stop a program that contains a **RequestStr** command inside an infinite loop:

- Handheld: Hold down the Gion 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 123.

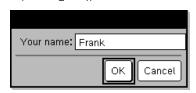
Catalogue > 🗐

Define a program:

Define requestStr_demo()=Prgm RequestStr "Your name:", name, 0 Disp "Response has ",dim(name)," characters." EndPrgm

Run the program and type a response:

requestStr demo()



Result after selecting **OK** (Note that the DispFlag argument of **0** omits the prompt and response from the history):

requestStr demo()

Response has 5 characters.

Return

Catalogue > 😰

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Define **factorial** (nn)= Func

Local answer, counter $1 \rightarrow answer$

For counter,1,nn

answer· counter → answer

EndFor

Return answer

factorial (3)

right() Catalogue > 🗐

 $right(List 1[, Num]) \Rightarrow list$

Returns the rightmost *Num* elements contained in *List1*.

If you omit *Num*, returns all of *List1*.

 $right(sourceString[, Num]) \Rightarrow string$

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

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

 $right(Comparison) \Rightarrow expression$

Returns the right side of an equation or inequality.

right({1,3,-2,4},3) {3,-2,4}

right("Hello",2) "lo"

rk23 () Catalogue > 🗐

rk23(Expr, Var, depVar, {Var0, VarMax}, depVar0, $VarStep[, diftol]) <math>\Rightarrow matrix$

rk23(SystemOfExpr, Var, ListOfDepVars, {Var0, VarMax}, ListOfDepVars0, $VarStep[, diftol]) \Rightarrow matrix$

rk23(ListOfExpr, Var, ListOfDepVars, {Var0, VarMax}, ListOfDepVars0, VarStep[, diftol]) $\Rightarrow matrix$

Differential equation:

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

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 \triangle and then use \triangleleft and \triangleright to move the cursor.

rk23 () Catalogue > 🕮

Uses the Runge-Kutta method to solve the system

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

with $depVar(Var\theta)=depVar\theta$ 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*).

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*VarStepis 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).

Same equation with diftol set to 1.E-6

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

rk23
$$\begin{bmatrix} yI+0.1 \cdot yI \cdot y2 \\ 3 \cdot y2-yI \cdot y2 \end{bmatrix}$$
, $t_1\{yI,y2\}$, $\{0,5\}$, $\{2,5\}$, 1
 $\begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 2. & 1.94103 & 4.78694 & 3.25253 & 1.82848 \\ 5. & 16.8311 & 12.3133 & 3.51112 & 6.27245 \end{bmatrix}$

root() root(Value) \Rightarrow root root(Value1, Value2) \Rightarrow root

 $\frac{3\sqrt{8}}{3\sqrt{3}}$ 2 1.44225

Catalogue > 🕮

root(*Value*) returns the square root of *Value*.

root(Value1, Value2) returns the Value2 root of Value1. Value1 can be a real or complex floating point constant or an integer or complex rational constant.

Note: See also Nth root template, page 2.

"-t-t-/\	Catalagua > [30]
rotate()	Catalogue > 👰

 $rotate(Integer1[,\#ofRotations]) \Rightarrow integer$

Rotates the bits in a binary integer. You can enter *Integer I* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *Integer I* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see **Base2**, page 16.

If #ofRotations is positive, the rotation is to the left. If #ofRotations is negative, the rotation is to the right. The default is -1 (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

0b0000000000001111010110000110101

Rightmost bit rotates to leftmost.

produces:

0b10000000000000111101011000011010

The result is displayed according to the Base mode.

 $rotate(List1[,\#ofRotations]) \Rightarrow list$

Returns a copy of *List1* rotated right or left by #of *Rotations* elements. Does not alter *List1*.

To see the entire result, press \triangle and then use \triangleleft and \triangleright to move the cursor.

In Hex base mode:

In Bin base mode:

rotate(0h78E)	0h3C7
rotate(0h78E,-2)	0h80000000000001E3
rotate(0h78E,2)	0h1E38

Important: To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).

In Dec base mode:

rotate({1,2,3,4})	{4,1,2,3}
rotate({1,2,3,4},-2)	{3,4,1,2}
rotate({1,2,3,4},1)	{2,3,4,1}

rotate()

Catalogue > 🕮

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).

 $rotate(String1[,\#ofRotations]) \Rightarrow string$

Returns a copy of *String1* rotated right or left by #ofRotations characters. Does not alter *String1*.

If #ofRotations is positive, the rotation is to the left. If #ofRotations is negative, the rotation is to the right. The default is -1 (rotate right one character).

rotate("abcd")	"dabc"
rotate("abcd",-2)	"cdab"
rotate("abcd" 1)	"bcda"

round()

Catalogue > 🕮

1.235

round(1.234567,3)

 $round(Value1[, digits]) \Rightarrow value$

Returns the argument rounded to the specified number of digits after the decimal point.

digits must be an integer in the range 0-12. If *digits* is not included, returns the argument rounded to 12 significant digits.

Note: Display digits mode may affect how this is displayed.

 $round(List1[, digits]) \Rightarrow list$

Returns a list of the elements rounded to the specified number of digits.

 $round(Matrix 1[, digits]) \Rightarrow matrix$

Returns a matrix of the elements rounded to the specified number of digits.

11 E. (a) A
round $(\{\pi,\sqrt{2},\ln(2)\},4)$
{3.1416,1.4142,0.6931}
(3.1410,1.4142,0.0931)

round
$$\begin{bmatrix} \ln(5) & \ln(3) \\ \pi & e^1 \end{bmatrix}$$
, 1 $\begin{bmatrix} 1.6 & 1.1 \\ 3.1 & 2.7 \end{bmatrix}$

rowAdd()

Catalogue > 🕮

 $rowAdd(Matrix1, rIndex1, rIndex2) \Rightarrow$ matrix

Returns a copy of Matrix 1 with row rIndex2 replaced by the sum of rows rIndex 1 and rIndex 2.

$$rowAdd \begin{bmatrix} 3 & 4 \\ -3 & -2 \end{bmatrix}, 1, 2$$

$$\begin{bmatrix} 3 & 4 \\ 0 & 2 \end{bmatrix}$$

rowDim()

 $rowDim(Matrix) \Rightarrow expression$

Returns the number of rows in *Matrix*.

Note: See also colDim(), page 23.

1	2	1	2
3	$4 \rightarrow m1$	3	4
5	6	5	6

rowDim(m1)

rowNorm()

 $rowNorm(Matrix) \Rightarrow expression$

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

Note: All matrix elements must simplify to numbers. See also **colNorm()**, page 23.

Catalogue > 23

3

Catalogue > 🕮

rowNorm $\begin{bmatrix} -5 & 6 & -7 \\ 3 & 4 & 9 \\ 9 & -9 & -7 \end{bmatrix}$ 25

rowSwap()

rowSwap(Matrix1, rIndex1, rIndex2) \Rightarrow matrix

Returns *Matrix1* with rows *rIndex1* and *rIndex2* exchanged.

Catalogue > 🗊

 $\begin{vmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{vmatrix} \rightarrow mat$ $\begin{vmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{vmatrix}$ rowSwap(mat,1,3) $\begin{vmatrix}
5 & 6 \\
3 & 4
\end{vmatrix}$

rref()

 $rref(Matrix 1[, Tol]) \Rightarrow matrix$

Returns the reduced row echelon form of *Matrix 1*.

Catalogue > 🗐

1 2

 $\begin{bmatrix} 1 & 0 & 0 & \frac{66}{71} \\ 0 & 1 & 0 & \frac{147}{71} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$

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.

 If you use ctrl enter or set the Auto or Approximate mode to Approximate, computations are done using floatingpoint arithmetic. rref()

Catalogue > 23

If *Tol* is omitted or not used, the default tolerance is calculated as: $5E-14 \cdot max(dim(Matrix I)) \cdot rowNorm$ (Matrix 1)

Note: See also ref(), page 121.

S

trig kev sec()

 $sec(Value1) \Rightarrow value$ $sec(List1) \Rightarrow list$

Returns the secant of Value 1 or returns a list containing the secants of all elements in List 1.

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

In Degree angle mode:

sec(45)	1.41421
sec({1,2.3,4})	{1.00015,1.00081,1.00244}

trig key sec-1()

 $sec-1(Value1) \Rightarrow value$ $sec-1(List1) \Rightarrow list$

Returns the angle whose secant is *Value1* 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:

In Gradian angle mode:

$$\sec^{-1}(\sqrt{2})$$
 50.

In Radian angle mode:

$$sec^{-1}(\{1,2,5\})$$
 $\{0,1.0472,1.36944\}$

$$sech(Value I) \Rightarrow value$$

 $sech(List I) \Rightarrow list$

Returns the hyperbolic secant of Value 1 or returns a list containing the hyperbolic secants of the List1 elements.

sech(3)	0.099328
sech({1,2.3,4})	
{0.648054,0.19	98522,0.036619}

sech-1()

 $sech-1(Value1) \Rightarrow value$ $sech-1(List1) \Rightarrow list$

Returns the inverse hyperbolic secant of *Value 1* 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 (...).

Catalogue > 🔯

In Radian angle and Rectangular complex mode:

$$\frac{\text{sech}^{-1}(1)}{\text{sech}^{-1}(\left\{1,^{-2},2.1\right\})} \\ \left\{0,2.0944 \cdot i,8.\epsilon^{-1}5+1.07448 \cdot i\right\}$$

Send

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 programme but not within a function.

Note: See also Get (page 58), GetStr (page 64), and eval() (page 47).

Hub Menu

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.

Send Hub Menu

n:=50		50
m:=4		4
Send "SET SOUND eval(m·n)"	Done
iostr.SendAns	"SET SC	OUND 200"

seq()

 $seq(Expr, Var, Low, High[, Step]) \Rightarrow list$

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.

Catalogue > 😰

$seq(n^2, n, 1, 6)$	{1,4,9,16,25,36}
$\operatorname{seq}\left(\frac{1}{n}, n, 1, 10, 2\right)$	$\left\{1, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}\right\}$
$\operatorname{sum}\left\{\operatorname{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right\}$	1968329 1270080

Note: To force an approximate result,

Handheld: Press ctrl enter.

Windows®: Press Ctrl+Enter.

Macintosh®: Press #Enter.

iPad®: Hold enter, and select = .

$$sum \left| seq \left(\frac{1}{n^2}, n, 1, 10, 1 \right) \right|$$
 1.54977

seqGen()

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

Catalogue > 😰

Generate the first 5 terms of the sequence u $(n) = u(n-1)^2/2$, with u(1)=2 and VarStep=1.

$$\frac{\left\{\frac{(u(n-1))^{2}}{n}, n, u, \{1,5\}, \{2\}\right\}}{\left\{2, 2, \frac{4}{3}, \frac{4}{9}, \frac{16}{405}\right\}}$$

Example in which Var0=2:

seqGen
$$\left(\frac{u(n-1)+1}{n}, n, u, \{2,5\}, \{3\}\right)$$
 $\left\{3, \frac{4}{3}, \frac{7}{12}, \frac{19}{60}\right\}$

Generates a matrix of terms for a system (or list) of sequences ListOfDepVars (Var)=ListOrSystemOfExpr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates ListOfDepVars(Var) for corresponding values of Var using ListOrSystemOfExpr formula and MatrixOfInitTerms, and returns the results as a matrix.

The original contents of *Var* are unchanged after **seqGen()** is completed.

The default value for VarStep = 1.

System of two sequences:

$$\frac{1}{\operatorname{seqGen}\left\{\frac{1}{n}, \frac{u2(n-1)}{2} + u1(n-1)\right\}, n, \{u1,u2\}, \{1,5\} \begin{bmatrix} 1 \\ 2 \end{bmatrix}}{\left[\frac{1}{2}\right]} \\
\left[\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right] \\
\left[\frac{1}{2}, \frac{2}{3}, \frac{13}{12}, \frac{19}{24}\right]$$

Note: The Void (_) in the initial term matrix above is used to indicate that the initial term for u1(n) is calculated using the explicit sequence formula u1(n)=1/n.

seqn()

seqn($Expr(u, n[, ListOfInitTerms[, nMax[, CeilingValue]]]) <math>\Rightarrow list$

Generates a list of terms for a sequence u (n)=Expr(u,n) as follows: Increments n from 1 through nMax by 1, evaluates u(n) for corresponding values of n using the Expr(u,n) formula and ListOfInitTerms, and returns the results as a list.

 $seqn(Expr(n[, nMax[, CeilingValue]]) \Rightarrow list$

Generates a list of terms for a non-recursive sequence u(n)=Expr(n) as follows: Increments n from 1 through nMax by 1, evaluates u(n) for corresponding values of n using the Expr(n) formula, and returns the results as a list.

If nMax is missing, nMax is set to 2500

If nMax=0, nMax is set to 2500

Note: seqn() calls seqGen() with $n\theta$ =1 and nstep =1

Catalogue > 🗐

Generate the first 6 terms of the sequence u(n) = u(n-1)/2, with u(1)=2.

$$seqn\left(\frac{u(n-1)}{n}, \{2\}, 6\right) \\
\left\{2, 1, \frac{1}{3}, \frac{1}{12}, \frac{1}{60}, \frac{1}{360}\right\}$$

$$\overline{\text{seqn}\left(\frac{1}{n^2}, 6\right)} \qquad \left\{1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right\}$$

Catalogue > 23 setMode()

setMode(modeNameInteger, settingInteger) $\Rightarrow integer$ $setMode(list) \Rightarrow integer\ list$

Valid only within a function or program.

setMode(modeNameInteger, settingInteger) temporarily sets mode modeNameInteger to the new setting settingInteger, and returns an integer corresponding to the original setting of that mode. The change is limited to the duration of the program/function's execution.

modeNameInteger specifies which mode you want to set. It must be one of the mode integers from the table below.

settingInteger specifies the new setting for the mode. It must be one of the setting integers listed below for the specific mode vou 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 $getMode(0) \rightarrow var$, you can use setMode(var) to restore those settings until the function or program exits. See getMode(), page 63.

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Display approximate value of π using the default setting for Display Digits, and then display π with a setting of Fix 2. Check to see that the default is restored after the program executes.

Done
1,16)
3.14159
3.14
Done

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
Vector Format	6	1=Rectangular, 2=Cylindrical, 3=Spherical
Base	7	1=Decimal, 2=Hex, 3=Binary

shift() Catalogue > 🗐

 $shift(Integer1[,\#ofShifts]) \Rightarrow integer$

Shifts the bits in a binary integer. You can enter *Integer I* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *Integer I* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see **Base2**, page 16.

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

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).

Inserts 0 if leftmost bit is 0. or 1 if leftmost bit is 1.

produces:

0b00000000000000111101011000011010

The result is displayed according to the Base mode. Leading zeros are not shown.

 $shift(List1[,\#ofShifts]) \Rightarrow list$

Returns a copy of *List1* shifted right or left by #ofShifts elements. Does not alter List 1.

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".

 $shift(String1[,\#ofShifts]) \Rightarrow string$

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. 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

shift("abcd")	" abc"
shift("abcd",-2)	" ab"
shift("abcd",1)	"bcd "

sign() Catalogue > 23

 $sign(Value 1) \Rightarrow value$ $sign(List1) \Rightarrow list$ $sign(Matrix 1) \Rightarrow matrix$

For real and complex *Value1*, returns Value1 / abs(Value1) when $Value1 \neq 0$.

Returns 1 if *Value I* is positive. Returns −1 if Value 1 is negative. sign(0) returns ± 1 if the complex format mode is Real: otherwise, it returns itself.

sign(-3.2) $sign({2,3,4,-5})$ $\{1,1,1,-1\}$

If complex format mode is Real:

 $sign([-3 \ 0 \ 3])$ -1 undef 1 **sign(0)** represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

simult()

Catalogue > 🔯

simult(coeffMatrix, constVector[, Tol]) \Rightarrow matrix

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also **linSolve()**, page 81.

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)

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 for x and y:

$$x + 2y = 1$$

$$3x + 4y = -1$$

$$\begin{array}{c|c}
simult \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix} & \begin{bmatrix} -3 \\ 2 \end{bmatrix}$$

The solution is x=-3 and y=2.

Solve:

ax + by = 1

cx + dy = 2

$ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow matx1 $	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
$\overline{\operatorname{simult}\left(\operatorname{matx} 1, \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)}$	$\begin{bmatrix} 0 \\ \frac{1}{2} \end{bmatrix}$

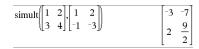
Solve:

$$x + 2y = 1$$

$$3x + 4y = -1$$

$$x + 2v = 2$$

$$3x + 4y = -3$$



For the first system, x=-3 and y=2. For the second system, x=-7 and y=9/2.

sin()



$$sin(Value 1) \Rightarrow value$$

 $sin(List 1) \Rightarrow list$

sin(Value 1) returns the sine of the argument.

sin(List1) returns a list of the sines of all elements in List 1.

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

$sin(squareMatrix1) \Rightarrow squareMatrix$

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

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

In Degree angle mode:

$\frac{1}{\sin\left(\!\left(\frac{\pi}{4}\right)^{\!r}\right)}$	0.707107
sin(45)	0.707107
sin({0,60,90})	{0.,0.866025,1.}

In Gradian angle mode:

sin(50) 0.70

In Radian angle mode:

$\frac{1}{\sin\left(\frac{\pi}{4}\right)}$	0.707107
sin(45°)	0.707107

In Radian angle mode:

$$\sin\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.9424 & -0.04542 & -0.031999 \\ -0.045492 & 0.949254 & -0.020274 \\ -0.048739 & -0.00523 & 0.961051 \end{bmatrix}$$

sin-1()



$sin-1(Value 1) \Rightarrow value$ $sin-1(List1) \Rightarrow list$

sin-1(Value 1) returns the angle whose sine is Value 1.

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:



In Gradian angle mode:

In Radian angle mode:

$\sin^{-1}(\{0,0.2,0.5\})$ {0.,0.	.201358,0.523599}
-----------------------------------	-------------------

sin-1()



 $sin-1(squareMatrix 1) \Rightarrow squareMatrix$

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

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

In Radian angle mode and Rectangular complex format mode:

$$\sin^{4}\begin{pmatrix} 1 & 5 \\ 4 & 2 \end{pmatrix}
\begin{bmatrix}
-0.174533 - 0.12198 \cdot \mathbf{i} & 1.74533 - 2.35591 \cdot \mathbf{i} \\
1.39626 - 1.88473 \cdot \mathbf{i} & 0.174533 - 0.593162 \cdot \mathbf{i}
\end{bmatrix}$$

sinh() Catalogue > 23

 $sinh(Numver1) \Rightarrow value$ $sinh(List1) \Rightarrow list$

sinh (*Value1*) returns the hyperbolic sine of the argument.

sinh (*List1*) returns a list of the hyperbolic sines of each element of *List1*.

 $sinh(squareMatrix1) \Rightarrow squareMatrix$

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

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

 $\begin{array}{ccc} \sinh(1.2) & 1.50946 \\ \sinh(\left\{0,1.2,3.\right\}) & \left\{0,1.50946,10.0179\right\} \end{array}$

In Radian angle mode:

$$\sinh \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 360.954 & 305.708 & 239.604 \\ 352.912 & 233.495 & 193.564 \\ 298.632 & 154.599 & 140.251 \end{bmatrix}$$

sinh-1() Catalogue > 23

 $sinh-1(Value I) \Rightarrow value$ $sinh-1(List I) \Rightarrow list$

sinh-1(*Value1*) returns the inverse hyperbolic sine of the argument.

sinh-1(List 1) returns a list of the inverse hyperbolic sines of each element of List 1.

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

sinh-1()

Catalogue > 23

 $sinh-1(squareMatrix 1) \Rightarrow squareMatrix$

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

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

In Radian angle mode:

SinReg

Catalogue > 🕮

SinReg X, Y[, [Iterations],[Period][, Category, Include11

Computes the sinusoidal regression on lists X and Y. A summary of results is stored in the stat.results variable. (See page 144.)

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 numeric or string 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 210.

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 <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified 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

SOFTA
SortA <i>List1</i> [, <i>List2</i>] [, <i>List3</i>]
SortA Vector1[. Vector2] [. Vector3]

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 210.

	Catalogue > 🕎
$\{2,1,4,3\} \rightarrow list1$	{2,1,4,3}
SortA list1	Done
list1	{1,2,3,4}
$\{4,3,2,1\} \rightarrow list2$	{4,3,2,1}
SortA list2,list1	Done
list2	{1,2,3,4}
list1	{4,3,2,1}

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 210.

$\{2,1,4,3\} \rightarrow list1$	{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

SortD

Catalogue > 🕮

Catalogue > 🕮

Vector ▶ Sphere

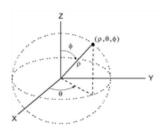
Note: You can insert this operator from the computer keyboard by typing @>Sphere.

Displays the row or column vector in spherical form $[\rho \angle \theta \angle \phi]$.

Vector must be of dimension 3 and can be either a row or a column vector.

Note: ► **Sphere** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

$$\left[2 \ \angle \frac{\pi}{4} \ 3\right] \triangleright \text{Sphere} \\
\left[3.60555 \ \angle 0.785398 \ \angle 0.588003\right]$$



sart()

Catalogue > 23

sqrt(Value	:1) ⇒	value
sqrt(List1)	$\Rightarrow l$	ist

$\sqrt{4}$	2
$\sqrt{\left\{9,2,4\right\}}$	{3,1.41421,2}

Returns the square root of the argument.

For a list, returns the square roots of all the elements in List1.

Note: See also Square root template, page 1.

stat.results

Displays results from a statistics calculation.

The results are displayed as a set of namevalue pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

Note: Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

$xlist:=\{1,2,3,4,5\}$	{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}"

stat.a	stat.dfDenom	stat.MedianY	stat.Q3X	stat.SSBlock
stat.AdjR ²	stat.dfBlock	stat.MEPred	stat.Q3Y	stat.SSCol
stat.b	stat.dfCol	stat.MinX	stat.r	stat.SSX
stat.b0	stat.dfError	stat.MinY	stat.r ²	stat.SSY
stat.b1	stat.dfInteract	stat.MS	stat.RegEqn	stat.SSError
stat.b2	stat.dfReg	stat.MSBlock	stat.Resid	stat.SSInteract
stat.b3	stat.dfNumer	stat.MSCol	stat. Resid Trans	stat.SSReg
stat.b4	stat.dfRow	stat.MSError	stat.σx	stat.SSRow
stat.b5	stat.DW	stat.MSInteract	stat.σy	stat.tList
stat.b6	stat.e	stat.MSReg	stat.σx1	stat.UpperPred
stat.b7	stat.ExpMatrix	stat.MSRow	stat.σx2	stat.UpperVal
stat.b8	stat.F	stat.n	stat. Σ x	$stat.\overline{X}$
stat.b9	stat.FBlock	Stat. $\hat{\pmb{p}}$	$stat.\Sigma x^2$	stat.X1
stat.b10	stat.Fcol	stat. \hat{p} 1	stat. Σ xy	stat. \overline{x} 2
stat.bList	stat.FInteract	stat. \hat{p} 2	stat. Σ y	stat.X Diff
$stat.\chi^2$	stat.FreqReg	stat. $\hat{\pmb{p}}$ Diff	$stat.\Sigmay^2$	stat.XList
stat.c	stat.Frow	stat.PList	stat.s	stat.XReg
stat.CLower	stat.Leverage	stat.PVal	stat.SE	stat.XVal
stat.CLowerList	stat.LowerPred	stat.PValBlock	stat.SEList	stat.XValList
stat.CompList	stat.LowerVal	stat.PValCol	stat.SEPred	stat. y
stat.CompMatrix	stat.m	stat.PValInteract	stat.sResid	stat. ŷ
stat.CookDist	stat.MaxX	stat.PValRow	stat.SEslope	stat. ŷ List
stat.CUpper	stat.MaxY	stat.Q1X	stat.sp	stat.YReg
				Juliance

stat.CUpperList stat.ME stat.O1Y stat.SS

stat.d stat.MedianX

Note: Each time the Lists & Spreadsheet application calculates statistical results, it copies the "stat." group variables to a "stat#." group, where # is a number that is incremented automatically. This lets you maintain previous results while performing multiple calculations.

stat.values Catalogue > 🕮

stat.values

See the stat.results example.

Displays a matrix of the values calculated for the most recently evaluated statistics function or command

Unlike stat.results, stat.values omits the names associated with the values.

You can copy a value and paste it into other locations.

Catalogue > [3] stDevPop()

 $stDevPop(List [, freqList]) \Rightarrow expression$

Returns the population standard deviation of the elements in List.

Each *freaList* 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 210.

 $stDevPop(Matrix 1[, freqMatrix]) \Rightarrow$ matrix

Returns a row vector of the population standard deviations of the columns in Matrix 1.

Each freaMatrix element counts the number of consecutive occurrences of the corresponding element in *Matrix 1*.

In Radian angle and auto modes:

$$stDevPop(\{1,2,5,-6,3,-2\}) \\ stDevPop(\{1.3,2.5,-6.4\},\{3,2,5\}) \\ 4.11107$$



Note: *Matrix I* must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 210.

stDevSamp()

Catalogue > 🗐

 $stDevSamp(List[, freqList]) \Rightarrow expression$

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 210.

stDevSamp(Matrix I[, freqMatrix]**)** \Rightarrow matrix

Returns a row vector of the sample standard deviations of the columns in *Matrix I*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

Note: *Matrix I* must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 210.

$$stDevSamp \begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{bmatrix}$$

$$[4. \quad 3.60555 \quad 2.]$$

$$stDevSamp \begin{bmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{bmatrix}$$

$$[2.7005 \quad 5.44695]$$

Stop Stop

Catalogue > 🕮

Programming command: Terminates the program.

Stop is not allowed in functions.

i:=0	0
Define prog1()=Prgm	Done
For <i>i</i> ,1,10,1	
If <i>i</i> =5	
Stop	
EndFor	
EndPrgm	
prog1()	Done
i	5

Catalogue > 🕄

Stop

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

Store

See \rightarrow (store), page 192.

string()		Catalogue > 📳
$string(Expr) \Rightarrow string$	string(1.2345)	"1.2345"
Simplifies Expr and returns the result as a	string(1+2)	"3"
character string		

subMat()		Catalogue > 📳
subMat($Matrix 1[$, $startRow][$, $startCol][$, $endRow][$, $endCol]] \Rightarrow matrix$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
Returns the specified submatrix of $\mathit{Matrix} 1$.	[7 8 9]	[7 8 9]
Defaults: startRow=1, startCol=1,	subMat(m1,2,1,3,2)	$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$
endRow=last row, endCol=last column.	$\operatorname{subMat}(m1,2,2)$	5 6 8 9
		[0 3]

Sum (Sigma)

see page 210.

See Σ (), page 185.

sum()	Catalogue > 🗐
$sum(List[, Start[, End]]) \Rightarrow expression$	$sum(\{1,2,3,4,5\})$ 15
Returns the sum of all elements in $List$.	$\operatorname{sum}(\{a,2\cdot a,3\cdot a\})$
Characteristic Anna antiqual Theoremsifica	"Error: Variable is not defined"
Start and End are optional. They specify a range of elements.	$\operatorname{sum}(\operatorname{seq}(n,n,1,10)) $ 55
Any void argument produces a void result.	$sum(\{1,3,5,7,9\},3)$ 21
Empty (void) elements in <i>List</i> are ignored. For more information on empty elements,	

sum()

 $sum(Matrix 1[, Start[, End]]) \Rightarrow matrix$

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 Matrix 1 are ignored. For more information on empty elements, see page 210.

sum	1	2	3 6	[5 7 9]
1	$\lfloor 4$	5	6∬	
=	1	2	3 6 9	[12 15 18]
sum	4	5	6	
)	7	8	9∬	
= I	1	2	3	[11 13 15]
sum	4	5	$\begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}, 2, 3 $	
	7	8	9]	

sumIf()

 $sumlf(List,Criteria[,SumList]) \Rightarrow value$

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 place holder 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.

Catalogue > 🕮

Catalogue > 🕮

$$\begin{aligned} & sumIf(&\{1,2,\pmb{e},3,\pi,4,5,6\},2.5<4.5)\\ & 12.859874482\\ & sumIf(&\{1,2,3,4\},2<?<5,&\{10,20,30,40\})\\ & & 70 \end{aligned}</math$$

sumIf()

Catalogue > [3]

Empty (void) elements are ignored. For more information on empty elements, see page 210.

Note: See also countif(), page 29.

sumSeq()

See Σ (), page 185.

Catalogue > 🗐

system()

system(Value1[, Value2[, Value3[, ...]]])

Returns a system of equations, formatted as a list. You can also create a system by using a template.

T

T (transpose)

Catalogue > [3]

trig key

 $Matrix l \mathbf{T} \Rightarrow matrix$

Returns the complex conjugate transpose of Matrix 1.

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

-								
	1	2	3		1	4	7]	
	4	5	6	T	2	5	8	
	7	8	9		3	6	9	

tan()

tan(Value1)⇒value $tan(List1) \Rightarrow list$

tan(Value 1) returns the tangent of the argument.

tan(List1) returns a list of the tangents of all elements in *List1*.

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

In Degree angle mode:

$\tan\left(\left(\frac{\pi}{4}\right)^{r}\right)$	1.
tan(45)	1.
tan({0,60,90})	{0.,1.73205,undef}

In Gradian angle mode:

tan()



$\tan\left(\left(\frac{\pi}{4}\right)^r\right)$	1.
tan(50)	1.
tan({0,50,100})	{0.,1.,undef}

In Radian angle mode:

$\frac{1}{\tan\left(\frac{\pi}{4}\right)}$	1.
tan(45°)	1.
$\tan\left\{\left\{\pi,\frac{\pi}{3},-\pi,\frac{\pi}{4}\right\}\right\}$	{0.,1.73205,0.,1.}

 $tan(squareMatrix 1) \Rightarrow squareMatrix$

Returns the matrix tangent of squareMatrix 1. This is not the same as calculating the tangent of each element. For information about the calculation method, refer to cos().

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

In Radian angle mode:

tan⁻¹()

 $tan^{-1}(Value 1) \Rightarrow value$ $tan^{-1}(List 1) \Rightarrow list$

 $tan^{-1}(Value 1)$ returns the angle whose tangent is Value 1.

 $tan^{-1}(List1)$ 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 (...).

 $tan^{-1}(squareMatrix 1) \Rightarrow squareMatrix$

In Degree angle mode:

 $tan^{-1}(1)$ 45

trig kev

In Gradian angle mode:

tan-1(1) 50

In Radian angle mode:

 $tan^{-1}(\{0,0.2,0.5\}) = \{0,0.197396,0.463648\}$

In Radian angle mode:

tan-1()



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 diagonalisable. The result always contains floating-point numbers.

tan-1 4	$\begin{bmatrix} 5 & 3 \\ 2 & 1 \\ -2 & 1 \end{bmatrix}$		
	-0.083658	1.26629	0.62263
	0.748539	0.630015	-0.070012
	1.68608	$^{-}1.18244$	0.455126

tanh() Catalogue > 🕮

 $tanh(Value1) \Rightarrow value$

 $tanh(List1) \Rightarrow list$

tanh(Value 1) returns the hyperbolic tangent of the argument.

tanh(List1) returns a list of the hyperbolic tangents of each element of List1.

 $tanh(squareMatrix1) \Rightarrow squareMatrix$

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 diagonalisable. The result always contains floating-point numbers.

tanh(1.2)	0.833655
tanh({0,1})	{0.,0.761594}

In Radian angle mode:

tanh-1() Catalog > [13]

tanh⁻¹(Value I)⇒value

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

tanh⁻¹(Value 1) returns the inverse hyperbolic tangent of the argument.

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)	0.
tanh-1({1,2.1,3})	
{undef,0.518046-1.5	708 ·i ,0.346574–1.570

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

tanh-1() Catalog > 🕮

 $tanh^{-1}(squareMatrix 1) \Rightarrow squareMatrix$

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 diagonalisable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format:

$$tanh^{-1}\begin{bmatrix} 4 & 2 & 1 \\ 4 & 2 & 1 \end{bmatrix} \\
\begin{bmatrix} -0.099353+0.164058 \cdot \mathbf{i} & 0.267834-1.4908 \\ -0.087596-0.725533 \cdot \mathbf{i} & 0.479679-0.94736 \end{bmatrix}$$

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

0.511463-2.08316·i

tCdf() Catalogue > 🗐

 $tCdf(lowBound,upBound,df) \Rightarrow number if lowBound$ and upBound are numbers, list if lowBound and upBound are lists

Computes the Student-t distribution probability between lowBound and upBound for the specified degrees of freedom df.

For $P(X \le upBound)$, set lowBound = -9E999.

Text

TextpromptString[, DispFlag]

Programming command: Pauses the programme and displays the character string promptString in a dialogue box.

When the user selects **OK**, programme 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 programme needs a typed response from the user, refer to Request, page 123, or RequestStr, page 125.

Note: You can use this command within a userdefined programme but not within a function.

Catalogue > 23

-0.878563+1.7901

Define a programme that pauses to display each of five random numbers in a dialogue 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 programme:

text_demo()

Sample of one dialogue box:



See If, page 67. Then

tInterval Catalogue > 👰

tInterval List[,Freq[,CLevel]]

(Data list input)

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 (page 144).

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
stat. $\overline{\mathbf{x}}$	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 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 (page 144).

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 210.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. \overline{x} 1- \overline{x} 2	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom
stat. \overline{x} 1, stat. \overline{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

tPdf() Catalogue > 🗐

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

Computes the probability density function (pdf) for the Student-t distribution at a specified x value with specified degrees of freedom df.

trace()

$trace(squareMatrix) \Rightarrow value$

Returns the trace (sum of all the elements on the main diagonal) of squareMatrix.

$\frac{1}{\text{trace} \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}}$	2 3 5 6 8 9	15
a:=12		12
$\overline{\operatorname{trace} \begin{bmatrix} a \\ 1 \end{bmatrix}}$	$\begin{bmatrix} 0 \\ a \end{bmatrix}$	24

Catalogue > 🕮

Try

Try

block1

Else

block2

EndTrv

Executes *block1* unless an error occurs. programme execution transfers to block2 if an error occurs in block1. System variable errCode contains the error code to allow the programme to perform error recovery. For a list of error codes, see "Error codes and messages," page 220.

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 programme and function definitions, refer to the Calculator section of your product guidebook.

Example 2

To see the commands Try, CirErr and **PassErr** in operation, enter the eigenvals() programme shown at the right. Run the programme by executing each of the following expressions.

eigenvals
$$\begin{bmatrix} -3\\ -41\\ 5 \end{bmatrix}$$
 $\begin{bmatrix} -1 & 2 & -3.1 \end{bmatrix}$

Catalogue > 🕮

Define *prog1*()=Prgm Try z := z + 1

> Disp "z incremented." Disp "Sorry, z undefined."

EndTrv EndPrgm

Done

z := 1 : prog I()

z incremented.

DelVar z:prog1()

Sorry, z undefined.

Done

Done

Define eigenvals(a,b)=Prgm

© programme eigenvals(A,B) displays eigenvalues of A·B

Try

Disp "A= ",a

Disp "B=",b

Disp " "

Disp "Eigenvalues of A·B are:",eigVI(a*b)

Catalogue > 🗐

Note: See also **CirErr**, page 22, and **PassErr**, page 108.

Else

If errCode=230 Then

Disp "Error: Product of A·B must be a

square matrix"

ClrErr

Else

PassErr

EndIf

EndTry

EndPrgm

tTest Catalogue > [2]

 $\mathsf{tTest} \ \mu 0$, List[Freq[Hypoth]]

(Data list input)

tTest $\mu \theta$, \overline{x} ,sx,n,[Hypoth]

(Summary stats input)

Performs a hypothesis test for a single unknown population mean μ when the population standard deviation σ is unknown. A summary of results is stored in the stat.results variable (page 144).

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 210.

Output variable	Description
stat.t	$(\overline{x} - \mu 0) / (stdev / sqrt(n))$
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected

Output variable	Description	
stat.df	Degrees of freedom	
$stat.\overline{\mathbf{x}}$	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

Catalogue > 😰

tTest_2Samp List1,List2[,Freq1[,Freq2[,Hypoth [,Pooled]]]]

(Data list input)

tTest 2Samp $\bar{x}1$,sx1,n1, $\bar{x}2$,sx2,n2[,Hypoth[,Pooled]]

(Summary stats input)

Computes a two-sample t test. A summary of results is stored in the *stat.results* variable (page 144).

Test H_0 : $\mu 1 = \mu 2$, against one of the following:

For H_a : μ 1< μ 2, set Hypoth<0

For H_a: $\mu 1 \neq \mu 2$ (default), set Hypoth=0

For H_a : μ 1> μ 2, 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 210.

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. \overline{x} 1, stat. \overline{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()

Catalogue > 🗐

tvmFV(N,I,PV,Pmt,[PpY],[CpY],[PmtAt]) $\Rightarrow value$

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 159. See also **amortTbl()**, page 7.

Catalogue > [1]

tvmI()

tvml(N,PV,Pmt,FV,[PpY],[CpY],[PmtAt]) $\Rightarrow value$

tvmI(240,100000,-1000,0,12,12)

10.5241

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 159. See also **amortTbl()**, page 7.

tvmN()

Catalogue > 😰

tvmN(I,PV,Pmt,FV,[PpY],[CpY],[PmtAt]) \Rightarrow value

tvmN(5,0,-500,77641,12,12)

120.

Financial function that calculates the number of payment periods.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 159. See also **amortTbl()**, page 7.

tvmPmt()

Catalogue > 🕡

tvmPmt(N,I,PV,FV,[PpY],[CpY], [PmtAt]) $\Rightarrow value$

tvmPmt(60,4,30000,0,12,12)

-552.496

Financial function that calculates the amount of each payment.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 159. See also amortTbl(),

page 7.

Catalogue > 🕄 tvmPV()

tvmPV(N,I,Pmt,FV,[PpY],[CpY],[PmtAt]) \Rightarrow value

tvmPV(48,4,-500,30000,12,12)

-3426.7

Financial function that calculates the present value.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 159. See also amortTbl(). page 7.

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
СрҮ	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 Catalogue > 23

TwoVar X, Y[, [Freq] [, Category, Include]]

Calculates the TwoVar statistics. A summary of results is stored in the *stat.results* variable (page 144).

Catalogue > 🗐

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

X and Y are lists of independent and dependent variables.

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

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

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

An empty (void) element in any of the lists X, Freq, or Category results in a void for the corresponding element of all those lists. An empty element in any of the lists XI through X20 results in a void for the corresponding element of all those lists. For more information on empty elements, see page 210.

Output variable	Description
stat.x	Mean of x values
stat. x	Sum of x values
stat. x2	Sum of x2 values
stat.sx	Sample standard deviation of x
stat. x	Population standard deviation of x
stat.n	Number of data points
stat. <u>y</u>	Mean of y values
stat. y	Sum of y values
stat. y ²	Sum of y2 values
stat.sy	Sample standard deviation of y
stat. y	Population standard deviation of y
stat. xy	Sum of x · y values
stat.r	Correlation coefficient
stat. MinX	Minimum of x values

Output variable	Description
stat.Q ₁ X	1st Quartile of x
stat.MedianX	Median of x
stat.Q ₃ X	3rd Quartile of x
stat.MaxX	Maximum of x values
stat.MinY	Minimum of y values
stat.Q ₁ Y	1st Quartile of y
stat.MedY	Median of y
stat.Q ₃ Y	3rd Quartile of y
stat.MaxY	Maximum of y values
stat. (x-) ²	Sum of squares of deviations from the mean of x
stat. (y-) ²	Sum of squares of deviations from the mean of y

U

unl ock

unitV() Catalogue > 📳

unitV(Vector1)⇒vector

Returns either a row- or column-unit vector, depending on the form of *Vector1*.

Vector 1 must be either a single-row matrix or a single-column matrix.

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

UIILOCK	
unLock Var1[, Var2] [, Var3]	a:=65
unLock Var.	Lock a
	getLockInfo(a)

Unlocks the specified variables or variable group. Locked variables cannot be modified or deleted.

See Lock, page 84, and getLockInfo(), page 63.

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

Catalogue > 🗐

varPop() Catalogue > [3]

 $varPop(List[, freqList]) \Rightarrow expression$

varPop({5,10,15,20,25,30}) 72.9167

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 210.

varSamp() Catalogue > [3]

 $varSamp(List[,freqList]) \Rightarrow expression$

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 210.

 $varSamp(Matrix 1[, freqMatrix]) \Rightarrow matrix$

Returns a row vector containing the sample variance of each column in *Matrix 1*.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

varSamp({1,2,5,-6,3,-2})	31
	2
varSamp({1,3,5},{4,6,2})	68
	33

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 210.

Note: Matrix 1 must contain at least two rows.

W

Wait Catalogue > 23

Wait timeInSeconds

Suspends execution for a period of timeInSeconds seconds.

Wait is particularly useful in a programme 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 Gion 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 programme 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(Expr1, StatusVar) $\Rightarrow expression$

Evaluates expression *Expr1*, returns the result and stores the codes of any generated warnings in the *StatusVar* list variable. If no warnings are generated, this function assigns Status Var an empty list.

Expr1 can be any valid TI-NspireTM or TI-Nspire™ CAS maths expression. You cannot use a command or assignment as Expr1.

Status Var must be a valid variable name.

For a list of warning codes and associated messages, see page 228.

<u> </u>	warnCodes(det([1.234	56 e -999]),warn)
wa	rn	{10029}

when() Catalogue > 🕮

when(Condition, trueResult [, falseResult] [, unknownResult]) $\Rightarrow expression$

Returns trueResult, falseResult, or unknownResult, depending on whether Condition is true, false, or unknown. Returns the input if there are too few arguments to specify the appropriate result.

Omit both falseResult and unknownResult to make an expression defined only in the region where Condition is true.

Use an **undef** falseResult to define an expression that graphs only on an interval.

when() is helpful for defining recursive functions.

$$when(x<0,x+3)|x=5$$
 undef

when $(n>0, n \cdot factoral(n-1), 1) \rightarrow factoral(n)$	
	Done
factoral(3)	6
3!	6

While Condition

Block

EndWhile

Executes the statements in *Block* as long as Condition is true.

Block can be either a single statement or a sequence of statements separated with the ":" character.

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

Define $sum_of_recip(n)$ =Func

sum of recip(3)

true xor true

5>3 xor 3>5

Local i,tempsum $1 \rightarrow i$

0 → tempsum While $i \le n$

 $i+1 \rightarrow i$ EndWhile Return tempsum

EndFunc Done 11

false

true

X

Catalogue > [3] xor

BooleanExpr1xorBooleanExpr2 returns Boolean expression

BooleanList1xorBooleanList2 returns Boolean list

BooleanMatrix lxorBooleanMatrix 2 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 106.

Integer 1 xor Integer $2 \Rightarrow integer$

In Hex base mode:

Important: Zero, not the letter O.

0h7AC36 xor 0h3D5F 0h79169

In Bin base mode:

xor

Catalogue > 🗐

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.

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 16.

Note: See or, page 106.

0b100101 xor 0b100

0b100001

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

zInterval

Catalogue > 😰

 $zInterval \sigma_{,}List[Freq[CLevel]]$

(Data list input)

zinterval σ, \overline{x}, n [, CLevel]

(Summary stats input)

Computes a *z* confidence interval. A summary of results is stored in the *stat.results* variable (page 144).

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
$stat.\overline{\mathbf{x}}$	Sample mean of the data sequence from the normal random distribution
stat.ME	Margin of error

Output variable	Description
stat.sx	Sample standard deviation
stat.n	Length of the data sequence with sample mean
stat.σ	Known population standard deviation for data sequence List

zInterval_1Prop

Catalogue > 23

zinterval 1Prop x,n [,CLevel]

Computes a one-proportion z confidence interval. A summary of results is stored in the stat.results variable (page 144).

x is a non-negative integer.

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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

Catalogue > 🗐

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 (page 144).

x1 and x2 are non-negative integers.

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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
stat. p 1	First sample proportion estimate

Output variable	Description
stat. p̂ 2	Second sample proportion estimate
stat.n1	Sample size in data sequence one
stat.n2	Sample size in data sequence two

zInterval_2Samp

Catalogue > 🕎

zInterval_2Samp σ_1, σ_2 , *List1*, *List2*[, *Freq1*[, *Freq2*, [CLevel]]]

(Data list input)

zInterval_2Samp σ_1 , σ_2 , \bar{x} l,nl, \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 (page 144).

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. \overline{x} 1- \overline{x} 2	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat. \overline{x} 1, stat. \overline{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 Catalogue > 👰

zTest $\mu \theta$, σ ,List,[Freq[,Hypoth]]

(Data list input)

zTest μ *θ*, σ , \overline{x} ,n[,Hypoth]

zTest

(Summary stats input)

Performs a z test with frequency freglist. A summary of results is stored in the stat.results variable (page 144).

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 210.

Output variable	Description
stat.z	$(\overline{x} - \mu 0) / (\sigma / \text{sqrt(n)})$
stat.P Value	Least probability at which the null hypothesis can be rejected
$\operatorname{stat}.\overline{\mathbf{x}}$	Sample mean of the data sequence in List
stat.sx	Sample standard deviation of the data sequence. Only returned for ${\it Data}$ input.
stat.n	Size of the sample

zTest_1Prop

Catalogue > 🗐

zTest_1Prop p0,x,n[,Hypoth]

Computes a one-proportion z test. A summary of results is stored in the stat.results variable (page 144).

x is a non-negative integer.

Test H_0 : $p = p\theta$ against one of the following:

For H_a : p > p0, set Hypoth>0

For H_a : $p \neq p0$ (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 210.

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

Catalogue > 🗐

 $zTest_2Prop x1,n1,x2,n2[,Hypoth]$

Computes a two-proportion z test. A summary of results is stored in the stat.results variable (page 144).

x1 and x2 are non-negative integers.

Test H_0 : p1 = p2, against one of the following:

For H_a : p1 > p2, set Hypoth > 0

For H_a : $p1 \neq p2$ (default), set Hypoth=0

For H_a : p < p0, set Hypoth < 0

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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

Catalogue > 🗐

zTest_2Samp σ_1, σ_2 ,List1,List2[,Freq1[,Freq2 [,Hypoth]]]

(Data list input)

zTest_2Samp $\sigma_1, \sigma_2, \overline{x} l, nl, \overline{x} 2, n2[Hypoth]$

(Summary stats input)

Computes a two-sample z test. A summary of results is stored in the stat.results variable (page 144).

Test H_0 : $\mu 1 = \mu 2$, against one of the following:

For H_a : $\mu 1 < \mu 2$, set Hypoth < 0

For H_a: $\mu 1 \neq \mu 2$ (default), set Hypoth=0

For H_a : $\mu 1 > \mu 2$, Hypoth > 0

For information on the effect of empty elements in a list, see "Empty (Void) Elements", page 210.

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. \overline{x} 1, stat. \overline{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)		+ key
Value1 + Value2 ⇒ value	56	56
Returns the sum of the two arguments.	56+4	60
	60+4	64
	64+4	68
	68+4	72

 $List1 + List2 \Rightarrow list$

 $Matrix1 + Matrix2 \Rightarrow matrix$

Returns a li sums of cor and List2 (

Matrix2 - matrix	[]	{10,5,1.5708}
ist (or matrix) containing the	$\left\{10,5,\frac{\pi}{2}\right\} \rightarrow l2$	(10,3,1.3700)
rresponding elements in <i>List l</i> (or <i>Matrix 1</i> and <i>Matrix 2</i>).	11+12	{32,8.14159,3.14159}
(0		

 $\left\{22,\pi,\frac{\pi}{2}\right\}\to 11$

Dimensions of the arguments must be equal.

$$Value + Listl \Rightarrow list$$

$$List1 + Value \Rightarrow list$$

Returns a list containing the sums of Value and each element in List 1.

 $Value + Matrix 1 \Rightarrow matrix$

$$Matrix 1 + Value \Rightarrow matrix$$

Returns a matrix with Value added to each element on the diagonal of *Matrix1*. *Matrix1* must be square.

Note: Use .+ (dot plus) to add an expression to each element.

15+{10,15,20}	{25,30,35}
{10,15,20}+15	{25,30,35}

{22,3.14159,1.5708}

20+	1	2	21	2
	3	4	3	24

- (subtract)		- key
Value1−Value2 ⇒ value	6-2	4
Returns Value 1 minus Value 2.	$\pi - \frac{\pi}{6}$	2.61799
List1 −List2⇒ list	$\left\{22,\pi,\frac{\pi}{2}\right\} - \left\{10,5,\frac{\pi}{2}\right\}$	{12,-1.85841,0.}
$Matrix1 - Matrix2 \Rightarrow matrix$	$ \frac{\begin{bmatrix} 22, 11, \frac{1}{2} \end{bmatrix} - \begin{bmatrix} 10, 3, \frac{1}{2} \end{bmatrix}}{\begin{bmatrix} 3 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 2 \end{bmatrix}} $	[2 2]

– (subtract)



Subtracts each element in List2 (or *Matrix2*) from the corresponding element in *List1* (or *Matrix1*), and returns the results.

Dimensions of the arguments must be egual.

$$Value - List1 \Rightarrow list$$

$$List1 - Value \Rightarrow list$$

Subtracts each List I element from Value or subtracts Value from each List1 element, and returns a list of the results.

$$Value - Matrix 1 \Rightarrow matrix$$

$$Matrix 1 - Value \Rightarrow matrix$$

Value - Matrix I returns a matrix of Value times the identity matrix minus Matrix1. Matrix1 must be square.

Matrix 1 - Value returns a matrix of Value times the identity matrix subtracted from Matrix1. Matrix1 must be square.

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

15-{10,15,20}	{5,0,-5}
{10,15,20}-15	{-5,0,5}

20-	1	2	19	-2
	3	4	-3	16

(multiply)

Value1•Value2 ⇒ value

Returns the product of the two arguments.

$$List1 \cdot List2 \Rightarrow list$$

Returns a list containing the products of the corresponding elements in *List1* and *List2*.

Dimensions of the lists must be equal.

 $Matrix 1 \cdot Matrix 2 \Rightarrow matrix$

Returns the matrix product of *Matrix1* and Matrix 2.

The number of columns in *Matrix1* must equal the number of rows in *Matrix2*.

$$\{1.,2,3\}\cdot\{4,5,6\}$$
 $\{4,10,18\}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 \\ 7 & 8 \\ 7 & 8 \end{bmatrix} \qquad \begin{bmatrix} 42 & 48 \\ 105 & 120 \end{bmatrix}$$

$$\pi \cdot \{4,5,6\}$$
 {12.5664,15.708,18.8496}

•(multiply)

× key

Value •List1 ⇒ list

List1•Value ⇒ list

Returns a list containing the products of *Value* and each element in *List I*.

 $Value \cdot Matrix 1 \Rightarrow matrix$

 $Matrix 1 \cdot Value \Rightarrow matrix$

Returns a matrix containing the products of *Value* and each element in *Matrix 1*.

Note: Use .•(dot multiply) to multiply an expression by each element.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 0.01$	0.01	0.0	02
[3 4]	[0.03	0.0	04]
6·identity(3)	[6	0 6	0
	0	6	0
	l 0	Λ	6

/(divide)

÷ key

0.57971

Value1/Value2 ⇒ value

Returns the quotient of *Value1* divided by *Value2*.

Note: See also Fraction template, page 1.

 $List1/List2 \Rightarrow list$

Returns a list containing the quotients of List1 divided by List2.

Dimensions of the lists must be equal.

 $Value/List1 \Rightarrow list$

 $List1/Value \Rightarrow list$

Returns a list containing the quotients of Value divided by List1 or List1 divided by Value.

 $Value / Matrix 1 \Rightarrow matrix$

 $Matrix1/Value \Rightarrow matrix$

Returns a matrix containing the quotients of Matrix 1/Value.

Note: Use ./ (dot divide) to divide an expression by each element.

{	1.,2,3}	

4,5,6

2

3.45

 $\left\{0.25, \frac{2}{5}, \frac{1}{2}\right\}$

 $\frac{6}{\left\{3,6,\sqrt{6}\right\}}$

{2,1,2.44949}

 $\frac{\{7,9,2\}}{7\cdot 9\cdot 2}$

 $\left\{\frac{1}{18}, \frac{1}{14}, \frac{1}{63}\right\}$

[7 9 2]

 $\frac{1}{18} \quad \frac{1}{14} \quad \frac{1}{63}$

^ (power)



Value1 ^ Value2⇒ value

 $List1 \land List2 \Rightarrow list$

4 ²	16
$\{2.4.6\}$ $\{1,2,3\}$	{2,16,216}

Returns the first argument raised to the power of the second argument.

Note: See also Exponent template, page 1.

For a list, returns the elements in *List1* raised to the power of the corresponding elements in List2.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

 $Value \land List1 \Rightarrow list$

Returns Value raised to the power of the elements in List1.

List1 ^ Value ⇒ list

Returns the elements in List1 raised to the power of Value.

 $squareMatrix1 \land integer \Rightarrow matrix$

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.

$\pi^{\{1,2,-3\}}$	{3.14159,9.8696,0.032252}
--------------------	---------------------------

$$\{1,2,3,4\}^{-2}$$
 $\{1,\frac{1}{4},\frac{1}{9},\frac{1}{16}\}$

$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$	2 2 4	7 15	10 22
$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$	2 4] ⁻¹	$\begin{bmatrix} -2\\ \frac{3}{2} \end{bmatrix}$	$\begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$
$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$	2]-2	$\frac{11}{2}$ $\frac{-15}{4}$	$\begin{bmatrix} \frac{-5}{2} \\ \frac{7}{4} \end{bmatrix}$

x² (square)

x² key

Value 12⇒ value

Returns the square of the argument.

 $List12 \Rightarrow list$

Returns a list containing the squares of the elements in *List1*.

 $squareMatrix 12 \Rightarrow matrix$

Returns the matrix square of squareMatrix I. This is not the same as calculating the square of each element. Use .^2 to calculate the square of each element.

$\frac{-}{4^2}$								16
$\frac{1}{2}$,4,6	}2					$\{4,1\}$	6,36}
$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$	4 5 6	6 7 8	2			40 49 58	64 79 94	88 109 130
$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$	4 5 6	6 7 8	.^ 2			$\begin{bmatrix} 4\\9\\16\end{bmatrix}$	16 25 5 36	5 36 5 49 5 64

.+ (dot add)

. + kevs

 $Matrix1 + Matrix2 \Rightarrow matrix$

Value .+ Matrix1⇒ matrix

Matrix1.+Matrix2 returns a matrix that is the sum of each pair of corresponding elements in Matrix1 and Matrix2.

Value .+ Matrix I returns a matrix that is the sum of Value and each element in Matrix I.

$ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} . + \begin{bmatrix} 10 & 30 \\ 20 & 40 \end{bmatrix} $	$\begin{bmatrix} 11 & 32 \\ 23 & 44 \end{bmatrix}$
$5 .+ \begin{bmatrix} 10 & 30 \\ 20 & 40 \end{bmatrix}$	[15 35] 25 45]

.- (dot subt.)

. – keys

Matrix1 .- Matrix2⇒ matrix

 $Value - Matrixl \Rightarrow matrix$

Matrix 1.— Matrix 2 returns a matrix that is the difference between each pair of corresponding elements in Matrix 1 and Matrix 2.

Value .— Matrix I returns a matrix that is the difference of Value and each element in Matrix I.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$	-9 -27	-18 -36
$5 \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$	-5	-15 -35
[30 40]	-25	-35]

.•(dot mult.)

Matrix1 .• Matrix2⇒ matrix

 $Value \cdot Matrix l \Rightarrow matrix$

Matrix1.• Matrix2 returns a matrix that is the product of each pair of corresponding elements in Matrix1 and Matrix2.

Value .• Matrix I returns a matrix containing the products of Value and each element in Matrix I

		_
$ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} $	[10	40 160
[3 4] [30 40]	90	160
$5 \cdot \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$	50 150	100
30 40	150	200

x kevs

□ i kove

./(dot divide)

 $Matrix1./Matrix2 \Rightarrow matrix$

 $Value ./Matrix 1 \Rightarrow matrix$

Matrix1./Matrix2 returns a matrix that is the quotient of each pair of corresponding elements in Matrix1 and Matrix2.

Value ./Matrix I returns a matrix that is the quotient of Value and each element in Matrix I

	· · · Keys
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} / \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$	$\begin{bmatrix} \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} \end{bmatrix}$
5 / \[\begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \]	$\begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{6} & \frac{1}{8} \end{bmatrix}$

.^ (dot power)

 $Matrix1 \land Matrix2 \Rightarrow matrix$

Value . ^ Matrix l ⇒ matrix

Matrix1.^ Matrix2 returns a matrix where each element in Matrix2 is the exponent for the corresponding element in Matrix1.

Value .^ *Matrix1* returns a matrix where each element in *Matrix1* is the exponent for *Value*

	. ^ keys
$ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} $	$\begin{bmatrix} 1 & 4 \\ 27 & \frac{1}{4} \end{bmatrix}$
$5 \begin{bmatrix} 0 & 2 \\ 3 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 25 \\ 125 & \frac{1}{5} \end{bmatrix}$

– (negate)

(-) key

-Value1 ⇒ value

 $-List1 \Rightarrow list$

 $-Matrix1 \Rightarrow matrix$

-2.43	-2.43
-{-1,0.4,1.2 E 19}	{1.,-0.4,-1.2 E 19}

- (negate)



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.

To see the entire result, press \triangle and then use \triangleleft and \triangleright to move the cursor.

% (percent)

ctrl 🕮 kevs

Value 1% ⇒ value

List1% ⇒ list

Matrix1% ⇒ *matrix*

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})%

0.13

{0.01,0.1,1.

<u>argument</u>

Returns

100

For a list or matrix, returns a list or matrix with each element divided by 100.

= (equal)



 $Expr1=Expr2 \Rightarrow Boolean \ expression$

List1= $List2 \Rightarrow Boolean\ list$

 $Matrix 1 = Matrix 2 \Rightarrow Boolean matrix$

Returns true if Expr1 is determined to be equal to Expr2.

Returns false if Expr1 is determined to not be equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Example function that uses maths test symbols: =, \neq , <, \leq , >, \geq

Define g(x)=Func

If *x*≤-5 Then

Return 5

ElseIf x > -5 and x < 0 Then

Return -x

ElseIf $x \ge 0$ and $x \ne 10$ Then

Return x

ElseIf x=10 Then

Return 3

EndIf

EndFunc

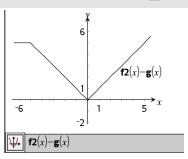
Done

Result of graphing g(x)

= (equal)

= key

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



\neq (not equal)

ctrl = keys

 $Expr1 \neq Expr2 \Rightarrow Boolean expression$

See "=" (equal) example.

 $List1 \neq List2 \Rightarrow Boolean \ list$

 $Matrix 1 \neq Matrix 2 \Rightarrow Boolean matrix$

Returns true if Expr1 is determined to be not equal to Expr2.

Returns false if Expr1 is determined to be equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing /=

< (less than)

ctrl = keys

 $Expr1 < Expr2 \Rightarrow Boolean expression$

See "=" (equal) example.

 $List1 < List2 \Rightarrow Boolean list$

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.

< (less than)

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

\leq (less or equal)

ctrl = kevs

 $Expr1 \leq Expr2 \Rightarrow Boolean expression$

See "=" (equal) example.

 $List1 \le List2 \Rightarrow Boolean\ list$

 $Matrix 1 \le Matrix 2 \Rightarrow Boolean matrix$

Returns true if *Expr1* is determined to be less than or equal to Expr2.

Returns false if Expr1 is determined to be greater than Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing <=

> (greater than)

ctrl = kevs

 $Expr1>Expr2 \Rightarrow Boolean expression$

See "=" (equal) example.

 $List1>List2 \Rightarrow Boolean\ list$

 $Matrix 1 > Matrix 2 \Rightarrow Boolean matrix$

Returns true if Expr1 is determined to be greater than Expr2.

Returns false if *Expr1* is determined to be less than or equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≥ (greater or equal)

 $Expr1 \ge Expr2 \Rightarrow Boolean expression$

See "=" (equal) example.

 $List1 > List2 \Rightarrow Boolean \ list$

 $Matrix1 \ge Matrix2 \Rightarrow Boolean \ matrix$

Returns true if Expr1 is determined to be greater than or equal to Expr2.

Returns false if *Expr1* is determined to be less than Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing >=

⇒ (logical implication)

ctrl = kevs

 $BooleanExpr1 \Rightarrow BooleanExpr2$ returns Boolean expression

 $BooleanList1 \Rightarrow BooleanList2$ returns Boolean list

 $BooleanMatrix1 \Rightarrow BooleanMatrix2$ returns Boolean matrix

 $Integer1 \Rightarrow Integer2$ returns Integer

Evaluates the expression not <argument1> or <argument2> and returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing =>

	iii keys
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}
$\{1,2,3\} \Rightarrow \{3,2,1\}$	{-1,-1,-3}

⇔ (logical double implication, XNOR)

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 <=>

true
false
7
-8
{2,0,2}
{-3,-1,-3}

! (factorial)		?!► key
Value1! ⇒ value	5!	120
$Listl! \Rightarrow list$	({5,4,3})!	{120,24,6}
$Matrix 1! \Rightarrow matrix$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$!	$\begin{bmatrix} 1 & 2 \\ 6 & 24 \end{bmatrix}$

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

& (append) $\text{ctrl} \boxtimes \text{keys}$ String1 & String2 \Rightarrow string "Hello "&"Nick" "Hello Nick"

Returns a text string that is *String2* appended to *String1*.

 $d(Expr1, Var[, Order]) \mid Var=Value \Rightarrow$ value

 $d(Expr1, Var[, Order]) \Rightarrow value$

 $d(List1, Var[, Order]) \Rightarrow list$

 $d(Matrix 1, Var[, Order]) \Rightarrow matrix$

Except when using the first syntax, you must store a numeric value in variable Var before evaluating d(). Refer to the examples.

d() can be used for calculating first and second order derivative at a point numerically, using auto differentiation methods.

Order, if included, must be=1 or 2. The default is 1.

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

Note: See also First derivative, page 5 or Second derivative, page 5.

Note: The d() algorithm has a limitation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of $x \cdot (x^2+x)^(1/3)$ at x=0 is equal to 0. However, because the first derivative of the subexpression $(x^2+x)^(1/3)$ is undefined at x=0, and this value is used to calculate the derivative of the total expression, d() reports the result as undefined and displays a warning message.

If you encounter this limitation, verify the solution graphically. You can also try using centralDiff().

$\frac{d}{dx}(x) x=0$	undef
$x:=0:\frac{d}{dx}(x)$	undef
$\overline{x:=3:\frac{d}{dx}(\left\{x^2,x^3,x^4\right\})}$	{6,27,108}

$$\frac{d}{dx}\left(x\cdot\left(x^2+x\right)^{\frac{1}{3}}\right)_{|x=0}$$
 undef centralDiff $\left(x\cdot\left(x^2+x\right)^{\frac{1}{3}}\right)_{|x=0}$ 0.000033

∫() (integral)

Catalogue > 💱

 $\int (Expr1, Var, Lower, Upper) \Rightarrow value$

Returns the integral of *Expr1* with respect to the variable *Var* from *Lower* to *Upper*. Can be used to calculate the definite integral numerically, using the same method as nlnt().

 $\int_{0}^{1} x^{2} dx$ 0.333333

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

Note: See also nint(), page 100, and Definiteintegral template, page 6.

() (square root)		ctrl x² keys
$\sqrt{(Value 1)} \Rightarrow value$	$\overline{\sqrt{4}}$	2
$\sqrt{(List 1)} \Rightarrow list$	$\sqrt{\left\{ 9,2,4\right\} }$	{3,1.41421,2}

Returns the square root of the argument.

For a list, returns the square roots of all the elements in *List1*.

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

Note: See also Square root template, page 1.

Π () (prodSeq)

Catalogue > 🕎

 $\Pi(Expr1, Var, Low, High) \Rightarrow expression$

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

Evaluates Expr I for each value of Var from Low to High, and returns the product of the results.

Note: See also Product template (Π), page 5.

 $\Pi(Expr1, Var, Low, Low-1) \Rightarrow 1$

 $\Pi(Expr1, Var, Low, High) \Rightarrow 1/\Pi(Expr1, Var, High+1, Low-1)$ if High < Low-1

$\left(\frac{1}{n}\right)$	120
$\frac{n=1}{5} \left\{ \left\{ \frac{1}{n}, n, 2 \right\} \right\}$	$\left\{\frac{1}{120},120,32\right\}$
$\frac{1 \left\{ \left(n' \right) \right\}}{n=1}$	

5

$$\frac{3}{\prod_{k=4}}(k)$$

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.

$\frac{1}{\left \frac{1}{k} \right }$	6
k=4	
$\frac{1}{\prod_{k=4}^{1} \left(\frac{1}{k}\right)} \cdot \frac{1}{\prod_{k=2}^{4} \left(\frac{1}{k}\right)}$	$\frac{1}{4}$

Σ () (sumSeq)

 Σ (Expr1, Var, Low, High) \Rightarrow expression

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

Evaluates *Expr1* for each value of *Var* from *Low* to *High*, and returns the sum of the results.

Note: See also Sum template, page 5.

 $\Sigma(Expr1, Var, Low, Low-1) \Rightarrow 0$

 $\Sigma(Expr1, Var, Low, High) \Rightarrow \mu$

 Σ (Expr1, Var, High+1, Low-1) if High < Low-1

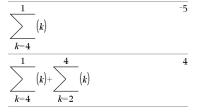
The summation formulas used are derived from the following reference:

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

Catalogue > 🗐

$$\sum_{n=1}^{5} \left(\frac{1}{n}\right) \qquad \frac{137}{60}$$

$$\sum_{k=4}^{3} \langle k \rangle$$



ΣInt() Catalogue > 🗊

 Σ **int**(*NPmt1*, *NPmt2*, *N*, *I*, *PV*, [*Pmt*], [*FV*], [*PpY*], [*CpY*], [*PmtAt*], [roundValue]) ⇒ value

 $\Sigma Int(NPmt1,NPmt2,amortTable) \Rightarrow value$

Amortization function that calculates the sum of the interest during a specified range of payments.

NPmt1 and *NPmt2* define the start and end boundaries of the payment range.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 159.

- 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.

Sint(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 7.

Note: See also Σ Prn(), below, and **Bal()**, page 15.

tbl:=amortTbl(12,12,4.75,20000,,12,12)					
	0	0.	0.	20000.	
	1	-77. 4 9	-1632.43	18367.6	
	2	-71.17	-1638.75	16728.8	
	3	$^{-}64.82$	$^{-}1645.1$	15083.7	
	4	-58.44	-1651.48	13432.2	
	5	-52.05	-1657.87	11774.4	
	6	-45.62	-1664.3	10110.1	
	7	-39.17	-1670.75	8439.32	
	8	-32.7	-1677.22	6762.1	
	9	-26.2	-1683.72	5078.38	
	10	-19.68	-1690.24	3388.14	
	11	-13.13	-1696.79	1691.35	
	12	-6.55	-1703.37	-12.02	
$\Sigma Int(1,3,tbl)$				-213.48	

 Σ Prn() Catalogue > \mathbb{Q}^3

 Σ Prn(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) \Rightarrow value

 Σ Prn(NPmt1, NPmt2, amortTable) \Rightarrow value

Amortization function that calculates the sum of the principal during a specified range of payments.

NPmt1 and *NPmt2* define the start and end boundaries of the payment range.

N, I, PV, Pmt, FV, PpY, CpY, and *PmtAt* are described in the table of TVM arguments, page 159.

 Σ Prn(1,3,12,4.75,20000,,12,12) -4916.28

Σ Prn()

Catalogue > 🗐

- 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 7.

Note: See also Σ Int(), above, and **Bal()**, page 15.

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
$\Sigma Prm(1,3,tbl)$ -4916.28				

(indirection)

varNameString

Refers to the variable whose name is varNameString. This lets you use strings to create variable names from within a function.

	 -
xyz:=12	12
#("x"&"y"&"z")	12

ctrl kevs

Creates or refers to the variable xyz.

$10 \rightarrow r$	10
"r" → s1	"r"
#s1	10

Returns the value of the variable (r) whose name is stored in variable s1.

E (scientific notation)

mantissaEexponent

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.

	ш кеу
23000.	23000.
2300000000.+4.1E15	4.1E15
3·10 ⁴	30000

 \Box

E (scientific notation)

| EE | kev

Note: You can insert this operator from the computer keyboard by typing @E. for example, type 2.3@E4 to enter 2.3E4.

g (gradian)

1 kev

 $Exprls \Rightarrow expression$

 $Listlg \Rightarrow list$

 $Matrix lg \Rightarrow matrix$

In Degree, Gradian or Radian mode:

cos(50 ⁹)	0.707107
$\cos(\{0,100^{g},200^{g}\})$	{1,0.,-1.}

This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies *Expr1* by $\pi/200$.

In Degree angle mode, multiplies *Expr1* by g/100.

In Gradian mode, returns *Expr1* unchanged.

Note: You can insert this symbol from the computer keyboard by typing @g.

r(radian)

11 kev

 $Value Ir \Rightarrow value$

 $List1r \Rightarrow list$

 $Matrix Ir \Rightarrow matrix$

This function gives you a way to specify a radian angle while in Degree or Gradian mode.

In Degree angle mode, multiplies the argument by $180/\pi$.

In Radian angle mode, returns the argument unchanged.

In Degree, Gradian or Radian angle mode:

$$\cos\left(\frac{\pi}{4^r}\right) \qquad 0.707107$$

$$\cos\left(\left\{0^r, \left(\frac{\pi}{12}\right)^r, -(\pi)^r\right\}\right) \qquad \left\{1, 0.965926, -1.\right\}$$

r(radian)

|1| kev

In Gradian mode, multiplies the argument by $200/\pi$.

Hint: Use r if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

Note: You can insert this symbol from the computer keyboard by typing @r.

° (degree)

1 kev

Value 1° ⇒ value

 $List1^{\circ} \Rightarrow list$

 $Matrix 1^{\circ} \Rightarrow matrix$

This function gives you a way to specify a degree angle while in Gradian or Radian mode.

In Radian angle mode, multiplies the argument by $\pi/180$.

In Degree angle mode, returns the argument unchanged.

In Gradian angle mode, multiplies the argument by 10/9.

Note: You can insert this symbol from the computer keyboard by typing @d.

In Degree, Gradian or Radian angle mode:

cos(45°) 0.707107

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 ≈ .

°, ', " (degree/minute/second)

 $dd^{\circ}mm'ss.ss'' \Rightarrow expression$

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.

In Degree angle mode:

25°13'17.5"	25.2215
25°30'	51
	2

°, ', " (degree/minute/second)

ctrl keys

• Enter time as hours/minutes/seconds.

Note: Follow ss. with two apostrophes ("), not a quote symbol (").

∠ (angle)

ctrl 🕮 keys

 $[Radius, \angle \theta_Angle] \Rightarrow vector$ (polar input)

 $[Radius, ∠ θ_Angle, Z_Coordinate] ⇒ vector$ (cylindrical input)

 $[Radius, ∠ θ_Angle, ∠ θ_Angle] \Rightarrow vector$ (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 @<.

 $(Magnitude \angle Angle) \Rightarrow complex Value$ (polar input)

Enters a complex value in $(r \angle \theta)$ polar form. The Angle is interpreted according to the current Angle mode setting.

In Radian mode and vector format set to: rectangular

cylindrical

spherical

In Radian angle mode and Rectangular complex format:

$$\frac{}{5+3 \cdot i - \left(10 \angle \frac{\pi}{4}\right)} \qquad ^{-2.07107 - 4.07107 \cdot i}$$

_ (underscore as an empty element)

See "Empty (Void) Elements," page 210.

10^()

Catalogue > [3]

31.6228

10^ (Value1) ⇒ value

10^{1.5}

10^ (Listl) $\Rightarrow list$

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in *List1*.

10^(squareMatrix 1**)** \Rightarrow 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.

5	3		
2	1		
-2	1		
	1.14336е7	8.17155E6	6.67589E6
	9.95651 E 6	7.11587 E 6	5.81342E6
	7.65298 e 6	5.46952 e 6	4.46845E6
		9.95651 E 6	5 3 2 1 -2 1 1 .14336E7 8.17155E6 9.95651E6 7.11587E6 7.65298E6 5.46952E6

^-1 (reciprocal)

Catalogue > [3]

Value 1 \land -1 \Rightarrow value

 $(3.1)^{-1}$

0.322581

 $List1 \land -1 \Rightarrow list$

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in *List1*.

 $squareMatrix1 \land -1 \Rightarrow squareMatrix$

Returns the inverse of *squareMatrix1*.

squareMatrix1 must be a non-singular square matrix.

[1	2]-1	-2	1
3	$\begin{bmatrix} 2 \\ 4 \end{bmatrix}^{-1}$	$\frac{3}{2}$	-1
L	73	2	2

(constraint operator)

ctrl 🕮 keys

Expr | BooleanExpr1[and BooleanExpr2]...

Expr | BooleanExpr1[orBooleanExpr2]...

The constraint ("|") symbol serves as a binary operator. The operand to the left of is an expression. The operand to the right of I 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:

x+1 x=3	4
$x+55 x=\sin(55)$	54.0002

(constraint operator)

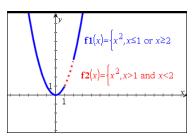
- Substitutions
- Interval constraints
- Exclusions

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.

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.

$x^3 - 2 \cdot x + 7 \rightarrow f(x)$	Done	
$f(x) x=\sqrt{3}$	8.73205	

$$\frac{\text{nSolve}(x^3 + 2 \cdot x^2 - 15 \cdot x = 0, x)}{\text{nSolve}(x^3 + 2 \cdot x^2 - 15 \cdot x = 0, x)|x > 0 \text{ and } x < 5} \quad 3.$$



Exclusions use the "not equals" (/= or \neq) relational operator to exclude a specific value from consideration.

\rightarrow (store)		ctrl var key
$Value \rightarrow Var$	$\frac{\pi}{\longrightarrow} mvvar$	0.785398

 $List \rightarrow Var$

 $Matrix \rightarrow Var$

 $Expr \rightarrow Function(Param 1,...)$

 $List \rightarrow Function(Param 1,...)$

 $Matrix \rightarrow Function(Param 1,...)$

If the variable Var does not exist, creates it and initializes it to Value, List, or Matrix.

If the variable Var already exists and is not locked or protected, replaces its contents with Value, List, or Matrix.

$\frac{\pi}{4} \rightarrow myvar$	0.785398
$2 \cdot \cos(x) \to y I(x)$	Done
$\{1,2,3,4\} \rightarrow lst5$	{1,2,3,4}
$ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \rightarrow matg $	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$
"Hello" → str1	"Hello"

\rightarrow (store)

var kev

Note: You can insert this operator from the keyboard by typing =: as a shortcut. For example, type pi/4 =: myvar.

:= (assign)

ctrl | | kevs

<i>Var</i> :=	Value
---------------	-------

Var := ListVar := Matrix

Function(Param1...) := Expr

Function(Paraml,...) := List

Function(Param1,...) := Matrix

If variable Var does not exist, creates Var and initializes it to Value, List, or Matrix.

If Var already exists and is not locked or protected, replaces its contents with Value, List. or Matrix.

π	.785398
$myvar:=\frac{\pi}{4}$	
$y1(x):=2\cdot\cos(x)$	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"

© (comment)

ctri 🕮 kevs

© [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 programme 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+i2

EndFor

Return result

EndFunc

Done

g(3)

14

0b, 0h

0 B keys, 0 H keys

0b binaryNumber

Oh hexadecimal Number

In Dec base mode:

0b, 0h	OB keys, OH keys
•	

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.

0b10+0hF+10	27
In Bin base mode:	
0b10+0hF+10	0b11011
In Hex base mode:	
0b10+0hF+10	0h1B

TI-Nspire[™] CX II - Draw Commands

This is a supplemental document for the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide, All TI-Nspire™ CX II commands will be incorporated and published in version 5.1 of the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide.

Graphics Programming

New commands have been added on TI-Nspire™ CX II Handhelds and TI-Nspire™ desktop applications for graphics programming.

The TI-Nspire™ CX II Handhelds will switch into this graphics mode while executing graphics commands and switch back to the context in which the program was executed after completion of the program.

The screen will display "Running..." in the top bar while the program is being executed. It will show "Finished" when the program completes. Any key-press will transition the system out of the graphics mode.

- The transition to graphics mode is triggered automatically when one of the Draw (graphics) commands is encountered during execution of the TI-Basic program.
- This transition will only happen when executing a program from calculator; in a document or calculator in scratchpad.
- The transition out of graphics mode happens upon termination of the program.
- The graphics mode is only available on the TI-Nspire™ CX II Handhelds and the desktop TI-Nspire™ CX II Handhelds view. This means it is not available in the computer document view or PublishView (.tnsp) on the desktop nor on iOS.
 - If a graphics command is encountered while executing a TI-Basic program from the incorrect context, an error message is displayed and the TI-Basic program is terminated.

Graphics Screen

The graphics screen will contain a header at the top of the screen that cannot be written to by graphics commands.

The graphics screen drawing area will be cleared (colour = 255,255,255) when the graphics screen is initialized.

Graphics Screen	Default
Height	212
Width	318
Colour	white: 255,255,255

Default View and Settings

- The status icons in the top bar (battery status, press-to-test status, network indicator etc.) will not be visible while a graphics program is running.
- Default drawing colour: Black (0.0.0)
- Default pen style normal, smooth
 - Thickness: 1 (thin), 2 (normal), 3 (thickest)
 - Style: 1 (smooth), 2 (dotted), 3 (dashed)
- All drawing commands will use the current colour and pen settings; either default values or those which were set via TI-Basic commands.
- Text font is fixed and cannot be changed.
- Any output to the graphics screen will be drawn within a clipping window which is the size of the graphics screen drawing area. Any drawn output that extends outside of this clipped graphics screen drawing area will not be drawn. No error message will be displayed.
- All x,y coordinates specified for drawing commands are defined such that 0,0 is at the top left corner of the graphics screen drawing area.
 - Exceptions:
 - **DrawText** uses the coordinates as the bottom left corner of the bounding box for the text.
 - **SetWindow** uses the bottom left corner of the screen
- All parameters for the commands can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.

Graphics Screen Errors Messages

If the validation fails, an error message will display.

Error Message	Description	View
Error Syntax	If the syntax checker finds any syntax errors, it displays an error message and tries to position the cursor near the first error so you can correct it.	Error Syntax
Error Too few arguments	The function or command is missing one or more arguments	Too few arguments The function or command is missing one or more arguments. OK
Error Too many arguments	The function or command contains and excessive number of arguments and cannot be evaluated.	Error Too many arguments The function or command contains an excessive number of arguments and cannot be evaluated. OK
Error Invalid data type	An argument is of the wrong data type.	Error Invalid data type An argument is of the wrong data type. OK

Invalid Commands While in Graphics Mode

Some commands are not allowed once the program switches to graphics mode. If these commands are encountered while in graphics mode an error will be displayed and the program will be terminated.

Disallowed Command	Error Message
Request	Request cannot be executed in graphics mode
RequestStr	RequestStr cannot be executed in graphics mode
Text	Text cannot be executed in graphics mode

The commands that print text to the calculator - disp and dispAt - will be supported commands in the graphics context. The text from these commands will be sent to the Calculator screen (not on Graphics) and will be visible after the program exits and the system switches back to the Calculator app

Clear	Catalogue > [1] CXII
Clear x, y, width, height	Clear
Clears entire screen if no parameters are specified.	Clears entire screen
If x , y , $width$ and $height$ are specified, the rectangle defined by the parameters will be cleared.	Clear 10,10,100,50 Clears a rectangle area with top left corner on (10, 10) and with width 100, height 50

DrawArc

Catalogue > 🔯

DrawArc x, y, width, height, startAngle, arcAngle

Draw an arc within the defined bounding rectangle with the provided start and arc angles.

x, y: upper left coordinate of bounding rectangle width, height: dimensions of bounding rectangle The "arc angle" defines the sweep of the arc.

These parameters can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.

DrawArc 20,20,100,100,0,90



DrawArc 50,50,100,100,0,180



See Also: FillArc

DrawCircle

Catalogue > 💷 CXII

DrawCircle x, y, radius

x, y: coordinate of centre radius: radius of the circle DrawCircle 150,150,40



See Also: FillCircle

DrawLine

Catalogue > 🗐

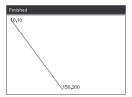
DrawLine x1, y1, x2, y2

Draw a line from x1, y1, x2, y2.

Expressions that evaluate to a number which is then rounded to the nearest integer.

Screen bounds: If the specified coordinates causes any part of the line to be drawn outside of the graphics screen, that part of the line will be clipped and no error message will be displayed.

DrawLine 10,10,150,200



DrawPoly

Catalogue > 🕮

The commands have two variants:

DrawPoly xlist, ylist

or

DrawPoly x1, y1, x2, y2, x3, y3...xn, yn

Note: DrawPoly *xlist*, *ylist*

Shape will connect x1, y1 to x2, y2, x2, y2 to x3, y3

and so on.

Note: DrawPoly *x1*, *y1*, *x2*, *y2*, *x3*, *y3*...*xn*, *yn* xn, yn will **NOT** be automatically connected to x1, y1.

Expressions that evaluate to a list of real floats xlist, ylist

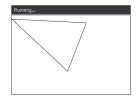
Expressions that evaluate to a single real float x1, y1...xn, yn = coordinates for vertices of polygon

Note: DrawPoly: Input size dimensions (width/height) relative to drawn lines.

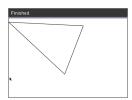
The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn polygon will be larger than the width and height.

See Also: FillPoly

xlist:={0,200,150,0} ylist:={10,20,150,10} DrawPoly xlist, ylist



DrawPoly 0,10,200,20,150,150,0,10





Catalogue > 🗐

CXII

DrawRect x, y, width, height

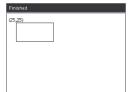
x, y: upper left coordinate of rectangle

width, height: width and height of rectangle (rectangle drawn down and right from starting coordinate).

Note: The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn rectangle will be larger than the width and height indicated.

See Also: FillRect

DrawRect 25,25,100,50



DrawText

DrawText x, y, exprOrString1 [,exprOrString2]...

x, y: coordinate of text output

Draws the text in *exprOrString* at the specified x, y coordinate location.

The rules for exprOrString are the same as for Disp DrawText can take multiple arguments.

DrawText 50,50,"Hello World"



FillArc

Catalogue > [[]] CXII

FillArc x, y, width, height startAngle, arcAngle

x, y: upper left coordinate of bounding rectangle

Draw and fill an arc within the defined bounding rectangle with the provided start and arc angles.

Default fill colour is black. The fill colour can be set by the SetColor command

The "arc angle" defines the sweep of the arc



FillArc 50,50,100,100,0,180

FillCircle

Catalogue > (CXII

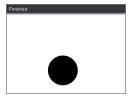
FillCircle *x*, *y*, radius

x, y: coordinate of centre

Draw and fill a circle at the specified centre with the specified radius.

Default fill colour is black. The fill colour can be set by the <u>SetColor</u> command.

FillCircle 150,150,40



Here!

FillPoly

Catalogue > [2] CXII

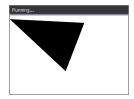
FillPoly xlist, ylist

or

FillPoly x1, y1, x2, y2, x3, y3...xn, yn

Note: The line and colour are specified by <u>SetColor</u> and <u>SetPen</u>

xlist:={0,200,150,0}
ylist:={10,20,150,10}
FillPoly xlist,ylist



FillPoly 0,10,200,20,150,150,0,10



FillRect

Catalogue > 🗐 CXII

FillRect x, y, width, height

x, y: upper left coordinate of rectangle

width, height: width and height of rectangle

Draw and fill a rectangle with the top left corner at the coordinate specified by (x,y)

Default fill colour is black. The fill colour can be set by the SetColor command

Note: The line and colour are specified by SetColor and SetPen



getPlatform() Catalogue > 📳 getPlatform() getPlatform() "dt" Returns: "dt" on desktop software applications

"hh" on TI-Nspire™ CX handhelds

"ios" on TI-Nspire™ CX iPad® app

PaintBuffer Catalogue > 23 **PaintBuffer** UseBuffer For n,1,10 Paint graphics buffer to screen x:=randInt(0,300) This command is used in conjunction with UseBuffer y:=randInt(0,200) to increase the speed of display on the screen when radius:=randInt(10,50) the program generates multiple graphical objects. Wait 0.5 DrawCircle x,y,radius EndFor PaintBuffer This program will display all the 10 circles at once. If the "UseBuffer" command is removed, each circle

See Also: UseBuffer

will be displayed as it is

drawn.

PlotXY x, y, shape

x, y: coordinate to plot shape

shape: a number between 1 and 13 specifying the shape

- 1 Filled circle
- 2 Empty circle
- 3 Filled square
- 4 Empty square
- 5 Cross
- 6 Plus
- 7 Thin
- 8 medium point, solid
- 9 medium point, empty
- 10 larger point, solid
- 11 larger point, empty
- 12 largest point, solid
- 13 largest point, empty

PlotXY 100,100,1

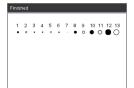


For n,1,13

DrawText 1+22*n,40,n

PlotXY 5+22*n,50,n

EndFor



SetColor

Catalogue > 🕮

SetColor

Red-value, Green-value, Blue-value

Valid values for red, green and blue are between 0 and 255

Sets the colour for subsequent Draw commands

SetColor 255,0,0 DrawCircle 150,150,100



SetPen

Catalogue > 🗐 CXII

SetPen

thickness, style

thickness: 1 <= thickness <= 3 | 1 is thinnest. 3 is thickest

style: 1 = Smooth, 2 = Dotted, 3 = Dashed

Sets the pen style for subsequent Draw commands

SetPen 3,3

DrawCircle 150,150,50



SetWindow



SetWindow

xMin, xMax, yMin, yMax

Establishes a logical window that maps to the graphics drawing area. All parameters are required.

If the part of drawn object is outside the window, the output will be clipped (not shown) and no error message is displayed.

SetWindow 0,160,0,120

will set the output window to have 0,0 in the bottom left corner with a width of 160 and a height of 120

DrawLine 0,0,100,100

SetWindow 0,160,0,120

SetPen 3,3

DrawLine 0,0,100,100

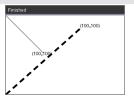


If xmin is greater than or equal to xmax or ymin is greater than or equal to ymax, an error message is shown.

Any objects drawn before a SetWindow command will not be re-drawn in the new configuration.

To reset the window parameters to the default, use:

SetWindow 0,0,0,0



UseBuffer Catalogue > 23 UseBuffer UseBuffer

Draw to an off screen graphics buffer instead of screen (to increase performance)

This command is used in conjunction with PaintBuffer to increase the speed of display on the screen when the program generates multiple graphical objects.

With UseBuffer, all the graphics are displayed only after the next PaintBuffer command is executed.

UseBuffer only needs to be called once in the program i.e. every use of PaintBuffer does not need a corresponding UseBuffer

For n,1,10

x:=randInt(0,300)

y:=randInt(0,200)

radius:=randInt(10,50)

Wait 0.5

DrawCircle x,y,radius

EndFor

PaintBuffer

This program will display all the 10 circles at once.

If the "UseBuffer" command is removed, each circle will be displayed as it is drawn.

See Also: PaintBuffer

Empty (Void) Elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ 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 38, and isVoid(), page 74.

Note: To enter an empty element manually in a maths expression, type " " or the keyword void. The keyword void is automatically converted to a " " symbol when

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, countlf, 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 $\begin{bmatrix} 1 & 2 \\ 3 & - \\ 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 4 & - \\ 9 & 8 \end{bmatrix}$

SortA and SortD move all void elements within the first argument to the bottom.

$\{5,4,3,_,1\} \rightarrow list1$	{5,4,3,_,1}
$\{5,4,3,2,1\} \rightarrow list2$	{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

$\{1,2,3,_,5\} \rightarrow list1$	{1,2,3,_,5}
$\{1,2,3,4,5\} \rightarrow list2$	{1,2,3,4,5}
SortD list1,list2	Done
list1	{5,3,2,1,_}
list2	{5,3,2,1,4}

In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

11:={1,2,3,4,5}: 12:={2,_,3,5,6.6}	}	
	{2,_,3,5,6.6}	
LinRegMx 11,12	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.

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

l1:={1,3,4,5}:	12:={2,3,5,6.6}	{2,3,5,6.6}
LinRegMx 11,1	2,{1,0,1,1}	Done
stat.Resid	{0.069231,_,-0.2	76923,0.207692}
stat.XReg		{1.,_,4.,5.}
stat.YReg		{2.,_,5.,6.6}
stat.FreqReg		{1.,_,1.,1.}

Shortcuts for Entering Maths Expressions

Shortcuts let you enter elements of maths expressions by typing instead of using the Catalogue or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type $\mathtt{sqrt}(6)$ on the entry line. When you press $\boxed{\mathtt{enter}}$, the expression $\mathtt{sqrt}(6)$ is changed to $\sqrt{6}$. 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
≤	<=
<u>></u>	>=
≠	/=
\Rightarrow (logical implication)	=>
dd⇔ (logical double implication, XNOR)	<=>
→ (store operator)	=:
(absolute value)	abs ()
√ ()	sqrt()
Σ () (Sum template)	sumSeq()
Π () (Product template)	prodSeq()
sin ⁻¹ (), cos ⁻¹ (),	arcsin(), arccos(),
ΔList()	deltaList()

From the Computer Keyboard

To enter this:	Type this shortcut:
i (imaginary constant)	@i
e (natural log base e)	@ e
E (scientific notation)	@ E
T (transpose)	@t
r (radians)	@r
° (degrees)	@d
g (gradians)	6 d

To enter this:	Type this shortcut:
∠ (angle)	@<
▶ (conversion)	@>
▶Decimal, ▶approxFraction () and so on.	<pre>@>Decimal, @>approxFraction() and so on.</pre>

EOS™ (Equation Operating System) Hierarchy

This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire[™] maths 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 ($^{\circ}$,',"), factorial (!), percentage (%), radian (r), subscript ([]), transpose (T)
5	Exponentiation, power operator (^)
6	Negation (-)
7	String concatenation (&)
8	Multiplication (*), division (/)
9	Addition (+), subtraction (-)
10	Equality relations: equal (=), not equal (\neq or /=), less than (<), less than or equal (\leq or <=), greater than (>), greater than or equal (\geq or >=)
11	Logical not
12	Logical and
13	Logical or
14	xor, nor, nand
15	Logical implication (⇒)
16	Logical double implication, XNOR (\Leftrightarrow)
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™ 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 programme. For example, if 10→r and "r" \rightarrow s1, then #s1=10.

Post Operators

Post operators are operators that come directly after an argument, such as 5!, 25%, or 60°15' 45". Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression 4³!, 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 2³2 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 -x2 is a negative number, and -92 = -81. Use parentheses to square a negative number such as (-9)2 to produce 81.

Constraint ("|")

The argument following the constraint ("|") operator provides a set of constraints that affect the evaluation of the argument preceding the operator.

TI-Nspire CX II - TI-Basic Programming Features

Auto-indentation in Programming Editor

The TI-Nspire™ program editor now auto-indents statements inside a block command.

Block commands are If/EndIf, For/EndFor, While/EndWhile, Loop/EndLoop, Try/EndTry

The editor will automatically prepend spaces to program commands inside a block command. The closing command of the block will be aligned with the opening command.

The example below shows auto-indentation in nested block commands.



Code fragments that are copied and pasted will retain the original indentation.

Opening a program created in an earlier version of the software will retain the original indentation.

Improved Error Messages for TI-Basic

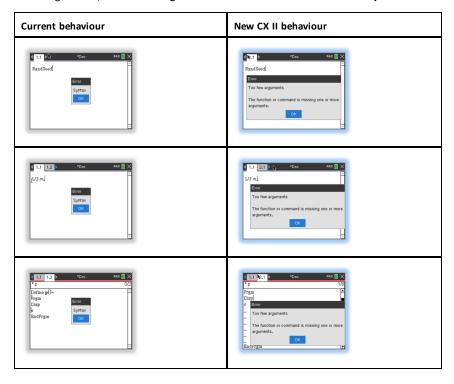
Errors

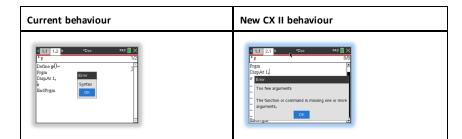
Error Condition	New message
Error in condition statement (If/While)	A conditional statement did not resolve to TRUE or FALSE NOTE : With the change to place the cursor on the line with the error, we no longer need to specify if the error is in an "If" statement or a "While" statement.
Missing EndIf	Expected EndIf but found a different end statement
Missing EndFor	Expected EndFor but found a different end statement
Missing EndWhile	Expected EndWhile but found a different end statement
Missing EndLoop	Expected EndLoop but found a different end statement

Error Condition	New message	
Missing EndTry	Expected EndTry but found a different end statement	
"Then" omitted after If <condition></condition>	Missing IfThen	
"Then" omitted after ElseIf <condition></condition>	Then missing in block: ElseIf.	
When "Then", "Else" and "Elself" were encountered outside of control blocks	Else invalid outside of blocks: IfThenEndIf or TryEndTry	
"Elself" appears outside of "IfThenEndIf" block	Elself invalid outside of block: IfThenEndIf	
"Then" appears outside of "IfEndIf" block	Then invalid outside of block: IfEndIf	

Syntax Errors

In case commands that expect one or more arguments are called with an incomplete list of arguments, a "Too few argument error" will be issued instead of "syntax" error





Note: When an incomplete list of arguments is not followed by a comma, the error message is: "too few arguments". This is the same as previous releases.



Constants and Values

The following table lists the constants and their values that are available when performing unit conversions. They can be typed in manually or selected from the Constants list in Utilities > Unit Conversions (Handheld: Press a).

Constant	Name	Value
_c	Speed of light	299792458 _m/_s
_Cc	Coulomb constant	8987551787.3682 _m/_F
_Fc	Faraday constant	96485.33289 _coul/_mol
_g	Acceleration of gravity	9.80665 _m/_s ²
_Gc	Gravitational constant	6.67408 E -11_m ³ /_kg/_s ²
_h	Planck's constant	6.626070040 E -34_J_s
_k	Boltzmann's constant	1.38064852 e -23_J/_°K
_μ0	Permeability of a vacuum	1.2566370614359E-6_N/_A ²
_µb	Bohr magneton	9.274009994E-24_J_m ² /_Wb
_Me	Electron rest mass	9.10938356 E -31_kg
_Μμ	Muon mass	1.883531594 E -28_kg
_Mn	Neutron rest mass	1.674927471 E -27_kg
_Mp	Proton rest mass	1.672621898 E -27_kg
_Na	Avogadro's number	6.022140857E23 /_mol
_q	Electron charge	1.6021766208E-19 _coul
_Rb	Bohr radius	5.2917721067 E -11 _m
_Rc	Molar gas constant	8.3144598 _J/_mol/_°K
_Rdb	Rydberg constant	10973731.568508/_m
_Re	Electron radius	2.8179403227 E -15 _m
_u	Atomic mass	1.660539040 E -27_kg
_Vm	Molar volume	2.2413962 E -2 _m ³ /_mol
_ε0	Permittivity of a vacuum	8.8541878176204E-12_F/_m
_σ	Stefan-Boltzmann constant	5.670367 E -8 _W/_m ² /_°K ⁴
_φ0	Magnetic flux quantum	2.067833831E-15 _Wb

Error Codes and Messages

When an error occurs, its code is assigned to variable *errCode*. User-defined programs and functions can examine errCode to determine the cause of an error. For an example of using errCode, See Example 2 under the Try command, page 155.

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 a="" b="" cause="" either="" error="" executed.<="" if="" is="" or="" statement="" td="" the="" this="" undefined="" when="" will="">	
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 ক্রিনা key was pressed during a long calculation or during programme 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 $(3x^2-4=0,x) \mid x<0 \text{ or } x>5 would produce this error message because the constraint is separated by "or" instead of "and."$
210	Invalid Data type
	An argument is of the wrong data type.
220	Dependent limit
230	Dimension
	A list or matrix index is not valid. For example, if the list $\{1,2,3,4\}$ is stored in L1, then L1[5] is a dimension error because L1 only contains four elements.
235	Dimension Error. Not enough elements in the lists.
240	Dimension mismatch
	Two or more arguments must be of the same dimension. For example, [1,2]+[1,2,3] is a dimension mismatch because the matrices contain a different number of elements.
250	Divide by zero
260	Domain error
	An argument must be in a specified domain. For example, rand(0) is not valid.
270	Duplicate variable name
280	Else and ElseIf invalid outside of IfEndIf 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

Error code	Description
	For example, solve $(3x^2-4,x)$ is invalid because the first argument is not an equation.
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 TryEndTry block
510	Invalid list or matrix
550	Invalid outside function or programme
	A number of commands are not valid outside a function or programme. For example, Local cannot be used unless it is in a function or programme.
560	Invalid outside LoopEndLoop, ForEndFor, or WhileEndWhile blocks
	For example, the Exit command is valid only inside these loop blocks.
565	Invalid outside programme
570	Invalid pathname
	For example, \var is invalid.
575	Invalid polar complex
580	Invalid programme reference
	Programs cannot be referenced within functions or expressions such as $1+p(x)$ where p is a programme.

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 diagonalisable
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 IfEndIf block
750	Name is not a function or programme
765	No functions selected
780	No solution found
800	Non-real result
	For example, if the software is in the Real setting, $\sqrt{ ext{(-1)}}$ is invalid.
	To allow complex results, change the "Real or Complex" Mode Setting to RECTANGULAR or POLAR.

Description
Overflow
programme not found
A programme reference inside another programme could not be found in the provided path during execution.
Rand type functions not allowed in graphing
Recursion too deep
Reserved name or system variable
Argument error
Median-median model could not be applied to data set.
Syntax error
Text not found
Too few arguments
The function or command is missing one or more arguments.
Too many arguments
The expression or equation contains an excessive number of arguments and cannot be evaluated.
Too many subscripts
Too many undefined variables
Variable is not defined
No value is assigned to variable. Use one of the following commands:
• sto →
• := • Define
to assign values to variables.
Unlicensed OS
Variable in use so references or changes are not allowed
Variable is protected
Invalid variable name
Make sure that the name does not exceed the length limitations
Window variables do main

Error code	Description	
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, $\sqrt{(-1)}$ 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 A pathname must be in the form xxx\yyy, where: The xxx part can have 1 to 16 characters.	

Error code	Description	
	The <i>yyy</i> 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 programme 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. Trigonometric conversion operators are not supported in Degree or Gradian angle modes.	
1250	Argument Error	

Error code	Description
	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 164.

Warning code	Message
10000	Operation might introduce false solutions.
	When applicable, try using graphical methods to verify the results.
10001	Differentiating an equation may produce a false equation.
10002	Questionable solution
	When applicable, try using graphical methods to verify the results.
10003	Questionable accuracy
	When applicable, try using graphical methods to verify the results.
10004	Operation might lose solutions.
	When applicable, try using graphical methods to verify the results.
10005	cSolve might specify more zeroes.
10006	Solve may specify more zeroes.
	When applicable, try using graphical methods to verify the results.
10007	More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess.
	Examples using solve():
	solve(Equation, Var=Guess) lowBound <var<upbound< td=""></var<upbound<>
	 solve(Equation, Var) lowBound solve(Equation, Var=Guess)
	When applicable, try using graphical methods to verify the results.
10008	Domain of the result might be smaller than the domain of the input.
10009	Domain of the result might be larger than the domain of the input.
10003	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

Warning code	Message
10017	Overflow replaced by ∞ or $-\infty$
10018	Operation requires and returns 64 bit value.
10019	Resource exhaustion, simplification might be incomplete.
10020	Trig function argument too big for accurate reduction.
10021	Input contains an undefined parameter.
	Result might not be valid for all possible parameter values.
10022	Specifying appropriate lower and upper bounds might produce a solution.
10023	Scalar has been multiplied by the identity matrix.
10024	Result obtained using approximate arithmetic.
10025	Equivalence cannot be verified in EXACT mode.
10026	Constraint might be ignored. Specify constraint in the form "\" 'Variable MathTestSymbol Constant' or a conjunct of these forms, for example 'x<3 and x>-12'

General Information

Online Help

education.ti.com/eguide

Select your country for more product information.

Contact TI Support

education.ti.com/ti-cares

Select your country for technical and other support resources.

Service and Warranty Information

education.ti.com/warranty

Select your country for information about the length and terms of the warranty or about product service.

Limited Warranty. This warranty does not affect your statutory rights.

Texas Instruments Incorporated

12500 TI Blvd.

Dallas, TX 75243

Index		^, power	175
_		1	
-, subtract	172	, constraint operator	191
	1/2	,	
ļ.		'minute notation	189
!, factorial	182	+	
"		+, add	172
", second notation	189	=	
#		≠, not equal	179
#, indirection	187	≤, less than or equal	180
#, indirection operator	215	≥, greater than or equal	181
2/		>, greater than	180
%		=, equal	178
%, percent	178	Π	
&		∏, product	184
&, append	182	Σ	
*		Σ(), sum	185
*	.=0	ΣInt()	185
*, multiply	173	∑Prn()	186
•		٧	
, dot subtraction	176	V, square root	104
.*, dot multiplication	177	v, square root	184
./, dot division	177	∠	
.^, dot power	177	/ (l-)	
.+, dot addition	176	∠ (angle)	190
/		l	
/, divide	174	ʃ, integral	184
:		•	
:=, assign	193	▶approxFraction()	13
_	193	▶approxFraction()▶Base10, display as decimal integer	13 17
:=, assign	193	**	_

►Base2, display as binary	16	absolute value	
►Cylind, display as cylindrical vector	34	template for	3-4
►DD, display as decimal angle	35	add, +	172
Decimal, display result as decimal	35	amortisation table, amortTbl()	7, 15
►DMS, display as		amortTbl(), amortisation table	7, 15
degree/minute/second	42	and, Boolean operator	8
▶Grad, convert to gradian angle	66	angle(), angle	9
▶Polar, display as polar vector	110	angle, angle()	9
▶Rad, convert to radian angle	117	ANOVA, one-way variance analysis	9
▶Rect, display as rectangular vector	120	ANOVA2way, two-way variance	
►Sphere, display as spherical vector .	143	analysis	10
		Ans, last answer	12
⇒		answer (last), Ans	12
⇒, logical implication181,	212	append, &	182
, 5 ,	,	approx(), approximate	12
→		approximate, approx()	12
N. atama wanialala		approxRational()	13
→, store variable	192	arccos(), cos ⁻¹ ()	13
⇔		arccosh(), cosh ⁻¹ ()	13
		arccot(), cot ⁻¹ ()	13
⇔ , logical double implication182,	, 212	arccoth(), coth ⁻¹ ()	13
		arccsc(), csc ⁻¹ ()	13
©		arccsch(), csch ⁻¹ ()	14
©, comment	193	arcsec(), sec ⁻¹ ()	14
		arcsech(), sech ⁻¹ ()	14
•		arcsin(), sin ⁻¹ ()	14
°, degree notation	100	arcsinh(), sinh ⁻¹ ()	14
°, degrees/minutes/seconds	189	arctan(), tan ⁻¹ ()	14
, degrees/illitates/secollus	189	arctanh(), tanh ⁻¹ ()	14
0		arguments in TVM functions	159
		augment(), augment/concatenate	14
0b, binary indicator	193	augment/concatenate, augment()	14
Oh, hexadecimal indicator	193	average rate of change, avgRC()	15
1		avgRC(), average rate of change	15
1		В	
10^(), power of ten	190	_	
		binary	
2		display, ►Base2	16
2-sample F Test	56	indicator, 0b	193
F	50	binomCdf()	18, 72
Α		binomPdf()	19
also() also a beta coal co	_	Boolean operators	04 040
abs(), absolute value	7	⇒1	81, 212

⇔	182	copy variable or function, CopyVar	24
and	182	correlation matrix, corrMat()	24
nand	97	corrMat(), correlation matrix	24
nor	•	cos ⁻¹ , arccosine	26
not	101	cos(), cosine	
	103	cosh ⁻¹ (), hyperbolic arccosine	25
or	106		27
xor	165	cosh(), hyperbolic cosine	26
С		cosine, cos()	25
-		cot ⁻¹ (), arccotangent	28
Cdf()	51	cot(), cotangent	28
ceiling(), ceiling	19	cotangent, cot()	28
ceiling, ceiling()	19, 30	coth ⁻¹ (), hyperbolic arccotangent	29
centralDiff()	19	coth(), hyperbolic cotangent	28
char(), character string	20	count days between dates, dbd()	34
character string, char()	20	count items in a list conditionally ,	20
characters		countif()	29
numeric code, ord()	107	count items in a list, count()	29
string, char()	20	count(), count items in a list	29
χ^2 2way	20	countif(), conditionally count items	29
χ ² Cdf()	21	in a listcPolyRoots()	30
$\chi^2 GOF$	21	cross product, crossP()	
$\chi^2 Pdf()$	21	crossP(), cross product	30
clear		csc ⁻¹ (), inverse cosecant	30
error, ClrErr	22	csc(), cosecant	31
Clear	198		31
ClearAZ	22	csch ⁻¹ (), inverse hyperbolic cosecant	32
ClrErr, clear error	22	csch(), hyperbolic cosecant	32
colAugment	23	cubic regression, CubicReg	32
colDim(), matrix column dimension	23	CubicReg, cubic regression	32
colNorm(), matrix column norm	23	cumulative sum, cumulativeSum().	33
combinations, nCr()	98	cumulativeSum(), cumulative sum	33
comment, ©	193	cycle, Cycle	33
complex		Cycle, cycle	33
conjugate, conj()	23	cylindrical vector display, ▶Cylind	34
conj(), complex conjugate	23	D	
constraint operator " "	191	b	
constraint operator, order of		d(), first derivative	183
evaluation	214	days between dates, dbd()	34
construct matrix, constructMat() .	23	dbd(), days between dates	34
constructMat(), construct matrix .	23	decimal	
convert		angle display, ►DD	35
▶Grad	66	integer display, ►Base10	17
▶Rad	117	Define	35

Define LibPriv	37	binomPdf()	19
Define LibPub	37	invNorm()	72
define, Define	35	invt()	72
Define, define	35	Invχ²()	71
defining		normCdf()	102
private function or programme	37	normPdf()	102
public function or programme .	37	poissCdf()	109
definite integral		poissPdf()	109
template for	6	tCdf()	152
degree notation, °	189	tPdf()	154
degree/minute/second display,		χ²2way()	20
►DMS	42	χ²Cdf()	21
degree/minute/second notation	189	χ²GOF()	21
delete		χ²Pdf()	21
void elements from list	38		
deleting		divide, /dot	174
variable, DelVar	38	addition, .+	176
deltaList()	37	division, ./	170
DelVar, delete variable	38	multiplication, .*	
delVoid(), remove void elements	38	power, .^	177
derivatives			177
first derivative, d()	183	product, dotP()	42
numeric derivative, nDeriv()	99-100	subtraction,	176
numeric derivative, nDerivative(dotP(), dot product	42
)	98	draw1	.99-201
det(), matrix determinant	38	E	
diag(), matrix diagonal	39	L	
dim(), dimension	39	e exponent	
dimension, dim()	39	template for	2
Disp, display data	40, 132	e to a power, e^()	43, 48
DispAt	40	E, exponent	187
display as		e^(), e to a power	43
binary, ►Base2	16	eff(), convert nominal to effective	
cylindrical vector, ▶Cylind	34	rate	43
decimal angle, ►DD	35	effective rate, eff()	43
decimal integer, ►Base10	17	eigenvalue, eigVI()	44
degree/minute/second, ►DMS .	42	eigenvector, eigVc()	44
hexadecimal, ▶Base16	18	eigVc(), eigenvector	44
polar vector, ▶Polar	110	eigVI(), eigenvalue	44
rectangular vector, ▶Rect	120	else if, ElseIf	45
spherical vector, ▶Sphere	143	else, Else	67
display data, Disp	_	Elself, else if	45
distribution functions	,	empty (void) elements	210
binomCdf()	18.72	P - / / /	210

end		financial functions, tvml()	158
for, EndFor	53	financial functions, tvmN()	158
function, EndFunc	56	financial functions, tvmPmt()	158
if, EndIf	67	financial functions, tvmPV()	159
loop, EndLoop	87	first derivative	
try, EndTry	155	template for	5
while, EndWhile	165	FiveNumSummary	51
end function, EndFunc	56	floor(), floor	52
end if, EndIf	67	floor, floor()	52
end loop, EndLoop	87	For	53
end while, EndWhile	165	for, For	53
EndTry, end try	155	For, for	53
EndWhile, end while	165	format string, format()	53
EOS (Equation Operating System)	214	format(), format string	53
equal, =	178	fpart(), function part	54
Equation Operating System (EOS)	214	fractions	
error codes and messages22	0. 228	propFrac	113
errors and troubleshooting	-, -	template for	1
clear error, ClrErr	22	freqTable()	54
pass error, PassErr	108	frequency()	55
euler(), Euler function	46	Frobenius norm, norm()	102
evaluate polynomial, polyEval()	110	Func, function	56
evaluation, order of	214	Func, programme function	56
exclusion with " " operator	191	functions	
exit, Exit	48	part, fpart()	54
Exit, exit	48	programme function, Func	56
exp(), e to a power	48	user-defined	35
exponent, E	187	functions and variables	
exponential regession, ExpReg	49	copying	24
exponents			
template for	1	G	
expr(), string to expression	49	g, gradians	188
ExpReg, exponential regession	49	gcd(), greatest common divisor	57
expressions		geomCdf()	57
string to expression, expr()	49	geomPdf()	58
_		Get	
F		get/return	00, 20 .
factor(), factor	50	denominator, getDenom()	59
factor, factor()	50	number, getNum()	64
factorial, !	182	variables injformation,	
fill20		getVarInfo()	62, 65
Fill, matrix fill	51	getDenom(), get/return	
financial functions, tvmFV()	158	denominator	59
	100		

getKey()	59	int(), integer	70
getLangInfo(), get/return language		intDiv(), integer divide	70
information	62	integer divide, intDiv()	70
getLockInfo(), tests lock status of		integer part, iPart()	73
variable or variable group .	63	integer, int()	70
getMode(), get mode settings	63	integral, §	184
getNum(), get/return number	64	interpolate(), interpolate	70
GetStr	64	inverse cumulative normal	, 0
getType(), get type of variable	65	distribution (invNorm()	72
getVarInfo(), get/return variables		inverse, ^-1	191
information	65	invF()	71
go to, Goto	66	invNorm(), inverse cumulative	, 1
Goto, go to	66	normal distribution)	72
gradian notation, g	188	invt()	72
greater than or equal, ≥	181	Invx²()	71
greater than, >	180	iPart(), integer part	73
greatest common divisor, gcd()	57	irr(), internal rate of return	,,
groups, locking and unlocking		internal rate of return, irr()	73
groups, testing lock status	63	isPrime(), prime test	73
		isVoid(), test for void	74
н		.5.0.0(//, 1050.10.10.00	74
hexadecimal		L	
display, ►Base16	18	label, LbI	7.1
indicator, 0h	193	language	74
hyperbolic	133	get language information	62
arccosine, cosh ⁻¹ ()	27	Lbl, label	74
arcsine, sinh ⁻¹ ()	140	lcm, least common multiple	74 75
arctangent, tanh ⁻¹ ()	151	least common multiple, lcm	
cosine, cosh()	26	•	75
sine, sinh()	140	left (), left	75
tangent, tanh()	151	left, left()	75
tungent, tunn()	131	length of string	39
1		less than or equal, ≤	180
		LibPriv	37
identity matrix, identity()	67	LibPub	37
identity(), identity matrix	67	library	
if, If	67	create shortcuts to objects	75
If, if	67	libShortcut(), create shortcuts to	75
ifFn()	68	library objects	75
imag(), imaginary part	69	linear regression, LinRegAx	77
imaginary part, imag()	69	linear regression, LinRegBx	76, 78
indirection operator (#)	215	LinRegBx, linear regression	76
indirection, #	187	LinRegMx, linear regression	77
		LinRegtIntervals, linear regression	78
inString(), within string	69		,,

LinRegtTest	79	loop, Loop	87
linSolve()	81	Loop, loop	87
Δlist(), list difference	81	LU, matrix lower-upper	0.
list to matrix, list►mat()	82	decomposition	88
list, conditionally count items in	29	·	
list, count items in	29	M	
list►mat(), list to matrix	82	mat ►list(), matrix to list	89
lists	02	matrices	09
augment/concatenate,		augment/concatenate,	
augment()	14	augment()	14
cross product, crossP()	30	column dimension, colDim()	23
cumulative sum,		column norm, colNorm()	23
cumulativeSum()	33	cumulative sum,	
differences in a list, Δlist()	81	cumulativeSum()	33
dot product, dotP()	42	determinant, det()	38
empty elements in	210	diagonal, diag()	39
list to matrix, list▶mat()	82	dimension, dim()	39
matrix to list, mat list()	89	dot addition, .+	176
maximum, max()	89	dot division, ./	177
mid-string, mid()	91	dot multiplication, .*	177
minimum, min()	92	dot power, .^	177
new, newList()	99	dot subtraction,	176
product, product()	113	eigenvalue, eigVl()	44
sort ascending, SortA	142	eigenvector, eigVc()	44
sort descending, SortD	143	filling, Fill	51
summation, sum()14	17-148	identity, identity()	67
ln(), natural logarithm	82	list to matrix, list rmat()	82
LnReg, logarithmic regression	83	lower-upper decomposition, LU	88
local variable, Local	84	matrix to list, mat list()	89
local, Local	84	maximum, max()	89
Local, local variable	84	minimum, min()	92
Lock, lock variable or variable group	84	new, newMat()	99
locking variables and variable groups	84	product, product()	113
Log		QR factorization, QR	114
template for	2	random, randMat()	119
logarithmic regression, LnReg	83	reduced row echelon form, rref(
logarithms	82)	130
logical double implication, ⇔	182	row addition, rowAdd()	129
logical implication, ⇒18	31, 212	row dimension, rowDim()	130
logistic regression, Logistic	85	row echelon form, ref()	121
logistic regression, LogisticD	86	row multiplication and addition,	
Logistic, logistic regression	85	mRowAdd()	94
LogisticD, logistic regression	86	row norm, rowNorm()	130

row operation, mRow()	94	multiply, *	173
row swap, rowSwap()	130	MultReg	94
submatrix, subMat()14		MultRegIntervals()	95
summation, sum()14		MultRegTests()	95
transpose, T	149	ζ (,	
matrix (1×2)	143	N	
template for	4		
matrix (2 × 1)		nand, Boolean operator	97
template for	4	natural logarithm, ln()	82
matrix (2 × 2)	-	nCr(), combinations	98
template for	4	nDerivative(), numeric derivative	98
matrix (m × n)		negation, entering negative numbers	215
template for	4	net present value, npv()	104
matrix to list, mat list()	89	new	
max(), maximum	89	list, newList()	99
maximum, max()	89	matrix, newMat()	99
mean(), mean	89	newList(), new list	99
mean, mean()	89	newMat(), new matrix	99
median(), median	90	nfMax(), numeric function	
median, median()	90	maximum	99
medium-medium line regression,	30	nfMin(), numeric function minimum	100
MedMed	90	nInt(), numeric integral	100
MedMed, medium-medium line	30	nom), convert effective to nominal	
regression	90	rate	101
mid-string, mid()	91	nominal rate, nom()	101
mid(), mid-string	91	nor, Boolean operator	101
min(), minimum	92	norm(), Frobenius norm	102
minimum, min()	92	normal distribution probability,	
minute notation, '	189	normCdf()	102
mirr(), modified internal rate of		normCdf()	102
return	93	normPdf()	102
mixed fractions, using propFrac()		not equal, ≠	179
with	113	not, Boolean operator	103
mod(), modulo	93	nPr(), permutations	103
mode settings, getMode()	63	npv(), net present value	104
modes		nSolve(), numeric solution	104
setting, setMode()	135	nth root	
modified internal rate of return, mirr		template for	2
()	93	numeric	
modulo, mod()	93	derivative, nDeriv()	99-100
mRow(), matrix row operation	94	derivative, nDerivative()	98
mRowAdd(), matrix row		integral, nInt()	100
multiplication and addition	94	solution, nSolve()	104
Multiple linear regression t test	95		

O		prodSeq()	113
objects		product(), product	113
create shortcuts to library	75	product, $\Pi()$	184
one-variable statistics, OneVar	105	template for	5
OneVar, one-variable statistics	105	product, product()	113
operators		programmes and programming	422
order of evaluation	214	display I/O screen, Disp	132
or (Boolean), or	106	programming define programme, Prgm	112
or, Boolean operator	106	display data, Disp	
ord(), numeric character code	107	pass error, PassErr	
		programs	108
Р		defining private library	37
P►Rx(), rectangular x coordinate	108	defining public library	37
P*Ry(), rectangular y coordinate	108	programs and programming	
pass error, PassErr	108	clear error, ClrErr	22
PassErr, pass error	108	display I/O screen, Disp	40
Pdf()	54	end try, EndTry	155
percent, %	178	try, Try	155
permutations, nPr()	103	proper fraction, propFrac	113
piecewise function (2-piece)	103	propFrac, proper fraction	113
template for	2		
piecewise function (N-piece)	_	Q	
template for	2	QR factorization, QR	114
piecewise()	109	QR, QR factorization	
poissCdf()	109	quadratic regression, QuadReg	114 115
poissPdf()	109	QuadReg, quadratic regression	_
polar		quartic regression, QuartReg	115
coordinate, R Pr()	117	QuartReg, quartic regression	116
coordinate, R►Pθ()	117	Quartheg, quartic regression	116
vector display, ▶Polar	110	R	
polyEval(), evaluate polynomial	110		
polynomials		R, radian	188
evaluate, polyEval()	110	R Pr(), polar coordinate	117
random, randPoly()	119	R•Pθ(), polar coordinate	117
PolyRoots()	111	radian, R	188
power of ten, 10^()	190	rand(), random number	118
power regression,		randBin, random number	118
PowerReg 111, 123, 125	, 152	randInt(), random integer	118
power, ^	175	randMat(), random matrix	119
PowerReg, power regression	111	randNorm(), random norm	119
Prgm, define programme	112	random	
prime number test, isPrime()	73	matrix, randMat()	119
probability densiy, normPdf()	102	norm, randNorm()	119

number seed, RandSeed	120	rotate(), rotate	128
polynomial, randPoly()	119	rotate, rotate()	128
random sample	120	round(), round	129
randPoly(), random polynomial	119	round, round()	129
randSamp()	120	row echelon form, ref()	121
RandSeed, random number seed	120	rowAdd(), matrix row addition	129
real(), real	120	rowDim(), matrix row dimension	130
real, real()	120	rowNorm(), matrix row norm	130
reciprocal, ^-1	191	rowSwap(), matrix row swap	130
rectangular-vector display, ▶Rect	120	rref(), reduced row echelon form	130
rectangular x coordinate, P►Rx()	108		
rectangular y coordinate, P*Ry()	108	S	
reduced row echelon form, rref()	130	sec ⁻¹ (), inverse secant	131
ref(), row echelon form	121	sec(), secant	131
RefreshProbeVars	122	sech ⁻¹ (), inverse hyperbolic secant	131
regressions		sech(), hyperbolic secant	132
cubic, CubicReg	32	second derivative	152
exponential, ExpReg	49	template for	5
linear regression, LinRegAx	77	second notation, "	189
linear regression, LinRegBx	76, 78	seq(), sequence	133
logarithmic, LnReg	83	seqGen()	133
Logistic	85	seqn()	134
logistic, Logistic	86	sequence, seq()	-
medium-medium line, MedMed	90	set	J 134
MultReg	94	mode, setMode()	135
power regression,		setMode(), set mode	135
PowerReg _ 111, 123, 1	25, 152	settings, get current	63
quadratic, QuadReg	115	shift(), shift	136
quartic, QuartReg	116	shift, shift()	136
sinusoidal, SinReg	141	sign(), sign	137
remain(), remainder	123	sign, sign()	137
remainder, remain()	123	simult(), simultaneous equations	138
remove		simultaneous equations, simult()	138
void elements from list	38	sin ⁻¹ (), arcsine	139
Request	123	sin(), sine	139
RequestStr	125	sine, sin()	139
result values, statistics	145	sinh ⁻¹ (), hyperbolic arcsine	140
results, statistics	144	sinh(), hyperbolic sine	140
return, Return	126	SinReg, sinusoidal regression	141
Return, return	126	sinusoidal regression, SinReg	141
right(), right	126	SortA, sort ascending	142
right, right()	•	SortD, sort descending	143
rk23(), Runge Kutta function	126		

sorting		indirection, # 187
ascending, SortA	142	left, left() 75
descending, SortD	143	mid-string, mid() 91
spherical vector display, ▶Sphere	143	right, right()46, 70, 126, 164
sqrt(), square root	143	rotate, rotate()
square root		shift, shift()
template for	1	string to expression, expr() 49
square root, v()143	, 184	using to create variable names . 215
standard deviation, stdDev() . 145-146		within, InString
stat.results	144	student-t distribution probability,
stat.values	145	tCdf()
statistics		student-t probability density, tPdf() 154
combinations, nCr()	98	subMat(), submatrix147, 149
factorial, !	182	submatrix, subMat()147, 149
mean, mean()	89	substitution with " " operator 191
median, median()	90	subtract,
one-variable statistics, OneVar	105	sum of interest payments 185
permutations, nPr()	103	sum of principal payments 186
random norm, randNorm()	119	sum(), summation
random number seed,		sum, ∑()
RandSeed	120	template for 5
standard deviation, stdDev		sumIf()
()145-146	, 162	
two-variable results, Two Var	159	
variance, variance()	162	sumSeq()
stdDevPop(), population standard		template for
deviation	145	system of equations (N-equation)
stdDevSamp(), sample standard		template for
deviation	146	template for
Stop command	146	Т
store variable (\rightarrow)	192	
storing		t test, tTest
symbol, &	193	T, transpose
string	20	tan ⁻¹ (), arctangent
dimension, dim()	39	tan(), tangent
length	39	tangent, tan() 149
string(), expression to string	147	tanh ⁻¹ (), hyperbolic arctangent 151
strings		tanh(), hyperbolic tangent 151
append, &	182	tCdf(), studentt distribution
character code, ord()	107	probability 152
character string, char()	20	templates
expression to string, string()	147	absolute value
format, format()	53	definite integral
formatting	53	e exponent 2

exponent	1	two-variable results, Two Var	159
first derivative	5	Two Var, two-variable results	159
fraction	1		
Log	2	U	
matrix (1 × 2)	4	unit vector, unitV()	161
matrix (2 × 1)	4	unitV(), unit vector	161
matrix (2 × 2)	4	unLock, unlock variable or variable	101
matrix (m × n)	4	group	161
nth root	2	unlocking variables and variable	
piecewise function (2-piece)	2	groups	161
piecewise function (N-piece)	2	user-defined functions	35
product, ∏()	5	user-defined functions and	
second derivative	5	programs	37
square root	1	v	
sum, ∑()	5	V	
system of equations (2-		variable	
equation)	3	creating name from a character	
system of equations (N-		string	215
equation)	3	variable and functions	
test for void, isVoid()	74	copying	24
Test_2S, 2-sample F test	56	variables	
Text command	152	clear all single-letter	22
time value of money, Future Value	158	delete, DelVar	38
time value of money, Interest	158	local, Local	84
time value of money, number of		variables, locking and unlocking 63, 8	
payments	158	variance, variance()	162
time value of money, payment	150	varPop()	162
amounttime value of money, present value	158	varSamp(), sample variance	162
tInterval, t confidence interval	159	vectors	
tInterval 2Samp, twosample t	153	cross product, crossP()	30
confidence interval	154	cylindrical vector display,	34
tPdf(), studentt probability density	154	►Cylinddot product, dotP()	34 42
trace()	155	unit, unitV()	
transpose, T	149	void elements	161
Try, error handling command	155	void elements, remove	210
tTest, t test	156	void, test for	38
tTest_2Samp, two-sample t test	157	void, test ioi	74
TVM arguments	159	w	
tvmFV()	158		
tvmI()	158	Wait command	163
tvmN()	158	warnCodes(), Warning codes	164
tvmPmt()	158	warning codes and messages	228
tvmPV()	159	when(), when	164
•••••••••••••••••	133		

when, when()	164
while, While	165
While, while	165
with,	191
within string, inString()	69
Х	
x², square	176
XNOR	182
xor, Boolean exclusive or	165
Z	
zInterval, z confidence interval	166
zInterval_1Prop, one-proportion z	
confidence interval	167
zInterval_2Prop, two-proportion z	
confidence interval	167
zinterval_2Samp, two-sample z	160
confidence interval	168
zTest	168
zTest_1Prop, one-proportion z test .	169
zTest_2Prop, two-proportion z test	170
zTest 2Samp, two-sample z test	170