



TI-*nspire*[™] **Reference Guide**

This guidebook applies to TI-Nspire™ software version 3.0. To obtain the latest version of the documentation, go to education.ti.com/guides.

Important Information

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

License

Please see the complete license installed in **C:\Program Files\TI Education\TI-Nspire**.

© 2006 - 2011 Texas Instruments Incorporated

Contents

Expression templates

Fraction template	1
Exponent template	1
Square root template	1
Nth root template	1
e exponent template	2
Log template	2
Piecewise template (2-piece)	2
Piecewise template (N-piece)	2
System of 2 equations template	3
System of N equations template	3
Absolute value template	3
dd ² mm'ss.ss'' template	3
Matrix template (2 x 2)	3
Matrix template (1 x 2)	4
Matrix template (2 x 1)	4
Matrix template (m x n)	4
Sum template (Σ)	4
Product template (II)	4
First derivative template	5
Second derivative template	5
Definite integral template	5

Alphabetical listing

A

abs()	6
amortTbl()	6
and	6
angle()	7
ANOVA	7
ANOVA2way	8
Ans	9
approx()	10
►approxFraction()	10
approxRational()	10
arccos()	10
arccosh()	10
arccot()	10
arccoth()	11
arccsc()	11
arccsch()	11
arcsec()	11
arcsech()	11
arcsin()	11
arcsinh()	11
arctan()	11
arctanh()	11
augment()	11
avgRC()	12

B

bal()	12
►Base2	12
►Base10	13
►Base16	14
binomCdf()	14
binomPdf()	14

C

ceiling()	14
centralDiff()	15
char()	15
χ^2 2way	15
χ^2 Cdf()	16
χ^2 GOF	16
χ^2 Pdf()	16
ClearAZ	16
ClrErr	17
colAugment()	17
colDim()	17
colNorm()	17
completeSquare()	18
conj()	18
constructMat()	18
CopyVar	18
corrMat()	19
cos()	19
cos ⁻¹ ()	20
cosh()	21
cosh ⁻¹ ()	21
cot()	21
cot ⁻¹ ()	22
coth()	22
coth ⁻¹ ()	22
count()	22
countif()	23
cPolyRoots()	23
crossP()	23
csc()	24
csc ⁻¹ ()	24
csch()	24
csch ⁻¹ ()	24
CubicReg	25
cumulativeSum()	25
Cycle	26
►Cylind	26

D

dbd()	26
►DD	27
►Decimal	27
Define	27
Define LibPriv	28
Define LibPub	28
deltaList()	29
DelVar	29
delVoid()	29
det()	29
diag()	30
dim()	30
Disp	30
►DMS	31
dotP()	31

E

e [^] ()	31
eff()	32

eigVc()	32
eigVl()	32
Else	32
Elseif	33
EndFor	33
EndFunc	33
EndIf	33
EndLoop	33
EndPrgm	33
EndTry	33
EndWhile	33
euler()	34
Exit	34
exp()	35
expr()	35
ExpReg	35

F

factor()	36
FCdf()	36
Fill	36
FiveNumSummary	37
floor()	37
For	38
format()	38
fPart()	38
FPdf()	38
freqTable▶list()	39
frequency()	39
FTest_2Samp	39
Func	40

G

gcd()	40
geomCdf()	41
geomPdf()	41
getDenom()	41
getLangInfo()	41
getLockInfo()	42
getMode()	42
getNum()	43
getType()	43
getVarInfo()	43
Goto	44
▶Grad	44

I

identity()	45
If	45
ifFn()	46
imag()	46
Indirection	47
inString()	47
int()	47
intDiv()	47
interpolate()	48
inv χ^2 ()	48
invF()	48
invNorm()	48
invt()	48
iPart()	49
irr()	49
isPrime()	49

isVoid()	49
----------	----

L

Lbl	50
lcm()	50
left()	50
libShortcut()	51
LinRegBx	51
LinRegMx	52
LinRegtIntervals	52
LinRegtTest	54
linSolve()	55
Δ List()	55
list▶mat()	55
ln()	55
LnReg	56
Local	57
Lock	57
log()	58
Logistic	58
LogisticD	59
Loop	60
LU	60

M

mat▶list()	60
max()	61
mean()	61
median()	61
MedMed	62
mid()	62
min()	63
mirr()	63
mod()	64
mRow()	64
mRowAdd()	64
MultReg	64
MultRegIntervals	65
MultRegTests	65

N

nCr()	66
nDerivative()	67
newList()	67
newMat()	67
nfMax()	67
nfMin()	68
nInt()	68
nom()	68
norm()	68
normCdf()	69
normPdf()	69
not	69
nPr()	69
npv()	70
nSolve()	70

O

OneVar	71
or	72
ord()	72

P

P►Rx()	72
P►Ry()	73
PassErr	73
piecewise()	73
poissCdf()	73
poissPdf()	73
►Polar	74
polyEval()	74
polyRoots()	74
PowerReg	75
Prgm	76
prodSeq()	76
Product (PI)	76
product()	76
propFrac()	77

Q

QR	77
QuadReg	78
QuartReg	78

R

R►Pθ()	79
R►Pr()	79
►Rad	80
rand()	80
randBin()	80
randInt()	80
randMat()	80
randNorm()	80
randPoly()	81
randSamp()	81
RandSeed	81
real()	81
►Rect	81
ref()	82
remain()	83
Request	83
RequestStr	84
Return	84
right()	84
rk23()	85
root()	85
rotate()	85
round()	86
rowAdd()	86
rowDim()	87
rowNorm()	87
rowSwap()	87
rref()	87

S

sec()	88
sec ⁻¹ ()	88
sech()	88
sech ⁻¹ ()	88
seq()	89
seqGen()	89
seqn()	90
setMode()	90
shift()	91
sign()	92
simult()	92

sin()	93
sin ⁻¹ ()	93
sinh()	94
sinh ⁻¹ ()	94
SinReg	95
SortA	95
SortD	96
►Sphere	96
sqrt()	96
stat.results	97
stat.values	98
stDevPop()	98
stDevSamp()	98
Stop	99
Store	99
string()	99
subMat()	99
Sum (Sigma)	99
sum()	99
sumIf()	100
sumSeq()	100
system()	100

T

T (transpose)	100
tan()	101
tan ⁻¹ ()	101
tanh()	102
tanh ⁻¹ ()	102
tCdf()	103
Text	103
Then	103
tInterval	103
tInterval_2Samp	104
tPdf()	104
trace()	104
Try	105
tTest	105
tTest_2Samp	106
tvmFV()	106
tvmI()	107
tvmN()	107
tvmPmt()	107
tvmPV()	107
TwoVar	108

U

unitV()	109
unLock	109

V

varPop()	109
varSamp()	110

W

warnCodes()	110
when()	110
While	111
“With”	111

X

xor	111
-----	-----

Z

zInterval	112
zInterval_1Prop	112
zInterval_2Prop	113
zInterval_2Samp	113
zTest	114
zTest_1Prop	114
zTest_2Prop	115
zTest_2Samp	115

Symbols

+ (add)	116
-(subtract)	116
· (multiply)	117
/ (divide)	117
^ (power)	118
x ² (square)	118
·+ (dot add)	119
·- (dot sub.)	119
·* (dot mult.)	119
·/ (dot divide)	119
·^ (dot power)	119
-(negate)	120
% (percent)	120
= (equal)	121
≠ (not equal)	121
< (less than)	121
≤ (less or equal)	122
> (greater than)	122
≥ (greater or equal)	122
! (factorial)	122
& (append)	122
d() (derivative)	123
∫() (integral)	123
√() (square root)	123
Π() (prodSeq)	124
Σ() (sumSeq)	124
ΣInt()	125

ΣPrn()	125
# (indirection)	126
E (scientific notation)	126
g (gradian)	126
r (radian)	126
° (degree)	127
° , ' , '' (degree/minute/second)	127
∠ (angle)	127
_ (underscore as an empty element)	127
10^()	128
^-1 (reciprocal)	128
("with")	128
→ (store)	129
:= (assign)	129
@ (comment)	129
0b, 0h	130

Empty (void) elements

Calculations involving void elements	131
List arguments containing void elements	131

Shortcuts for entering math expressions

EOS™ (Equation Operating System) hierarchy

Error codes and messages

Warning codes and messages

Texas Instruments Support and Service

TI-Nspire™ Reference Guide

This guide lists the templates, functions, commands, and operators available for evaluating math expressions.

Expression templates

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

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 **ctrl enter** to evaluate the expression.

Fraction template

ctrl **÷** **keys**



Note: See also **/ (divide)**, page 117.

Example:

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

Exponent template

^ **key**



Note: Type the first value, press **^**, and then type the exponent.

To return the cursor to the baseline, press right arrow **►**.

Note: See also **^ (power)**, page 118.

Example:

2^3	8
-------	---

Square root template

ctrl **x²** **keys**



Note: See also **√()** **(square root)**, page 123.

Example:

$\sqrt{4}$	2
$\sqrt{\{9, a, 4\}}$	$\{3, \sqrt{(a)}, 2\}$
$\sqrt{4}$	2
$\sqrt{\{9, 16, 4\}}$	$\{3, 4, 2\}$

Nth root template

ctrl **^** **keys**



Note: See also **root()**, page 85.

Example:

$\sqrt[3]{8}$	2
$\sqrt[3]{\{8, 27, 15\}}$	$\{2, 3, 2.46621\}$

e exponent template**e^x** keys**e**Natural exponential e raised to a power**Note:** See also **e^()**, page 31.

Example:

$$e^1 = 2.71828182846$$

Log template**ctrl** **10^x** **key****log**

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

Note: See also **log()**, page 58.

Example:

$$\log_4(2) = 0.5$$

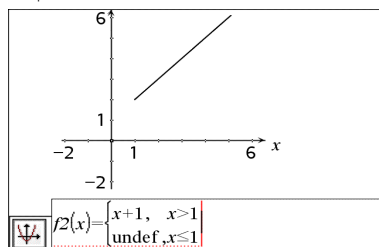
Piecewise template (2-piece)**Catalog** > **10¹⁰**

$$\left\{ \begin{array}{l} () \\ () \end{array} \right.$$

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

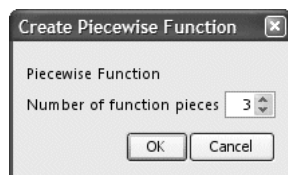
Note: See also **piecewise()**, page 73.

Example:

**Piecewise template (N-piece)****Catalog** > **10¹⁰**Lets you create expressions and conditions for an N -piece piecewise function. Prompts for N .

Example:

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

**Note:** See also **piecewise()**, page 73.

System of 2 equations template

Catalog > 



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 100.

Example:

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

$$\text{solve} \left\{ \begin{array}{l} y=x^2-2 \\ x+2 \cdot y=-1 \end{array} \right\}, x, y$$

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

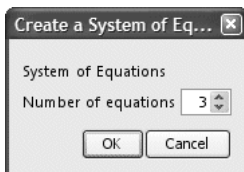
System of N equations template

Catalog > 

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

Example:

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



Note: See also **system()**, page 100.

Absolute value template

Catalog > 



Note: See also **abs()**, page 6.

Example:

$$\left\{ 2, -3, 4, -4^3 \right\} \quad \left\{ 2, 3, 4, 6, 4 \right\}$$

dd°mm'ss.ss" template

Catalog > 



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

Example:

$$30^\circ 15' 10'' \quad 0.528011$$

Matrix template (2 x 2)

Catalog > 



Creates a 2 x 2 matrix.

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 5 \quad \begin{bmatrix} 5 & 10 \\ 15 & 20 \end{bmatrix}$$

Matrix template (1 x 2)Catalog > 

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

Example:

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

Matrix template (2 x 1)Catalog > 

$$\begin{bmatrix} \square \\ \square \end{bmatrix}$$

Example:

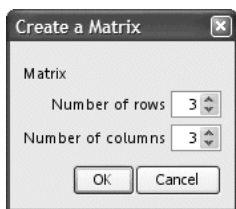
$$\begin{bmatrix} 5 \\ 8 \end{bmatrix} \cdot 0.01 \quad \begin{bmatrix} 0.05 \\ 0.08 \end{bmatrix}$$

Matrix template (m x n)Catalog > 

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

Example:

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

**Note:** If you create a matrix with a large number of rows and columns, it may take a few moments to appear.**Sum template (Σ)**Catalog > 

$$\begin{array}{c} \square \\ \Sigma \\ \square = \square \end{array} \left(\begin{bmatrix} \square \end{bmatrix} \right)$$

Example:

$$\frac{7}{\sum_{n=3}^{} \left(\begin{bmatrix} \square \end{bmatrix} \right)} \quad 25$$

Note: See also $\Sigma()$ ([sumSeq](#)), page 124.**Product template (Π)**Catalog > 

$$\begin{array}{c} \square \\ \Pi \\ \square = \square \end{array} \left(\begin{bmatrix} \square \end{bmatrix} \right)$$

Example:

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

Note: See also $\Pi()$ ([prodSeq](#)), page 124.

First derivative templateCatalog > 

$$\frac{d}{d\boxed{}}\left(\boxed{}\right)$$

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 123.

Example:

$$\frac{d}{dx}\left(|x|\right)|_{x=0}$$

undef

Second derivative templateCatalog > 

$$\frac{d^2}{d\boxed{}^2}\left(\boxed{}\right)$$

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

Note: See also **d()** (**derivative**), page 123.

Example:

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

18

Definite integral templateCatalog > 

$$\int_{\boxed{}}^{\boxed{}}\boxed{}d\boxed{}$$

The definite integral template can be used to calculate the definite integral numerically, using the same method as nint().

183Note: See also **nint()**, page 68.

Example:


$$\int_0^{10} x^2 dx$$

333.333

Alphabetical listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, starting on page 116. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

abs() Catalog > 


abs(Value1) \Rightarrow value
abs(List1) \Rightarrow list
abs(Matrix1) \Rightarrow matrix

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.

$\left \left[\frac{\pi}{2}, \frac{\pi}{3} \right] \right $	{ 1.5708, 1.0472 }
$ 2-3 \cdot i $	3.60555

amortTbl() Catalog > 

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

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

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

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY*, and *PmtAt* are described in the table of TVM arguments, page 107.

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


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

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

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

You can use the output matrix as input for the other amortization functions $\Sigma\mathbf{Int}()$ and $\Sigma\mathbf{Prn}()$, page 125, and **bal()**, page 12.

amortTbl(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	

and Catalog > 

BooleanExpr1 and BooleanExpr2 \Rightarrow Boolean expression
BooleanList1 and BooleanList2 \Rightarrow Boolean list
BooleanMatrix1 and BooleanMatrix2 \Rightarrow Boolean matrix

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

and

Catalog >

Integer1 and Integer2 ⇒ *integer*

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

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

In Hex base mode:

0h7AC36 and 0h3D5F 0h2C16

Important: Zero, not the letter O.

In Bin base mode:

0b100101 and 0b100 0b100

In Dec base mode:

37 and 0b100 4

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

angle()

Catalog >

angle(Value) ⇒ *value*

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

In Degree angle mode:

angle(0+2*i*) 90

In Gradian angle mode:

angle(0+3*i*) 100

In Radian angle mode:

angle(1+i) 0.785398
$$\mathbf{angle}\left(\left\{1+2\cdot i, 3+0\cdot i, 0-4\cdot i\right\}\right)$$

$$\left\{1.10715, 0, -1.5708\right\}$$

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

$$\left\{\frac{\pi}{2}-\tan^{-1}\left(\frac{1}{2}\right), 0, \frac{\pi}{2}\right\}$$
angle(List1) ⇒ *list***angle(Matrix1)** ⇒ *matrix*

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

ANOVA

Catalog >

ANOVA *List1, List2[, List3, ..., List20][, Flag]*

Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the *stat.results* variable. (See page 97.)

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
stat.dfError	Degrees of freedom of the errors

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

ANOVA2way

Catalog > 

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

Computes a two-way analysis of variance for comparing the means of two to 10 populations. A summary of results is stored in the *stat.results* variable. (See page 97.)

LevRow=0 for Block

LevRow=2,3,...,Len-1, for Two Factor, where

Len=length(*List1*)=length(*List2*) = ... = length(*List10*) and
LevRow ∈ {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
stat.SSError	Sum of squares of the errors
stat.MSError	Mean squares for the errors
stat.s	Standard deviation of the error

COLUMN FACTOR Outputs

Output variable	Description
stat.Fcol	F statistic of the column factor

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

ROW FACTOR Outputs

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

INTERACTION Outputs

Output variable	Description
stat.FInteract	F statistic of the interaction
stat.PValInteract	Probability value of the interaction
stat.dfInteract	Degrees of freedom of the interaction
stat.SSInteract	Sum of squares of the interaction
stat.MSInteract	Mean squares for interaction

ERROR Outputs

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

Ans

ctrl **(←)** **keys**

Ans ⇒ *value*

Returns the result of the most recently evaluated expression.

56	56
56+4	60
60+4	64

approx()

Catalog >

approx(*Value!*) \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 .

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

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

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

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

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

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

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

approx(*List!*) \Rightarrow *list***approx**(*Matrix!*) \Rightarrow *matrix*Returns a list or *matrix* where each element has been evaluated to a decimal value, when possible.**approxFraction()**

Catalog >

Value! \blacktriangleright **approxFraction**(*Tol!*) \Rightarrow *value**List!* \blacktriangleright **approxFraction**(*Tol!*) \Rightarrow *list**Matrix!* \blacktriangleright **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**(...).

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

$$0.8333333333333333 \blacktriangleright \text{approxFraction}(5.E-14) \quad \frac{5}{6}$$

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

approxRational()

Catalog >

approxRational(*Value!*, *Tol!*) \Rightarrow *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.

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

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

arccos()See $\cos^{-1}()$, page 20.**arcosh()**See $\cosh^{-1}()$, page 21.**arccot()**See $\cot^{-1}()$, page 22.

arccoth() See coth^{-1} , page 22.

arccsc() See csc^{-1} , page 24.

arccsch() See csch^{-1} , page 24.

arcsec() See sec^{-1} , page 88.

arcsech() See sech^{-1} , page 88.

arcsin() See sin^{-1} , page 93.

arcsinh() See sinh^{-1} , page 94.

arctan() See tan^{-1} , page 101.

arctanh() See tanh^{-1} , page 102.

augment()

Catalog > 

augment(*List1*, *List2*) \Rightarrow *list*

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

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

augment(*Matrix1*, *Matrix2*) \Rightarrow *matrix*

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

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

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

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

avgRC()

Catalog >

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

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

Expr1 can be a user-defined function name (see **Func**).When *Value* is specified, it overrides any prior variable assignment or any current “with” 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.

$x:=2$	2
$\text{avgRC}(x^2-x+2,x)$	3.001
$\text{avgRC}(x^2-x+2,x,1)$	3.1
$\text{avgRC}(x^2-x+2,x,3)$	6

B**bal()**

Catalog >

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

Amortization function that calculates schedule balance after a specified payment.

N, *I*, *PV*, *Pmt*, *FV*, *PpY*, *CpY*, and *PmtAt* are described in the table of TVM arguments, page 107.*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 107.

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

roundValue specifies the number of decimal places for rounding. Default=2.**bal**(*NPmt*,*amortTable*) calculates the balance after payment number *NPmt*, based on amortization table *amortTable*. The *amortTable* argument must be a matrix in the form described under **amortTbl()**, page 6.**Note:** See also **ΣInt()** and **ΣPrn()**, page 125.

$\text{bal}(5,6,5.75,5000,,12,12)$	833.11																												
$\text{tbl}:=\text{amortTbl}(6,6,5.75,5000,,12,12)$																													
<table border="1"><tr><td>0</td><td>0.</td><td>0.</td><td>5000.</td></tr><tr><td>1</td><td>-23.35</td><td>-825.63</td><td>4174.37</td></tr><tr><td>2</td><td>-19.49</td><td>-829.49</td><td>3344.88</td></tr><tr><td>3</td><td>-15.62</td><td>-833.36</td><td>2511.52</td></tr><tr><td>4</td><td>-11.73</td><td>-837.25</td><td>1674.27</td></tr><tr><td>5</td><td>-7.82</td><td>-841.16</td><td>833.11</td></tr><tr><td>6</td><td>-3.89</td><td>-845.09</td><td>-11.98</td></tr></table>	0	0.	0.	5000.	1	-23.35	-825.63	4174.37	2	-19.49	-829.49	3344.88	3	-15.62	-833.36	2511.52	4	-11.73	-837.25	1674.27	5	-7.82	-841.16	833.11	6	-3.89	-845.09	-11.98	
0	0.	0.	5000.																										
1	-23.35	-825.63	4174.37																										
2	-19.49	-829.49	3344.88																										
3	-15.62	-833.36	2511.52																										
4	-11.73	-837.25	1674.27																										
5	-7.82	-841.16	833.11																										
6	-3.89	-845.09	-11.98																										
$\text{bal}(4,\text{tbl})$	1674.27																												

►Base2

Catalog >

Integer1 ►**Base2** ⇒ *integer***Note:** You can insert this operator from the computer keyboard by typing **@>Base2**.Converts *Integer1* to a binary number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

$256 \blacktriangleright \text{Base2}$	0b100000000
$0h1F \blacktriangleright \text{Base2}$	0b11111

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

0b *binaryNumber*
0h *hexadecimalNumber*

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

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

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

-1 is displayed as
0hFFFFFFFFFFFFFFF in Hex base mode
0b111...111 (64 1's) in Binary base mode

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

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

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

2⁶⁴ becomes 0 and is displayed as
0h0 in Hex base mode
0b0 in Binary base mode

-2⁶³ - 1 becomes 2⁶³ - 1 and is displayed as
0h7FFFFFFFFFFFFFFF in Hex base mode
0b111...111 (64 1's) in Binary base mode

►Base10

Integer1 ►Base10 ⇒ *integer*

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

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

0b *binaryNumber*
0h *hexadecimalNumber*

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

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

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

0b10011►Base10	19
0h1F►Base10	31

Base16Catalog > *Integer1* ▶ **Base16** ⇒ *integer***Note:** You can insert this operator from the computer keyboard by typing @>**Base16**.Converts *Integer1* to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.0b *binaryNumber*0h *hexadecimalNumber*

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

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

Without a prefix, *Integer1* is treated as decimal (base 10). The result is displayed in hexadecimal, regardless of the Base mode.If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ▶**Base2**, page 12.

256▶Base16	0h100
0b111100001111▶Base16	0hF0F

binomCdf()Catalog > **binomCdf**(*n,p*) ⇒ *number***binomCdf**(*n,p,lowBound,upBound*) ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists**binomCdf**(*n,p,upBound*) for $P(0 \leq X \leq upBound)$ ⇒ *number* if *upBound* is a number, *list* if *upBound* is a listComputes a cumulative probability for the discrete binomial distribution with *n* number of trials and probability *p* of success on each trial.For $P(X \leq upBound)$, set *lowBound*=0**binomPdf()**Catalog > **binomPdf**(*n,p*) ⇒ *number***binomPdf**(*n,p,XVal*) ⇒ *number* if *XVal* is a number, *list* if *XVal* is a listComputes a probability for the discrete binomial distribution with *n* number of trials and probability *p* of success on each trial.**C****ceiling()**Catalog > **ceiling**(*Value1*) ⇒ *value*

Returns the nearest integer that is ≥ the argument.

The argument can be a real or a complex number.

Note: See also **floor()**.**ceiling**(*List1*) ⇒ *list***ceiling**(*Matrix1*) ⇒ *matrix*

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

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

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

centralDiff()

Catalog >

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

Returns the numerical derivative using the central difference quotient formula.

When *Value* is specified, it overrides any prior variable assignment or any current "with" 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()**.

$$\text{centralDiff}\left(\cos(x), x\right)\Big|_{x=\frac{\pi}{2}} = -1.$$

char()

Catalog >

char(*Integer*) ⇒ *character*

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

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

 χ^2 2way

Catalog >

 χ^2 2way *obsMatrix***chi22way** *obsMatrix*

Computes a χ^2 test for association on the two-way table of counts in the observed matrix *obsMatrix*. A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a matrix, see "Empty (void) elements" on page 131.

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

$\chi^2\text{Cdf}()$ Catalog > 

$\chi^2\text{Cdf}(\text{lowBound}, \text{upBound}, \text{df}) \Rightarrow$ number if *lowBound* and *upBound* are numbers, list if *lowBound* and *upBound* are lists
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 $P(X \leq \text{upBound})$, set *lowBound* = 0.

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

 $\chi^2\text{GOF}$ Catalog > 

$\chi^2\text{GOF}$ *obsList*, *expList*, *df*
chi2GOF *obsList*, *expList*, *df*

Performs a test to confirm that sample data is from a population that conforms to a specified distribution. *obsList* is a list of counts and must contain integers. A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

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

 $\chi^2\text{Pdf}()$ Catalog > 

$\chi^2\text{Pdf}(XVal, df) \Rightarrow$ 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" on page 131.

ClearAZ

Catalog > 

ClearAZ

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

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See **unLock**, page 109.

5	\rightarrow b	5
	b	5
	ClearAZ	Done
	b	"Error: Variable is not defined"



ClrErrCatalog > **ClrErr**

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

For an example of **ClrErr**, see Example 2 under the **Try** command, page 105.

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

Note: See also **PassErr**, page 73, and **Try**, page 105.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

colAugment()Catalog > 

colAugment(*Matrix1*, *Matrix2*) \Rightarrow *matrix*

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

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

colDim()Catalog > 

colDim(*Matrix*) \Rightarrow *expression*

Returns the number of columns contained in *Matrix*.

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

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

colNorm()Catalog > 

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**() .

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

completeSquare()Catalog > **completeSquare**(ExprOrEqn, Var) \Rightarrow expression or equation**completeSquare**(ExprOrEqn, Var^Power) \Rightarrow expression or equation**completeSquare**(ExprOrEqn, Var1, Var2 [...]) \Rightarrow expression or equation**completeSquare**(ExprOrEqn, {Var1, Var2 [...]}) \Rightarrow expression or equationConverts a quadratic polynomial expression of the form $a \cdot x^2 + b \cdot x + c$ into the form $a \cdot (x-h)^2 + k$

- or -

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

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

The Second argument must be a single univariate term or a single univariate term raised to a rational power, for example x , y^2 , or $z^{(1/3)}$.The third and fourth syntax attempt to complete the square with respect to variables $Var1$, $Var2$ [...]).

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

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

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

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

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

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

conj()Catalog > **conj**(Value1) \Rightarrow value**conj**(List1) \Rightarrow list**conj**(Matrix1) \Rightarrow matrix

Returns the complex conjugate of the argument.

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

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

constructMat()Catalog > **constructMat**(Expr, Var1, Var2, numRows, numCols)
 \Rightarrow matrix

Returns a matrix based on the arguments.

Expr is an expression in variables *Var1* and *Var2*. Elements in the resulting matrix are formed by evaluating *Expr* for each incremented value of *Var1* and *Var2*.*Var1* is automatically incremented from **1** through *numRows*. Within each row, *Var2* is incremented from **1** through *numCols*.

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

CopyVarCatalog > **CopyVar** Var1, Var2**CopyVar** Var1., Var2.**CopyVar** Var1, Var2 copies the value of variable *Var1* to variable *Var2*, creating *Var2* if necessary. Variable *Var1* must have a value.If *Var1* is the name of an existing user-defined function, copies the definition of that function to function *Var2*. Function *Var1* must be defined.*Var1* must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.

Define $a(x)=\frac{1}{x}$	Done
---------------------------	------

Define $b(x)=x^2$	Done
-------------------	------

CopyVar a,c: c(4)	$\frac{1}{4}$
-------------------	---------------

CopyVar b,c: c(4)	16
-------------------	----

CopyVar

Catalog > 

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

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

$aa.a:=45$	45																		
$aa.b:=6.78$	6.78																		
$aa.c:=8.9$	8.9																		
getVarInfo()	<table border="1"> <tr> <td>aa.a</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>aa.b</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>aa.c</td> <td>"NUM"</td> <td>"0"</td> </tr> </table>	aa.a	"NUM"	"0"	aa.b	"NUM"	"0"	aa.c	"NUM"	"0"									
aa.a	"NUM"	"0"																	
aa.b	"NUM"	"0"																	
aa.c	"NUM"	"0"																	
CopyVar aa.,bb.	Done																		
getVarInfo()	<table border="1"> <tr> <td>aa.a</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>aa.b</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>aa.c</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>bb.a</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>bb.b</td> <td>"NUM"</td> <td>"0"</td> </tr> <tr> <td>bb.c</td> <td>"NUM"</td> <td>"0"</td> </tr> </table>	aa.a	"NUM"	"0"	aa.b	"NUM"	"0"	aa.c	"NUM"	"0"	bb.a	"NUM"	"0"	bb.b	"NUM"	"0"	bb.c	"NUM"	"0"
aa.a	"NUM"	"0"																	
aa.b	"NUM"	"0"																	
aa.c	"NUM"	"0"																	
bb.a	"NUM"	"0"																	
bb.b	"NUM"	"0"																	
bb.c	"NUM"	"0"																	

corrMat()

Catalog > 

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

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

cos()

 key

cos(*Value1*) \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 $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode temporarily.

In Degree angle mode:

$\cos\left(\left(\frac{\pi}{4}\right)^{\text{r}}$	0.707107
$\cos(45)$	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
$\cos(45^{\circ})$	0.707107

cos() **key****cos**(*squareMatrix1*) \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 *squareMatrix1* (A), the result is calculated by the algorithm:Compute the eigenvalues (λ_i) and eigenvectors (V_i) of A .*squareMatrix1* must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

Form the matrices:

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

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

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

All computations are performed using floating-point arithmetic.

In Radian angle mode:

$$\cos \left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix} \right) = \begin{bmatrix} 0.212493 & 0.205064 & 0.121389 \\ 0.160871 & 0.259042 & 0.037126 \\ 0.248079 & -0.090153 & 0.218972 \end{bmatrix}$$

cos⁻¹() **key****cos⁻¹**(*Value1*) \Rightarrow *value***cos⁻¹**(*List1*) \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**(*squareMatrix1*) \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()**.*squareMatrix1* 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:

$$\cos^{-1} \left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix} \right) = \begin{bmatrix} 1.73485+0.064606 \cdot i & -1.49086+2.10514 \\ -0.725533+1.51594 \cdot i & 0.623491+0.778369 \\ -2.08316+2.63205 \cdot i & 1.79018-1.27182 \cdot i \end{bmatrix}$$

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

cosh()

Catalog >

cosh(*Value1*) ⇒ *value***cosh**(*List1*) ⇒ *list***cosh**(*Value1*) returns the hyperbolic cosine of the argument.**cosh**(*List1*) returns a list of the hyperbolic cosines of each element of *List1*.**cosh**(*squareMatrix1*) ⇒ *squareMatrix*Returns the matrix hyperbolic cosine of *squareMatrix1*. This is not the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

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

$$\cosh^{-1}\{1,2,1,3\} \quad \{0,1.37286,1.76275\}$$

In Radian angle mode:

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

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

cosh⁻¹()

Catalog >

cosh⁻¹(*Value1*) ⇒ *value***cosh⁻¹**(*List1*) ⇒ *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**(*squareMatrix1*) ⇒ *squareMatrix*Returns the matrix inverse hyperbolic cosine of *squareMatrix1*. This is not the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

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

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

In Radian angle mode and In Rectangular Complex Format:

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

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

To see the entire result, press **▲** and then use **◀** and **▶** to move the cursor.**cot()** **key****cot**(*Value1*) ⇒ *value***cot**(*List1*) ⇒ *list*Returns the cotangent of *Value1* or returns a list of the cotangents of all elements in *List1*.**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use **°**, **∇**, or **⊞** to override the angle mode temporarily.

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

$$\cot(\{1,2,1,3\}) \quad \{0.642093,-0.584848,-7.01525\}$$

cot⁻¹()
 **key**
cot⁻¹(Value1) ⇒ value**cot⁻¹(List1)** ⇒ list

Returns the angle whose cotangent is *Value1* or returns a list containing the inverse cotangents 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 **arccot (...)**.

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

$\cot^{-1}(1)$.785398
----------------	---------

coth()
Catalog > 
coth(Value1) ⇒ value**coth(List1)** ⇒ list

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

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

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

coth⁻¹()
Catalog > 
coth⁻¹(Value1) ⇒ value**coth⁻¹(List1)** ⇒ list

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 (...)**.

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

$\coth^{-1}(\{-2.2, 1.6\})$	{ -0.549306, 0.518046, 0.168236 }
-----------------------------	-----------------------------------

count()
Catalog > 
count(Value1orList1 [, Value2orList2 [, ...]]) ⇒ value

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

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

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

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 131.

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

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

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

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

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

Empty (void) elements in the list are ignored. For more information on empty elements, see page 131.

Note: See also **sumif()**, page 100, and **frequency()**, page 39.

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

Counts the number of elements equal to 3.

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

Counts the number of elements equal to "def."

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

Counts 1 and 3.

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

Counts 3, 5, and 7.

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

Counts 1, 3, 7, and 9.

cPolyRoots()Catalog > **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 $y^2 \cdot y + 1$ or $x \cdot x + 2 \cdot x + 1$.The second syntax, **cPolyRoots**(*ListOfCoeffs*), returns a list of complex roots for the coefficients in *ListOfCoeffs*.**Note:** See also **polyRoots()**, page 74.

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

$\text{cPolyRoots}(y^3+1,y)$	$\{-1, 0.5-0.866025i, 0.5+0.866025i\}$
------------------------------	--

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

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

crossP()Catalog > **crossP**(*List1*, *List2*) \Rightarrow *list*Returns the cross product of *List1* and *List2* as a list.*List1* and *List2* must have equal dimension, and the dimension must be either 2 or 3.**crossP**(*Vector1*, *Vector2*) \Rightarrow *vector*Returns a row or column vector (depending on the arguments) that is the cross product of *Vector1* and *Vector2*.Both *Vector1* and *Vector2* must be row vectors, or both must be column vectors. Both vectors must have equal dimension, and the dimension must be either 2 or 3.

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

$\text{crossP}([1 \ 2 \ 3],[4 \ 5 \ 6])$	$[-3 \ 6 \ -3]$
--	-----------------

$\text{crossP}([1 \ 2],[3 \ 4])$	$[0 \ 0 \ -2]$
----------------------------------	----------------

csc() trig key**csc**(*Value1*) ⇒ *value***csc**(*List1*) ⇒ *list*Returns the cosecant of *Value1* or returns a list containing the cosecants of all elements in *List1*.

In Degree angle mode:

$$\text{csc}(45) \quad 1.41421$$

In Gradian angle mode:

$$\text{csc}(50) \quad 1.41421$$

In Radian angle mode:

$$\text{csc}\left(\left\{1, \frac{\pi}{2}, \frac{\pi}{3}\right\}\right) \quad \{1.1884, 1., 1.1547\}$$

csc⁻¹() trig key**csc⁻¹**(*Value1*) ⇒ *value***csc⁻¹**(*List1*) ⇒ *list*Returns the angle whose cosecant is *Value1* or returns a list containing the inverse cosecants of each element of *List1*.**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.**Note:** You can insert this function from the keyboard by typing **arccsc** (...).

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

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

csch() Catalog > **csch**(*Value1*) ⇒ *value***csch**(*List1*) ⇒ *list*Returns the hyperbolic cosecant of *Value1* or returns a list of the hyperbolic cosecants of all elements of *List1*.

$$\text{csch}(3) \quad 0.099822$$

$$\text{csch}(\{1, 2, 1, 4\}) \quad \{0.850918, 0.248641, 0.036644\}$$

csch⁻¹() Catalog > **csch⁻¹**(*Value*) ⇒ *value***csch⁻¹**(*List1*) ⇒ *list*Returns the inverse hyperbolic cosecant of *Value1* or returns a list containing the inverse hyperbolic cosecants of each element of *List1*.**Note:** You can insert this function from the keyboard by typing **arccsch** (...).

$$\text{csch}^{-1}(1) \quad 0.881374$$

$$\text{csch}^{-1}(\{1, 2, 1, 3\}) \quad \{0.881374, 0.459815, 0.32745\}$$

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

Computes the cubic polynomial regression $y = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

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

cumulativeSum()

cumulativeSum(*List1*) \Rightarrow *list*

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

$$\text{cumulativeSum}\{1, 2, 3, 4\} \quad \{1, 3, 6, 10\}$$

cumulativeSum(*Matrix1*) \Rightarrow *matrix*

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

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

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 131.

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

Cycle

Catalog >

Cycle

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

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

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing

instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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

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

Cylind

Catalog >

Vector **Cylind**

Note: You can insert this operator from the computer keyboard by typing $\text{\textcircled{C}}\text{\textcircled{Y}}\text{\textcircled{L}}\text{\textcircled{I}}\text{\textcircled{N}}\text{\textcircled{D}}$.

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
	$[2.82843 \ \angle 0.785398 \ 3.]$

D**dbd()**

Catalog >

dbd($date1, date2$) \Rightarrow *value*

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

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

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

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

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

DDMM.YY (format use commonly in Europe)

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

►DD

Catalog >

Expr1 ►DD ⇒ *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°
$(45^\circ 22' 14.3")$ ►DD	45.3706°
$(\{45^\circ 22' 14.3", 60^\circ 0' 0"\})$ ►DD	$\{45.3706^\circ, 60^\circ\}$

In Gradian angle mode:

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

In Radian angle mode:

(1.5) ►DD	85.9437°
-------------	----------

►Decimal

Catalog >

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.

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

Define

Catalog >



Define *Var* = *Expression***Define** *Function*(*Param1*, *Param2*, ...) = *Expression*Defines the variable *Var* or the user-defined function *Function*.Parameters, such as *Param1*, provide placeholders for passing arguments to the function. When calling a user-defined function, you must supply arguments (for example, values or variables) that correspond to the parameters. When called, the function evaluates *Expression* using the supplied arguments.*Var* and *Function* cannot be the name of a system variable or built-in function or command.**Note:** This form of **Define** is equivalent to executing the expression: *expression* → *Function*(*Param1*, *Param2*).

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

DefineCatalog > **Define Function**(Param1, Param2, ...) = **Func***Block***EndFunc****Define Program**(Param1, Param2, ...) = **Prgm***Block***EndPrgm**

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

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

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Note: See also **Define LibPriv**, page 28, and **Define LibPub**, page 28.

Define $g(x,y)=$ Func

Done

If $x>y$ ThenReturn x

Else

Return y

EndIf

EndFunc

 $g(3,-7)$

3

Define $g(x,y)=$ PrgmIf $x>y$ ThenDisp $x,$ " greater than ", y

Else

Disp $x,$ " not greater than ", y

EndIf

EndPrgm

Done

 $g(3,-7)$

3 greater than -7

Done

Define LibPrivCatalog > **Define LibPriv** *Var* = *Expression***Define LibPriv** *Function*(Param1, Param2, ...) = *Expression***Define LibPriv** *Function*(Param1, Param2, ...) = **Func***Block***EndFunc****Define LibPriv** *Program*(Param1, Param2, ...) = **Prgm***Block***EndPrgm**

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

Note: See also **Define**, page 27, and **Define LibPub**, page 28.

Define LibPubCatalog > **Define LibPub** *Var* = *Expression***Define LibPub** *Function*(Param1, Param2, ...) = *Expression***Define LibPub** *Function*(Param1, Param2, ...) = **Func***Block***EndFunc****Define LibPub** *Program*(Param1, Param2, ...) = **Prgm***Block***EndPrgm**

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

Note: See also **Define**, page 27, and **Define LibPriv**, page 28.

DelVar

Catalog > **DelVar** *Var1*, *Var2* [, *Var3*] ...**DelVar** *Var*.

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 109.

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 <i>a</i>		Done									
$(a+2)^2$	"Error: Variable is not defined"										
<i>aa.a</i> :=45		45									
<i>aa.b</i> :=5.67		5.67									
<i>aa.c</i> :=78.9		78.9									
getVarInfo()	<table border="1"> <tr> <td><i>aa.a</i></td> <td>"NUM"</td> <td>"{"</td> </tr> <tr> <td><i>aa.b</i></td> <td>"NUM"</td> <td>"{"</td> </tr> <tr> <td><i>aa.c</i></td> <td>"NUM"</td> <td>"{"</td> </tr> </table>	<i>aa.a</i>	"NUM"	"{"	<i>aa.b</i>	"NUM"	"{"	<i>aa.c</i>	"NUM"	"{"	
<i>aa.a</i>	"NUM"	"{"									
<i>aa.b</i>	"NUM"	"{"									
<i>aa.c</i>	"NUM"	"{"									
DelVar <i>aa</i> .		Done									
getVarInfo()		"NONE"									

delVoid()

Catalog > **delVoid**(*List1*) \Rightarrow *list*

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

For more information on empty elements, see page 131.



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

det()

Catalog > **det**(*squareMatrix*, *Tolerance*) \Rightarrow *expression*

Returns the determinant of *squareMatrix*.

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

- If you use   or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tolerance* is omitted or not used, the default tolerance is calculated as:

$$5E^{-14} \cdot \mathbf{max}(\mathbf{dim}(\mathbf{squareMatrix})) \cdot \mathbf{rowNorm}(\mathbf{squareMatrix})$$

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

diag()

Catalog >

diag(List) \Rightarrow matrix**diag(rowMatrix)** \Rightarrow matrix**diag(columnMatrix)** \Rightarrow matrix

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

diag(squareMatrix) \Rightarrow rowMatrix

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

squareMatrix must be square.

$\text{diag}\{2, 4, 6\}$	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$
--------------------------	---

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

dim()

Catalog >

dim(List) \Rightarrow integer

Returns the dimension of List.

dim(Matrix) \Rightarrow list

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

dim(String) \Rightarrow integer

Returns the number of characters contained in character string String.

$\text{dim}\{0, 1, 2\}$	3
-------------------------	---

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

$\text{dim}(\text{"Hello"})$	5
------------------------------	---

$\text{dim}(\text{"Hello " \& "there"})$	11
--	----

Disp

Catalog >

Disp [exprOrString1] [, exprOrString2] ...

Displays the arguments in the Calculator history. The arguments are displayed in succession, with thin spaces as separators.

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

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define chars(start,end)=Prgm
  For i,start,end
    Disp i," ",char(i)
  EndFor
EndPrgm
Done
```

$\text{chars}(240,243)$	
	240 ó
	241 ñ
	242 ò
	243 ó
	Done

DMS

Catalog >

Value ►DMS*List* ►DMS*Matrix* ►DMS

Note: You can insert this operator from the computer keyboard by typing $\text{e} \rightarrow \text{DMS}$.

Interprets the argument as an angle and displays the equivalent DMS (DDDDDD°MM'SS.ss'') number. See °, ', '' on page 127 for DMS (degree, minutes, seconds) format.

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

In Degree angle mode:

$\{45.371\}$ ►DMS	45°22'15.6"
$\{\{45.371,60\}\}$ ►DMS	$\{45^\circ 22' 15.6", 60^\circ\}$

dotP()

Catalog >

dotP(*List1*, *List2*) \Rightarrow *expression*

Returns the "dot" product of two lists.

$\text{dotP}(\{1,2\},\{5,6\})$	17
--------------------------------	----

dotP(*Vector1*, *Vector2*) \Rightarrow *expression*

Returns the "dot" product of two vectors.

$\text{dotP}(\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, \begin{bmatrix} 4 & 5 & 6 \end{bmatrix})$	32
---	----

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

E**e^()** **key****e^**(*Value1*) \Rightarrow *value*Returns **e** raised to the *Value1* power.

e^1	2.71828
-------	---------

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

e^{3^2}	8103.08
-----------	---------

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

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

e^(*List1*) \Rightarrow *list*Returns **e** raised to the power of each element in *List1*.

$e^{\{1,1.,.05\}}$	$\{2.71828, 2.71828, 1.64872\}$
--------------------	---------------------------------

e^(*SquareMatrix1*) \Rightarrow *squareMatrix*

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

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

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

eff()

Catalog >

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.

nominalRate must be a real number, and *CpY* must be a real number > 0 .

Note: See also **nom()**, page 68.

 $\text{eff}(5.75, 12)$

5.90398

eigVc()

Catalog >

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 $V = [x_1, x_2, \dots, x_n]$, then:

$$x_1^2 + x_2^2 + \dots + x_n^2 = 1$$

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

In Rectangular Complex Format:

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

$$\text{eigVc}(mI) \begin{bmatrix} -0.800906 & 0.767947 & \\ 0.484029 & 0.573804+0.052258 \cdot i & 0.5738 \\ 0.352512 & 0.262687+0.096286 \cdot i & 0.2626 \end{bmatrix}$$

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

eigVl()

Catalog >

eigVl(*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:

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

$$\text{eigVl}(mI) \{ -4.40941, 2.20471+0.763006 \cdot i, 2.20471-0.763006 \cdot i \}$$

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

Else


See If, page 45.


ElseifCatalog > **If** *BooleanExpr1* **Then***Block1***Elseif** *BooleanExpr2* **Then***Block2*

⋮

Elseif *BooleanExprN* **Then***BlockN***Endif**

⋮

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing 

instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Define $g(x)=\text{Func}$ If $x \leq 5$ Then

Return 5

ElseIf $x > 5$ and $x < 0$ ThenReturn $-x$ ElseIf $x \geq 0$ and $x \neq 10$ ThenReturn x ElseIf $x = 10$ Then

Return 3

EndIf

EndFunc

*Done***EndFor**

See For, page 38.

EndFunc

See Func, page 40.

EndIf

See If, page 45.

EndLoop

See Loop, page 60.

EndPrgm

See Prgm, page 76.

EndTry

See Try, page 105.

EndWhile

See While, page 111.

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

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

with $\text{depVar}(\text{Var0}) = \text{depVar0}$ 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.

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

Differential equation:

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

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

0.	1.	2.	3.	4.
10.	10.9	11.8712	12.9174	14.042

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

System of equations:

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

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

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

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

Exit

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: In the Calculator application on the handheld, you can enter multi-line definitions by pressing $\left[\text{↵} \right]$

instead of $\left[\text{enter} \right]$ at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Function listing:

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

$g()$ 21

exp()

ex key

exp(Value) \Rightarrow valueReturns **e** raised to the *Value* power.**Note:** See also **e** exponent template, page 2.

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.

exp(List) \Rightarrow listReturns **e** raised to the power of each element in *List*.**exp(squareMatrix)** \Rightarrow squareMatrix

Returns the matrix exponential of *squareMatrix*. 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 must be diagonalizable. The result always contains floating-point numbers.

e^1	2.71828
-------	---------

e^{3^2}	8103.08
-----------	---------

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

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

expr()

Catalog >

expr(String) \Rightarrow expressionReturns 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"

$\text{expr}(\text{funcstr})$	Done
-------------------------------	------

$\text{cube}(2)$	8
------------------	---

ExpReg

Catalog >

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

Computes the exponential regression $y = a \cdot (b)^x$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot (b)^x$
stat.a, stat.b	Regression coefficients
stat.r ²	Coefficient of linear determination for transformed data
stat.r	Correlation coefficient for transformed data ($x, \ln(y)$)

Output variable	Description
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 <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

F



factor()

Catalog > 

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

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

To stop a calculation manually,

- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **Handheld:** Hold down the  key and press  repeatedly.

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.

F Cdf()

Catalog > 

F Cdf(*lowBound*,*upBound*,*dfNumer*,*dfDenom*) \Rightarrow number if *lowBound* and *upBound* are numbers, list if *lowBound* and *upBound* are lists

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

Computes the **F** distribution probability between *lowBound* and *upBound* for the specified *dfNumer* (degrees of freedom) and *dfDenom*.

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

Fill

Catalog > 

Fill *Value*, *matrixVar* \Rightarrow matrix

Replaces each element in variable *matrixVar* with *Value*.
matrixVar must already exist.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\rightarrow amatrix$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Fill 1.01, <i>amatrix</i>		Done
<i>amatrix</i>		$\begin{bmatrix} 1.01 & 1.01 \\ 1.01 & 1.01 \end{bmatrix}$

Fill

Catalog >

Fill $Value, listVar \Rightarrow list$ Replaces each element in variable $listVar$ with $Value$. $listVar$ must already exist.

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

FiveNumSummary

Catalog >

FiveNumSummary $X[, [Freq], [Category], [Include]]$ Provides an abbreviated version of the 1-variable statistics on list X . A summary of results is stored in the $stat.results$ variable. (See page 97.) 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 131.

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()

Catalog >

floor($Value1$) $\Rightarrow integer$ 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**($Matrix1$) $\Rightarrow matrix$

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

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

$\text{floor}(-2.14)$	-3.
$\text{floor}\left(\left\{\frac{3}{2}, 0, -5.3\right\}\right)$	$\{1, 0, -6\}$
$\text{floor}\left(\begin{bmatrix} 1.2 & 3.4 \\ 2.5 & 4.8 \end{bmatrix}\right)$	$\begin{bmatrix} 1. & 3. \\ 2. & 4. \end{bmatrix}$



ForCatalog > **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: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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

$g()$ 5050

format()Catalog > **format**(*Value*, *formatString*) \Rightarrow *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.

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.

format(1.234567, "f3")	"1.235"
format(1.234567, "s2")	"1.23E0"
format(1.234567, "e3")	"1.235E0"
format(1.234567, "g3")	"1.235"
format(1234.567, "g3")	"1,234.567"
format(1.234567, "g3,r:")	"1:235"

fPart()Catalog > **fPart**(*Expr1*) \Rightarrow *expression***fPart**(*List1*) \Rightarrow *list***fPart**(*Matrix1*) \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.

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

Fpdf()Catalog > 

Fpdf(*XVal*, *dfNumer*, *dfDenom*) \Rightarrow *number* if *XVal* is a number, *list* if *XVal* is a list

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

freqTable►list()

Catalog >

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

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

List1 can be any valid list.

freqIntegerList must have the same dimension as *List1* and must contain non-negative integer elements only. Each element specifies the number of times the corresponding *List1* element will be repeated in the result list. A value of zero excludes the corresponding *List1* element.

Note: You can insert this function from the computer keyboard by typing **freqTable►list**(...).

Empty (void) elements are ignored. For more information on empty elements, see page 131.

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

frequency()

Catalog >

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

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

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

Each element of the result corresponds to the number of elements from *List1* that are in the range of that bin. Expressed in terms of the **countf()** function, the result is {countf(list, ?≤b(1)), countf(list, b(1)<?≤b(2)), ..., countf(list, b(n-1)<?≤b(n)), countf(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 131.

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

Note: See also **countf()**, page 23.

```
datalist:={1,2,e,3,π,4,5,6,"hello",7}
          {1,2,2.71828,3,3.14159,4,5,6,"hello",7}
frequency(datalist,{2.5,4.5})           {2,4,3}
```

Explanation of result:

2 elements from *Datalist* are ≤2.5

4 elements from *Datalist* are >2.5 and ≤4.5

3 elements from *Datalist* are >4.5

The element "hello" is a string and cannot be placed in any of the defined bins.

FTest_2Samp

Catalog >

FTest_2Samp *List1*,*List2*[,*Freq1*[,*Freq2*[,*Hypoht*]]]**FTest_2Samp** *List1*,*List2*[,*Freq1*[,*Freq2*[,*Hypoht*]]]

(Data list input)

FTest_2Samp *sx1*,*n1*,*sx2*,*n2*[,*Hypoht*]**FTest_2Samp** *sx1*,*n1*,*sx2*,*n2*[,*Hypoht*]

(Summary stats input)

Performs a two-sample **F** test. A summary of results is stored in the *stat.results* variable. (See page 97.)

For H_a : $\sigma_1 > \sigma_2$, set *Hypoht*>0

For H_a : $\sigma_1 \neq \sigma_2$ (default), set *Hypoht* = 0

For H_a : $\sigma_1 < \sigma_2$, set *Hypoht*<0

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

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 <i>List 1</i> and <i>List 2</i>
stat.x1_bar stat.x2_bar	Sample means of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Size of the samples

Func

Catalog >

Func

Block
EndFunc

Template for creating a user-defined function.

Block can be a single statement, a series of statements separated with the ":" character, or a series of statements on separate lines. The function can use the **Return** instruction to return a specific result.

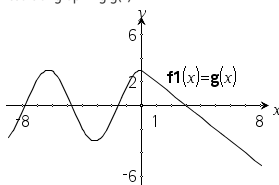
Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of **enter** at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Define a piecewise function:

```
Define g(x)=Func
    If x<0 Then
    Return 3*cos(x)
    Else
    Return 3-x
    EndIf
EndFunc
```

Done

Result of graphing $g(x)$



G

gcd()

Catalog >

$\text{gcd}(\text{Number1}, \text{Number2}) \Rightarrow \text{expression}$

Returns the greatest common divisor of the two arguments. The **gcd** of two fractions is the **gcd** of their numerators divided by the **lcm** of their denominators.

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

$\text{gcd}(\text{List1}, \text{List2}) \Rightarrow \text{list}$

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

$\text{gcd}(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix}$

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

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

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

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

geomCdf()

Catalog >

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

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

geomPdf()

Catalog >

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 .

getDenom()

Catalog >

getDenom($Fraction1$) \Rightarrow value

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

$x:=5; y:=6$	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

getLangInfo()

Catalog >

getLangInfo() \Rightarrow string

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

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"

getLangInfo() "en"

getLockInfo()Catalog > **getLockInfo**(*Var*) ⇒ *value*Returns the current locked/unlocked state of variable *Var*.*value* =0: *Var* is unlocked or does not exist.*value* =1: *Var* is locked and cannot be modified or deleted.See **Lock**, page 57, and **unLock**, page 109.

<i>a</i> :=65	65
Lock <i>a</i>	Done
getLockInfo(<i>a</i>)	1
<i>a</i> :=75	"Error: Variable is locked."
DelVar <i>a</i>	"Error: Variable is locked."
Unlock <i>a</i>	Done
<i>a</i> :=75	75
DelVar <i>a</i>	Done

getMode()Catalog > **getMode**(*ModeNameInteger*) ⇒ *value***getMode**(0) ⇒ *list***getMode**(*ModeNameInteger*) returns a value representing the current setting of the *ModeNameInteger* mode.**getMode**(0) returns a list containing number pairs. Each pair consists of a mode integer and a setting integer.

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

If you save the settings with **getMode**(0) → *var*, you can use **setMode**(*var*) in a function or program to temporarily restore the settings within the execution of the function or program only. See **setMode**(), page 90.

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

Mode Name	Mode Integer	Setting Integers
Display Digits	1	1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12, 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12
Angle	2	1=Radian, 2=Degree, 3=Gradian
Exponential Format	3	1=Normal, 2=Scientific, 3=Engineering
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()

Catalog >

getNum(Fraction1) \Rightarrow *value*

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

$x:=5; y:=6$	6
$\text{getNum}\left(\frac{x+2}{y-3}\right)$	7
$\text{getNum}\left(\frac{2}{7}\right)$	2
$\text{getNum}\left(\frac{1}{x} + \frac{1}{y}\right)$	11

getType()

Catalog >

getType(var) \Rightarrow *string*

Returns a string that indicates the data type of variable *var*.

If *var* has not been defined, returns the string "NONE".

$\{1,2,3\} \rightarrow temp$	$\{1,2,3\}$
$\text{getType}(temp)$	"LIST"
$2.3 \cdot i \rightarrow temp$	$3 \cdot i$
$\text{getType}(temp)$	"EXPR"
$\text{DelVar } temp$	Done
$\text{getType}(temp)$	"NONE"

getVarInfo()

Catalog >

getVarInfo() \Rightarrow *matrix or string***getVarInfo(LibNameString)** \Rightarrow *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.

$\text{getVarInfo}()$	"NONE"												
Define $x=5$	Done												
Lock x	Done												
Define LibPriv $y=\{1,2,3\}$	Done												
Define LibPub $z(x)=3 \cdot x^2 - x$	Done												
$\text{getVarInfo}()$	<table border="1"> <tr> <td>x</td> <td>"NUM"</td> <td>"{}"</td> <td>1</td> </tr> <tr> <td>y</td> <td>"LIST"</td> <td>"LibPriv"</td> <td>0</td> </tr> <tr> <td>z</td> <td>"FUNC"</td> <td>"LibPub"</td> <td>0</td> </tr> </table>	x	"NUM"	"{}"	1	y	"LIST"	"LibPriv"	0	z	"FUNC"	"LibPub"	0
x	"NUM"	"{}"	1										
y	"LIST"	"LibPriv"	0										
z	"FUNC"	"LibPub"	0										
$\text{getVarInfo}(tmp3)$	"Error: Argument must be a string"												
$\text{getVarInfo}("tmp3")$	$[volcyl2 \text{ "NONE" "LibPub" } 0]$												

getVarInfo()

Catalog >

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) reevaluates to a matrix.

This error could also occur when using *Ans* to reevaluate a **getVarInfo()** result.

The system gives the above error because the current version of the software does not support a generalized matrix structure where an element of a matrix can be either a matrix or a list.

$a:=1$	1
$b:=\begin{bmatrix} 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \end{bmatrix}$
$c:=\begin{bmatrix} 1 & 3 & 7 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 & 7 \end{bmatrix}$
$vs:=\text{getVarInfo}()$	$\begin{bmatrix} a & \text{"NUM"} & \text{"["}] & \text{"} & 0 \\ b & \text{"MAT"} & \text{"["}] & \text{"} & 0 \\ c & \text{"MAT"} & \text{"["}] & \text{"} & 0 \end{bmatrix}$
$vs[1]$	$\begin{bmatrix} 1 & \text{"NUM"} & \text{"["}] & \text{"} & 0 \end{bmatrix}$
$vs[1,1]$	1
$vs[2]$	"Error: Invalid list or matrix"
$vs[2,1]$	$\begin{bmatrix} 1 & 2 \end{bmatrix}$

Goto

Catalog >

Goto *labelName*

Transfers control to the label *labelName*.

labelName must be defined in the same function using a **Lbl** instruction.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of **enter** at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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

►Grad

Catalog >

Expr1 ► **Grad** \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:



$\begin{pmatrix} 1.5 \end{pmatrix} \blacktriangleright \text{Grad}$	$\begin{pmatrix} 1.66667 \end{pmatrix}^{\text{g}}$
---	--

In Radian angle mode:

$\begin{pmatrix} 1.5 \end{pmatrix} \blacktriangleright \text{Grad}$	$\begin{pmatrix} 95.493 \end{pmatrix}^{\text{g}}$
---	---

identity()Catalog > **identity(Integer)** \Rightarrow matrixReturns the identity matrix with a dimension of *Integer*.*Integer* must be a positive integer.

identity(4)	1 0 0 0
	0 1 0 0
	0 0 1 0
	0 0 0 1

IfCatalog > **If** *BooleanExpr*
*Statement***If** *BooleanExpr* **Then**
*Block***EndIf**If *BooleanExpr* evaluates to true, executes the single statement *Statement* or the block of statements *Block* before continuing execution.If *BooleanExpr* evaluates to false, continues execution without executing the statement or block of statements.*Block* can be either a single statement or a sequence of statements separated with the ":" character.**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.**If** *BooleanExpr* **Then**
*Block1***Else**
*Block2***EndIf**If *BooleanExpr* evaluates to true, executes *Block1* and then skips *Block2*.If *BooleanExpr* evaluates to false, skips *Block1* but executes *Block2*.*Block1* and *Block2* can be a single statement.

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

g(-2)	4
-------	---

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

g(12)	12
g(-12)	12

If *BooleanExpr1* **Then**
Block1
Elseif *BooleanExpr2* **Then**
Block2
 ⋮
Elseif *BooleanExprN* **Then**
BlockN
Endif

Allows for branching. If *BooleanExpr1* evaluates to true, executes *Block1*. If *BooleanExpr1* evaluates to false, evaluates *BooleanExpr2*, and so on.

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

	<i>Done</i>
$g(-4)$	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*.

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},{8,9,10})
                                     {5,6,10}
```

Test value of **1** is less than 2.5, so its corresponding *Value_If_True* element of **5** is copied to the result list.

Test value of **2** is less than 2.5, so its corresponding *Value_If_True* element of **6** is copied to the result list.

Test value of **3** is not less than 2.5, so its corresponding *Value_If_False* element of **10** is copied to the result list.

```
ifFn({1,2,3}<2.5,4,{8,9,10})        {4,4,10}
```

Value_If_true is a single value and corresponds to any selected position.

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

Value_If_false is not specified. Undef is used.

```
ifFn({2,"a"<2.5,{6,7},{9,10},"err")
                                     {6,"err"}
```

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

imag()

imag(*Value1*) ⇒ *value*

Returns the imaginary part of the argument.

```
imag(1+2i)                          2
```

imag(*List1*) ⇒ *list*

Returns a list of the imaginary parts of the elements.

```
imag({-3,4-i,i})                   {0,-1,1}
```

imag()Catalog > **imag**(*Matrix1*) ⇒ *matrix*

Returns a matrix of the imaginary parts of the elements.

$\text{imag}\left(\begin{bmatrix} 1 & 2 \\ i \cdot 3 & i \cdot 4 \end{bmatrix}\right)$	$\begin{bmatrix} 0 & 0 \\ 3 & 4 \end{bmatrix}$
--	--

Indirection

See #(), page 126.

inString()Catalog > **inString**(*srcString*, *subString* [, *Start*]) ⇒ *integer*Returns the character position in string *srcString* at which the first occurrence of string *subString* begins.*Start*, if included, specifies the character position within *srcString* where the search begins. Default = 1 (the first character of *srcString*).If *srcString* does not contain *subString* or *Start* is > the length of *srcString*, returns zero.

$\text{inString}(\text{"Hello there"}, \text{"the"})$	7
$\text{inString}(\text{"ABCEFG"}, \text{"D"})$	0

int()Catalog > **int**(*Value*) ⇒ *integer***int**(*List1*) ⇒ *list***int**(*Matrix1*) ⇒ *matrix*Returns the greatest integer that is less than or equal to the argument. This function is identical to **floor()**.

The argument can be a real or a complex number.

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

$\text{int}(-2.5)$	-3.
$\text{int}\left(\begin{bmatrix} -1.234 & 0 & 0.37 \end{bmatrix}\right)$	$\begin{bmatrix} -2. & 0 & 0. \end{bmatrix}$

intDiv()Catalog > **intDiv**(*Number1*, *Number2*) ⇒ *integer***intDiv**(*List1*, *List2*) ⇒ *list***intDiv**(*Matrix1*, *Matrix2*) ⇒ *matrix*Returns the signed integer part of (*Number1* ÷ *Number2*).

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

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

interpolate()Catalog > **interpolate**(*xValue*, *xList*, *yList*, *yPrimeList*) \Rightarrow *list*

This function does the following:

Given *xList*, *yList*=**f**(*xList*), and *yPrimeList*=**f'**(*xList*) for some unknown function **f**, a cubic interpolant is used to approximate the function **f** at *xValue*. It is assumed that *xList* is a list of monotonically increasing or decreasing numbers, but this function may return a value even when it is not. This function walks through *xList* looking for an interval [*xList*[*i*], *xList*[*i*+1]] that contains *xValue*. If it finds such an interval, it returns an interpolated value for **f**(*xValue*); otherwise, it returns **undef**.

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

xValue can be a number or a list of numbers.

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

$$rk := rk23(-3y + 6t + 5, y, \{0, 10\}, 5, 1)$$

0.	1.	2.	3.	4.
5.	3.19499	5.00394	6.99957	9.00593

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

Use the interpolate() function to calculate the function values for the *xvalueList*:

```

xvalueList := seq(i, i, 0, 10, 0.5)
              {0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5}
xlist := mat▶list(rk[1])
              {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
ylist := mat▶list(rk[2])
              {5, 3.19499, 5.00394, 6.99957, 9.00593, 10.9978}
yprimelist := -3y + 6t + 5 | y=ylist and t=xlist
              {-10, 1.41503, 1.98819, 2.00129, 1.98221, 2.006}
interpolate(xvalueList, xlist, ylist, yprimelist)
              {5, 2.67062, 3.19499, 4.02782, 5.00394, 6.0001}
  
```

inv χ^2 ()Catalog > **inv χ^2** (*Area*, *df*)**invChi2**(*Area*, *df*)

Computes the Inverse cumulative χ^2 (chi-square) probability function specified by degree of freedom, *df* for a given *Area* under the curve.

invF()Catalog > **invF**(*Area*, *dfNumer*, *dfDenom*)**invF**(*Area*, *dfNumer*, *dfDenom*)

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

invNorm()Catalog > **invNorm**(*Area*, μ , σ)

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

invT()Catalog > **invT**(*Area*, *df*)

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

iPart()

Catalog >

iPart(*Number*) \Rightarrow integer**iPart**(*List1*) \Rightarrow list**iPart**(*Matrix1*) \Rightarrow matrix

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.

$iPart(-1.234)$	-1.
$iPart\left(\left\{\frac{3}{2}, -2.3, 7.003\right\}\right)$	$\{1, -2, 7\}$

irr()

Catalog >

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

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

CF0 is the initial cash flow at time 0; it must be a real number.*CFList* is a list of cash flow amounts after the initial cash flow CF0.*CFFreq* is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.**Note:** See also **mirr()**, page 63.

$list1 := \{6000, -8000, 2000, -3000\}$	$\{6000, -8000, 2000, -3000\}$
$list2 := \{2, 2, 2, 1\}$	$\{2, 2, 2, 1\}$
$irr(5000, list1, list2)$	-4.64484

isPrime()

Catalog >

isPrime(*Number*) \Rightarrow Boolean constant expressionReturns true or false to indicate if *number* is a whole number ≥ 2 that is evenly divisible only by itself and 1.If *Number* exceeds about 306 digits and has no factors ≤ 1021 , **isPrime**(*Number*) displays an error message.**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

$isPrime(5)$	true
$isPrime(6)$	false

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

isVoid()

Catalog >

isVoid(*Var*) \Rightarrow Boolean constant expression**isVoid**(*Expr*) \Rightarrow Boolean constant expression**isVoid**(*List*) \Rightarrow list of Boolean constant expressions

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

For more information on void elements, see page 131.

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

L

Lbl



Catalog > 

Lbl *labelName*

Defines a label with the name *labelName* within a function.

You can use a **Goto** *labelName* instruction to transfer control to the instruction immediately following the label.

labelName must meet the same naming requirements as a variable name.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define g() $\Rightarrow$ Func
    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
```

Done

$g()$ 55

lcm()

Catalog > 

lcm(*Number1*, *Number2*) \Rightarrow *expression*

lcm(*List1*, *List2*) \Rightarrow *list*

lcm(*Matrix1*, *Matrix2*) \Rightarrow *matrix*

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

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

$\text{lcm}(6,9)$ 18

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

left()

Catalog > 

left(*sourceString*[, *Num*]) \Rightarrow *string*

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

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

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

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.

$\text{left}(\text{"Hello"}, 2)$ "He"

$\text{left}(\{1, 3, -2, 4\}, 3)$ $\{1, 3, -2\}$

libShortcut()Catalog > 

libShortcut(*LibNameString*, *ShortcutNameString*
[, *LibPrivFlag*]) ⇒ *list of variables*

Creates a variable group in the current problem that contains references to all the objects in the specified library document *libNameString*. Also adds the group members to the Variables menu. You can then refer to each object using its *ShortcutNameString*.

Set *LibPrivFlag*=**0** to exclude private library objects (default)
Set *LibPrivFlag*=**1** to include private library objects

To copy a variable group, see **CopyVar** on page 18.
To delete a variable group, see **DelVar** on page 29.

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

```
getVarInfo("linalg2")
  {
    clearmat  "FUNC"  "LibPub  "
    gauss1   "PRGM"  "LibPriv "
    gauss2   "FUNC"  "LibPub  "
  }

libShortcut("linalg2", "la")
  {la.clearmat, la.gauss2}

libShortcut("linalg2", "la", 1)
  {la.clearmat, la.gauss1, la.gauss2}
```

LinRegBxCatalog > 

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

Computes the linear regression $y = a + b \cdot x$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

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

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

Computes the linear regression $y = m \cdot x + b$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.RegEqn	Regression Equation: $y = m \cdot x + b$
stat.m, stat.b	Regression coefficients
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories
stat.FreqReg	List of frequencies corresponding to $stat.XReg$ and $stat.YReg$

LinRegIntervals**LinRegIntervals** $X, Y, F, 0, [CLev]]$

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

LinRegIntervals $X, Y, F, 1, Xval, [CLev]]$

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

A summary of results is stored in the *stat.results* variable. (See page 97.)

All the lists must have equal dimension.

X and Y are lists of independent and dependent variables.

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

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.RegEqn	Regression Equation: $a + b \cdot x$
stat.a, stat.b	Regression coefficients
stat.df	Degrees of freedom
stat.r ²	Coefficient of determination
stat.r	Correlation coefficient
stat.Resid	Residuals from the regression

For Slope type only

Output variable	Description
[stat.CLower, stat.CUpper]	Confidence interval for the slope
stat.ME	Confidence interval margin of error
stat.SESlope	Standard error of slope
stat.s	Standard error about the line

For Response type only

Output variable	Description
[stat.CLower, stat.CUpper]	Confidence interval for the mean response
stat.ME	Confidence interval margin of error
stat.SE	Standard error of mean response
[stat.LowerPred, stat.UpperPred]	Prediction interval for a single observation
stat.MEPred	Prediction interval margin of error
stat.SEPred	Standard error for prediction
stat. \hat{y}	$a + b \cdot X_{\text{Val}}$

LinRegTTest $X, Y [, Freq [, Hypoth]]$

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

All the lists must have equal dimension.

X and Y are lists of independent and dependent variables.

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

$Hypoth$ is an optional value specifying one of three alternative hypotheses against which the null hypothesis ($H_0: \beta = \rho = 0$) will be tested.

For $H_a: \beta \neq 0$ and $\rho \neq 0$ (default), set $Hypoth = 0$

For $H_a: \beta < 0$ and $\rho < 0$, set $Hypoth < 0$

For $H_a: \beta > 0$ and $\rho > 0$, set $Hypoth > 0$

A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

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()

Catalog >

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

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

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

$$\text{linSolve}\left(\left\{\begin{array}{l} \text{apple}+4\text{pear}=23 \\ 5\text{apple}-\text{pear}=17 \end{array}\right\}, \{\text{apple},\text{pear}\}\right)$$

$$\left\{\begin{array}{l} 13 \\ 3 \end{array}, \begin{array}{l} 14 \\ 3 \end{array}\right\}$$

$$\text{linSolve}\left(\left\{\begin{array}{l} \text{apple}\cdot 4+\frac{\text{pear}}{3}=14 \\ -\text{apple}+\text{pear}=6 \end{array}\right\}, \{\text{apple},\text{pear}\}\right)$$

$$\left\{\begin{array}{l} 36 \\ 13 \end{array}, \begin{array}{l} 114 \\ 13 \end{array}\right\}$$

ΔList()

Catalog >

ΔList(*List1*) ⇒ *list***Note:** You can insert this function from the keyboard by typing **ΔList**(...).Returns a list containing the differences between consecutive elements in *List1*. Each element of *List1* is subtracted from the next element of *List1*. The resulting list is always one element shorter than the original *List1*.

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

list▶mat()

Catalog >

list▶mat(*List* [, *elementsPerRow*]) ⇒ *matrix*Returns a matrix filled row-by-row with the elements from *List*.*elementsPerRow*, if included, specifies the number of elements per row. Default is the number of elements in *List* (one row).If *List* does not fill the resulting matrix, zeros are added.**Note:** You can insert this function from the computer keyboard by typing **list▶mat**(...).

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

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

ln() **keys****ln**(*Value1*) ⇒ *value***ln**(*List1*) ⇒ *list*

Returns the natural logarithm of the argument.

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

$$\ln(2.) \quad 0.693147$$

If complex format mode is Real:

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

"Error: Non-real calculation"

If complex format mode is Rectangular:

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

ln()ctrl e^x keys**ln**(*squareMatrix1*) ⇒ *squareMatrix*

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

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

In Radian angle mode and Rectangular complex format:

$$\ln \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} = \begin{bmatrix} 1.83145+1.73485 \cdot i & 0.009193-1.49086 \\ 0.448761-0.725533 \cdot i & 1.06491+0.623491 \cdot i \\ -0.266891-2.08316 \cdot i & 1.12436+1.79018 \cdot i \end{bmatrix}$$

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

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

Computes the logarithmic regression $y = a + b \cdot \ln(x)$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.


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

LocalCatalog > **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: In the Calculator application on the handheld, you can enter multi-line definitions by pressing 

instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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

Done

rollcount() 16

rollcount() 3

LockCatalog > **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 *tm*.

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

See **unLock**, page 109, and **getLockInfo()**, page 42.

a:=65 65

Lock a Done

getLockInfo(a) 1

a:=75 "Error: Variable is locked."

DelVar a "Error: Variable is locked."

Unlock a Done

a:=75 75

DelVar a Done

log()ctrl 10^x keys**log(Value1[,Value2])** ⇒ *value***log(List1[,Value2])** ⇒ *list*Returns the base-*Value2* logarithm of the first argument.**Note:** See also **Log template**, page 2.For a list, returns the base-*Value2* logarithm of the elements.

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

$$\log_{10} \left(\begin{matrix} 2. \\ \end{matrix} \right) \quad 0.30103$$

$$\log_4 \left(\begin{matrix} 2. \\ \end{matrix} \right) \quad 0.5$$

$$\log_3 \left(\begin{matrix} 10 \\ \end{matrix} \right) - \log_3 \left(\begin{matrix} 5 \\ \end{matrix} \right) \quad 0.63093$$

If complex format mode is Real:

$$\log_{10} \left(\left\{ \begin{matrix} -3, 1.2, 5 \end{matrix} \right\} \right)$$

"Error: Non-real calculation"

If complex format mode is Rectangular:

$$\log_{10} \left(\left\{ \begin{matrix} -3, 1.2, 5 \end{matrix} \right\} \right)$$

$$\{ 0.477121 + 1.36438 \cdot i, 0.079181, 0.69897 \}$$

In Radian angle mode and Rectangular complex format:

$$\log_{10} \left(\begin{matrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{matrix} \right)$$

$$\begin{bmatrix} 0.795387 + 0.753438 \cdot i & 0.003993 - 0.6474 \cdot i \\ 0.194895 - 0.315095 \cdot i & 0.462485 + 0.2707 \cdot i \\ -0.115909 - 0.904706 \cdot i & 0.488304 + 0.7774 \cdot i \end{bmatrix}$$

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

log(squareMatrix1[,Value1]) ⇒ *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 diagonalizable. The result always contains floating-point numbers.

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

LogisticCatalog > **Logistic** *X*, *Y*, [*Freq*] [, *Category*, *Include*]Computes the logistic regression $y = (d/(1+a \cdot e^{-bx}))$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 97.)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" on page 131.

Output variable	Description
stat.RegEqn	Regression equation: $c/(1+a \cdot e^{-bx})$
stat.a, stat.b, stat.c	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

LogisticD

Catalog > 

LogisticD *X*, *Y* [, [*Iterations*], [*Freq*] [, *Category*, *Include*]

Computes the logistic regression $y = c/(1+a \cdot e^{-bx})+d$ on lists *X* and *Y* with frequency *Freq*, using a specified number of *Iterations*. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.RegEqn	Regression equation: $c/(1+a \cdot e^{-bx})+d$
stat.a, stat.b, stat.c, stat.d	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

Loop

Catalog >

Loop
Block
EndLoop

Repeatedly executes the statements in *Block*. Note that the loop will be executed endlessly, unless a **Goto** or **Exit** instruction is executed within *Block*.

Block is a sequence of statements separated with the ":" character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define rollcount() $\Rightarrow$ Func
  Local i
  1  $\rightarrow$  i
  Loop
  If randInt(1,6) $\Rightarrow$ randInt(1,6)
  Goto end
  i+1  $\rightarrow$  i
EndLoop
Lbl end
Return i
EndFunc
```

Done

rollcount()	16
rollcount()	3

LU

Catalog >

LU *Matrix, lMatrix, uMatrix, pMatrix, Tol]*

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

$$lMatrix \cdot uMatrix = pMatrix \cdot matrix$$

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

The LU factorization algorithm uses partial pivoting with row interchanges.

$\begin{bmatrix} 6 & 12 & 18 \\ 5 & 14 & 31 \\ 3 & 8 & 18 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 6 & 12 & 18 \\ 5 & 14 & 31 \\ 3 & 8 & 18 \end{bmatrix}$
---	--

LU *m1, lower, upper, perm* Done

<i>lower</i>	$\begin{bmatrix} 1 & 0 & 0 \\ 5 & 1 & 0 \\ 6 & & \end{bmatrix}$
	$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & \end{bmatrix}$

<i>upper</i>	$\begin{bmatrix} 6 & 12 & 18 \\ 0 & 4 & 16 \\ 0 & 0 & 1 \end{bmatrix}$
--------------	--

<i>perm</i>	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
-------------	---

M**mat►list()**

Catalog >

mat►list(*Matrix*) \Rightarrow *list*

Returns a list filled with the elements in *Matrix*. The elements are copied from *Matrix* row by row.

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

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

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

mat►list (<i>m1</i>)	{1,2,3,4,5,6}
-------------------------------	---------------

max()Catalog > **max**(Value1, Value2) \Rightarrow expression**max**(List1, List2) \Rightarrow list**max**(Matrix1, Matrix2) \Rightarrow 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 expressionReturns the maximum element in *List*.**max**(Matrix1) \Rightarrow matrixReturns a row vector containing the maximum element of each column in *Matrix1*.

Empty (void) elements are ignored. For more information on empty elements, see page 131.

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

$\max(2.3, 1.4)$	2.3
$\max(\{1, 2\}, \{-4, 3\})$	$\{1, 3\}$

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

$\max\left(\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}\right)$	$\begin{bmatrix} 1 & 0 & 7 \end{bmatrix}$
---	---

mean()Catalog > **mean**(List, freqList) \Rightarrow expressionReturns the mean of the elements in *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**mean**(Matrix1, freqMatrix) \Rightarrow matrixReturns a row vector of the means of all the columns in *Matrix1*.Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix1*.

Empty (void) elements are ignored. For more information on empty elements, see page 131.

$\text{mean}(\{0.2, 0, 1, -0.3, 0.4\})$	0.26
---	------

$\text{mean}(\{1, 2, 3\}, \{3, 2, 1\})$	$\frac{5}{3}$
---	---------------

In Rectangular vector format:

$\text{mean}\left(\begin{bmatrix} 0.2 & 0 \\ -1 & 3 \\ 0.4 & -0.5 \end{bmatrix}\right)$	$\begin{bmatrix} -0.133333 & 0.833333 \end{bmatrix}$
---	--

$\text{mean}\left(\begin{bmatrix} \frac{1}{5} & 0 \\ -1 & 3 \\ \frac{2}{5} & \frac{-1}{2} \end{bmatrix}\right)$	$\begin{bmatrix} \frac{-2}{15} & \frac{5}{6} \end{bmatrix}$
---	---

$\text{mean}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \begin{bmatrix} 5 & 3 \\ 4 & 1 \\ 6 & 2 \end{bmatrix}\right)$	$\begin{bmatrix} \frac{47}{15} & \frac{11}{3} \end{bmatrix}$
--	--

median()Catalog > **median**(List, freqList) \Rightarrow expressionReturns the median of the elements in *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.

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

median()Catalog > **median**(*MatrixI*, *freqMatrix*) ⇒ *matrix*Returns a row vector containing the medians of the columns in *MatrixI*.Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *MatrixI*.**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 131.

$$\text{median} \begin{pmatrix} 0.2 & 0 \\ 1 & -0.3 \\ 0.4 & -0.5 \end{pmatrix} \quad [0.4 \quad -0.3]$$

MedMedCatalog > **MedMed** *X*, *Y* [, *Freq*] [, *Category*, *Include*]Computes the median-median line $y = (m \cdot x + b)$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 97.)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" on page 131.

Output variable	Description
stat.RegEqn	Median-median line equation: $m \cdot x + b$
stat.m, stat.b	Model coefficients
stat.Resid	Residuals from the median-median line
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

mid()Catalog > **mid**(*sourceString*, *Start*, *Count*) ⇒ *string*Returns *Count* characters from character string *sourceString*, beginning with character number *Start*.If *Count* is omitted or is greater than the dimension of *sourceString*, returns all characters from *sourceString*, beginning with character number *Start*.*Count* must be ≥ 0. If *Count* = 0, returns an empty string.

$$\begin{array}{ll} \text{mid}(\text{"Hello there"}, 2) & \text{"ello there"} \\ \text{mid}(\text{"Hello there"}, 7, 3) & \text{"the"} \\ \text{mid}(\text{"Hello there"}, 1, 5) & \text{"Hello"} \\ \text{mid}(\text{"Hello there"}, 1, 0) & \text{""} \end{array}$$

mid()Catalog > **mid**(sourceList, Start[, Count]) \Rightarrow listReturns *Count* elements from *sourceList*, beginning with element number *Start*.If *Count* is omitted or is greater than the dimension of *sourceList*, returns all elements from *sourceList*, beginning with element number *Start*.*Count* must be ≥ 0 . If *Count* = 0, returns an empty list.**mid**(sourceStringList, Start[, Count]) \Rightarrow listReturns *Count* strings from the list of strings *sourceStringList*, beginning with element number *Start*.

$$\text{mid}(\{9,8,7,6\},3) \quad \{7,6\}$$

$$\text{mid}(\{9,8,7,6\},2,2) \quad \{8,7\}$$

$$\text{mid}(\{9,8,7,6\},1,2) \quad \{9,8\}$$

$$\text{mid}(\{9,8,7,6\},1,0) \quad \{\}$$

$$\text{mid}(\{"A","B","C","D"\},2,2) \quad \{"B","C"\}$$

min()Catalog > **min**(Value1, Value2) \Rightarrow expression**min**(List1, List2) \Rightarrow list**min**(Matrix1, Matrix2) \Rightarrow 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 expressionReturns the minimum element of *List*.**min**(Matrix1) \Rightarrow matrixReturns a row vector containing the minimum element of each column in *Matrix1*.**Note:** See also **max()**.

$$\text{min}(2.3,1.4) \quad 1.4$$

$$\text{min}(\{1,2\},\{-4,3\}) \quad \{-4,2\}$$

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

$$\text{min}\left(\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}\right) \quad [-4 \ -3 \ 0.3]$$

mirr()Catalog > **mirr**(financeRate, reinvestRate, CF0, CFList[, CFFreq])

Financial function that returns the modified internal rate of return of an investment.

financeRate is the interest rate that you pay on the cash flow amounts.*reinvestRate* is the interest rate at which the cash flows are reinvested.*CF0* is the initial cash flow at time 0; it must be a real number.*CFList* is a list of cash flow amounts after the initial cash flow *CF0*.*CFFreq* is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.**Note:** See also **irr()**, page 49.

$$\text{list1} := \{6000, -8000, 2000, -3000\} \quad \{6000, -8000, 2000, -3000\}$$

$$\text{list2} := \{2, 2, 2, 1\} \quad \{2, 2, 2, 1\}$$

$$\text{mirr}(4.65, 12, 5000, \text{list1}, \text{list2}) \quad 13.41608607$$

mod()Catalog > **mod**(*Value1*, *Value2*) \Rightarrow *expression***mod**(*List1*, *List2*) \Rightarrow *list***mod**(*Matrix1*, *Matrix2*) \Rightarrow *matrix*

Returns the first argument modulo the second argument as defined by the identities:

$$\text{mod}(x,0) = x$$

$$\text{mod}(x,y) = x - y \text{ floor}(x/y)$$

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

If the arguments are two lists or two matrices, returns a list or matrix containing the modulo of each pair of corresponding elements.

Note: See also **remain()**, page 83

$\text{mod}(7,0)$	7
$\text{mod}(7,3)$	1
$\text{mod}(-7,3)$	2
$\text{mod}(7,-3)$	-2
$\text{mod}(-7,-3)$	-1
$\text{mod}(\{12,-14,16\},\{9,7,-5\})$	$\{3,0,-4\}$

mRow()Catalog > **mRow**(*Value*, *Matrix1*, *Index*) \Rightarrow *matrix*Returns a copy of *Matrix1* with each element in row *Index* of *Matrix1* multiplied by *Value*.

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

mRowAdd()Catalog > **mRowAdd**(*Value*, *Matrix1*, *Index1*, *Index2*) \Rightarrow *matrix*Returns a copy of *Matrix1* with each element in row *Index2* of *Matrix1* replaced with:

$$\text{Value} \cdot \text{row } \text{Index1} + \text{row } \text{Index2}$$

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

MultRegCatalog > **MultReg** *Y*, *X1*[, *X2*[, *X3*, ..., [, *X10*]]]Calculates multiple linear regression of list *Y* on lists *X1*, *X2*, ..., *X10*. A summary of results is stored in the *stat.results* variable. (See page 97.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.RegEqn	Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots$
stat.b0, stat.b1, ...	Regression coefficients
stat.R ²	Coefficient of multiple determination
stat. \hat{y} List	\hat{y} List = $b_0 + b_1 \cdot x_1 + \dots$
stat.Resid	Residuals from the regression

MultRegIntervals $Y, X1[,X2[,X3,...[,X10]]], XValList[,CLeve]$

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

A summary of results is stored in the *stat.results* variable. (See page 97.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (void) elements” on page 131.

Output variable	Description
stat.RegEqn	Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots$
stat. \hat{y}	A point estimate: $\hat{y} = b_0 + b_1 \cdot x_1 + \dots$ for <i>XValList</i>
stat.dfError	Error degrees of freedom
stat.CLower, stat.CUpper	Confidence interval for a mean response
stat.ME	Confidence interval margin of error
stat.SE	Standard error of mean response
stat.LowerPred, stat.UpperrPred	Prediction interval for a single observation
stat.MEPred	Prediction interval margin of error
stat.SEPred	Standard error for prediction
stat.bList	List of regression coefficients, $\{b_0, b_1, b_2, \dots\}$
stat.Resid	Residuals from the regression

MultRegTests

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

Multiple linear regression test computes a multiple linear regression on the given data and provides the global F test statistic and t test statistics for the coefficients.

A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see “Empty (void) elements” on page 131.

Outputs

Output variable	Description
stat.RegEqn	Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots$
stat.F	Global F test statistic
stat.PVal	P-value associated with global F statistic
stat.R ²	Coefficient of multiple determination

Output variable	Description
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	{b ₀ , b ₁ , ...} 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	\hat{y} List = b ₀ +b ₁ ·x ₁ + . . .
stat.Resid	Residuals from the regression
stat.sResid	Standardized residuals; obtained by dividing a residual by its standard deviation
stat.CookDist	Cook's distance; measure of the influence of an observation based on the residual and leverage
stat.Leverage	Measure of how far the values of the independent variable are from their mean values

N

nCr()

Catalog > 

nCr(Value1, Value2) ⇒ *expression*

For integer *Value1* and *Value2* with *Value1* ≥ *Value2* ≥ 0, **nCr()** is the number of combinations of *Value1* things taken *Value2* at a time. (This is also known as a binomial coefficient.)

$$\frac{nCr(z, 3)|_{z=5}}{\quad\quad\quad} \quad 10$$

$$\frac{nCr(z, 3)|_{z=6}}{\quad\quad\quad} \quad 20$$

nCr(Value, 0) ⇒ 1

nCr(Value, negInteger) ⇒ 0

nCr(Value, posInteger) ⇒ *Value* · (*Value* - 1) · ...
(*Value* - *posInteger* + 1)!

nCr(Value, nonInteger) ⇒ *expression!*
((*Value* - *nonInteger*)! · *nonInteger*!)

nCr(List1, List2) ⇒ *list*

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

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

nCr(Matrix1, Matrix2) ⇒ *matrix*

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

$$\frac{nCr\left(\begin{bmatrix} 6 & 5 \\ 4 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}\right)}{\quad\quad\quad} \quad \begin{bmatrix} 15 & 10 \\ 6 & 3 \end{bmatrix}$$

nDerivative()

Catalog >

nDerivative(*Expr1*, *Var*=*Value*, *Order*) ⇒ *value***nDerivative**(*Expr1*, *Var*[, *Order*]) | *Var*=*Value* ⇒ *value*

Returns the numerical derivative calculated using auto differentiation methods.

When *Value* is specified, it overrides any prior variable assignment or any current "with" 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**.**Note:** The **nDerivative()** 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, **nDerivative()** 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()**.

$nDerivative(x , x=1)$	1
-------------------------	---

$nDerivative(x x , x=0)$	undef
---------------------------	-------

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

$nDerivative\left(x \cdot (x^2 + x)^{\frac{1}{3}}, x, 1\right) x=0$	undef
---	-------

$centralDiff\left(x \cdot (x^2 + x)^{\frac{1}{3}}, x\right) x=0$	0.000033
--	----------

newList()

Catalog >

newList(*numElements*) ⇒ *list*Returns a list with a dimension of *numElements*. Each element is zero.

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

newMat()

Catalog >

newMat(*numRows*, *numColumns*) ⇒ *matrix*Returns a matrix of zeros with the dimension *numRows* by *numColumns*.

$newMat(2,3)$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
---------------	--

nfMax()

Catalog >

nfMax(*Expr*, *Var*) ⇒ *value***nfMax**(*Expr*, *Var*, *lowBound*) ⇒ *value***nfMax**(*Expr*, *Var*, *lowBound*, *upBound*) ⇒ *value***nfMax**(*Expr*, *Var*) | *lowBound*<*Var*<*upBound* ⇒ *value*Returns a candidate numerical value of variable *Var* where the local maximum of *Expr* occurs.If you supply *lowBound* and *upBound*, the function looks between those values for the local maximum.

$nfMax(-x^2 - 2 \cdot x - 1, x)$	-1.
----------------------------------	-----

$nfMax(0.5 \cdot x^3 - x - 2, x, -5, 5)$	-0.816497
--	-----------

nfMin()Catalog > **nfMin**(*Expr*, *Var*) ⇒ *value***nfMin**(*Expr*, *Var*, *lowBound*) ⇒ *value***nfMin**(*Expr*, *Var*, *lowBound*, *upBound*) ⇒ *value***nfMin**(*Expr*, *Var*) | *lowBound* < *Var* < *upBound* ⇒ *value*Returns a candidate numerical value of variable *Var* where the local minimum of *Expr* occurs.If you supply *lowBound* and *upBound*, the function looks between those values for the local minimum.

$\text{nfMin}(x^2+2\cdot x+5,x)$	-1.
----------------------------------	-----

$\text{nfMin}(0.5\cdot x^3-x-2,x,-5,5)$	0.816497
---	----------

nInt()Catalog > **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 **nInt()** returns an approximation of $\int(\text{Expr1}, \text{Var}, \text{Lower}, \text{Upper})$. This approximation is a weighted average of some sample values of the integrand in the interval *Lower* < *Var* < *Upper*.

The goal is six significant digits. The adaptive algorithm terminates when it seems likely that the goal has been achieved, or when it seems unlikely that additional samples will yield a worthwhile improvement.

A warning is displayed ("Questionable accuracy") when it seems that the goal has not been achieved.

Nest **nInt()** to do multiple numeric integration. Integration limits can depend on integration variables outside them.

$\text{nInt}(e^{-x^2}, x, -1, 1)$	1.49365
-----------------------------------	---------

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

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

nom()Catalog > **nom**(*effectiveRate*, *CpY*) ⇒ *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 32.

$\text{nom}(5.90398, 12)$	5.75
---------------------------	------

norm()Catalog > **norm**(*Matrix*) ⇒ *expression***norm**(*Vector*) ⇒ *expression*

Returns the Frobenius norm.

$\text{norm}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\right)$	5.47723
--	---------

$\text{norm}\left(\begin{bmatrix} 1 & 2 \end{bmatrix}\right)$	2.23607
---	---------

$\text{norm}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$	2.23607
--	---------

normCdf()

Catalog >

normCdf(*lowBound*,*upBound*[,*μ*],[*σ*]) ⇒ *number* if *lowBound* and *upBound* are numbers, *list* if *lowBound* and *upBound* are lists

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

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

normPdf()

Catalog >

normPdf(*XVal*[,*μ*],[*σ*]) ⇒ *number* if *XVal* is a number, *list* if *XVal* is a list

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

not

Catalog >

not *BooleanExpr* ⇒ *Boolean expression*

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

not *Integer1* ⇒ *integer*

Returns the one's complement of a real integer. Internally, *Integer1* is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one's complement. Results are displayed according to the Base mode.

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

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ►**Base2**, page 12.

not (2≥3)	true
not 0hB0►Base16	0hFFFFFFFFFFFFFFF4F
not not 2	2

In Hex base mode:

└─**Important:** Zero, not the letter O.

not 0h7AC36	0hFFFFFFFFFFFF853C9
-------------	---------------------

In Bin base mode:

0b100101►Base10	37
not 0b100101	0b11111111111111111111111111111111
not 0b100101►Base10	-38

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

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

nPr()

Catalog >

nPr(*Value1*, *Value2*) ⇒ *expression*

For integer *Value1* and *Value2* with $Value1 \geq Value2 \geq 0$, **nPr**() is the number of permutations of *Value1* things taken *Value2* at a time.

nPr(*Value*, 0) ⇒ 1

nPr(*Value*, *negInteger*) ⇒ $1 / ((Value+1) \cdot (Value+2) \cdot \dots \cdot (Value-negInteger))$

nPr(*Value*, *posInteger*) ⇒ $Value \cdot (Value-1) \cdot \dots \cdot (Value-posInteger+1)$

nPr(*Value*, *nonInteger*) ⇒ $Value! / (Value-nonInteger)!$

nPr(*List1*, *List2*) ⇒ *list*

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

nPr(z,3)z=5	60
nPr(z,3)z=6	120
nPr({5,4,3},{2,4,2})	{20,24,6}
nPr($\begin{bmatrix} 6 & 5 \\ 4 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix})$	$\begin{bmatrix} 30 & 20 \\ 12 & 6 \end{bmatrix}$
nPr({5,4,3},{2,4,2})	{20,24,6}

nPr()

Catalog >

nPr(Matrix1, Matrix2) \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.

$nPr\left(\begin{bmatrix} 6 & 5 \\ 4 & 3 \end{bmatrix}, \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}\right)$	$\begin{bmatrix} 30 & 20 \\ 12 & 6 \end{bmatrix}$
--	---

npv()

Catalog >

npv(InterestRate,CFO,CFList,CFFreq)

Financial function that calculates net present value; the sum of the present values for the cash inflows and outflows. A positive result for npv indicates a profitable investment.

InterestRate is the rate by which to discount the cash flows (the cost of money) over one period.

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

CFList is a list of cash flow amounts after the initial cash flow *CF0*.

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.

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

nSolve()

Catalog >

nSolve(Equation,Var[=Guess]) \Rightarrow number or error_string**nSolve**(Equation,Var[=Guess],lowBound) \Rightarrow number or error_string**nSolve**(Equation,Var[=Guess],lowBound,upBound) \Rightarrow number or error_string**nSolve**(Equation,Var[=Guess]) | lowBound<Var<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."

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

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

$nSolve(x^2+5\cdot x-25=9,x) x<0$	-8.84429
$nSolve\left(\frac{(1+r)^{24}-1}{r}=26,r\right) r>0 \text{ and } r<0.25$	0.006886
$nSolve(x^2=-1,x)$	"No solution found"

O

OneVar

Catalog > 

OneVar [**1**,] X_1 [*Freq*][*Category*,*Include*]

OneVar [n_1] X_1 , X_2 [X_3],...[, X_{20}]]

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the *stat.results* variable. (See page 97.)

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

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

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

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

An empty (void) element in any of the lists X , *Freq*, or *Category* results in a void for the corresponding element of all those lists. An empty element in any of the lists X_1 through X_{20} results in a void for the corresponding element of all those lists. For more information on empty elements, see page 131.

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

or

Catalog >

BooleanExpr1 or BooleanExpr2
⇒ *Boolean expression*

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

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See **xor**.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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 12.

Note: See **xor**.

ord()

Catalog >

ord(String) ⇒ *integer*
ord(List1) ⇒ *list*

Returns the numeric code of the first character in character string *String*, or a list of the first characters of each list element.

P

P►Rx()

Catalog >

P►Rx(rExpr, θExpr) ⇒ *expression*
P►Rx(rList, θList) ⇒ *list*
P►Rx(rMatrix, θMatrix) ⇒ *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 °, G or R to override the angle mode setting temporarily.

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

Define $g(x)=\text{Func}$ Done
If $x \leq 0$ or $x \geq 5$
Goto end
Return $x \cdot 3$
Lbl end
EndFunc

$g(3)$ 9
 $g(0)$ A function did not return a value

In Hex base mode:

0h7AC36 or 0h3D5F 0h7BD7F

Important: Zero, not the letter O.

In Bin base mode:

0b100101 or 0b100 0b100101

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("hello") 104
char(104) "h"
ord(char(24)) 24
ord({"alpha", "beta"}) {97,98}

In Radian angle mode:

P►Rx(4,60°) 2.
P►Rx({-3,10,1.3}, { $\frac{\pi}{3}, \frac{\pi}{4}, 0$ })
{-1.5,7.07107,1.3}

P►Ry()

Catalog >

P►Ry($rValue$, $\theta Value$) \Rightarrow $value$ **P►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.^{OT}**Note:** You can insert this function from the computer keyboard by typing **P@>Ry** (...).

In Radian angle mode:

$P\blacktriangleright Ry(4, 60^\circ)$	3.4641
--	--------

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

PassErr

Catalog >

PassErr

Passes an error to the next level.

If system variable $errCode$ is zero, **PassErr** does not do anything.The **Else** clause of the **Try...Else...EndTry** block should use **ClrErr** or **PassErr**. If the error is to be processed or ignored, use **ClrErr**. If what to do with the error is not known, use **PassErr** to send it to the next error handler. If there are no more pending **Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.**Note:** See also **ClrErr**, page 17, and **Try**, page 105.**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of **enter** at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.For an example of **PassErr**, See Example 2 under the **Try** command, page 105.**piecewise()**

Catalog >

piecewise($Expr1$ |, $Cond1$ |, $Expr2$ |, $Cond2$ |, ... |)])

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

Note: See also **Piecewise template**, page 2.

Define $p(x) = \begin{cases} x, & x > 0 \\ \text{undef}, & x \leq 0 \end{cases}$	Done
--	------

$p(1)$	1
$p(-1)$	undef

poissCdf()

Catalog >

poissCdf(λ , $lowBound$, $upBound$) \Rightarrow $number$ if $lowBound$ and $upBound$ are numbers, $list$ if $lowBound$ and $upBound$ are lists**poissCdf**(λ , $upBound$) for $P(0 \leq X \leq upBound) \Rightarrow$ $number$ if $upBound$ is a number, $list$ if $upBound$ is a listComputes a cumulative probability for the discrete Poisson distribution with specified mean λ .For $P(X \leq upBound)$, set $lowBound=0$ **poissPdf()**

Catalog >

poissPdf(λ , $XVal$) \Rightarrow $number$ if $XVal$ is a number, $list$ if $XVal$ is a listComputes a probability for the discrete Poisson distribution with the specified mean λ .

Polar

Catalog >

Vector ► **Polar**

Note: You can insert this operator from the computer keyboard by typing @>Po1a.r.

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 81.

complexValue ► **Polar**

Displays *complexVector* in polar form.

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

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

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

$$\left[\begin{array}{c} 1 \\ 3 \end{array} \right] \text{►Polar} \quad \left[3.16228 \angle 71.5651 \right]$$

In Radian angle mode:

$$(3+4 \cdot i) \text{►Polar} \quad e^{.927295 \cdot i} \cdot 5$$

$$\left(\left(4 \angle \frac{\pi}{3} \right) \right) \text{►Polar} \quad e^{1.0472 \cdot i} \cdot 4$$

In Gradian angle mode:

$$(4 \cdot i) \text{►Polar} \quad (4 \angle 100)$$

In Degree angle mode:

$$(3+4 \cdot i) \text{►Polar} \quad (5 \angle 53.1301)$$

polyEval()

Catalog >

polyEval(*List1*, *Expr1*) \Rightarrow *expression*

polyEval(*List1*, *List2*) \Rightarrow *expression*

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

$$\text{polyEval}(\{1, 2, 3, 4\}, 2) \quad 26$$

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

polyRoots()

Catalog >

polyRoots(*Poly*, *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 $y^2 \cdot y + 1$ 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 23.

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

$$\text{cPolyRoots}(y^3 + 1, y) \quad \{-1, 0.5 - 0.866025 \cdot i, 0.5 + 0.866025 \cdot i\}$$

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

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

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

Computes the power regression $y = (a \cdot (x)^b)$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

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

Prgm*Block***EndPrgm**

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

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

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of **enter** at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Calculate GCD and display intermediate results.

```
Define proggcd(a,b)=Prgm
  Local d
  While b≠0
    d:=mod(a,b)
    a:=b
    b:=d
  Disp a," ",b
  EndWhile
  Disp "GCD=",a
EndPrgm
```

Done

```
proggcd(4560,450)
```

450 60

60 30

30 0

GCD=30

*Done***prodSeq()**See **Π()**, page 124.**Product (PI)**See **Π()**, page 124.**product()**

product(*List*₁, *Start*₁, *End*₁) ⇒ *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*₁, *End*₁) ⇒ *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 131.

```
product({1,2,3,4})
```

 24

```
product({4,5,8,9},2,3)
```

 40

```
product(
  (
    1 2 3
    4 5 6
    7 8 9
  )
)
```

 [28 80 162]

```
product(
  (
    1 2 3
    4 5 6
    7 8 9
  )
,1,2)
```

 [4 10 18]

propFrac()Catalog > **propFrac**(*Value1*, *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.

$$\text{propFrac}\left(\frac{4}{3}\right) \quad 1 + \frac{1}{3}$$

$$\text{propFrac}\left(\frac{-4}{3}\right) \quad -1 - \frac{1}{3}$$

propFrac(*rational_expression*, *Var*) returns the sum of proper ratios and a polynomial with respect to *Var*. The degree of *Var* in the denominator exceeds the degree of *Var* in the numerator in each proper ratio. Similar powers of *Var* are collected. The terms and their factors are sorted with *Var* as the main variable.If *Var* is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.You can use the **propFrac()** function to represent mixed fractions and demonstrate addition and subtraction of mixed fractions.

$$\text{propFrac}\left(\frac{11}{7}\right) \quad 1 + \frac{4}{7}$$

$$\text{propFrac}\left(3 + \frac{1}{11} + 5 + \frac{3}{4}\right) \quad 8 + \frac{37}{44}$$

$$\text{propFrac}\left(3 + \frac{1}{11} - \left(5 + \frac{3}{4}\right)\right) \quad -2 - \frac{29}{44}$$

Q**QR**Catalog > **QR** *Matrix*, *qMatrix*, *rMatrix*, *Tol*Calculates the Householder QR factorization of a real or complex matrix. The resulting Q and R matrices are stored to the specified *Matrix*. The Q matrix is unitary. The R matrix is upper triangular.Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

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

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

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9. \end{bmatrix} \rightarrow m1 \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9. \end{bmatrix}$$

QR *m1*, *qm*, *rm* Done

<i>qm</i>	0.123091	0.904534	0.408248
	0.492366	0.301511	-0.816497
	0.86164	-0.301511	0.408248

<i>rm</i>	8.12404	9.60114	11.0782
	0.	0.904534	1.80907
	0.	0.	0.

ClearAZ DoneThe 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 $X, Y [, Freq] [, Category, Include]$

Computes the quadratic polynomial regression $y = a \cdot x^2 + b \cdot x + c$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

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 <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

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

Computes the quartic polynomial regression

$y = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$ on lists X and Y with frequency $Freq$. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.RegEqn	Regression equation: $a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e$
stat.a, stat.b, stat.c, stat.d, stat.e	Regression coefficients
stat.R ²	Coefficient of determination
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified <i>X List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified <i>Y List</i> actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

R

R►Pθ()

Catalog > 

R►Pθ (*xValue*, *yValue*) ⇒ *value*

In Degree angle mode:

R►Pθ (*xList*, *yList*) ⇒ *list*

$$\text{R►P}\theta(2,2) \quad 45.$$

R►Pθ (*xMatrix*, *yMatrix*) ⇒ *matrix*

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

In Gradian angle mode:

$$\text{R►P}\theta(2,2) \quad 50.$$

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

In Radian angle mode:

$$\text{R►P}\theta(3,2) \quad 0.588003$$

Note: You can insert this function from the computer keyboard by typing **R►Ptheta** (...).

$$\text{R►P}\theta\left(\begin{bmatrix} 3 & -4 & 2 \\ 0 & \frac{\pi}{4} & 1.5 \end{bmatrix}\right) \quad \begin{bmatrix} 0. & 2.94771 & 0.643501 \end{bmatrix}$$

R►Pr()

Catalog > 

R►Pr (*xValue*, *yValue*) ⇒ *value*

In Radian angle mode:

R►Pr (*xList*, *yList*) ⇒ *list*

$$\text{R►Pr}(3,2) \quad 3.60555$$

R►Pr (*xMatrix*, *yMatrix*) ⇒ *matrix*

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

$$\text{R►Pr}\left(\begin{bmatrix} 3 & -4 & 2 \\ 0 & \frac{\pi}{4} & 1.5 \end{bmatrix}\right) \quad \begin{bmatrix} 3 & 4.07638 & \frac{5}{2} \end{bmatrix}$$

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

►Rad

Catalog >

Value ►Rad ⇒ *value*

Converts the argument to radian angle measure.

Note: You can insert this operator from the computer keyboard by typing Θ ►Rad.

In Degree angle mode:

$(1.5) \blacktriangleright \text{Rad}$	$(0.02618)^{\circ}$
--	---------------------

In Gradian angle mode:

$(1.5) \blacktriangleright \text{Rad}$	$(0.023562)^{\circ}$
--	----------------------

rand()

Catalog >

rand() ⇒ *expression***rand**(#Trials) ⇒ *list***rand()** returns a random value between 0 and 1.**rand**(#Trials) returns a list containing #Trials random values between 0 and 1.

Sets the random-number seed.

RandSeed 1147	Done
rand (2)	{0.158206, 0.717917}

randBin()

Catalog >

randBin(*n*, *p*) ⇒ *expression***randBin**(*n*, *p*, #Trials) ⇒ *list***randBin**(*n*, *p*) returns a random real number from a specified Binomial distribution.**randBin**(*n*, *p*, #Trials) returns a list containing #Trials random real numbers from a specified Binomial distribution.

randBin (80, 5)	34.
randBin (80, 5, 3)	{47., 41., 46.}

randInt()

Catalog >

randInt(*lowBound*, *upBound*) ⇒ *expression***randInt**(*lowBound*, *upBound*, #Trials) ⇒ *list***randInt**(*lowBound*, *upBound*) returns a random integer within the range specified by *lowBound* and *upBound* integer bounds.**randInt**(*lowBound*, *upBound*, #Trials) returns a list containing #Trials random integers within the specified range.

randInt (3, 10)	7.
randInt (3, 10, 4)	{8., 9., 4., 4.}

randMat()

Catalog >

randMat(*numRows*, *numColumns*) ⇒ *matrix*

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

Both arguments must simplify to integers.

RandSeed 1147	Done									
randMat (3, 3)	<table border="1"> <tr><td>8</td><td>-3</td><td>6</td></tr> <tr><td>-2</td><td>3</td><td>-6</td></tr> <tr><td>0</td><td>4</td><td>-6</td></tr> </table>	8	-3	6	-2	3	-6	0	4	-6
8	-3	6								
-2	3	-6								
0	4	-6								

Note: The values in this matrix will change each time you press **enter**.**randNorm()**

Catalog >

randNorm(μ , σ) ⇒ *expression***randNorm**(μ , σ , #Trials) ⇒ *list***randNorm**(μ , σ) returns a decimal number from the specified normal distribution. It could be any real number but will be heavily concentrated in the interval $[\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma]$.**randNorm**(μ , σ , #Trials) returns a list containing #Trials decimal numbers from the specified normal distribution.

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

randPoly()

Catalog >

randPoly(*Var*, *Order*) ⇒ *expression*

Returns a polynomial in *Var* of the specified *Order*. The coefficients are random integers in the range -9 through 9. The leading coefficient will not be zero.

Order must be 0–99.

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

randSamp()

Catalog >

randSamp(*List*, #*Trials*, *noRepl*) ⇒ *list*

Returns a list containing a random sample of #*Trials* trials from *List* with an option for sample replacement (*noRepl*=0), or no sample replacement (*noRepl*=1). The default is with sample replacement.

Define list3={1,2,3,4,5}	Done
Define list4=randSamp(list3,6)	Done
list4	{5,1,.,3.,3.,4.,4.}

RandSeed

Catalog >

RandSeed *Number*

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

RandSeed 1147	Done
rand()	0.158206

real()

Catalog >

real(*Value1*) ⇒ *value*

Returns the real part of the argument.

real(*List1*) ⇒ *list*

Returns the real parts of all elements.

real(*Matrix1*) ⇒ *matrix*

Returns the real parts of all elements.

$\text{real}(2+3 \cdot i)$	2
$\text{real}\{1+3 \cdot i, 3, i\}$	{1,3,0}
$\text{real}\left(\begin{matrix} 1+3 \cdot i & 3 \\ 2 & i \end{matrix}\right)$	$\begin{bmatrix} 1 & 3 \\ 2 & 0 \end{bmatrix}$

►Rect

Catalog >

Vector ►**Rect**

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

Displays *Vector* in rectangular form [x, y, z]. The vector must be of dimension 2 or 3 and can be a row or a column.

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

Note: See also ►**Polar**, page 74.

$\left(3 \angle \frac{\pi}{4} \angle \frac{\pi}{6}\right) \blacktriangleright \text{Rect}$	$[1.06066 \ 1.06066 \ 2.59808]$
--	---------------------------------

complexValue ►Rect

Displays *complexValue* in rectangular form $a+bi$. The *complexValue* can have any complex form. However, an $re^{i\theta}$ entry causes an error in Degree angle mode.

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

In Radian angle mode:

$\left(4e^{\frac{\pi}{3}}\right)$ ►Rect	11.3986
$\left(4\angle\frac{\pi}{3}\right)$ ►Rect	$2+3.4641\cdot i$

In Gradian angle mode:

$\left(1\angle 100\right)$ ►Rect	i
----------------------------------	-----

In Degree angle mode:

$\left(4\angle 60\right)$ ►Rect	$2+3.4641\cdot i$
---------------------------------	-------------------

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

ref()

ref(*MatrixI*, *Tol*) \Rightarrow *matrix*

Returns the row echelon form of *MatrixI*.

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

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

Avoid undefined elements in *MatrixI*. They can lead to unexpected results.

For example, if *a* is undefined in the following expression, a warning message appears and the result is shown as:

$$\text{ref}\left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) \Rightarrow \begin{bmatrix} 1 & \frac{1}{a} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The warning appears because the generalized element $1/a$ would not be valid for $a=0$.

You can avoid this by storing a value to *a* beforehand or by using the "I" substitution mechanism, as shown in the following example.

$$\text{ref}\left(\begin{bmatrix} a & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) | a=0 \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Note: See also **rref()**, page 87.

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

remain()

Catalog >

remain(Value1, Value2) \Rightarrow value**remain**(List1, List2) \Rightarrow list**remain**(Matrix1, Matrix2) \Rightarrow matrix

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

remain(x,0) = x

remain(x,y) = x - y · iPart(x/y)

As a consequence, note that **remain**(-x,y) = -**remain**(x,y). The result is either zero or it has the same sign as the first argument.**Note:** See also **mod()**, page 64.

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}

remain($\begin{bmatrix} 9 & -7 \\ 6 & 4 \end{bmatrix}, \begin{bmatrix} 4 & 3 \\ 4 & -3 \end{bmatrix}$)	$\begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix}$
--	---

Request

Catalog >

Request promptString, var[, DispFlag [, statusVar]]**Request** promptString, func(arg1, ...argn)
[, DispFlag [, statusVar]]Programming command: Pauses the program and displays a dialog box containing the message *promptString* and an input box for the user's response.When the user types a response and clicks **OK**, the contents of the input box are assigned to variable *var*.If the user clicks **Cancel**, the program proceeds without accepting any input. The program uses the previous value of *var* if *var* was already defined.The optional *DispFlag* argument can be any expression.

- If *DispFlag* is omitted or evaluates to **1**, the prompt message and user's response are displayed in the Calculator history.
- If *DispFlag* evaluates to **0**, the prompt and response are not displayed in the history.

The optional *statusVar* argument gives the program a way to determine how the user dismissed the dialog box. Note that *statusVar* requires the *DispFlag* argument.

- If the user clicked **OK** or pressed **Enter** or **Ctrl+Enter**, variable *statusVar* is set to a value of **1**.
- Otherwise, variable *statusVar* is set to a value of **0**.

The *func()* argument allows a program to store the user's response as a function definition. This syntax operates as if the user executed the command:Define *func*(arg1, ...argn) = user's responseThe program can then use the defined function *func()*. The *promptString* should guide the user to enter an appropriate user's response that completes the function definition.**Note:** You can use the **Request** command within a user-defined program but not within a function.To stop a program that contains a **Request** command inside an infinite loop:

- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **Handheld:** Hold down the key and press repeatedly.

Note: See also **RequestStr**, page 84.

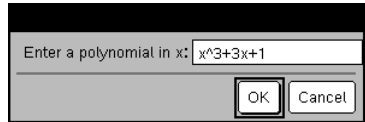
Define a program:

```
Define request_demo()=Prgm
Request "Radius: ",r
Disp "Area = ",pi*r^2
EndPrgm
```

Run the program and type a response:
request_demo()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 selecting **OK**:Enter a polynomial in x: x^3+3x+1
Real roots are: {-0.322185}

RequestStr



Catalog > 

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:

- **Windows®:** Hold down the **F12** key and press **Enter** repeatedly.
- **Macintosh®:** Hold down the **F5** key and press **Enter** repeatedly.
- **Handheld:** Hold down the  **on** key and press  repeatedly.

Note: See also **Request**, page 83.

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

Catalog > 

Return [*Expr*]

Returns *Expr* as the result of the function. Use within a **Func...EndFunc** block.

Note: Use **Return** without an argument within a **Prgm...EndPrgm** block to exit a program.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define factorial(n)=Func
Local answer,count
1 → answer
For count,1,n
answer·count → answer
EndFor
Return answer
EndFunc
```

Done

```
factorial(3) 6
```

right()

Catalog > 

right(*List1*, *Num*) ⇒ *list*

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

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

right(*sourceString*, *Num*) ⇒ *string*

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

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

right(*Comparison*) ⇒ *expression*

Returns the right side of an equation or inequality.

```
right({1,3,-2,4},3) {3,-2,4}
```

```
right("Hello",2) "lo"
```

rk23()

rk23(Expr, Var, depVar, {Var0, VarMax}, depVar0, VarStep [, diftol]) \Rightarrow matrix

rk23(SystemOfExpr, Var, ListOfDepVars, {Var0, VarMax}, ListOfDepVars0, VarStep [, diftol]) \Rightarrow matrix

rk23(ListOfExpr, Var, ListOfDepVars, {Var0, VarMax}, ListOfDepVars0, VarStep [, diftol]) \Rightarrow matrix

Uses the Runge-Kutta method to solve the system

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

with $\text{depVar}(\text{Var0}) = \text{depVar0}$ on the interval $[\text{Var0}, \text{VarMax}]$. Returns a matrix whose first row defines the *Var* output values as defined by *VarStep*. The second row defines the value of the first solution component at the corresponding *Var* values, and so on.

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

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

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

Var is the independent variable.

ListOfDepVars is a list of dependent variables.

{*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: $\text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax} - \text{Var0})$ and solutions are returned at $\text{Var0} + i * \text{VarStep}$ for all $i = 0, 1, 2, \dots$ such that $\text{Var0} + i * \text{VarStep}$ is in $[\text{Var0}, \text{VarMax}]$ (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).

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

$$\text{rk23}(0.001 \cdot y \cdot (100 - y), t, y, \{0, 100\}, 10, 1) \Rightarrow \begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 10.9367 & 11.9493 & 13.042 & 14.2 \end{bmatrix}$$

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

Same equation with *diftol* set to $1.E-6$

$$\text{rk23}(0.001 \cdot y \cdot (100 - y), t, y, \{0, 100\}, 10, 1, 1.E-6) \Rightarrow \begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 10. & 10.9367 & 11.9495 & 13.0423 & 14.2189 \end{bmatrix}$$

System of equations:

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

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

$$\text{rk23}\left(\begin{cases} -y1 + 0.1 \cdot y1 \cdot y2 \\ 3 \cdot y2 - y1 \cdot y2 \end{cases}, t, \{y1, y2\}, \{0, 5\}, \{2, 5\}, 1\right) \Rightarrow \begin{bmatrix} 0. & 1. & 2. & 3. & 4. \\ 2. & 1.94103 & 4.78694 & 3.25253 & 1.82848 \\ 5. & 16.8311 & 12.3133 & 3.51112 & 6.27245 \end{bmatrix}$$

root()

root(*Value*) \Rightarrow root

root(*Value1*, *Value2*) \Rightarrow root

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 1.

$$\begin{aligned} \sqrt[3]{8} &= 2 \\ \sqrt[3]{3} &= 1.44225 \end{aligned}$$

rotate()

rotate(*Integer1* [, #ofRotations]) \Rightarrow integer

Rotates the bits in a binary integer. You can enter *Integer1* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *Integer1* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see **Base2**, page 12.

In Bin base mode:

$$\begin{aligned} \text{rotate}(0b11) &= 0b1001 \\ \text{rotate}(256, 1) &= 0b1000000000 \end{aligned}$$

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

rotate()

Catalog >

If *#ofRotations* is positive, the rotation is to the left. If *#ofRotations* is negative, the rotation is to the right. The default is -1 (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

0b00000000000001111010110000110101

Rightmost bit rotates to leftmost.

produces:

0b10000000000000111101011000011010

The result is displayed according to the Base mode.

rotate(*List1*, *#ofRotations*) ⇒ *list*

Returns a copy of *List1* rotated right or left by *#ofRotations* elements. Does not alter *List1*.

If *#ofRotations* is positive, the rotation is to the left. If *#ofRotations* is negative, the rotation is to the right. The default is -1 (rotate right one element).

rotate(*String1*, *#ofRotations*) ⇒ *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).

In Hex base mode:

rotate (0h78E)	0h3C7
rotate (0h78E,-2)	0h800000000000001E3
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 ("abcd")	"dabc"
rotate ("abcd",-2)	"cdab"
rotate ("abcd",1)	"bcda"

round()

Catalog >

round(*Value1*, *digits*) ⇒ *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*) ⇒ *list*

Returns a list of the elements rounded to the specified number of digits.

round(*Matrix1*, *digits*) ⇒ *matrix*

Returns a matrix of the elements rounded to the specified number of digits.

round (1.234567,3)	1.235
---------------------------	-------

round ({ π , $\sqrt{2}$, $\ln(2)$ },4)	{3.1416,1.4142,0.6931}
--	------------------------

round ($\begin{pmatrix} \ln(5) & \ln(3) \\ \pi & e^1 \end{pmatrix}$,1)	$\begin{bmatrix} 1.6 & 1.1 \\ 3.1 & 2.7 \end{bmatrix}$
---	--

rowAdd()

Catalog >

rowAdd(*Matrix1*, *rIndex1*, *rIndex2*) ⇒ *matrix*

Returns a copy of *Matrix1* with row *rIndex2* replaced by the sum of rows *rIndex1* and *rIndex2*.

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

rowDim()

Catalog >

rowDim(*Matrix*) \Rightarrow *expression*Returns the number of rows in *Matrix*.**Note:** See also **colDim()**, page 17.

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

rowNorm()

Catalog >

rowNorm(*Matrix*) \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 17.

rowNorm $\left(\begin{bmatrix} -5 & 6 & -7 \\ 3 & 4 & 9 \\ 9 & -9 & -7 \end{bmatrix}\right)$	25
---	----

rowSwap()

Catalog >

rowSwap(*Matrix1*, *rIndex1*, *rIndex2*) \Rightarrow *matrix*Returns *Matrix1* with rows *rIndex1* and *rIndex2* exchanged.

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

rref()

Catalog >

rref(*Matrix1*, *Tol*) \Rightarrow *matrix*Returns the reduced row echelon form of *Matrix1*.

rref $\left(\begin{bmatrix} -2 & -2 & 0 & -6 \\ 1 & -1 & 9 & -9 \\ -5 & 2 & 4 & -4 \end{bmatrix}\right)$	$\begin{bmatrix} 1 & 0 & 0 & \frac{66}{71} \\ 0 & 1 & 0 & \frac{147}{71} \\ 0 & 0 & 1 & \frac{-62}{71} \end{bmatrix}$
---	---

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

- If you use **ctrl enter** or set the **Auto or Approximate** mode to Approximate, computations are done using floating-point arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as:
 $5E-14 \cdot \max(\dim(\text{Matrix1})) \cdot \text{rowNorm}(\text{Matrix1})$

Note: See also **rref()**, page 82.

S

sec()

 **key**

$\text{sec}(Value1) \Rightarrow value$

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

Returns the secant of $Value1$ or returns a list containing the secants 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 $^{\circ}$, $^{\text{G}}$, or $^{\text{r}}$ to override the angle mode temporarily.

In Degree angle mode:

$\text{sec}(45)$	1.41421
------------------	---------

$\text{sec}\{1,2,3,4\}$	$\{1.00015,1.00081,1.00244\}$
-------------------------	-------------------------------

$\text{sec}^{-1}()$

 **key**

$\text{sec}^{-1}(Value1) \Rightarrow value$

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

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:

$\text{sec}^{-1}(1)$	0
----------------------	---

In Gradian angle mode:

$\text{sec}^{-1}(\sqrt{2})$	50
-----------------------------	----

In Radian angle mode:

$\text{sec}^{-1}\{1,2,5\}$	$\{0,1.0472,1.36944\}$
----------------------------	------------------------

sech()

Catalog > 

$\text{sech}(Value1) \Rightarrow value$

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

Returns the hyperbolic secant of $Value1$ or returns a list containing the hyperbolic secants of the $List1$ elements.

$\text{sech}(3)$	0.099328
------------------	----------

$\text{sech}\{1,2,3,4\}$	$\{0.648054,0.198522,0.036619\}$
--------------------------	----------------------------------

$\text{sech}^{-1}()$

Catalog > 

$\text{sech}^{-1}(Value1) \Rightarrow value$

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

Returns the inverse hyperbolic secant of $Value1$ or returns a list containing the inverse hyperbolic secants of each element of $List1$.

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

In Radian angle and Rectangular complex mode:

$\text{sech}^{-1}(1)$	0
-----------------------	---

$\text{sech}^{-1}\{1,-2,2.1\}$	$\{0,2.0944i,8.E-15+1.07448i\}$
--------------------------------	---------------------------------

seq()

Catalog >

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.

$$\text{seq}\left(n^2, n, 1, 6\right) \quad \left\{1, 4, 9, 16, 25, 36\right\}$$

$$\text{seq}\left(\frac{1}{n}, n, 1, 10, 2\right) \quad \left\{1, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}\right\}$$

$$\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right) \quad \frac{1968329}{1270080}$$

Press **Ctrl+Enter** (Macintosh®: + **Enter**) to evaluate:

$$\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right) \quad 1.54977$$

seqGen()

Catalog >

seqGen(*Expr*, *Var*, *depVar*, {*Var0*, *VarMax*}, *ListOfInitTerms* [, *VarStep* [, *CeilingValue*]]) \Rightarrow *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*]]) \Rightarrow *matrix*

Generates a matrix of terms for a system (or list) of sequences *ListOfDepVars*(*Var*)=*ListOrSystemOfExpr* as follows: Increments independent variable *Var* from *Var0* through *VarMax* by *VarStep*, evaluates *ListOfDepVars*(*Var*) for corresponding values of *Var* using *ListOrSystemOfExpr* formula and *MatrixOfInitTerms*, and returns the results as a matrix.

The original contents of *Var* are unchanged after **seqGen()** is completed.

The default value for *VarStep* = 1.

Generate the first 5 terms of the sequence $u(n) = u(n-1)^2/2$, with $u(1)=2$ and *VarStep*=1.

$$\text{seqGen}\left(\frac{u(n-1)^2}{n}, n, u, \{1, 5\}, \{2\}\right)$$

$$\left\{2, 2, \frac{4}{3}, \frac{4}{9}, \frac{16}{405}\right\}$$

Example in which *Var0*=2:

$$\text{seqGen}\left(\frac{u(n-1)+1}{n}, n, u, \{2, 5\}, \{3\}\right)$$

$$\left\{3, \frac{4}{3}, \frac{7}{12}, \frac{19}{60}\right\}$$

System of two sequences:

$$\text{seqGen}\left(\left\{\frac{1}{n}, \frac{u_2(n-1)}{2} + u_1(n-1)\right\}, n, \{u_1, u_2\}, \{1, 5\}, \left[\begin{array}{c} - \\ 2 \end{array}\right]\right)$$

$$\left[\begin{array}{ccccc} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ 2 & 2 & \frac{3}{2} & \frac{13}{12} & \frac{19}{24} \end{array}\right]$$

Note: The Void () in the initial term matrix above is used to indicate that the initial term for $u_1(n)$ is calculated using the explicit sequence formula $u_1(n)=1/n$.

seqn()

Catalog >

seqn(*Expr*(*u*, *n* [, *ListOfInitTerms* [, *nMax* [, *CeilingValue*]])]) ⇒ *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*]])]) ⇒ *list*

Generates a list of terms for a non-recursive sequence $u(n)=Expr(n)$ as follows: Increments n from 1 through $nMax$ by 1, evaluates $u(n)$ for corresponding values of n using the $Expr(n)$ formula, and returns the results as a list.

If $nMax$ is missing, $nMax$ is set to 2500

If $nMax=0$, $nMax$ is set to 2500

Note: **seqn()** calls **seqGen()** with $n0=1$ and $nstep=1$

Generate the first 6 terms of the sequence $u(n) = u(n-1)/2$, with $u(1)=2$.

$$\text{seqn}\left(\frac{u(n-1)}{n}, \{2\}, 6\right)$$

$$\left\{2, 1, \frac{1}{3}, \frac{1}{12}, \frac{1}{60}, \frac{1}{360}\right\}$$

$$\text{seqn}\left(\frac{1}{n^2}, 6\right) \quad \left\{1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}\right\}$$

setMode()

Catalog >

setMode(*modeNameInteger*, *settingInteger*) ⇒ *integer*
setMode(*list*) ⇒ *integer list*

Valid only within a function or program.

setMode(*modeNameInteger*, *settingInteger*) temporarily sets mode *modeNameInteger* to the new setting *settingInteger*, and returns an integer corresponding to the original setting of that mode. The change is limited to the duration of the program/function's execution.

modeNameInteger specifies which mode you want to set. It must be one of the mode integers from the table below.

settingInteger specifies the new setting for the mode. It must be one of the setting integers listed below for the specific mode you are setting.

setMode(*list*) lets you change multiple settings. *list* contains pairs of mode integers and setting integers. **setMode**(*list*) returns a similar list whose integer pairs represent the original modes and settings.

If you have saved all mode settings with **getMode(0)** → *var*, you can use **setMode**(*var*) to restore those settings until the function or program exits. See **getMode()**, page 42.

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: In the Calculator application on the handheld, you can enter multi-line definitions by pressing instead of at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Display approximate value of π using the default setting for Display Digits, and then display π with a setting of Fix2. Check to see that the default is restored after the program executes.

```
Define prog1()=Prgm                               Done
    Disp  $\pi$ 
    setMode(1,16)
    Disp  $\pi$ 
    EndPrgm
```

```
prog1()
```

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

Mode Name	Mode Integer	Setting Integers
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()

Catalog > 

shift(IntegerI [,#ofShifts]) ⇒ integer

Shifts the bits in a binary integer. You can enter *IntegerI* in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of *IntegerI* is too large for this form, a symmetric modulo operation brings it within the range. For more information, see **Base2**, page 12.

If *#ofShifts* is positive, the shift is to the left. If *#ofShifts* is negative, the shift is to the right. The default is -1 (shift right one bit).

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

For example, in a right shift:

Each bit shifts right.

```
0b0000000000000111101011000011010
```

Inserts 0 if leftmost bit is 0,
or 1 if leftmost bit is 1.

produces:

```
0b00000000000000111101011000011010
```

The result is displayed according to the Base mode. Leading zeros are not shown.

shift(ListI [,#ofShifts]) ⇒ list

Returns a copy of *ListI* shifted right or left by *#ofShifts* elements. Does not alter *ListI*.

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(StringI [,#ofShifts]) ⇒ string

Returns a copy of *StringI* shifted right or left by *#ofShifts* characters. Does not alter *StringI*.

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 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).

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()Catalog > **sign(Value1)** \Rightarrow *value***sign(List1)** \Rightarrow *list***sign(Matrix1)** \Rightarrow *matrix*

For real and complex *Value1*, returns *Value1* / **abs(Value1)** when *Value1* \neq 0.

Returns 1 if *Value1* is positive.Returns -1 if *Value1* is negative.

sign(0) returns ± 1 if the complex format mode is Real; otherwise, it returns itself.

sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

$\text{sign}(-3.2)$	-1
$\text{sign}\{2,3,4,-5\}$	{1,1,1,-1}

If complex format mode is Real:

$\text{sign}\begin{pmatrix} -3 & 0 & 3 \end{pmatrix}$	$\begin{bmatrix} -1 & \text{undef} & 1 \end{bmatrix}$
---	---

simult()Catalog > **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 55.

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:

$$5E-14 \cdot \max(\dim(\text{coeffMatrix})) \cdot \text{rowNorm}(\text{coeffMatrix})$$

simult(coeffMatrix, constMatrix, Tol) \Rightarrow *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:

$x + 2y = 1$

$3x + 4y = -1$

$\text{simult}\left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}\right)$	$\begin{bmatrix} -3 \\ 2 \end{bmatrix}$
---	---

The solution is $x=-3$ and $y=2$.

Solve:

$ax + by = 1$

$cx + dy = 2$

$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \rightarrow \text{matX1}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
---	--

$\text{simult}\left(\text{matX1}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}\right)$	$\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$
--	---

Solve:

$x + 2y = 1$

$3x + 4y = -1$

$x + 2y = 2$

$3x + 4y = -3$

$\text{simult}\left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ -1 & -3 \end{pmatrix}\right)$	$\begin{bmatrix} -3 & -7 \\ 2 & 9 \\ & 2 \end{bmatrix}$
--	---

For the first system, $x=-3$ and $y=2$. For the second system, $x=-7$ and $y=9/2$.

sin()**sin**(*Value1*) ⇒ *value***sin**(*List1*) ⇒ *list***sin**(*Value1*) returns the sine of the argument.**sin**(*List1*) returns a list of the sines of all elements in *List1*.**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, G, or r to override the angle mode setting temporarily.

In Degree angle mode:

$\sin\left(\frac{\pi}{4}\right)$	0.707107
----------------------------------	----------

$\sin(45)$	0.707107
------------	----------

$\sin(\{0,60,90\})$	{0,.866025,1}
---------------------	---------------

In Gradian angle mode:

$\sin(50)$	0.707107
------------	----------

In Radian angle mode:

$\sin\left(\frac{\pi}{4}\right)$	0.707107
----------------------------------	----------

$\sin(45^\circ)$	0.707107
------------------	----------

In Radian angle mode:

$\sin\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$	$\begin{bmatrix} 0.9424 & -0.04542 & -0.031999 \\ -0.045492 & 0.949254 & -0.020274 \\ -0.048739 & -0.00523 & 0.961051 \end{bmatrix}$
---	--

sin(*squareMatrix1*) ⇒ *squareMatrix*Returns the matrix sine of *squareMatrix1*. This is not the same as calculating the sine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.**sin⁻¹()****sin⁻¹**(*Value1*) ⇒ *value***sin⁻¹**(*List1*) ⇒ *list***sin⁻¹**(*Value1*) returns the angle whose sine is *Value1*.**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**(...).**sin⁻¹**(*squareMatrix1*) ⇒ *squareMatrix*Returns the matrix inverse sine of *squareMatrix1*. This is not the same as calculating the inverse sine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

$\sin^{-1}(\{0,0,2,0.5\})$	{0,0.201358,0.523599}
----------------------------	-----------------------

In Radian angle mode and Rectangular complex format mode:

$\sin^{-1}\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$	$\begin{bmatrix} -0.164058-0.064606\cdot i & 1.49086-2.1051\cdot i \\ 0.725533-1.51594\cdot i & 0.947305-0.7783\cdot i \\ 2.08316-2.63205\cdot i & -1.79018+1.2718\cdot i \end{bmatrix}$
--	--

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

sinh()Catalog > **sinh**(*Number1*) \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()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

$\sinh(1.2)$	1.50946
$\sinh(\{0,1,2,3\})$	$\{0,1.50946,10.0179\}$

In Radian angle mode:

$\sinh \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$	$\begin{bmatrix} 360.954 & 305.708 & 239.604 \\ 352.912 & 233.495 & 193.564 \\ 298.632 & 154.599 & 140.251 \end{bmatrix}$
--	---

sinh⁻¹()Catalog > **sinh⁻¹**(*Value1*) \Rightarrow *value***sinh⁻¹**(*List1*) \Rightarrow *list***sinh⁻¹** (*Value1*) returns the inverse hyperbolic sine of the argument.**sinh⁻¹** (*List1*) returns a list of the inverse hyperbolic sines of each element of *List1*.**Note:** You can insert this function from the keyboard by typing **arcsinh** (...).**sinh⁻¹**(*squareMatrix1*) \Rightarrow *squareMatrix*Returns the matrix inverse hyperbolic sine of *squareMatrix1*. This is not the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

$\sinh^{-1}(0)$	0
$\sinh^{-1}(\{0,2,1,3\})$	$\{0,1.48748,1.81845\}$

In Radian angle mode:

$\sinh^{-1} \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}$	$\begin{bmatrix} 0.041751 & 2.15557 & 1.1582 \\ 1.46382 & 0.926568 & 0.112557 \\ 2.75079 & -1.5283 & 0.57268 \end{bmatrix}$
---	---

SinReg $X, Y [, [Iterations], [Period] [, Category, Include]]$

Computes the sinusoidal regression on lists X and Y . A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.RegEqn	Regression Equation: $a \cdot \sin(bx+c)+d$
stat.a, stat.b, stat.c, stat.d	Regression coefficients
stat.Resid	Residuals from the regression
stat.XReg	List of data points in the modified X List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.YReg	List of data points in the modified Y List actually used in the regression based on restrictions of <i>Freq</i> , <i>Category List</i> , and <i>Include Categories</i>
stat.FreqReg	List of frequencies corresponding to <i>stat.XReg</i> and <i>stat.YReg</i>

SortA

SortA $List1 [, List2] [, List3] \dots$

SortA $Vector1 [, Vector2] [, Vector3] \dots$

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 131.

$\{2,1,4,3\} \rightarrow list1$	$\{2,1,4,3\}$
SortA <i>list1</i>	<i>Done</i>
<i>list1</i>	$\{1,2,3,4\}$
$\{4,3,2,1\} \rightarrow list2$	$\{4,3,2,1\}$
SortA <i>list2, list1</i>	<i>Done</i>
<i>list2</i>	$\{1,2,3,4\}$
<i>list1</i>	$\{4,3,2,1\}$

SortD

Catalog > 

SortD $List1$, $List2$ [, $List3$] ...

SortD $Vector1$ [, $Vector2$] [, $Vector3$] ...

Identical to **SortA**, except **SortD** sorts the elements in descending order.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 131.

$\{2,1,4,3\} \rightarrow list1$	$\{2,1,4,3\}$
$\{1,2,3,4\} \rightarrow list2$	$\{1,2,3,4\}$
SortD $list1, list2$	Done
$list1$	$\{4,3,2,1\}$
$list2$	$\{3,4,1,2\}$

►Sphere

Catalog > 

Vector ►Sphere

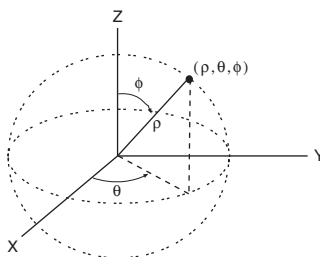
Note: You can insert this operator from the computer keyboard by typing @>**Sphere**.

Displays the row or column vector in spherical form $[r \angle \theta \angle \phi]$.

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.

$[1 \ 2 \ 3] \blacktriangleright \text{Sphere}$	$[3.74166 \ \angle 1.10715 \ \angle 0.640522]$
$\left[2 \ \angle \frac{\pi}{4} \ 3\right] \blacktriangleright \text{Sphere}$	$[3.60555 \ \angle 0.785398 \ \angle 0.588003]$



sqrt()

Catalog > 

sqrt($Value1$) \Rightarrow $value$

sqrt($List1$) \Rightarrow $list$

Returns the square root of the argument.

For a list, returns the square roots of all the elements in $List1$.

Note: See also **Square root template**, page 1.

$\sqrt{4}$	2
$\sqrt{\{9,2,4\}}$	$\{3,1.41421,2\}$

stat.results

Displays results from a statistics calculation.

The results are displayed as a set of name-value pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

Note: Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

$$xlist: = \{1,2,3,4,5\} \quad \{1,2,3,4,5\}$$

$$ylist: = \{4,8,11,14,17\} \quad \{4,8,11,14,17\}$$

LinRegMx *xlist,ylist,1: stat.results*

"Title"	"Linear Regression (mx+b)"
"RegEqn"	"m*x+b"
"m"	3.2
"b"	1.2
"r ² "	0.996109
"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.AdjR²
stat.b
stat.b0
stat.b1
stat.b2
stat.b3
stat.b4
stat.b5
stat.b6
stat.b7
stat.b8
stat.b9
stat.b10
stat.bList
stat.χ²
stat.c
stat.CLower
stat.CLowerList
stat.CompList
stat.CompMatrix
stat.CookDist
stat.CUpper
stat.CUpperList
stat.d

stat.dfDenom
stat.dfBlock
stat.dfCol
stat.dfError
stat.dfInteract
stat.dfReg
stat.dfNumer
stat.dfRow
stat.DW
stat.e
stat.ExpMatrix
stat.F
stat.FBlock
stat.Fcol
stat.FInteract
stat.FreqReg
stat.Frow
stat.Leverage
stat.LowerPred
stat.LowerVal
stat.m
stat.MaxX
stat.MaxY
stat.ME
stat.MedianX

stat.MedianY
stat.MEPred
stat.MinX
stat.MinY
stat.MS
stat.MSBlock
stat.MSCol
stat.MSError
stat.MSInteract
stat.MSReg
stat.MSRow
stat.n
stat.β̂
stat.β̂1
stat.β̂2
stat.β̂Diff
stat.PList
stat.PVal
stat.PValBlock
stat.PValCol
stat.PValInteract
stat.PValRow
stat.Q1X
stat.Q1Y
stat.Q3X

stat.Q3Y
stat.r
stat.r²
stat.RegEqn
stat.Resid
stat.ResidTrans
stat.σx
stat.σy
stat.σx1
stat.σx2
stat.Σx
stat.Σxy
stat.Σy
stat.Σy²
stat.s
stat.SE
stat.SEList
stat.SEPred
stat.sResid
stat.SEslope
stat.sp
stat.SS
stat.SSBlock

stat.SSCol
stat.SSX
stat.SSY
stat.SSError
stat.SSInteract
stat.SSReg
stat.SSRow
stat.tList
stat.UpperPred
stat.UpperVal
stat.X̄
stat.X̄1
stat.X̄2
stat.X̄Diff
stat.X̄List
stat.XReg
stat.XVal
stat.XValList
stat.ȳ
stat.ŷ
stat.ŷList
stat.YReg

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.valuesCatalog > **stat.values**See the **stat.results** example.

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike **stat.results**, **stat.values** omits the names associated with the values.

You can copy a value and paste it into other locations.

stDevPop()Catalog > **stDevPop**(*List*1, *freqList*) \Rightarrow *expression*

Returns the population standard deviation of the elements in *List*.

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

Note: *List* must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 131.

stDevPop(*Matrix*1, *freqMatrix*) \Rightarrow *matrix*

Returns a row vector of the population standard deviations of the columns in *Matrix*1.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix*1.

Note: *Matrix*1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 131.

In Radian angle and auto modes:

$\text{stDevPop}\{1, 2, 5, -6, 3, -2\}$	3.59398
$\text{stDevPop}\{1, 3, 2, 5, -6, 4\}, \{3, 2, 5\}$	4.11107

$\text{stDevPop}\begin{pmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{pmatrix}$	$[3.26599 \quad 2.94392 \quad 1.63299]$
$\text{stDevPop}\begin{pmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{pmatrix}, \begin{pmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{pmatrix}$	$[2.52608 \quad 5.21506]$

stDevSamp()Catalog > **stDevSamp**(*List*1, *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 131.

stDevSamp(*Matrix*1, *freqMatrix*) \Rightarrow *matrix*

Returns a row vector of the sample standard deviations of the columns in *Matrix*1.

Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *Matrix*1.

Note: *Matrix*1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 131.

$\text{stDevSamp}\{1, 2, 5, -6, 3, -2\}$	3.937
$\text{stDevSamp}\{1, 3, 2, 5, -6, 4\}, \{3, 2, 5\}$	4.33345

$\text{stDevSamp}\begin{pmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{pmatrix}$	$[3.26599 \quad 2.94392 \quad 1.63299]$
$\text{stDevSamp}\begin{pmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{pmatrix}, \begin{pmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{pmatrix}$	$[2.52608 \quad 5.21506]$

Stop

Catalog >

Stop

Programming command: Terminates the program.

Stop is not allowed in functions.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing

instead of **enter** at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

$i:=0$	0
Define $prog1()$ =Prgm	Done
For $i,1,10,1$	
If $i=5$	
Stop	
EndFor	
EndPrgm	
$prog1()$	Done
i	5

Store

See → (store), page 129.

string()

Catalog >

string(*Expr*) ⇒ *string*

Simplifies *Expr* and returns the result as a character string.

$string(1.2345)$	"1.2345"
$string(1+2)$	"3"

subMat()

Catalog >

subMat(*Matrix1* [, *startRow*] [, *startCol*] [, *endRow*] [, *endCol*])
⇒ *matrix*

Returns the specified submatrix of *Matrix1*.

Defaults: *startRow*=1, *startCol*=1, *endRow*=last row, *endCol*=last column.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow m1$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
$subMat(m1,2,1,3,2)$	$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}$
$subMat(m1,2,2)$	$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$

Sum (Sigma)See $\Sigma()$, page 124.**sum()**

Catalog >

sum(*List* [, *Start* [, *End*]]) ⇒ *expression*

Returns the sum of all elements in *List*.

Start and *End* are optional. They specify a range of elements.

Any void argument produces a void result. Empty (void) elements in *List* are ignored. For more information on empty elements, see page 131.

$sum(\{1,2,3,4,5\})$	15
$sum(\{a,2 \cdot a,3 \cdot a\})$	"Error: Variable is not defined"
$sum(seq(n,n,1,10))$	55
$sum(\{1,3,5,7,9\},3)$	21

sum()Catalog > **sum**(*Matrix1*[, *Start*[, *End*]]) ⇒ *matrix*Returns a row vector containing the sums of all elements in the columns in *Matrix1*.*Start* and *End* are optional. They specify a range of rows.Any void argument produces a void result. Empty (void) elements in *Matrix1* are ignored. For more information on empty elements, see page 131.

sum	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$[5 \ 7 \ 9]$
sum	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$[12 \ 15 \ 18]$
sum	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, 2, 3$	$[11 \ 13 \ 15]$

sumIf()Catalog > **sumIf**(*List*, *Criteria*[, *SumList*]) ⇒ *value*Returns the accumulated sum of all elements in *List* that meet the specified *Criteria*. Optionally, you can specify an alternate list, *sumList*, to supply the elements to accumulate.*List* can be an expression, list, or matrix. *SumList*, if specified, must have the same dimension(s) as *List*.*Criteria* can be:

- A value, expression, or string. For example, **34** accumulates only those elements in *List* that simplify to the value 34.
- A Boolean expression containing the symbol ? as a placeholder for each element. For example, **?<10** accumulates only those elements in *List* that are less than 10.

When a *List* element meets the *Criteria*, the element is added to the accumulating sum. If you include *sumList*, the corresponding element from *sumList* is added to the sum instead.Within the Lists & Spreadsheet application, you can use a range of cells in place of *List* and *sumList*.

Empty (void) elements are ignored. For more information on empty elements, see page 131.

Note: See also **countIf()**, page 23.

sumIf	$\{ \{ 1, 2, e, 3, \pi, 4, 5, 6 \}, 2.5 < ? < 4.5 \}$	12.859874482
sumIf	$\{ \{ 1, 2, 3, 4 \}, 2 < ? < 5, \{ 10, 20, 30, 40 \} \}$	70

sumSeq()See $\Sigma()$, page 124.**system()**Catalog > **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)**Catalog > *Matrix1*^T ⇒ *matrix*Returns the complex conjugate transpose of *Matrix1*.**Note:** You can insert this operator from the computer keyboard by typing @t.

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

tan() **key****tan**(*Value1*) \Rightarrow *value***tan**(*List1*) \Rightarrow *list***tan**(*Value1*) 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 $^{\circ}$, $^{\text{G}}$ or $^{\text{r}}$ to override the angle mode setting temporarily.

In Degree angle mode:

$\tan\left(\left(\frac{\pi}{4}\right)^{\text{r}}\right)$	1.
--	----

$\tan(45)$	1.
------------	----

$\tan(\{0,60,90\})$	$\{0.,1.73205,\text{undef}\}$
---------------------	-------------------------------

In Gradian angle mode:

$\tan\left(\left(\frac{\pi}{4}\right)^{\text{r}}\right)$	1.
--	----

$\tan(50)$	1.
------------	----

$\tan(\{0,50,100\})$	$\{0.,1.,\text{undef}\}$
----------------------	--------------------------

In Radian angle mode:

$\tan\left(\frac{\pi}{4}\right)$	1.
----------------------------------	----

$\tan(45^{\circ})$	1.
--------------------	----

$\tan\left(\left\{\pi,\frac{\pi}{3},\pi,\frac{\pi}{4}\right\}\right)$	$\{0.,1.73205,0.,1.\}$
---	------------------------

tan(*squareMatrix1*) \Rightarrow *squareMatrix*Returns the matrix tangent of *squareMatrix1*. This is not the same as calculating the tangent of each element. For information about the calculation method, refer to **cos()**.*squareMatrix1* must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

$\tan\left(\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}\right)$	$\begin{bmatrix} -28.2912 & 26.0887 & 11.1142 \\ 12.1171 & -7.83536 & -5.48138 \\ 36.8181 & -32.8063 & -10.4594 \end{bmatrix}$
---	--

tan⁻¹() **key****tan⁻¹**(*Value1*) \Rightarrow *value***tan⁻¹**(*List1*) \Rightarrow *list***tan⁻¹**(*Value1*) returns the angle whose tangent is *Value1*.**tan⁻¹**(*List1*) returns a list of the inverse tangents of each element of *List1*.**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.**Note:** You can insert this function from the keyboard by typing **arctan (...)**.

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

$\tan^{-1}(\{0,0.2,0.5\})$	$\{0,0.197396,0.463648\}$
----------------------------	---------------------------

tan⁻¹()

trig key

tan⁻¹(squareMatrix1) ⇒ squareMatrix

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

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

In Radian angle mode:

$$\tan^{-1} \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \begin{bmatrix} -0.083658 & 1.26629 & 0.62263 \\ 0.748539 & 0.630015 & -0.070012 \\ 1.68608 & -1.18244 & 0.455126 \end{bmatrix}$$

tanh()Catalog > **tanh(Value1)** ⇒ value**tanh(List1)** ⇒ list**tanh(Value1)** returns the hyperbolic tangent of the argument.**tanh(List1)** returns a list of the hyperbolic tangents of each element of *List1*.**tanh(squareMatrix1)** ⇒ squareMatrix

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

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

tanh(1.2)	0.833655
tanh({0,1})	{0,.0761594}

In Radian angle mode:

$$\tanh \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \begin{bmatrix} -0.097966 & 0.933436 & 0.425972 \\ 0.488147 & 0.538881 & -0.129382 \\ 1.28295 & -1.03425 & 0.428817 \end{bmatrix}$$

tanh⁻¹()Catalog > **tanh⁻¹(Value1)** ⇒ value**tanh⁻¹(List1)** ⇒ list**tanh⁻¹(Value1)** 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(...)**.

tanh⁻¹(squareMatrix1) ⇒ squareMatrix

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

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

In Rectangular complex format:

tanh ⁻¹ (0)	0.
tanh ⁻¹ {1,2,1,3}	{undef,0.518046-1.5708 <i>i</i> ,0.346574-1.5708 <i>i</i> }

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

In Radian angle mode and Rectangular complex format:

$$\tanh^{-1} \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \begin{bmatrix} -0.099353+0.164058*i* & 0.267834-1.4908*i* \\ -0.087596-0.725533*i* & 0.479679-0.94730*i* \\ 0.511463-2.08316*i* & -0.878563+1.7901*i* \end{bmatrix}$$

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

tCdf()

Catalog > 

tCdf(*lowBound*,*upBound*,*df*) \Rightarrow number if *lowBound* and *upBound* are numbers, list if *lowBound* and *upBound* are lists

Computes the Student-*t* distribution probability between *lowBound* and *upBound* for the specified degrees of freedom *df*.

For $P(X \leq \textit{upBound})$, set *lowBound* = $-9E999$.

Text

Catalog > 

Text *promptString* [, *DispFlag*]

Programming command: Pauses the program and displays the character string *promptString* in a dialog box.

When the user selects **OK**, program execution continues.


The optional *flag* argument can be any expression.

- If *DispFlag* is omitted or evaluates to **1**, the text message is added to the Calculator history.
- If *DispFlag* evaluates to **0**, the text message is not added to the history.

If the program needs a typed response from the user, refer to **Request**, page 83, or **RequestStr**, page 84.

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

Define a program that pauses to display each of five random numbers in a dialog box.

Within the Prgm...EndPrgm template, complete each line by pressing  instead of **enter**. On the computer keyboard, hold down **Alt** and press **Enter**.

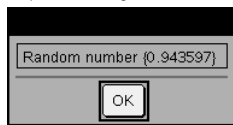
Define text_demo()=Prgm

```
For i,1,5
  strInfo:="Random number " & string(rand(i))
  Text strInfo
Next
EndPrgm
```

Run the program:

text_demo()

Sample of one dialog box:



Then

See **If**, page 45.

Interval

Catalog > 

Interval *List* [, *Freq* [, *CLevel*]]

(Data list input)

Interval \bar{x} , *sx*, *n* [, *CLevel*]

(Summary stats input)

Computes a *t* confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
stat. \bar{x}	Sample mean of the data sequence from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom
stat. σ_x	Sample standard deviation

Output variable	Description
stat.n	Length of the data sequence with sample mean

tInterval_2Samp

Catalog > 

tInterval_2Samp

List1,List2[,Freq1[,Freq2[,CLevel[,Pooled]]]]

(Data list input)

tInterval_2Samp $\bar{x}_1, s_1, n_1, \bar{x}_2, s_2, n_2[, CLevel[, Pooled]]$

(Summary stats input)

Computes a two-sample t confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. $\bar{x}_1 - \bar{x}_2$	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat.df	Degrees of freedom
stat. \bar{x}_1 , stat. \bar{x}_2	Sample means of the data sequences from the normal random distribution
stat. s_1 , stat. s_2	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 <i>Pooled</i> = YES

tPdf()

Catalog > 

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

Computes the probability density function (pdf) for the Student- t distribution at a specified x value with specified degrees of freedom df .

trace()

Catalog > 

trace(*squareMatrix*) \Rightarrow value



Returns the trace (sum of all the elements on the main diagonal) of *squareMatrix*.

trace $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	15
$a:=12$	12
trace $\begin{pmatrix} a & 0 \\ 1 & a \end{pmatrix}$	24

Try*block1***Else***block2***EndTry**

Executes *block1* unless an error occurs. Program execution transfers to *block2* if an error occurs in *block1*. System variable *errCode* contains the error code to allow the program to perform error recovery. For a list of error codes, see "Error codes and messages," page 137.

block1 and *block2* can be either a single statement or a series of statements separated with the ";" character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Define *prog1()*=Prgm

Try

z:=z+1

Disp "z incremented."

Else

Disp "Sorry, z undefined."

EndTry

EndPrgm

Done

z:=1:prog1()

z incremented.

Done

DelVar z:prog1()

Sorry, z undefined.

Done

Example 2

To see the commands **Try**, **ClrErr**, and **PassErr** in operation, enter the *eigenvals()* program shown at the right. Run the program by executing each of the following expressions.

$$\text{eigenvals}\left(\begin{bmatrix} -3 \\ -41 \\ 5 \end{bmatrix}, \begin{bmatrix} -1 & 2 & -3.1 \end{bmatrix}\right)$$

Note: See also **ClrErr**, page 17, and **PassErr**, page 73.

Define *eigenvals(a,b)*=Prgm© Program *eigenvals(A,B)* displays eigenvalues of A-B

Try

Disp "A=" ,a

Disp "B=" ,b

Disp " "

Disp "Eigenvalues of A-B are:" ,eigVl(a*b)

Else

If errCode=230 Then

Disp "Error: Product of A-B must be a square matrix"

ClrErr

Else

PassErr

EndIf

EndTry

EndPrgm

tTest**tTest** μ_0 ,List[,Freq[,Hypoth]]

(Data list input)

tTest μ_0 , \bar{x} ,s \bar{x} ,n[,Hypoth]

(Summary stats input)

Performs a hypothesis test for a single unknown population mean μ when the population standard deviation σ is unknown. A summary of results is stored in the *stat.results* variable. (See page 97.)

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" on page 131.

Output variable	Description
stat.t	$(\bar{x} - \mu_0) / (\text{stdev} / \sqrt{\text{qt}(n)})$
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat.df	Degrees of freedom
stat. \bar{x}	Sample mean of the data sequence in <i>List</i>
stat.sx	Sample standard deviation of the data sequence
stat.n	Size of the sample

tTest_2Samp

Catalog > 

tTest_2Samp *List1, List2, Freq1, Freq2, Hypoth, Pooled*]]]

(Data list input)

tTest_2Samp $\bar{x}1, sx1, n1, \bar{x}2, sx2, n2, Hypoth, Pooled$]

(Summary stats input)

Computes a two-sample *t* test. A summary of results is stored in the *stat.results* variable. (See page 97.)

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 $H_a: \mu_1 > \mu_2$, 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" on page 131.

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 <i>t</i> -statistic
stat. $\bar{x}1$, stat. $\bar{x}2$	Sample means of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Size of the samples
stat.sp	The pooled standard deviation. Calculated when <i>Pooled</i> =1.

tvmFV()

Catalog > 

tvmFV(*N, I, PV, Pmt, [PpY], [CpY], [PmtAt]*) \Rightarrow *value*

Financial function that calculates the future value of money.

$\text{tvmFV}(120, 5, 0, -500, 12, 12)$ 77641.1

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 107. See also **amortTbl()**, page 6.

tvmI()

Catalog >

tvmI($N, PV, Pmt, FV, [PpY], [CpY], [PmtAt]$) \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 107. See also **amortTbl()**, page 6.**tvmN()**

Catalog >

tvmN($I, PV, Pmt, FV, [PpY], [CpY], [PmtAt]$) \Rightarrow *value*

tvmN(5, -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 107. See also **amortTbl()**, page 6.**tvmPmt()**

Catalog >

tvmPmt($N, I, PV, FV, [PpY], [CpY], [PmtAt]$) \Rightarrow *value*

tvmPmt(60, 4, 30000, 0, 12, 12) -552.496

Financial function that calculates the amount of each payment.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 107. See also **amortTbl()**, page 6.**tvmPV()**

Catalog >

tvmPV($N, I, Pmt, FV, [PpY], [CpY], [PmtAt]$) \Rightarrow *value*

tvmPV(48, 4, -500, 30000, 12, 12) -3426.7

Financial function that calculates the present value.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 107. See also **amortTbl()**, page 6.

TVM argument*	Description	Data type
N	Number of payment periods	real number
I	Annual interest rate	real number
PV	Present value	real number
Pmt	Payment amount	real number
FV	Future value	real number
PpY	Payments per year, default=1	integer > 0
CpY	Compounding periods per year, default=1	integer > 0
$PmtAt$	Payment due at the end or beginning of each period, default=end	integer (0=end, 1=beginning)

* These time-value-of-money argument names are similar to the TVM variable names (such as **tvm.pv** and **tvm.pmt**) that are used by the Calculator application's finance solver. Financial functions, however, do not store their argument values or results to the TVM variables.

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

Calculates the TwoVar statistics. A summary of results is stored in the *stat.results* variable. (See page 97.)

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

X and Y are lists of independent and dependent variables.

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

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

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

An empty (void) element in any of the lists X , *Freq*, or *Category* results in a void for the corresponding element of all those lists. An empty element in any of the lists $X1$ through $X20$ results in a void for the corresponding element of all those lists. For more information on empty elements, see page 131.

Output variable	Description
stat. \bar{x}	Mean of x values
stat. Σx	Sum of x values
stat. Σx^2	Sum of x ² values
stat.sx	Sample standard deviation of x
stat. σ_x	Population standard deviation of x
stat.n	Number of data points
stat. \bar{y}	Mean of y values
stat. Σy	Sum of y values
stat. Σy^2	Sum of y ² values
stat.sy	Sample standard deviation of y
stat. σ_y	Population standard deviation of y
stat. Σxy	Sum of x · y values
stat.r	Correlation coefficient
stat.MinX	Minimum of x values
stat.Q ₁ X	1st Quartile of x
stat.MedianX	Median of x
stat.Q ₃ X	3rd Quartile of x
stat.MaxX	Maximum of x values
stat.MinY	Minimum of y values
stat.Q ₁ Y	1st Quartile of y

Output variable	Description
stat.MedY	Median of y
stat.Q3Y	3rd Quartile of y
stat.MaxY	Maximum of y values
stat. $\sum(x-\bar{x})^2$	Sum of squares of deviations from the mean of x
stat. $\sum(y-\bar{y})^2$	Sum of squares of deviations from the mean of y

U

unitV()

Catalog > 

unitV(*Vector1*) ⇒ *vector*

Returns either a row- or column-unit vector, depending on the form of *Vector1*.

Vector1 must be either a single-row matrix or a single-column matrix.

unitV($\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$)	$\begin{bmatrix} 0.408248 & 0.816497 & 0.408248 \end{bmatrix}$
unitV($\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$)	$\begin{bmatrix} 0.267261 \\ 0.534522 \\ 0.801784 \end{bmatrix}$

unLock

Catalog > 

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

unLock *Var*.

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

See **Lock**, page 57, and **getLockInfo()**, page 42.

<i>a</i> :=65	65
Lock <i>a</i>	Done
getLockInfo(<i>a</i>)	1
<i>a</i> :=75	"Error: Variable is locked."
DelVar <i>a</i>	"Error: Variable is locked."
Unlock <i>a</i>	Done
<i>a</i> :=75	75
DelVar <i>a</i>	Done

V

varPop()

Catalog > 

varPop(*List*, *freqList*) ⇒ *expression*

Returns the population variance of *List*.

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

Note: *List* must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 131.

varPop($\{5,10,15,20,25,30\}$)	72.9167
----------------------------------	---------

varSamp()

Catalog >

varSamp(*List*₁, *freqList*) ⇒ *expression*Returns the sample variance of *List*.Each *freqList* element counts the number of consecutive occurrences of the corresponding element in *List*.**Note:** *List* must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 131.

varSamp(*MatrixI*₁, *freqMatrix*) ⇒ *matrix*Returns a row vector containing the sample variance of each column in *MatrixI*.Each *freqMatrix* element counts the number of consecutive occurrences of the corresponding element in *MatrixI*.

If an element in either matrix is empty (void), that element is ignored, and the corresponding element in the other matrix is also ignored. For more information on empty elements, see page 131.

Note: *MatrixI* must contain at least two rows.

$\text{varSamp}\{\{1,2,5,-6,3,-2\}\}$	$\frac{31}{2}$
$\text{varSamp}\{\{1,3,5\},\{4,6,2\}\}$	$\frac{68}{33}$

$\text{varSamp}\begin{pmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ .5 & .7 & 3 \end{pmatrix}$	$[4.75 \ 1.03 \ 4]$
$\text{varSamp}\begin{pmatrix} -1.1 & 2.2 \\ 3.4 & 5.1 \\ -2.3 & 4.3 \end{pmatrix}, \begin{pmatrix} 6 & 3 \\ 2 & 4 \\ 5 & 1 \end{pmatrix}$	$[3.91731 \ 2.08411]$

W**warnCodes()**

Catalog >

warnCodes(*Expr1*, *StatusVar*) ⇒ *expression*Evaluates expression *Expr1*, returns the result, and stores the codes of any generated warnings in the *StatusVar* list variable. If no warnings are generated, this function assigns *StatusVar* an empty list.*Expr1* can be any valid TI-Nspire™ or TI-Nspire™ CAS math expression. You cannot use a command or assignment as *Expr1*.*StatusVar* must be a valid variable name.

For a list of warning codes and associated messages, see page 143.

$\text{warnCodes}\left(\text{solve}\left(\sin(10 \cdot x) = \frac{x^2}{x}, x\right), \text{warn}\right)$	
$x = -0.84232$ or $x = -0.706817$ or $x = -0.285234$ or $x = \emptyset$	
<i>warn</i>	$\{10007, 10009\}$

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

Catalog >



when(*Condition*, *trueResult* [, *falseResult*][, *unknownResult*]) ⇒ *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.


$\text{when}(x < 0, x + 3), x = 5$	undef
$\text{when}(n > 0, n \cdot \text{factorial}(n-1), 1) \rightarrow \text{factorial}(n)$	Done
$\text{factorial}(3)$	6
3!	6

WhileCatalog > **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: In the Calculator application on the handheld, you can enter multi-line definitions by pressing  instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

instead of  at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

```
Define sum_of_recip(n)=Func
```

```
Local i,tempsum
```

```
1 → i
```

```
0 → tempsum
```

```
While i ≤ n
```

```
tempsum + 1/i → tempsum
```

```
i + 1 → i
```

```
EndWhile
```

```
Return tempsum
```

```
EndFunc
```

```
Done
```

```
sum_of_recip(3)
```

```
11  
6
```

"With"See **|** ("with"), page 128.**X****xor**Catalog > 

BooleanExpr1 **xor** *BooleanExpr2* ⇒ *Boolean expression*

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 72.

Integer1 **xor** *Integer2* ⇒ *integer*

Compares two real integers bit-by-bit using an **xor** operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

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 12.

Note: See **or**, page 72.

true xor true	false
5 > 3 xor 3 > 5	true

In Hex base mode:

Important: Zero, not the letter O.

0h7AC36 xor 0h3D5F	0h79169
--------------------	---------

In Bin base mode:

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

Catalog > 

zInterval $\sigma, List[, Freq[, CLevel]]$

(Data list input)

zInterval $\sigma, \bar{x}, n [, CLevel]$

(Summary stats input)

Computes a z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval for an unknown population mean
stat. \bar{x}	Sample mean of the data sequence from the normal random distribution
stat.ME	Margin of error
stat.sx	Sample standard deviation
stat.n	Length of the data sequence with sample mean
stat. σ	Known population standard deviation for data sequence <i>List</i>

zInterval_1Prop

Catalog > 

zInterval_1Prop $x, n [, CLevel]$

Computes a one-proportion z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

x is a non-negative integer.

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

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 $x1, n1, x2, n2, [CLevel]$

Computes a two-proportion z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

$x1$ and $x2$ are non-negative integers.

For information on the effect of empty elements in a list, see “Empty (void) elements” on page 131.

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. \hat{p} 1	First sample proportion estimate
stat. \hat{p} 2	Second sample proportion estimate
stat.n1	Sample size in data sequence one
stat.n2	Sample size in data sequence two

zInterval_2Samp $\sigma_1, \sigma_2, List1, List2, Freq1, Freq2, [CLevel]$

(Data list input)

zInterval_2Samp $\sigma_1, \sigma_2, \bar{x}1, n1, \bar{x}2, n2, [CLevel]$

(Summary stats input)

Computes a two-sample z confidence interval. A summary of results is stored in the *stat.results* variable. (See page 97.)

For information on the effect of empty elements in a list, see “Empty (void) elements” on page 131.

Output variable	Description
stat.CLower, stat.CUpper	Confidence interval containing confidence level probability of distribution
stat. $\bar{x}1 - \bar{x}2$	Sample means of the data sequences from the normal random distribution
stat.ME	Margin of error
stat. $\bar{x}1$, stat. $\bar{x}2$	Sample means of the data sequences from the normal random distribution
stat. $\sigma x1$, stat. $\sigma x2$	Sample standard deviations for <i>List 1</i> and <i>List 2</i>
stat.n1, stat.n2	Number of samples in data sequences
stat.r1, stat.r2	Known population standard deviations for data sequence <i>List 1</i> and <i>List 2</i>

zTest $\mu_0, \sigma, List, [Freq, Hypoth]$

(Data list input)

zTest $\mu_0, \sigma, \bar{x}, n, [Hypoth]$

(Summary stats input)

Performs a z test with frequency *freqlist*. A summary of results is stored in the *stat.results* variable. (See page 97.)Test $H_0: \mu = \mu_0$, against one of the following:For $H_a: \mu < \mu_0$, set *Hypoth*<0For $H_a: \mu \neq \mu_0$ (default), set *Hypoth*=0For $H_a: \mu > \mu_0$, set *Hypoth*>0

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.z	$(\bar{x} - \mu_0) / (\sigma / \sqrt{n})$
stat.P Value	Least probability at which the null hypothesis can be rejected
stat. \bar{x}	Sample mean of the data sequence in <i>List</i>
stat.sx	Sample standard deviation of the data sequence. Only returned for <i>Data</i> input.
stat.n	Size of the sample

zTest_1Prop**zTest_1Prop** $p_0, x, n, [Hypoth]$ Computes a one-proportion z test. A summary of results is stored in the *stat.results* variable. (See page 97.)*x* is a non-negative integer.Test $H_0: p = p_0$ against one of the following:For $H_a: p > p_0$, set *Hypoth*>0For $H_a: p \neq p_0$ (default), set *Hypoth*=0For $H_a: p < p_0$, set *Hypoth*<0

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

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 $x1, n1, x2, n2[, Hypoth]$

Computes a two-proportion z test. A summary of results is stored in the *stat.results* variable. (See page 97.)

$x1$ 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" on page 131.

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**zTest_2Samp** $\sigma_1, \sigma_2, [List1, List2[, Freq1[, Freq2[, Hypoth]]]$

(Data list input)

zTest_2Samp $\sigma_1, \sigma_2, \bar{x}1, n1, \bar{x}2, n2[, Hypoth]$

(Summary stats input)

Computes a two-sample z test. A summary of results is stored in the *stat.results* variable. (See page 97.)

Test $H_0: \mu1 = \mu2$, against one of the following:

For $H_a: \mu1 < \mu2$, set *Hypoth*<0

For $H_a: \mu1 \neq \mu2$ (default), set *Hypoth*=0

For $H_a: \mu1 > \mu2$, *Hypoth*>0

For information on the effect of empty elements in a list, see "Empty (void) elements" on page 131.

Output variable	Description
stat.z	Standard normal value computed for the difference of means
stat.PVal	Smallest level of significance at which the null hypothesis can be rejected
stat. $\bar{x}1$, stat. $\bar{x}2$	Sample means of the data sequences in <i>List1</i> and <i>List2</i>
stat.sx1, stat.sx2	Sample standard deviations of the data sequences in <i>List1</i> and <i>List2</i>
stat.n1, stat.n2	Size of the samples

Symbols

+ (add)		+ key
$Value1 + Value2 \Rightarrow value$	56	56
Returns the sum of the two arguments.	56+4	60
	60+4	64
	64+4	68
	68+4	72
$List1 + List2 \Rightarrow list$		
$Matrix1 + Matrix2 \Rightarrow matrix$	$\left\{22, \pi, \frac{\pi}{2}\right\} \rightarrow I1$	$\{22, 3.14159, 1.5708\}$
Returns a list (or matrix) containing the sums of corresponding elements in <i>List1</i> and <i>List2</i> (or <i>Matrix1</i> and <i>Matrix2</i>).	$\left\{10, 5, \frac{\pi}{2}\right\} \rightarrow I2$	$\{10, 5, 1.5708\}$
Dimensions of the arguments must be equal.	$I1+I2$	$\{32, 8.14159, 3.14159\}$
$Value + List1 \Rightarrow list$	$15 + \{10, 15, 20\}$	$\{25, 30, 35\}$
$List1 + Value \Rightarrow list$	$\{10, 15, 20\} + 15$	$\{25, 30, 35\}$
Returns a list containing the sums of <i>Value</i> and each element in <i>List1</i> .	$20 + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 21 & 2 \\ 3 & 24 \end{bmatrix}$
$Value + Matrix1 \Rightarrow matrix$		
$Matrix1 + Value \Rightarrow matrix$		
Returns a matrix with <i>Value</i> added to each element on the diagonal of <i>Matrix1</i> . <i>Matrix1</i> must be square.		

Note: Use **⋅+** (dot plus) to add an expression to each element.

− (subtract)		− key
$Value1 - Value2 \Rightarrow value$	6−2	4
Returns <i>Value1</i> minus <i>Value2</i> .	$\pi - \frac{\pi}{6}$	2.61799
$List1 - List2 \Rightarrow list$		
$Matrix1 - Matrix2 \Rightarrow matrix$	$\left\{22, \pi, \frac{\pi}{2}\right\} - \left\{10, 5, \frac{\pi}{2}\right\}$	$\{12, -1.85841, 0\}$
Subtracts each element in <i>List2</i> (or <i>Matrix2</i>) from the corresponding element in <i>List1</i> (or <i>Matrix1</i>), and returns the results.	$\begin{bmatrix} 3 & 4 \end{bmatrix} - \begin{bmatrix} 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 \end{bmatrix}$
Dimensions of the arguments must be equal.		
$Value - List1 \Rightarrow list$	$15 - \{10, 15, 20\}$	$\{5, 0, -5\}$
$List1 - Value \Rightarrow list$	$\{10, 15, 20\} - 15$	$\{-5, 0, 5\}$
Subtracts each <i>List1</i> element from <i>Value</i> or subtracts <i>Value</i> from each <i>List1</i> element, and returns a list of the results.		

- (subtract)**- key** $Value - Matrix1 \Rightarrow matrix$ $Matrix1 - Value \Rightarrow matrix$

$20 - \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 19 & -2 \\ -3 & 16 \end{bmatrix}$
---	--

$Value - Matrix1$ returns a matrix of $Value$ times the identity matrix minus $Matrix1$. $Matrix1$ must be square.

$Matrix1 - Value$ 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.

· (multiply)**· key** $Value1 \cdot Value2 \Rightarrow value$

Returns the product of the two arguments.

$2 \cdot 3.45$	6.9
----------------	-------

 $List1 \cdot List2 \Rightarrow list$ Returns a list containing the products of the corresponding elements in $List1$ and $List2$.

$\{1,2,3\} \cdot \{4,5,6\}$	$\{4,10,18\}$
-----------------------------	---------------

Dimensions of the lists must be equal.

 $Matrix1 \cdot Matrix2 \Rightarrow matrix$ Returns the matrix product of $Matrix1$ and $Matrix2$.The number of columns in $Matrix1$ must equal the number of rows in $Matrix2$.

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 7 & 8 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 42 & 48 \\ 105 & 120 \end{bmatrix}$
--	--

 $Value \cdot List1 \Rightarrow list$ $List1 \cdot Value \Rightarrow list$ Returns a list containing the products of $Value$ and each element in $List1$.

$\pi \cdot \{4,5,6\}$	$\{12.5664, 15.708, 18.8496\}$
-----------------------	--------------------------------

 $Value \cdot Matrix1 \Rightarrow matrix$ $Matrix1 \cdot Value \Rightarrow matrix$ Returns a matrix containing the products of $Value$ and each element in $Matrix1$.

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 0.01$	$\begin{bmatrix} 0.01 & 0.02 \\ 0.03 & 0.04 \end{bmatrix}$
---	--

Note: Use \cdot (dot multiply) to multiply an expression by each element.

$6 \cdot \text{identity}(3)$	$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix}$
------------------------------	---

/ (divide)**÷ key** $Value1 / Value2 \Rightarrow value$ Returns the quotient of $Value1$ divided by $Value2$.

$\frac{2}{3.45}$	$.57971$
------------------	----------

Note: See also **Fraction template**, page 1.

 $List1 / List2 \Rightarrow list$ Returns a list containing the quotients of $List1$ divided by $List2$.

$\frac{\{1,2,3\}}{\{4,5,6\}}$	$\left\{0.25, \frac{2}{5}, \frac{1}{2}\right\}$
-------------------------------	---

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 $Value$.

$\frac{6}{\{3,6,\sqrt{6}\}}$	$\{2, 1, 2.44949\}$
------------------------------	---------------------

$\frac{\{7,9,2\}}{7 \cdot 9 \cdot 2}$	$\left\{\frac{1}{18}, \frac{1}{14}, \frac{1}{63}\right\}$
---------------------------------------	---

/ (divide)

 \div key $Value / Matrix1 \Rightarrow matrix$ $Matrix1 / Value \Rightarrow matrix$

$$\frac{\begin{bmatrix} 7 & 9 & 2 \end{bmatrix}}{7 \cdot 9 \cdot 2} \qquad \begin{bmatrix} \frac{1}{18} & \frac{1}{14} & \frac{1}{63} \end{bmatrix}$$

Returns a matrix containing the quotients of $Matrix1/Value$.**Note:** Use $\cdot /$ (dot divide) to divide an expression by each element.

^ (power)

 \wedge key $Value1 \wedge Value2 \Rightarrow value$ $List1 \wedge List2 \Rightarrow list$

$$4^2 \qquad 16$$

$$\{2,4,6\} \{1,2,3\} \qquad \{2,16,216\}$$

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.

 $Value \wedge List1 \Rightarrow list$ Returns $Value$ raised to the power of the elements in $List1$.

$$\pi \{1,2,3\} \qquad \{3.14159,9.8696,0.032252\}$$

 $List1 \wedge Value \Rightarrow list$ Returns the elements in $List1$ raised to the power of $Value$.

$$\{1,2,3,4\}^{-2} \qquad \left\{1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}\right\}$$

 $squareMatrix1 \wedge integer \Rightarrow matrix$ 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.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^2 \qquad \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} \qquad \begin{bmatrix} -2 & 1 \\ 3 & -1 \\ 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-2} \qquad \begin{bmatrix} 11 & -5 \\ 2 & 2 \\ -15 & 7 \\ 4 & 4 \end{bmatrix}$$

 x^2 (square) x^2 key $Value^2 \Rightarrow value$

Returns the square of the argument.

 $List1^2 \Rightarrow list$ Returns a list containing the squares of the elements in $List1$. $squareMatrix1^2 \Rightarrow matrix$ Returns the matrix square of $squareMatrix1$. This is not the same as calculating the square of each element. Use $\wedge 2$ to calculate the square of each element.

.+ (dot add)  **keys***Matrix1* .+ *Matrix2* \Rightarrow *matrix**Value* .+ *Matrix1* \Rightarrow *matrix*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .+ \begin{bmatrix} 10 & 30 \\ 20 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} 11 & 32 \\ 23 & 44 \end{bmatrix}$$

Matrix1 .+ *Matrix2* returns a matrix that is the sum of each pair of corresponding elements in *Matrix1* and *Matrix2*.*Value* .+ *Matrix1* returns a matrix that is the sum of *Value* and each element in *Matrix1*.



$$5 .+ \begin{bmatrix} 10 & 30 \\ 20 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} 15 & 35 \\ 25 & 45 \end{bmatrix}$$

.- (dot sub.)  **keys***Matrix1* .- *Matrix2* \Rightarrow *matrix**Value* .- *Matrix1* \Rightarrow *matrix*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .- \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} -9 & -18 \\ -27 & -36 \end{bmatrix}$$

Matrix1 .- *Matrix2* returns a matrix that is the difference between each pair of corresponding elements in *Matrix1* and *Matrix2*.*Value* .- *Matrix1* returns a matrix that is the difference of *Value* and each element in *Matrix1*.



$$5 .- \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} -5 & -15 \\ -25 & -35 \end{bmatrix}$$

.· (dot mult.)  **keys***Matrix1* .· *Matrix2* \Rightarrow *matrix**Value* .· *Matrix1* \Rightarrow *matrix*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .· \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} 10 & 40 \\ 90 & 160 \end{bmatrix}$$

Matrix1 .· *Matrix2* returns a matrix that is the product of each pair of corresponding elements in *Matrix1* and *Matrix2*.*Value* .· *Matrix1* returns a matrix containing the products of *Value* and each element in *Matrix1*.

$$5 .· \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} 50 & 100 \\ 150 & 200 \end{bmatrix}$$

.| (dot divide)  **keys***Matrix1* .| *Matrix2* \Rightarrow *matrix**Value* .| *Matrix1* \Rightarrow *matrix*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .| \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} \end{bmatrix}$$

Matrix1 .| *Matrix2* returns a matrix that is the quotient of each pair of corresponding elements in *Matrix1* and *Matrix2*.*Value* .| *Matrix1* returns a matrix that is the quotient of *Value* and each element in *Matrix1*.

$$5 .| \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{6} & \frac{1}{8} \end{bmatrix}$$

.^ (dot power)  **keys***Matrix1* .^ *Matrix2* \Rightarrow *matrix**Value* .^ *Matrix1* \Rightarrow *matrix*

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .^ \begin{bmatrix} 0 & 2 \\ 3 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 4 \\ 27 & \frac{1}{4} \end{bmatrix}$$

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*.

$$5 .^ \begin{bmatrix} 0 & 2 \\ 3 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 25 \\ 125 & \frac{1}{5} \end{bmatrix}$$

= (equal)

= key

 $Expr1 = Expr2 \Rightarrow$ Boolean expression $List1 = List2 \Rightarrow$ Boolean list $Matrix1 = Matrix2 \Rightarrow$ Boolean matrixReturns 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.

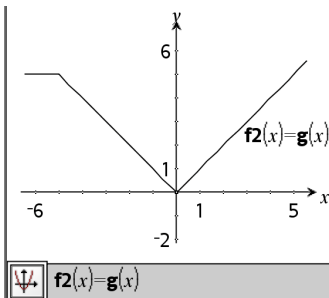
For lists and matrices, returns comparisons element by element.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing $\left[\downarrow \right]$ instead of $\left[\text{enter} \right]$ at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

Example function that uses math 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>=0 and x<=10 Then
    Return x
  ElseIf x=10 Then
    Return 3
  EndIf
EndFunc
```

Done

Result of graphing $g(x)$  **\neq (not equal)**

ctrl = keys

 $Expr1 \neq Expr2 \Rightarrow$ Boolean expression $List1 \neq List2 \Rightarrow$ Boolean list $Matrix1 \neq Matrix2 \Rightarrow$ Boolean matrixReturns true if $Expr1$ is determined to be not equal to $Expr2$.Returns false if $Expr1$ is determined to be equal to $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing \neq

See "=" (equal) example.

< (less than)

ctrl = keys

 $Expr1 < Expr2 \Rightarrow$ Boolean expression $List1 < List2 \Rightarrow$ Boolean list $Matrix1 < Matrix2 \Rightarrow$ Boolean matrixReturns true if $Expr1$ is determined to be less than $Expr2$.Returns false if $Expr1$ is determined to be greater than or equal to $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

See "=" (equal) example.

≤ (less or equal)

ctrl [=] keys

 $Expr1 \leq Expr2 \Rightarrow$ Boolean expression

See "=" (equal) example.

 $List1 \leq List2 \Rightarrow$ Boolean list $Matrix1 \leq Matrix2 \Rightarrow$ Boolean matrixReturns 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 [=] keys

 $Expr1 > Expr2 \Rightarrow$ Boolean expression

See "=" (equal) example.

 $List1 > List2 \Rightarrow$ Boolean list $Matrix1 > Matrix2 \Rightarrow$ Boolean matrixReturns 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)

ctrl [=] keys

 $Expr1 \geq Expr2 \Rightarrow$ Boolean expression

See "=" (equal) example.

 $List1 \geq List2 \Rightarrow$ Boolean list $Matrix1 \geq Matrix2 \Rightarrow$ Boolean matrixReturns 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 >=**! (factorial)**

ctrl [!] key

 $Value! \Rightarrow$ value $List! \Rightarrow$ list $Matrix! \Rightarrow$ matrix

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

5!	120
$\{5,4,3\}!$	$\{120,24,6\}$
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}!$	$\begin{bmatrix} 1 & 2 \\ 6 & 24 \end{bmatrix}$

& (append)

ctrl [⌘] keys

 $String1 \& String2 \Rightarrow$ stringReturns a text string that is $String2$ appended to $String1$.

"Hello "&"Nick" "Hello Nick"

d() (derivative)

Catalog >

 $d(\text{Expr1}, \text{Var} [, \text{Order}]) | \text{Var}=\text{value} \Rightarrow \text{value}$ $d(\text{Expr1}, \text{Var} [, \text{Order}]) \Rightarrow \text{value}$ $d(\text{List1}, \text{Var} [, \text{Order}]) \Rightarrow \text{list}$ $d(\text{Matrix1}, \text{Var} [, \text{Order}]) \Rightarrow \text{matrix}$

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()**.

$\frac{d}{dx} \left(x \right)_{x=0}$	undef
---	-------

$x:=0: \frac{d}{dx} \left(x \right)$	undef
---	-------

$x:=3: \frac{d}{dx} \left(\left\{ x^2, x^3, x^4 \right\} \right)$	$\{6, 27, 108\}$
--	------------------

$\frac{d}{dx} \left\{ x \cdot (x^2 + x)^{\frac{1}{3}} \right\} \Big _{x=0}$	undef
---	-------

$\text{centralDiff} \left\{ x \cdot (x^2 + x)^{\frac{1}{3}}, x \right\} \Big _{x=0}$	0.000033
--	----------

∫() (integral)

Catalog >

 $\int(\text{Expr1}, \text{Var}, \text{Lower}, \text{Upper}) \Rightarrow \text{value}$

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 **nInt()**.

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

Note: See also **nInt()**, page 68, and **Definite integral template**, page 5.

$\int_0^1 x^2 dx$	0.333333
-------------------	----------

√() (square root) **keys** $\sqrt{\text{Value1}} \Rightarrow \text{value}$ $\sqrt{\text{List1}} \Rightarrow \text{list}$

Returns the square root of the argument.

For a list, returns the square roots of all the elements in *List1*.

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

Note: See also **Square root template**, page 1.

$\sqrt{4}$	2
$\sqrt{\{9, 2, 4\}}$	$\{3, 1.41421, 2\}$

$\prod()$ (prodSeq)

Catalog >

 $\prod(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \text{expression}$ **Note:** You can insert this function from the keyboard by typing **prodSeq (...)**.Evaluates *Expr1* for each value of *Var* from *Low* to *High*, and returns the product of the results.**Note:** See also **Product template** (\prod), page 4.

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

$$\prod_{n=1}^5 \left\{ \left(\frac{1}{n}, n, 2 \right) \right\} \quad \left\{ \frac{1}{120}, 120, 32 \right\}$$

 $\prod(\text{Expr1}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 1$ $\prod(\text{Expr1}, \text{Var}, \text{Low}, \text{High})$ $\Rightarrow 1/\prod(\text{Expr1}, \text{Var}, \text{High}+1, \text{Low}-1)$ if $\text{High} < \text{Low}-1$

The product formulas used are derived from the following reference:

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Reading, Massachusetts: Addison-Wesley, 1994.

$$\prod_{k=4}^3 (k) \quad 1$$

$$\prod_{k=4}^1 \left(\frac{1}{k} \right) \quad 6$$

$$\prod_{k=4}^1 \left(\frac{1}{k} \right) \cdot \prod_{k=2}^4 \left(\frac{1}{k} \right) \quad \frac{1}{4}$$

 $\Sigma()$ (sumSeq)

Catalog >

 $\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \text{expression}$ **Note:** You can insert this function from the keyboard by typing **sumSeq (...)**.Evaluates *Expr1* for each value of *Var* from *Low* to *High*, and returns the sum of the results.**Note:** See also **Sum template**, page 4. $\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 0$ $\Sigma(\text{Expr1}, \text{Var}, \text{Low}, \text{High})$ $\Rightarrow -\Sigma(\text{Expr1}, \text{Var}, \text{High}+1, \text{Low}-1)$ if $\text{High} < \text{Low}-1$

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.

$$\sum_{n=1}^5 \left(\frac{1}{n} \right) \quad \frac{137}{60}$$

$$\sum_{k=4}^3 (k) \quad 0$$

$$\sum_{k=4}^1 (k) \quad -5$$

$$\sum_{k=4}^1 (k) + \sum_{k=2}^4 (k) \quad 4$$

ΣInt()

Catalog >

 $\Sigma\text{Int}(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) \Rightarrow \text{value}$
 $\Sigma\text{Int}(1,3,12,4.75,20000,,12,12)$ -213.48

 $\Sigma\text{Int}(NPmt1, NPmt2, amortTable) \Rightarrow \text{value}$

Amortization function that calculates the sum of the interest during a specified range of payments.

 $tbl:=\text{amortTbl}(12,12,4.75,20000,,12,12)$

$NPmt1$ and $NPmt2$ define the start and end boundaries of the payment range.

0	0.	0.	20000.
1	-77.49	-1632.43	18367.6
2	-71.17	-1638.75	16728.8
3	-64.82	-1645.1	15083.7
4	-58.44	-1651.48	13432.2
5	-52.05	-1657.87	11774.4
6	-45.62	-1664.3	10110.1
7	-39.17	-1670.75	8439.32
8	-32.7	-1677.22	6762.1
9	-26.2	-1683.72	5078.38
10	-19.68	-1690.24	3388.14
11	-13.13	-1696.79	1691.35
12	-6.55	-1703.37	-12.02

N , I , PV , Pmt , FV , PpY , CpY , and $PmtAt$ are described in the table of TVM arguments, page 107.

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

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

$\Sigma\text{Int}(NPmt1, NPmt2, amortTable)$ calculates the sum of the interest based on amortization table $amortTable$. The $amortTable$ argument must be a matrix in the form described under **amortTbl()**, page 6.

Note: See also **ΣPrn()**, below, and **Bal()**, page 12.

 $\Sigma\text{Int}(1,3,tbl)$ -213.48
ΣPrn()

Catalog >

 $\Sigma\text{Prn}(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) \Rightarrow \text{value}$
 $\Sigma\text{Prn}(1,3,12,4.75,20000,,12,12)$ -4916.28

 $\Sigma\text{Prn}(NPmt1, NPmt2, amortTable) \Rightarrow \text{value}$

Amortization function that calculates the sum of the principal during a specified range of payments.

 $tbl:=\text{amortTbl}(12,12,4.75,20000,,12,12)$

$NPmt1$ and $NPmt2$ define the start and end boundaries of the payment range.

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

N , I , PV , Pmt , FV , PpY , CpY , and $PmtAt$ are described in the table of TVM arguments, page 107.

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

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

$\Sigma\text{Prn}(NPmt1, NPmt2, amortTable)$ calculates the sum of the principal paid based on amortization table $amortTable$. The $amortTable$ argument must be a matrix in the form described under **amortTbl()**, page 6.

Note: See also **ΣInt()**, above, and **Bal()**, page 12.

 $\Sigma\text{Prn}(1,3,tbl)$ -4916.28

(indirection)ctrl  keys# *varNameString*Refers to the variable whose name is *varNameString*. This lets you use strings to create variable names from within a function.

$xyz:=12$	12
$\#{"x" \& "y" \& "z"}$	12

Creates or refers to the variable xyz.

$10 \rightarrow r$	10
$"r" \rightarrow s1$	"r"
$\#s1$	10

Returns the value of the variable (r) whose name is stored in variable s1.

E (scientific notation) key*mantissa*E*exponent*Enters a number in scientific notation. The number is interpreted as *mantissa* $\times 10^{\text{exponent}}$.Hint: If you want to enter a power of 10 without causing a decimal value result, use 10^{\wedge} *integer*.**Note:** You can insert this operator from the computer keyboard by typing @E. for example, type 2.3@E4 to enter 2.3E4.

23000.	23000.
2300000000.+4.1E15	4.1E15
$3 \cdot 10^4$	30000

g (gradian) key*Expr1*^g \Rightarrow *expression**List1*^g \Rightarrow *list**Matrix1*^g \Rightarrow *matrix*

This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies *Expr1* by $\pi/200$.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.

In Degree, Gradian or Radian mode:

$\cos(50^g)$	0.707107
$\cos(\{0,100^g,200^g\})$	{1.,0.,-1.}

r (radian) key*Value1*^r \Rightarrow *value**List1*^r \Rightarrow *list**Matrix1*^r \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 Gradian mode, multiplies the argument by 200/ π .Hint: Use ^r if you want to force radians in a function definition regardless of the mode that prevails when the function is used.**Note:** You can insert this symbol from the computer keyboard by typing @r.

In Degree, Gradian or Radian angle mode:

$\cos\left(\frac{\pi}{4^r}\right)$	0.707107
$\cos\left(\left\{0^r, \left(\frac{\pi}{12}\right)^r, -(\pi)^r\right\}\right)$	{1.,0.965926,-1.}

° (degree)

key

$Value1^\circ \Rightarrow value$

$List1^\circ \Rightarrow list$

$Matrix1^\circ \Rightarrow matrix$

This function gives you a way to specify a degree angle while in Gradian or Radian mode.

In Radian angle mode, multiplies the argument by $\pi/180$.

In Degree angle mode, returns the argument unchanged.

In Gradian angle mode, multiplies the argument by 10/9.

Note: You can insert this symbol from the computer keyboard by typing @d.

In Degree, Gradian or Radian angle mode:

$\cos(45^\circ)$	0.707107
------------------	----------

In Radian angle mode:

$\cos\left(\left\{0, \frac{\pi}{4}, 90^\circ, 30.12^\circ\right\}\right)$	$\{1, .0707107, 0., 0.864976\}$
---	---------------------------------

° , ' , '' (degree/minute/second)

keys

$dd^\circ mm' ss.ss'' \Rightarrow expression$

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.
- Enter time as hours/minutes/seconds.

Note: Follow *ss.ss* with two apostrophes (''), not a quote symbol (").

In Degree angle mode:

$25^\circ 13' 17.5''$	25.2215
-----------------------	---------

$25^\circ 30'$	$\frac{51}{2}$
----------------	----------------

∠ (angle)

keys

$[Radius, \angle \theta_Angle] \Rightarrow vector$
(polar input)

$[Radius, \angle \theta_Angle, Z_Coordinate] \Rightarrow vector$
(cylindrical input)

$[Radius, \angle \theta_Angle, \angle \theta_Angle] \Rightarrow vector$
(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 @<.

In Radian mode and vector format set to: rectangular

$[5 \angle 60^\circ \angle 45^\circ]$	$[1.76777 \ 3.06186 \ 3.53553]$
---------------------------------------	---------------------------------

cylindrical

$[5 \angle 60^\circ \angle 45^\circ]$	$[3.53553 \ \angle 1.0472 \ 3.53553]$
---------------------------------------	---------------------------------------

spherical

$[5 \angle 60^\circ \angle 45^\circ]$	$[5, \ \angle 1.0472 \ \angle 0.785398]$
---------------------------------------	--

In Radian angle mode and Rectangular complex format:

$5+3 \cdot i - \left(10 \angle \frac{\pi}{4}\right)$	$-2.07107 - 4.07107 \cdot i$
--	------------------------------

$(Magnitude \angle Angle) \Rightarrow complexValue$
(polar input)

Enters a complex value in $(r\angle\theta)$ polar form. The *Angle* is interpreted according to the current Angle mode setting.

_ (underscore as an empty element)

See "Empty (void) elements", page 131.

10[^]()Catalog > **10[^]** (Value1) \Rightarrow value**10[^]** (List1) \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[^](squareMatrix1) \Rightarrow squareMatrixReturns 10 raised to the power of squareMatrix1. This is not the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to **cos()**.

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

$10^{1.5}$	31.6228
------------	---------

$10^{$	$\begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{bmatrix}$	
		$\begin{bmatrix} 1.14336\text{E}7 & 8.17155\text{E}6 & 6.67589\text{E}6 \\ 9.95651\text{E}6 & 7.11587\text{E}6 & 5.81342\text{E}6 \\ 7.65298\text{E}6 & 5.46952\text{E}6 & 4.46845\text{E}6 \end{bmatrix}$

^{^-1}(reciprocal)Catalog > Value1 ^{^-1} \Rightarrow valueList1 ^{^-1} \Rightarrow list

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in List1.

squareMatrix1 ^{^-1} \Rightarrow squareMatrix

Returns the inverse of squareMatrix1.

squareMatrix1 must be a non-singular square matrix.

$(3.1)^{-1}$	0.322581
--------------	----------

$10^{$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1}$	$\begin{bmatrix} -2 & 1 \\ 3 & -1 \\ 2 & 2 \end{bmatrix}$

| ("with")ctrl  keys

Expr | BooleanExpr1 (and BooleanExpr2)...

The "with" (|) symbol serves as a binary operator. The operand to the left of | is an expression. The operand to the right of | specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after | must be joined by a logical "and".

The "with" operator provides three basic types of functionality: substitutions, interval constraints, and 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" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

Exclusions use the "not equals" (\neq or \neq) relational operator to exclude a specific value from consideration.

$x+1 x=3$	4
-----------	---

$x+55 x=\sin(55)$	54.0002
-------------------	---------

$x^3-2\cdot x+7\rightarrow f(x)$	Done
----------------------------------	------

$f(x) x=\sqrt{3}$	8.73205
-------------------	---------

$\text{nSolve}(x^3+2\cdot x^2-15\cdot x=0,x)$	0.
---	----

$\text{nSolve}(x^3+2\cdot x^2-15\cdot x=0,x) x>0 \text{ and } x<5$	3.
--	----

→ (store)

ctrl var key

Value → Var

List → Var

Matrix → Var

Expr → Function(Param1,...)

List → Function(Param1,...)

Matrix → Function(Param1,...)

If the variable *Var* does not exist, creates it and initializes it to *Value*, *List*, or *Matrix*.

If the variable *Var* already exists and is not locked or protected, replaces its contents with *Value*, *List*, or *Matrix*.

Note: You can insert this operator from the keyboard by typing =: as a shortcut. For example, type $\pi/4$ =: **myvar**.

$\frac{\pi}{4} \rightarrow myvar$	0.785398
$2 \cdot \cos(x) \rightarrow y1(x)$	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" → <i>str1</i>	"Hello"

:= (assign)

ctrl [=] keys

Var := Value

Var := List

Var := Matrix

Function(Param1,...) := Expr

Function(Param1,...) := List

Function(Param1,...) := Matrix

If variable *Var* does not exist, creates *Var* and initializes it to *Value*, *List*, or *Matrix*.

If *Var* already exists and is not locked or protected, replaces its contents with *Value*, *List*, or *Matrix*.

$myvar := \frac{\pi}{4}$.785398
$y1(x) := 2 \cdot \cos(x)$	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)

ctrl [⊙] keys

⊙ [text]

⊙ processes *text* as a comment line, allowing you to annotate functions and programs that you create.

⊙ can be at the beginning or anywhere in the line. Everything to the right of ⊙, to the end of the line, is the comment.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing $\left[\leftarrow \right]$ instead of $\left[\text{enter} \right]$ at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

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

⊙ Declare variables

Local *i, result*

result:=0

For *i,1,n,1* ⊙ Loop *n* times

result:=result+i²

EndFor

Return result

EndFunc

Done

$g(3)$ 14

0b, 0h**0 B keys, 0 H keys****0b** *binaryNumber***0h** *hexadecimalNumber*

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

In Dec base mode:

$0b10+0hF+10$	27
---------------	----

In Bin base mode:

$0b10+0hF+10$	0b11011
---------------	---------

In Hex base mode:



$0b10+0hF+10$	0h1B
---------------	------

Empty (void) elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ 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 29, and **isVoid()**, page 49.

Note: To enter an empty element manually in a math expression, type " _ " or the keyword **void**. The keyword **void** is automatically converted to a " _ " symbol when the expression is evaluated. To type " _ " on the handheld, press  .

Calculations involving void elements

The majority of calculations involving a void input will produce a void result. See special cases below.

$_$	$_$
$\gcd(100, _)$	$_$
$3 + _$	$_$
$\{5, _, 10\} - \{3, 6, 9\}$	$\{2, _, 1\}$

List arguments containing void elements

The following functions and commands ignore (skip) void elements found in list arguments.

count, **countIf**, **cumulativeSum**, **freqTableList**, **frequency**, **max**, **mean**, **median**, **product**, **stDevPop**, **stDevSamp**, **sum**, **sumIf**, **varPop**, and **varSamp**, as well as regression calculations, **OneVar**, **TwoVar**, and **FiveNumSummary** statistics, confidence intervals, and stat tests

$\text{sum}\{\{2, _, 3, 5, 6, 6\}\}$	16.6
$\text{median}\{\{1, 2, _, _, 3\}\}$	2
$\text{cumulativeSum}\{\{1, 2, _, 4, 5\}\}$	$\{1, 3, _, 7, 12\}$
$\text{cumulativeSum}\left(\begin{bmatrix} 1 & 2 \\ 3 & _ \\ 5 & 6 \end{bmatrix}\right)$	$\begin{bmatrix} 1 & 2 \\ 4 & _ \\ 9 & 8 \end{bmatrix}$

SortA and **SortD** move all void elements within the first argument to the bottom.

$\{5, 4, 3, _, 1\} \rightarrow \text{list1}$	$\{5, 4, 3, _, 1\}$
$\{5, 4, 3, 2, 1\} \rightarrow \text{list2}$	$\{5, 4, 3, 2, 1\}$
$\text{SortA list1, list2}$	<i>Done</i>
list1	$\{1, 3, 4, 5, _ \}$
list2	$\{1, 3, 4, 5, 2\}$
$\{1, 2, 3, _, 5\} \rightarrow \text{list1}$	$\{1, 2, 3, _, 5\}$
$\{1, 2, 3, 4, 5\} \rightarrow \text{list2}$	$\{1, 2, 3, 4, 5\}$
$\text{SortD list1, list2}$	<i>Done</i>
list1	$\{5, 3, 2, 1, _ \}$
list2	$\{5, 3, 2, 1, 4\}$

List arguments containing void elements(continued)

In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

$I1:=\{1,2,3,4,5\}; I2:=\{2,3,5,6,6\}$	$\{2,3,5,6,6\}$
LinRegMx $I1,I2$	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.

$I1:=\{1,3,4,5\}; I2:=\{2,3,5,6,6\}$	$\{2,3,5,6,6\}$
cat:="M", "M", "F", "F": incl:="F"	$\{ "F" \}$
LinRegMx $I1,I2,1,cat,incl$	Done
stat.Resid	$\{ _, _, 0, 0 \}$
stat.XReg	$\{ _, _, 4, 5 \}$
stat.YReg	$\{ _, _, 5, 6, 6 \}$
stat.FreqReg	$\{ _, _, 1, 1 \}$

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

$I1:=\{1,3,4,5\}; I2:=\{2,3,5,6,6\}$	$\{2,3,5,6,6\}$
LinRegMx $I1,I2,\{1,0,1,1\}$	Done
stat.Resid	$\{0.069231, _, -0.276923, 0.207692\}$
stat.XReg	$\{1, _, 4, 5\}$
stat.YReg	$\{2, _, 5, 6, 6\}$
stat.FreqReg	$\{1, _, 1, 1\}$

Shortcuts for entering math expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type `sqrt(6)` on the entry line. When you press `enter`, the expression `sqrt(6)` is changed to $\sqrt{6}$. Some shortcuts are useful from both the handheld and the computer keyboard. Others are useful primarily from the computer keyboard.

From the handheld or computer keyboard

To enter this:	Type this shortcut:
π	<code>pi</code>
θ	<code>theta</code>
∞	<code>infinity</code>
\leq	<code><=</code>
\geq	<code>>=</code>
\neq	<code>/=</code>
\rightarrow (store operator)	<code>=:</code>
$ $ (absolute value)	<code>abs(...)</code>
$\sqrt{\quad}$	<code>sqrt(...)</code>
$\Sigma()$ (Sum template)	<code>sumSeq(...)</code>
$\Pi()$ (Product template)	<code>prodSeq(...)</code>
$\sin^{-1}()$, $\cos^{-1}()$, ...	<code>arcsin(...)</code> , <code>arccos(...)</code> , ...
Δ List()	<code>deltaList(...)</code>

From the computer keyboard

To enter this:	Type this shortcut:
i (imaginary constant)	<code>@i</code>
e (natural log base e)	<code>@e</code>
E (scientific notation)	<code>@E</code>
\top (transpose)	<code>@t</code>
r (radians)	<code>@r</code>
$^{\circ}$ (degrees)	<code>@d</code>
$^{\text{g}}$ (gradians)	<code>@g</code>
\sphericalangle (angle)	<code>@<</code>

To enter this:	Type this shortcut:
▶ (conversion)	@>
▶Decimal, ▶approxFraction(), and so on.	@>Decimal, @>approxFraction(), and so on.

EOS™ (Equation Operating System) hierarchy

This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire™ math and science learning technology. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS™ software evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

Order of evaluation

Level	Operator
1	Parentheses (), brackets [], braces { }
2	Indirection (#)
3	Function calls
4	Post operators: degrees-minutes-seconds ([°] , ['] , ["]), factorial (!), percentage (%), radian ([∘]), subscript ([]), transpose (^T)
5	Exponentiation, power operator (^)
6	Negation (¯)
7	String concatenation (&)
8	Multiplication (*), division (/)
9	Addition (+), subtraction (-)
10	Equality relations: equal (=), not equal (\neq or \neq), less than (<), less than or equal (\leq or \leq), greater than (>), greater than or equal (\geq or \geq)
11	Logical not
12	Logical and
13	Logical or , exclusive logical xor
14	Constraint "with" operator ()
15	Store (\rightarrow)

Parentheses, brackets, and braces

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression $4(1+2)$, EOS™ software first evaluates the portion of the expression inside the parentheses, $1+2$, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, $(1+2)/(3+4$ will display the error message "Missing)."

Note: Because the TI-Nspire™ software allows you to define your own functions, a variable name followed by an expression in parentheses is considered a "function call" instead of implied multiplication. For example $a(b+c)$ is the function a evaluated by $b+c$. To multiply the expression $b+c$ by the variable a , use explicit multiplication: $a*(b+c)$.

Indirection

The indirection operator (#) converts a string to a variable or function name. For example, #("x"&"y"&"z") creates the variable name xyz. Indirection also allows the creation and modification of variables from inside a program. For example, if $10 \rightarrow r$ and $r \rightarrow s1$, then $\#s1 = 10$.

Post operators

Post operators are operators that come directly after an argument, such as $5!$, 25% , or $60^\circ 15' 45''$. Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression $4^3!$, $3!$ is evaluated first. The result, 6, then becomes the exponent of 4 to yield 4096.

Exponentiation

Exponentiation (^) and element-by-element exponentiation (.^) are evaluated from right to left. For example, the expression 2^3^2 is evaluated the same as $2^{(3^2)}$ to produce 512. This is different from $(2^3)^2$, which is 64.

Negation

To enter a negative number, press $\boxed{-}$ followed by the number. Post operations and exponentiation are performed before negation. For example, the result of $-x^2$ is a negative number, and $-9^2 = -81$. Use parentheses to square a negative number such as $(-9)^2$ to produce 81.



Constraint (|)

The argument following the "with" (|) operator provides a set of constraints that affect the evaluation of the argument preceding the "with" operator.

Error codes and messages

When an error occurs, its code is assigned to variable *errCode*. User-defined programs and functions can examine *errCode* to determine the cause of an error. For an example of using *errCode*, See Example 2 under the **Try** command, page [105](#).

Note: Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.

Error code	Description
10	A function did not return a value
20	A test did not resolve to TRUE or FALSE. Generally, undefined variables cannot be compared. For example, the test $If\ a < b$ will cause this error if either <i>a</i> or <i>b</i> is undefined when the <i>If</i> statement is executed.
30	Argument cannot be a folder name.
40	Argument error
50	Argument mismatch Two or more arguments must be of the same type.
60	Argument must be a Boolean expression or integer
70	Argument must be a decimal number
90	Argument must be a list
100	Argument must be a matrix
130	Argument must be a string
140	Argument must be a variable name. Make sure that the name: <ul style="list-style-type: none">• does not begin with a digit• does not contain spaces or special characters• does not use underscore or period in invalid manner• does not exceed the length limitations See the Calculator section in the documentation for more details.
160	Argument must be an expression
165	Batteries too low for sending or receiving Install new batteries before sending or receiving.
170	Bound The lower bound must be less than the upper bound to define the search interval.
180	Break The  or  key was pressed during a long calculation or during program execution.
190	Circular definition This message is displayed to avoid running out of memory during infinite replacement of variable values during simplification. For example, $a+1 \rightarrow a$, where <i>a</i> 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

Error code	Description
230	Dimension A list or matrix index is not valid. For example, if the list {1,2,3,4} is stored in L1, then L1[5] is a dimension error because L1 only contains four elements.
235	Dimension Error. Not enough elements in the lists.
240	Dimension mismatch Two or more arguments must be of the same dimension. For example, [1,2]+[1,2,3] is a dimension mismatch because the matrices contain a different number of elements.
250	Divide by zero
260	Domain error An argument must be in a specified domain. For example, rand(0) is not valid.
270	Duplicate variable name
280	Else and Elseif invalid outside of If...EndIf block
290	EndTry is missing the matching Else statement
295	Excessive iteration
300	Expected 2 or 3-element list or matrix
310	The first argument of nSolve must be an equation in a single variable. It cannot contain a non-valued variable other than the variable of interest.
320	First argument of solve or cSolve must be an equation or inequality For example, solve(3x^2-4,x) is invalid because the first argument is not an equation.
345	Inconsistent units
350	Index out of range
360	Indirection string is not a valid variable name
380	Undefined Ans Either the previous calculation did not create Ans, or no previous calculation was entered.
390	Invalid assignment
400	Invalid assignment value
410	Invalid command
430	Invalid for the current mode settings
435	Invalid guess
440	Invalid implied multiply For example, x(x+1) is invalid; whereas, x*(x+1) is the correct syntax. This is to avoid confusion between implied multiplication and function calls.
450	Invalid in a function or current expression Only certain commands are valid in a user-defined function.
490	Invalid in Try...EndTry block
510	Invalid list or matrix
550	Invalid outside function or program A number of commands are not valid outside a function or program. For example, Local cannot be used unless it is in a function or program.
560	Invalid outside Loop...EndLoop, For...EndFor, or While...EndWhile blocks For example, the Exit command is valid only inside these loop blocks.
565	Invalid outside program

Error code	Description
570	Invalid pathname For example, lvar is invalid.
575	Invalid polar complex
580	Invalid program reference Programs cannot be referenced within functions or expressions such as 1+p(x) where p is a program.
600	Invalid table
605	Invalid use of units
610	Invalid variable name in a Local statement
620	Invalid variable or function name
630	Invalid variable reference
640	Invalid vector syntax
650	Link transmission A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends.
665	Matrix not diagonalizable
670	Low Memory 1. Delete some data in this document 2. Save and close this document If 1 and 2 fail, pull out and re-insert batteries
680	Missing (
690	Missing)
700	Missing "
710	Missing]
720	Missing }
730	Missing start or end of block syntax
740	Missing Then in the If..EndIf block
750	Name is not a function or program
765	No functions selected
780	No solution found
800	Non-real result For example, if the software is in the Real setting, $\sqrt{-1}$ is invalid. To allow complex results, change the "Real or Complex" Mode Setting to RECTANGULAR or POLAR.
830	Overflow
850	Program not found A program reference inside another program could not be found in the provided path during execution.
855	Rand type functions not allowed in graphing
860	Recursion too deep
870	Reserved name or system variable
900	Argument error Median-median model could not be applied to data set.

Error code	Description
920	Text not found
930	Too few arguments The function or command is missing one or more arguments.
940	Too many arguments The expression or equation contains an excessive number of arguments and cannot be evaluated.
950	Too many subscripts
955	Too many undefined variables
960	Variable is not defined No value is assigned to variable. Use one of the following commands: <ul style="list-style-type: none"> • sto → • := • Define to assign values to variables.
965	Unlicensed OS
970	Variable in use so references or changes are not allowed
980	Variable is protected
990	Invalid variable name Make sure that the name does not exceed the length limitations
1000	Window variables domain
1010	Zoom
1020	Internal error
1030	Protected memory violation
1040	Unsupported function. This function requires Computer Algebra System. Try TI-Nspire™ CAS.
1045	Unsupported operator. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.
1050	Unsupported feature. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.
1060	Input argument must be numeric. Only inputs containing numeric values are allowed.
1070	Trig function argument too big for accurate reduction
1080	Unsupported use of Ans. This application does not support Ans.
1090	Function is not defined. Use one of the following commands: <ul style="list-style-type: none"> • 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.

Error code	Description
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: <ul style="list-style-type: none"> • The xxx part can have 1 to 16 characters. • The yyy part can have 1 to 15 characters. See the Library section in the documentation for more details.
1170	Invalid use of library pathname <ul style="list-style-type: none"> • A value cannot be assigned to a pathname using Define, :=, or sto →. • A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition.
1180	Invalid library variable name. Make sure that the name: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • Verify library is in the MyLib folder. • Refresh Libraries. See the Library section in the documentation for more details.
1200	Library variable not found: <ul style="list-style-type: none"> • Verify library variable exists in the first problem in the library. • Make sure library variable has been defined as LibPub or LibPriv. • Refresh Libraries. See the Library section in the documentation for more details.
1210	Invalid library shortcut name. Make sure that the name: <ul style="list-style-type: none"> • Does not contain a period • Does not begin with an underscore • Does not exceed 16 characters • Is not a reserved name 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 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.

Error code	Description
1310	Argument error: A function could not be evaluated for one or more of its arguments.

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 [110](#).

Warning code	Message
10000	Operation might introduce false solutions.
10001	Differentiating an equation may produce a false equation.
10002	Questionable solution
10003	Questionable accuracy
10004	Operation might lose solutions.
10005	cSolve might specify more zeros.
10006	Solve may specify more zeros.
10007	More solutions may exist.
10008	Domain of the result might be smaller than the domain of the input.
10009	Domain of the result might be larger than the domain of the input.
10012	Non-real calculation
10013	$\infty^{\wedge}0$ or $\text{undef}^{\wedge}0$ replaced by 1
10014	$\text{undef}^{\wedge}0$ replaced by 1
10015	$1^{\wedge}\infty$ or 1^{\wedge}undef replaced by 1
10016	1^{\wedge}undef replaced by 1
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'

Service and Support

Texas Instruments Support and Service

For general information

For more information about TI products and services, contact TI by e-mail or visit the TI Internet address.

E-mail inquiries: ti-cares@ti.com

Home Page: education.ti.com

Service and warranty information

For information about the length and terms of the warranty or about product service, refer to the warranty statement enclosed with this product or contact your local Texas Instruments retailer/distributor.

Index

Symbols

\wedge , power 118
 \wedge^{-1} , reciprocal 128
 $:=$, assign 129
 $!$, factorial 122
 \cdot^\wedge , dot power 119
 \cdot^* , dot multiplication 119
 $\cdot+$, dot addition 119
 $\cdot-$, dot subtraction 119
 $\cdot\div$, dot division 119
' , minute notation 127
" , second notation 127
 \leq , less than or equal 122
©, comment 129
 $\Delta\text{list}()$, list difference 55
 $^\circ$, degree notation 127
 $^\circ$, degrees/minutes/seconds 127
 \int , integral 123
 $\sqrt{\quad}$, square root 123
 \neq , not equal 121
 $-$, subtract 116
 \div , divide 117
 Π , product 124
 $\Sigma()$, sum 124
 $*$, multiply 117
&, append 122
 \rightarrow , store 129
#, indirection 126
#, indirection operator 136
%, percent 120
+, add 116
<, less than 121
=, equal 121
>, greater than 122
 \geq , greater than or equal 122
 $|$, with 128

Numerics

0b, binary indicator 130
0h, hexadecimal indicator 130
 $10^\wedge()$, power of ten 128
2-sample F Test 39
▶approxFraction() 10

A

abs(), absolute value 6
absolute value
 template for 3
add, + 116
amortization table, amortTbl() 6, 12
amortTbl(), amortization table 6, 12
and, Boolean and 6
angle, angle() 7
angle(), angle 7
ANOVA, one-way variance analysis 7
ANOVA2way, two-way variance
 analysis 8
Ans, last answer 9
answer (last), Ans 9
append, & 122
approx(), approximate 10
approximate, approx() 10
approxRational() 10
arccos() 10
arccosh() 10
arccosine, $\cos^{-1}()$ 20
arccot() 10
arccoth() 11
arccsc() 11
arccsch() 11
arcsec() 11
arcsech() 11
arcsin() 11
arcsine, $\sin^{-1}()$ 93
arcsinh() 11
arctan() 11
arctangent, $\tan^{-1}()$ 101
arctanh() 11
arguments in TVM functions 107
augment(), augment/concatenate
 11
augment/concatenate, augment()
 11
average rate of change, avgRC() 12
avgRC(), average rate of change 12

B

►Base10, display as decimal integer
13

►Base16, display as hexadecimal 14

►Base2, display as binary 12

binary

display, ►Base2 12

indicator, Ob 130

binomCdf() 14

binomPdf() 14

Boolean

and, and 6

exclusive or, xor 111

not, not 69

or, or 72

C

χ^2 2way 15

χ^2 Cdf() 16

χ^2 GOF 16

χ^2 Pdf() 16

Cdf() 36

ceiling, ceiling() 14, 15, 23

ceiling(), ceiling 14

centralDiff() 15

char(), character string 15

character string, char() 15

characters

numeric code, ord() 72

string, char() 15

clear

error, ClrErr 17

ClearAZ 16

ClrErr, clear error 17

colAugment 17

colDim(), matrix column dimension
17

colNorm(), matrix column norm 17

combinations, nCr() 66

comment, © 129

completeSquare(), complete square
18

complex

conjugate, conj() 18

conj(), complex conjugate 18

construct matrix, constructMat() 18

constructMat(), construct matrix 18

contact information 145

convert

►Grad 44

►Rad 80

copy variable or function, CopyVar
18

copyright statement *ii*

correlation matrix, corrMat() 19

corrMat(), correlation matrix 19

cos(), cosine 19

\cos^{-1} , arccosine 20

cosh(), hyperbolic cosine 21

\cosh^{-1} (), hyperbolic arccosine 21

cosine, cos() 19

cot(), cotangent 21

\cot^{-1} (), hyperbolic arccotangent 22

cotangent, cot() 21

coth(), hyperbolic cotangent 22

\coth^{-1} (), hyperbolic arccotangent 22

count days between dates, dbd() 26

count items in a list conditionally,
countif() 23

count items in a list, count() 22

count(), count items in a list 22

countif(), conditionally count items
in a list 23

cPolyRoots() 23

cross product, crossP() 23

crossP(), cross product 23

csc(), cosecant 24

\csc^{-1} (), inverse cosecant 24

csch(), hyperbolic cosecant 24

csch^{-1} (), inverse hyperbolic cosecant
24

cubic regression, CubicReg 25

CubicReg, cubic regression 25

cumulative sum, cumulativeSum()
25

cumulativeSum(), cumulative sum
25

customer support and service 145

Cycle, cycle 26

cycle, Cycle 26

►Cylind, display as cylindrical vector
26

cylindrical vector display, ►Cylind 26

D

`d ()`, first derivative 123
days between dates, `dbd ()` 26
`dbd ()`, days between dates 26
▶DD, display as decimal angle 27
▶Decimal, display result as decimal 27
decimal
 angle display, ▶DD 27
 integer display, ▶Base10 13
Define 27
Define LibPriv 28
Define LibPub 28
Define, define 27
define, Define 27
defining
 private function or program 28
 public function or program 28
definite integral
 template for 5
degree notation, ° 127
degree/minute/second display, ▶DMS 31
degree/minute/second notation 127
delete
 void elements from list 29
deleting
 variable, `DelVar` 29
`deltaList ()` 29
`DelVar`, delete variable 29
`delVoid ()`, remove void elements 29
derivatives
 first derivative, `d ()` 123
 numeric derivative, `nDeriv ()` 67, 68
 numeric derivative,
 `nDerivative ()` 67
`det ()`, matrix determinant 29
`diag ()`, matrix diagonal 30
`dim ()`, dimension 30
dimension, `dim ()` 30
`Disp`, display data 30
display as
 binary, ▶Base2 12
 cylindrical vector, ▶Cylind 26
 decimal angle, ▶DD 27
 decimal integer, ▶Base10 13

degree/minute/second, ▶DMS 31
hexadecimal, ▶Base16 14
polar vector, ▶Polar 74
rectangular vector, ▶Rect 81
spherical vector, ▶Sphere 96

display data, `Disp` 30

distribution functions

`binomCdf ()` 14

`binomPdf ()` 14

χ^2 2way () 15

χ^2 Cdf () 16

χ^2 GOF () 16

χ^2 Pdf () 16

`Inv χ^2 ()` 48

`invNorm ()` 48

`invt ()` 48

`normCdf ()` 69

`normPdf ()` 69

`poissCdf ()` 73

`poissPdf ()` 73

`tCdf ()` 103

`tPdf ()` 104

divide, ÷ 117

▶DMS, display as degree/minute/
second 31

dot

addition, .+ 119

division, ÷ 119

multiplication, .* 119

power, .^ 119

product, dotP () 31

subtraction, .- 119

`dotP ()`, dot product 31

E

e exponent

 template for 2

e to a power, `e^ ()` 31, 35

`e^ ()`, *e* to a power 31

E, exponent 126

`eff ()`, convert nominal to effective
rate 32

effective rate, `eff ()` 32

eigenvalue, `eigVl ()` 32

eigenvector, `eigVc ()` 32

`eigVc ()`, eigenvector 32

`eigVl ()`, eigenvalue 32

- else if, ElseIf 33
- else, Else 45
- ElseIf, else if 33
- empty (void) elements 131
- end
 - for, EndFor 38
 - function, EndFunc 40
 - if, EndIf 45
 - loop, EndLoop 60
 - program, EndPrgm 76
 - try, EndTry 105
 - while, EndWhile 111
- end function, EndFunc 40
- end if, EndIf 45
- end loop, EndLoop 60
- end while, EndWhile 111
- EndTry, end try 105
- EndWhile, end while 111
- EOS (Equation Operating System) 135
- equal, = 121
- Equation Operating System (EOS) 135
- error codes and messages 137
- errors and troubleshooting
 - clear error, ClrErr 17
 - pass error, PassErr 33
- euler(), Euler function 34
- evaluate polynomial, polyEval() 74
- evaluation, order of 135
- exclusive or (Boolean), xor 111
- Exit, exit 34
- exit, Exit 34
- exp(), e to a power 35
- exponent, E 126
- exponential regression, ExpReg 35
- exponents
 - template for 1
- expr(), string to expression 35
- ExpReg, exponential regression 35
- expressions
 - string to expression, expr() 35

F

- factor, factor() 36
- factor(), factor 36
- factorial, ! 122

- Fill, matrix fill 36
- financial functions, tvml() 107
- financial functions, tvml() 107
- financial functions, tvml() 107
- financial functions, tvml() 107
- financial functions, tvml() 107
- first derivative
 - template for 5
- FiveNumSummary 37
- floor, floor() 37
- floor(), floor 37
- For 38
- For, for 38
- for, For 38
- format string, format() 38
- format(), format string 38
- fpart(), function part 38
- fractions
 - propFrac 77
 - template for 1
- freqTable() 39
- frequency() 39
- Frobenius norm, norm() 68
- Func, function 40
- Func, program function 40
- functions
 - part, fpart() 38
 - program function, Func 40
 - user-defined 27
- functions and variables
 - copying 18

G

- g , gradians 126
- gcd(), greatest common divisor 40
- geomCdf() 41
- geomPdf() 41
- get/return
 - denominator, getDenom() 41
 - number, getNum() 43
 - variables information, getVarInfo() 41, 43
- getDenom(), get/return denominator 41
- getLangInfo(), get/return language information 41

getLockInfo(), tests lock status of variable or variable group 42
getMode(), get mode settings 42
getNum(), get/return number 43
getType(), get type of variable 43
getVarInfo(), get/return variables information 43
go to, Goto 44
Goto, go to 44
►, convert to gradian angle 44
gradian notation, ⁹ 126
greater than or equal, \geq 122
greater than, $>$ 122
greatest common divisor, gcd() 40
groups, locking and unlocking 57, 109
groups, testing lock status 42

H

hexadecimal
 display, ►Base16 14
 indicator, 0h 130
hyperbolic
 arccosine, $\cosh^{-1}()$ 21
 arcsine, $\sinh^{-1}()$ 94
 arctangent, $\tanh^{-1}()$ 102
 cosine, cosh() 21
 sine, sinh() 94
 tangent, tanh() 102

I

identity matrix, identity() 45
identity(), identity matrix 45
If, if 45
if, If 45
ifFn() 46
imag(), imaginary part 46
imaginary part, imag() 46
indirection operator (#) 136
indirection, # 126
inString(), within string 47
int(), integer 47
intDiv(), integer divide 47
integer divide, intDiv() 47
integer part, iPart() 49
integer, int() 47
integral, \int 123

interpolate(), interpolate 48
Inv χ^2 () 48
inverse cumulative normal distribution (invNorm()) 48
inverse, \wedge^{-1} 128
invF() 48
invNorm(), inverse cumulative normal distribution) 48
invt() 48
iPart(), integer part 49
irr(), internal rate of return
 internal rate of return, irr() 49
isPrime(), prime test 49
isVoid(), test for void 49

L

label, Lbl 50
language
 get language information 41
Lbl, label 50
lcm, least common multiple 50
least common multiple, lcm 50
left, left() 50
left(), left 50
length of string 30
less than or equal, \leq 122
less than, $<$ 121
LibPriv 28
LibPub 28
library
 create shortcuts to objects 51
libShortcut(), create shortcuts to library objects 51
linear regression, LinRegAx 52
linear regression, LinRegBx 51, 52
LinRegBx, linear regression 51
LinRegMx, linear regression 52
LinRegtIntervals, linear regression 52
LinRegtTest 54
linSolve() 55
list to matrix, list►mat() 55
list, conditionally count items in 23
list, count items in 22
list►mat(), list to matrix 55
lists

- augment/concatenate,
 - augment() 11
 - cross product, crossP() 23
 - cumulative sum,
 - cumulativeSum() 25
 - difference, Δ list() 55
 - differences in a list, Δ list() 55
 - dot product, dotP() 31
 - empty elements in 131
 - list to matrix, list▶mat() 55
 - matrix to list, mat▶list() 60
 - maximum, max() 61
 - mid-string, mid() 62
 - minimum, min() 63
 - new, newList() 67
 - product, product() 76
 - sort ascending, SortA 95
 - sort descending, SortD 96
 - summation, sum() 99, 100
 - ln(), natural logarithm 55
 - LnReg, logarithmic regression 56
 - local variable, Local 57
 - local, Local 57
 - Local, local variable 57
 - Lock, lock variable or variable group 57
 - locking variables and variable groups 57
 - Log
 - template for 2
 - logarithmic regression, LnReg 56
 - logarithms 55
 - logistic regression, Logistic 58
 - logistic regression, LogisticD 59
 - Logistic, logistic regression 58
 - LogisticD, logistic regression 59
 - Loop, loop 60
 - loop, Loop 60
 - LU, matrix lower-upper decomposition 60
- M**
- mat▶list(), matrix to list 60
 - matrices
 - augment/concatenate,
 - augment() 11
 - column dimension, colDim() 17
 - column norm, colNorm() 17
 - cumulative sum,
 - cumulativeSum() 25
 - determinant, det() 29
 - diagonal, diag() 30
 - dimension, dim() 30
 - dot addition, .+ 119
 - dot division, .÷ 119
 - dot multiplication, .* 119
 - dot power, .^ 119
 - dot subtraction, .- 119
 - eigenvalue, eigVl() 32
 - eigenvector, eigVc() 32
 - filling, Fill 36
 - identity, identity() 45
 - list to matrix, list▶mat() 55
 - lower-upper decomposition, LU 60
 - matrix to list, mat▶list() 60
 - maximum, max() 61
 - minimum, min() 63
 - new, newMat() 67
 - product, product() 76
 - QR factorization, QR 77
 - random, randMat() 80
 - reduced row echelon form,
 - rref() 87
 - row addition, rowAdd() 86
 - row dimension, rowDim() 87
 - row echelon form, ref() 82
 - row multiplication and addition,
 - mRowAdd() 64
 - row norm, rowNorm() 87
 - row operation, mRow() 64
 - row swap, rowSwap() 87
 - submatrix, subMat() 99, 100
 - summation, sum() 99, 100
 - transpose, ^T 100
 - matrix (1 × 2)
 - template for 4
 - matrix (2 × 1)
 - template for 4
 - matrix (2 × 2)
 - template for 3
 - matrix (m × n)
 - template for 4
 - matrix to list, mat▶list() 60
 - max(), maximum 61

maximum, `max()` 61
mean, `mean()` 61
`mean()`, mean 61
median, `median()` 61
`median()`, median 61
medium-medium line regression,
 `MedMed` 62
`MedMed`, medium-medium line
 regression 62
`mid()`, mid-string 62
mid-string, `mid()` 62
`min()`, minimum 63
minimum, `min()` 63
minute notation, ' 127
`mirr()`, modified internal rate of
 return 63
mixed fractions, using `propFrac()`
 with 77
`mod()`, modulo 64
mode settings, `getMode()` 42
modes
 setting, `setMode()` 90
modified internal rate of return,
 `mirr()` 63
modulo, `mod()` 64
`mRow()`, matrix row operation 64
`mRowAdd()`, matrix row
 multiplication and addition 64
Multiple linear regression t test 65
multiply, * 117
`MultReg` 64
`MultRegIntervals()` 65
`MultRegTests()` 65

N

natural logarithm, `ln()` 55
`nCr()`, combinations 66
`nDerivative()`, numeric derivative 67
negation, entering negative
 numbers 136
net present value, `npv()` 70
new
 list, `newList()` 67
 matrix, `newMat()` 67
`newList()`, new list 67
`newMat()`, new matrix 67

`nfMax()`, numeric function
 maximum 67
`nfMin()`, numeric function minimum
 68
`nInt()`, numeric integral 68
`nom()`, convert effective to nominal
 rate 68
nominal rate, `nom()` 68
`norm()`, Frobenius norm 68
normal distribution probability,
 `normCdf()` 69
`normCdf()` 69
`normPdf()` 69
`not` (Boolean), `not` 69
`not` equal, 121
`not`, Boolean `not` 69
`nPr()`, permutations 69
`npv()`, net present value 70
`nSolve()`, numeric solution 70
nth root
 template for 1
numeric
 derivative, `nDeriv()` 67, 68
 derivative, `nDerivative()` 67
 integral, `nInt()` 68
 solution, `nSolve()` 70

O

objects
 create shortcuts to library 51
`OneVar`, one-variable statistics 71
one-variable statistics, `OneVar` 71
operators
 order of evaluation 135
or (Boolean), `or` 72
or, Boolean `or` 72
`ord()`, numeric character code 72

P

`P►Rx()`, rectangular x coordinate 72
`P►Ry()`, rectangular y coordinate 73
pass error, `PassErr` 73
`PassErr`, pass error 73
`Pdf()` 38
percent, % 120
permutations, `nPr()` 69
piecewise function (2-piece)

- template for 2
- piecewise function (N-piece)
 - template for 2
- piecewise() 73
- poissCdf() 73
- poissPdf() 73
- Polar, display as polar vector 74
- polar
 - coordinate, $R\blacktriangleright P\theta()$ 79
 - coordinate, $R\blacktriangleright Pr()$ 79
 - vector display, ►Polar 74
- polyEval(), evaluate polynomial 74
- polynomials
 - evaluate, polyEval() 74
 - random, randPoly() 81
- PolyRoots() 74
- power of ten, $10^{\wedge}()$ 128
- power regression, PowerReg 74, 75, 83, 84, 103
- power, \wedge 118
- PowerReg, power regression 75
- Prgm, define program 76
- prime number test, isPrime() 49
- probability density, normPdf() 69
- prodSeq() 76
- product (Π)
 - template for 4
- product, $\Pi()$ 124
- product, product() 76
- product(), product 76
- programming
 - define program, Prgm 76
 - display data, Disp 30
 - pass error, PassErr 73
- programs
 - defining private library 28
 - defining public library 28
- programs and programming
 - clear error, ClrErr 17
 - display I/O screen, Disp 30
 - end program, EndPrgm 76
 - end try, EndTry 105
 - try, Try 105
- proper fraction, propFrac 77
- propFrac, proper fraction 77

Q

- QR factorization, QR 77
- QR, QR factorization 77
- quadratic regression, QuadReg 78
- QuadReg, quadratic regression 78
- quartic regression, QuartReg 78
- QuartReg, quartic regression 78

R

- r , radian 126
- $R\blacktriangleright P\theta()$, polar coordinate 79
- $R\blacktriangleright Pr()$, polar coordinate 79
- Rad, convert to radian angle 80
- radian, r 126
- rand(), random number 80
- randBin, random number 80
- randInt(), random integer 80
- randMat(), random matrix 80
- randNorm(), random norm 80
- random
 - matrix, randMat() 80
 - norm, randNorm() 80
 - number seed, RandSeed 81
 - polynomial, randPoly() 81
- random sample 81
- randPoly(), random polynomial 81
- randSamp() 81
- RandSeed, random number seed 81
- real, real() 81
- real(), real 81
- reciprocal, \wedge^{-1} 128
- Rect, display as rectangular vector 81
- rectangular x coordinate, $P\blacktriangleright Rx()$ 72
- rectangular y coordinate, $P\blacktriangleright Ry()$ 73
- rectangular-vector display, ►Rect 81
- reduced row echelon form, rref() 87
- ref(), row echelon form 82
- regressions
 - cubic, CubicReg 25
 - exponential, ExpReg 35
 - linear regression, LinRegAx 52
 - linear regression, LinRegBx 51, 52
 - logarithmic, LnReg 56
 - Logistic 58
 - logistic, Logistic 59

medium-medium line, MedMed
 62
 MultReg 64
 power regression, PowerReg 74,
 75, 83, 84, 103
 quadratic, QuadReg 78
 quartic, QuartReg 78
 sinusoidal, SinReg 95
 remain(), remainder 83
 remainder, remain() 83
 remove
 void elements from list 29
 Request 83
 RequestStr 84
 result values, statistics 98
 results, statistics 97
 Return, return 84
 return, Return 84
 right, right() 18, 34, 48, 84, 85, 110
 right(), right 84
 rk23(), Runge Kutta function 85
 rotate, rotate() 85
 rotate(), rotate 85
 round, round() 86
 round(), round 86
 row echelon form, ref() 82
 rowAdd(), matrix row addition 86
 rowDim(), matrix row dimension 87
 rowNorm(), matrix row norm 87
 rowSwap(), matrix row swap 87
 rref(), reduced row echelon form 87

S

sec(), secant 88
 sec⁻¹(), inverse secant 88
 sech(), hyperbolic secant 88
 sech⁻¹(), inverse hyperbolic secant 88
 second derivative
 template for 5
 second notation, " 127
 seq(), sequence 89
 seqGen() 89
 seqn() 90
 sequence, seq() 89, 90
 service and support 145
 set
 mode, setMode() 90

setMode(), set mode 90
 settings, get current 42
 shift, shift() 91
 shift(), shift 91
 sign, sign() 92
 sign(), sign 92
 similt(), simultaneous equations 92
 simultaneous equations, similt() 92
 sin(), sine 93
 sin⁻¹(), arcsine 93
 sine, sin() 93
 sinh(), hyperbolic sine 94
 sinh⁻¹(), hyperbolic arcsine 94
 SinReg, sinusoidal regression 95
 ΣInt() 125
 sinusoidal regression, SinReg 95
 SortA, sort ascending 95
 SortD, sort descending 96
 sorting
 ascending, SortA 95
 descending, SortD 96
 ►Sphere, display as spherical vector
 96
 spherical vector display, ►Sphere 96
 ΣPrn() 125
 sqrt(), square root 96
 square root
 template for 1
 square root, √() 96, 123
 standard deviation, stdDev() 98, 109
 stat.results 97
 stat.values 98
 statistics
 combinations, nCr() 66
 factorial, ! 122
 mean, mean() 61
 median, median() 61
 one-variable statistics, OneVar
 71
 permutations, nPr() 69
 random norm, randNorm() 80
 random number seed, RandSeed
 81
 standard deviation, stdDev() 98,
 109
 two-variable results, TwoVar 108
 variance, variance() 110

stdDevPop(), population standard deviation 98
 stdDevSamp(), sample standard deviation 98
 Stop command 99
 storing
 symbol, → 129
 string
 dimension, dim() 30
 length 30
 string(), expression to string 99
 strings
 append, & 122
 character code, ord() 72
 character string, char() 15
 expression to string, string() 99
 format, format() 38
 formatting 38
 indirection, # 126
 left, left() 50
 mid-string, mid() 62
 right, right() 18, 34, 48, 84, 85, 110
 rotate, rotate() 85
 shift, shift() 91
 string to expression, expr() 35
 using to create variable names 136
 within, InString 47
 student-*t* distribution probability, tCdf() 103
 student-*t* probability density, tPdf() 104
 subMat(), submatrix 99, 100
 submatrix, subMat() 99, 100
 subtract, - 116
 sum (Σ)
 template for 4
 sum of interest payments 125
 sum of principal payments 125
 sum, Σ () 124
 sum(), summation 99
 sumIf() 100
 summation, sum() 99
 sumSeq() 100
 support and service 145
 system of equations (2-equation)
 template for 3

system of equations (N-equation)
 template for 3

T

t test, tTest 105
 T , transpose 100
 tan(), tangent 101
 $\tan^{-1}()$, arctangent 101
 tangent, tan() 101
 tanh(), hyperbolic tangent 102
 $\tanh^{-1}()$, hyperbolic arctangent 102
 tCdf(), student-*t* distribution probability 103
 templates
 absolute value 3
 definite integral 5
 e exponent 2
 exponent 1
 first derivative 5
 fraction 1
 Log 2
 matrix (1 × 2) 4
 matrix (2 × 1) 4
 matrix (2 × 2) 3
 matrix (m × n) 4
 nth root 1
 piecewise function (2-piece) 2
 piecewise function (N-piece) 2
 product (II) 4
 second derivative 5
 square root 1
 sum (Σ) 4
 system of equations (2-equation) 3
 system of equations (N-equation) 3
 test for void, isVoid() 49
 Test_2S, 2-sample F test 39
 Text command 103
 time value of money, Future Value 106
 time value of money, Interest 107
 time value of money, number of payments 107
 time value of money, payment amount 107

time value of money, present value
107
tInterval_2Samp, two-sample t
confidence interval 104
tInterval, t confidence interval 103
tPdf(), student- t probability density
104
trace() 104
transpose, \top 100
Try, error handling command 105
tTest_2Samp, two-sample t test 106
tTest, t test 105
TVM arguments 107
tvmFV() 106
tvmI() 107
tvmN() 107
tvmPmt() 107
tvmPV() 107
TwoVar, two-variable results 108
two-variable results, TwoVar 108

U

unit vector, unitV() 109
unitV(), unit vector 109
unLock, unlock variable or variable
group 109
unlocking variables and variable
groups 109
user-defined functions 27
user-defined functions and
programs 28

V

variable
creating name from a character
string 136
variable and functions
copying 18
variables
clear all single-letter 16
delete, DelVar 29
local, Local 57
variables, locking and unlocking 42,
57, 109
variance, variance() 110
varPop() 109
varSamp(), sample variance 110

vectors
cross product, crossP() 23
cylindrical vector display, \blacktriangleright Cylind
26
dot product, dotP() 31
unit, unitV() 109
void elements 131
void elements, remove 29
void, test for 49

W

warnCodes(), Warning codes 110
warning codes and messages 143
when, when() 110
when(), when 110
While, while 111
while, While 111
with, | 128
within string, inString() 47

X

x2, square 118
xor, Boolean exclusive or 111

Z

zInterval_1Prop, one-proportion z
confidence interval 112
zInterval_2Prop, two-proportion z
confidence interval 113
zInterval_2Samp, two-sample z
confidence interval 113
zInterval, z confidence interval 112
zTest 114
zTest_1Prop, one-proportion z test
114
zTest_2Prop, two-proportion z test
115
zTest_2Samp, two-sample z test 115

