TI-Nspire™ CAS
TI-Nspire™ CX CAS
Reference Guide

This guidebook applies to TI-Nspire™ software version 3.2. To obtain the latest version of the documentation, go to education.ti.com/guides.
**Important Information**

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

**License**

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

© 2006 - 2012 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°mm’ss.ss” template ....................................... 3
Matrix template (m x n) .................................. 4
Matrix template (1 x 2) .................................... 4
Matrix template (2 x 2) .................................... 3
Matrix template (m x n) .................................... 4
Sum template (Σ) ............................................. 4
Product template (Π) ....................................... 4
First derivative template .............................. 5
Second derivative template ............................ 5
Nth derivative template ................................... 5
Definite integral template .............................. 5
Indefinite integral template ............................ 5
Limit template ................................................ 6

Alphabetical Listing

A
abs() .......................................................... 7
amortTbI() .................................................. 7
and .................................................................. 7
angle() .......................................................... 8
ANOVA ....................................................... 8
ANOVA2way .................................................. 9
Ans .............................................................. 11
approx() ...................................................... 11
approxFraction() ........................................... 11
approxRational() .......................................... 11
arccos() ...................................................... 11
arccosh() ..................................................... 12
arccot() ...................................................... 12
arccoth() ..................................................... 12
arccsch() ..................................................... 12
arccsch() ..................................................... 12
arcsch() ...................................................... 12
arccsc() ...................................................... 12
arcLen() ...................................................... 12
arccsc() ...................................................... 12
arccsch() ..................................................... 12
arcsin() ...................................................... 12
arcsinh() ..................................................... 12
arctan() ..................................................... 12
arctanh() ..................................................... 12
augment() ................................................... 12
avgRC() ...................................................... 13

B
bal() ............................................................ 13
Base2 ......................................................... 14
Base10 ......................................................... 14

C
ceiling() ..................................................... 15
centralDiff() ............................................... 16
cFactor() ..................................................... 16
char() ......................................................... 17
charPoly() ................................................... 17
χ²(2) ........................................................... 17
χ²cdf() ....................................................... 17
χ²(2) ........................................................... 18
χ²pdf() ....................................................... 18
ClearAZ ..................................................... 18
ClrErr ....................................................... 19
colAugment() .............................................. 19
colDim() ..................................................... 19
colNorm() ................................................... 19
comDenom() ............................................... 19
completeSquare() ....................................... 20
con() ......................................................... 21
constructMat() ............................................ 21
CopyVar .................................................... 21
corrMat() .................................................. 22

D
dbd() ........................................................ 33
DD ............................................................ 33
Decimal ..................................................... 33
Define ...................................................... 34
Define LibPriv ............................................ 34
Define LibPub ............................................. 35
deltaList() .................................................. 35
deltaTmpCnv() ............................................. 35
DelVar ..................................................... 35
delVoid() ................................................... 35
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrRx()</td>
<td>87</td>
</tr>
<tr>
<td>Pr Ry()</td>
<td>88</td>
</tr>
<tr>
<td>PassErr</td>
<td>88</td>
</tr>
<tr>
<td>piecewise()</td>
<td>88</td>
</tr>
<tr>
<td>poissPdf()</td>
<td>88</td>
</tr>
<tr>
<td>Polar</td>
<td>89</td>
</tr>
<tr>
<td>polyCoeffs()</td>
<td>89</td>
</tr>
<tr>
<td>polyDegree()</td>
<td>90</td>
</tr>
<tr>
<td>polyEval()</td>
<td>90</td>
</tr>
<tr>
<td>polyGcd()</td>
<td>90</td>
</tr>
<tr>
<td>polyQuotient()</td>
<td>91</td>
</tr>
<tr>
<td>polyRemainder()</td>
<td>91</td>
</tr>
<tr>
<td>polyRoots()</td>
<td>91</td>
</tr>
<tr>
<td>PowerReg</td>
<td>92</td>
</tr>
<tr>
<td>prg()</td>
<td>93</td>
</tr>
<tr>
<td>prodSeq()</td>
<td>93</td>
</tr>
<tr>
<td>Product (P!)</td>
<td>93</td>
</tr>
<tr>
<td>product()</td>
<td>93</td>
</tr>
<tr>
<td>propFrac()</td>
<td>94</td>
</tr>
<tr>
<td>RandPolynomial()</td>
<td>98</td>
</tr>
<tr>
<td>RandSample()</td>
<td>98</td>
</tr>
<tr>
<td>RandSeed</td>
<td>99</td>
</tr>
<tr>
<td>Real()</td>
<td>99</td>
</tr>
<tr>
<td>Rect()</td>
<td>99</td>
</tr>
<tr>
<td>ref()</td>
<td>100</td>
</tr>
<tr>
<td>remain()</td>
<td>100</td>
</tr>
<tr>
<td>Request</td>
<td>101</td>
</tr>
<tr>
<td>RequestStr</td>
<td>102</td>
</tr>
<tr>
<td>return()</td>
<td>102</td>
</tr>
<tr>
<td>root()</td>
<td>102</td>
</tr>
<tr>
<td>rotate()</td>
<td>104</td>
</tr>
<tr>
<td>round()</td>
<td>104</td>
</tr>
<tr>
<td>rowAdd()</td>
<td>105</td>
</tr>
<tr>
<td>rowDim()</td>
<td>105</td>
</tr>
<tr>
<td>rowNorm()</td>
<td>105</td>
</tr>
<tr>
<td>rowSwap()</td>
<td>105</td>
</tr>
<tr>
<td>rref()</td>
<td>105</td>
</tr>
<tr>
<td>sec()</td>
<td>106</td>
</tr>
<tr>
<td>sec^(-1)()</td>
<td>106</td>
</tr>
<tr>
<td>sech()</td>
<td>106</td>
</tr>
<tr>
<td>sech^(-1)()</td>
<td>107</td>
</tr>
<tr>
<td>seq()</td>
<td>107</td>
</tr>
<tr>
<td>seqGen()</td>
<td>108</td>
</tr>
<tr>
<td>seqn()</td>
<td>108</td>
</tr>
<tr>
<td>series()</td>
<td>109</td>
</tr>
<tr>
<td>setModel()</td>
<td>110</td>
</tr>
<tr>
<td>shift()</td>
<td>111</td>
</tr>
<tr>
<td>sign()</td>
<td>111</td>
</tr>
<tr>
<td>simul()</td>
<td>112</td>
</tr>
<tr>
<td>sin()</td>
<td>112</td>
</tr>
<tr>
<td>sinh()</td>
<td>113</td>
</tr>
<tr>
<td>sinh^(-1)()</td>
<td>113</td>
</tr>
<tr>
<td>sinh^(-1)()</td>
<td>114</td>
</tr>
<tr>
<td>SinReg</td>
<td>115</td>
</tr>
<tr>
<td>solve()</td>
<td>115</td>
</tr>
<tr>
<td>SortA</td>
<td>118</td>
</tr>
<tr>
<td>SortD</td>
<td>118</td>
</tr>
<tr>
<td>Sphere</td>
<td>119</td>
</tr>
<tr>
<td>sqrt()</td>
<td>119</td>
</tr>
<tr>
<td>statResults</td>
<td>120</td>
</tr>
<tr>
<td>statValues</td>
<td>121</td>
</tr>
<tr>
<td>stDevPop()</td>
<td>121</td>
</tr>
<tr>
<td>stDevSamp()</td>
<td>121</td>
</tr>
<tr>
<td>Store</td>
<td>122</td>
</tr>
<tr>
<td>string()</td>
<td>122</td>
</tr>
<tr>
<td>subMat()</td>
<td>122</td>
</tr>
<tr>
<td>Sum (Sigma)</td>
<td>122</td>
</tr>
<tr>
<td>sum()</td>
<td>123</td>
</tr>
<tr>
<td>sumIf()</td>
<td>123</td>
</tr>
<tr>
<td>sumSeq()</td>
<td>123</td>
</tr>
<tr>
<td>system()</td>
<td>123</td>
</tr>
</tbody>
</table>

**T**

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (transpose)</td>
<td>124</td>
</tr>
<tr>
<td>tan()</td>
<td>124</td>
</tr>
<tr>
<td>tan^(-1)()</td>
<td>125</td>
</tr>
</tbody>
</table>
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**

Example:

\[
\frac{12}{8} \cdot 2 \Rightarrow \frac{3}{4}
\]

*Note:* See also `/ (divide), page 144.

**Exponent template**

Example:

\[
2^3 \Rightarrow 8
\]

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

To return the cursor to the baseline, press right arrow (\(\triangleright\)).

*Note:* See also `^` (power), page 145.

**Square root template**

Example:

\[
\sqrt{4} \Rightarrow 2
\]

\[
\sqrt{\{9, a, 4\}} \Rightarrow \{3, \sqrt{a}, 2\}
\]

\[
\sqrt{4} \Rightarrow 2
\]

\[
\sqrt{\{9, 16, 4\}} \Rightarrow \{3, 4, 2\}
\]

*Note:* See also `\(\sqrt{()}\)` (square root), page 152.

**Nth root template**

Example:

\[
\sqrt[3]{8} \Rightarrow 2
\]

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

*Note:* See also `root()`, page 103.
### $e$ exponent template

\[ e \]

Natural exponential $e$ raised to a power

**Note:** See also $e^{}$, page 40.

---

### Log template

\[ \log \]

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

**Note:** See also $\log()$, page 71.

---

### Piecewise template (2-piece)

\[ \begin{cases} \text{piece1} & \text{condition1} \\ \text{piece2} & \text{condition2} \end{cases} \]

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

**Note:** See also $\text{piecewise}()$, page 88.

---

### Piecewise template (N-piece)

\[ \text{Catalog} > \text{piecewise}() \]

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 $\text{piecewise}()$, page 88.
System of 2 equations template

\[
\begin{align*}
\{ & x + y = 0 \\
& x - y = 5
\end{align*}
\]

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

Note: See also \texttt{system()}, page 123.

Example:
\[
\text{solve}\left\{ \begin{array}{l}
x + y = 0 \\
x - y = 5
\end{array} \right. \quad \text{so} \quad x = \frac{5}{2} \quad \text{and} \quad y = \frac{-5}{2}
\]

System of \(N\) equations template

Let you create a system of \(N\) equations. Prompts for \(N\).

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

Absolute value template

\[
\mid \quad 2, -3, 4 \quad \mid
\]

Note: See also \texttt{abs()}, page 7.

Example:
\[
\{ 2, -3, 4, \quad -4 \quad -3 \} \quad \quad \{ 2, 3, 4, 64 \}
\]

\(dd\text{"mm"ss ss}"\) template

\[
\text{"dd" mm ss ss}"\]

Let you enter angles in \(dd\text{"mm"ss ss}"\) format, where \(dd\) is the number of decimal degrees, \(mm\) is the number of minutes, and \(ss\text{ ss}"\) is the number of seconds.

Example:
\[
30^\circ 15'10" \quad \quad 10891.414800
\]

Matrix template (2 x 2)

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]

Create a 2 x 2 matrix.

Example:
\[
\begin{bmatrix}
a & 2 \cdot a \\
3 \cdot a & 4 \cdot a
\end{bmatrix}
\]
Matrix template (1 x 2)

Example:
\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]
\[
\begin{bmatrix}
0 & 0 \\
2 & 0
\end{bmatrix}
\]

Matrix template (2 x 1)

Example:
\[
\begin{bmatrix}
5 \\
8
\end{bmatrix}
\cdot 0.01
\begin{bmatrix}
0.05 \\
0.08
\end{bmatrix}
\]

Matrix template (m x n)

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

Example:
\[
\begin{bmatrix}
4 & 2 & 6 \\
1 & 2 & 3 \\
5 & 7 & 9
\end{bmatrix}
\]
\[
\begin{bmatrix}
4 \\
2 \\
9
\end{bmatrix}
\]

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)

Example:
\[
\sum_{n=3}^{7} (n) = 25
\]

Note: See also Σ() (sumSeq), page 153.

Product template (Π)

Example:
\[
\prod_{n=1}^{5} \left(\frac{1}{n}\right) = \frac{1}{120}
\]

Note: See also Π() (prodSeq), page 152.
### First derivative template

\[ \frac{d}{dx} (x^3) \]

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

**Note:** See also \( d() \) (derivative), page 150.

---

Example:

\[ \frac{d}{dx} (x^3) \]

\[ 3 \cdot x^2 \]

\[ \frac{d}{dx} (x^3) \bigg|_{x=3} \]

\[ 27 \]

### Second derivative template

\[ \frac{d^2}{dx^2} (x^3) \]

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

**Note:** See also \( d() \) (derivative), page 150.

---

Example:

\[ \frac{d^2}{dx^2} (x^3) \]

\[ 6 \cdot x \]

\[ \frac{d^2}{dx^2} (x^3) \bigg|_{x=3} \]

\[ 18 \]

### Nth derivative template

\[ \frac{d^n}{dx^n} (x^3) \]

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

**Note:** See also \( d() \) (derivative), page 150.

---

Example:

\[ \frac{d^3}{dx^3} (x^3) \bigg|_{x=3} \]

\[ 6 \]

### Definite integral template

\[ \int_{a}^{b} x^2 \, dx \]

**Note:** See also \( \int() \) integral), page 151.

---

Example:

\[ \int_{a}^{b} x^2 \, dx \]

\[ \frac{b^3}{3} - \frac{a^3}{3} \]

### Indefinite integral template

\[ \int x^2 \, dx \]

**Note:** See also \( \int() \) integral), page 151.

---

Example:

\[ \int x^2 \, dx \]

\[ \frac{x^3}{3} \]
Limit template

\[
\lim_{x \to a} \left( f(x) \right)
\]

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

**Note:** See also `limit()`, page 63.

Example:

\[
\lim_{x \to 5} (2x + 3)
\]
Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, starting on page 143. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

\textbf{abs()}

\begin{align*}
\text{abs(Expr1)} & \Rightarrow \text{expression} \\
\text{abs(List1)} & \Rightarrow \text{list} \\
\text{abs(Matrix1)} & \Rightarrow \text{matrix}
\end{align*}

Returns the absolute value of the argument.

\textbf{Note:} See also \textit{Absolute value template}, page 3.

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

\textbf{Note:} All undefined variables are treated as real variables.

\textbf{amortTbl()}

\begin{align*}
\text{amortTbl(NPmt, N, PV, [Pmt], [FY], [PpY], [CpY], [PmtAt], [roundValue])} & \Rightarrow \text{matrix}
\end{align*}

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

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


\begin{itemize}
\item If you omit \textit{Pmt}, it defaults to
\textit{Pmt} = \textit{tvmPmt(N, I, PV, FY, PpY, CpY, PmtAt)}.
\item If you omit \textit{FY}, it defaults to \textit{FY} = 0.
\item The defaults for \textit{PpY}, \textit{CpY}, and \textit{PmtAt} are the same as for the TVM functions.
\end{itemize}

\textit{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 \textit{n} is the balance after payment \textit{n}.

You can use the output matrix as input for the other amortization functions \textit{ΣInt()} and \textit{ΣPrnt()}, page 154, and \textit{bal()}, page 13.

\textbf{and}

\begin{align*}
\text{BooleanExpr1 and BooleanExpr2} & \Rightarrow \text{Boolean expression} \\
\text{BooleanList1 and BooleanList2} & \Rightarrow \text{Boolean list} \\
\text{BooleanMatrix1 and BooleanMatrix2} & \Rightarrow \text{Boolean matrix}
\end{align*}

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

\[ \text{Integer1 and Integer2} \Rightarrow \text{integer} \]

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).

\[ \text{In Hex base mode:} \]
\[ 0\text{b7AC36 and 0\text{b}3D5F} \Rightarrow 0\text{h2C16} \]

**Important:** Zero, not the letter O.

\[ \text{In Bin base mode:} \]
\[ 0\text{b100101 and 0b100} \Rightarrow 0\text{b100} \]

\[ \text{In Dec base mode:} \]
\[ 37 \text{ and 0b100} \Rightarrow 4 \]

**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()

\[ \text{angle(Eexpr)} \Rightarrow \text{expression} \]

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

**Note:** All undefined variables are treated as real variables.

\[ \text{In Degree angle mode:} \]
\[ \text{angle}(0+2\cdot i) \Rightarrow 90 \]

\[ \text{In Gradian angle mode:} \]
\[ \text{angle}(0+3\cdot i) \Rightarrow 100 \]

\[ \text{In Radian angle mode:} \]
\[ \text{angle}(1+i) \Rightarrow \frac{\pi}{4} \]
\[ \text{angle}(z) \Rightarrow \frac{\pi \cdot \text{sign}(z) \cdot 2}{2} \]
\[ \text{angle}(x+i \cdot y) \Rightarrow \frac{\pi \cdot \text{sign}(y) \cdot \tan^{-1}\left(\frac{x}{y}\right)}{2} \]

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

ANOVA

\[ \text{ANOVA(List1,List2,[List3,...,List20][,Flag])} \]

Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the \text{stat.results} variable. (See page 120.)

\[ \text{Flag}=0 \text{ for Data, Flag}=1 \text{ for Stats} \]

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.F}</td>
<td>Value of the \text{F} statistic</td>
</tr>
<tr>
<td>\text{stat.PVal}</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>\text{stat.df}</td>
<td>Degrees of freedom of the groups</td>
</tr>
<tr>
<td>\text{stat.SS}</td>
<td>Sum of squares of the groups</td>
</tr>
</tbody>
</table>
Output variable | Description
---|---
stat.MS | Mean squares for the groups
stat.dfError | Degrees of freedom of the errors
stat.SSError | Sum of squares of the errors
stat.MSError | Mean square for the errors
stat.sp | Pooled standard deviation
stat.xbarlist | Mean of the input of the lists
stat.CLowerList | 95% confidence interval for the mean of each input list
stat.CUpperList | 95% confidence intervals for the mean of each input list

**ANOVA2way**

`Catalog > ANOVA2way List1,List2,..,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 120.)

`LevRow=0` for Block

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

`Len=max(length(List1),length(List2)=...=length(List10) and Len/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

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.$F_{col}$</td>
<td>$F$ statistic of the column factor</td>
</tr>
<tr>
<td>stat.PValCol</td>
<td>Probability value of the column factor</td>
</tr>
<tr>
<td>stat.dfCol</td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td>stat.SSCol</td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td>stat.MSCol</td>
<td>Mean squares for column factor</td>
</tr>
</tbody>
</table>

### ROW FACTOR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.$F_{Row}$</td>
<td>$F$ statistic of the row factor</td>
</tr>
<tr>
<td>stat.PValRow</td>
<td>Probability value of the row factor</td>
</tr>
<tr>
<td>stat.dfRow</td>
<td>Degrees of freedom of the row factor</td>
</tr>
<tr>
<td>stat.SSRow</td>
<td>Sum of squares of the row factor</td>
</tr>
<tr>
<td>stat.MSRow</td>
<td>Mean squares for row factor</td>
</tr>
</tbody>
</table>

### INTERACTION Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.$F_{Interact}$</td>
<td>$F$ statistic of the interaction</td>
</tr>
<tr>
<td>stat.PValInteract</td>
<td>Probability value of the interaction</td>
</tr>
<tr>
<td>stat.dfInteract</td>
<td>Degrees of freedom of the interaction</td>
</tr>
<tr>
<td>stat.SSInteract</td>
<td>Sum of squares of the interaction</td>
</tr>
<tr>
<td>stat.MSInteract</td>
<td>Mean squares for interaction</td>
</tr>
</tbody>
</table>

### ERROR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSError</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Mean squares for the errors</td>
</tr>
<tr>
<td>$s$</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>
Ans

\( \text{Ans} \rightarrow \text{value} \)
Returns the result of the most recently evaluated expression.

\[
\begin{align*}
56 + 4 & \rightarrow 60 \\
60 + 4 & \rightarrow 64 \\
\end{align*}
\]

approx()\hspace{1cm} \text{Catalog >} \hspace{1cm} \text{keys}

\( \text{approx(Expr)} \rightarrow \text{expression} \)
Returns the evaluation of the argument as an expression containing decimal values, when possible, regardless of the current \text{Auto} or \text{Approximate} mode.

This is equivalent to entering the argument and pressing \( \text{ctrl enter} \).

\[
\begin{align*}
\text{approx}\left(\frac{1}{3}\right) & \rightarrow 0.333333 \\
\text{approx}\left(\frac{1}{3} + \frac{1}{9}\right) & \rightarrow \{0.333333, 0.111111\} \\
\text{approx}\{\sin(\pi), \cos(\pi)\} & \rightarrow \{0., 1.\} \\
\text{approx}\left[\sqrt{2} \cdot \sqrt{3}\right] & \rightarrow \{1.41421, 1.73205\} \\
\text{approx}\left[\frac{1}{3} + \frac{1}{9}\right] & \rightarrow \{0.333333, 0.111111\} \\
\text{approx}\{\sin(\pi), \cos(\pi)\} & \rightarrow \{0., 1.\} \\
\text{approx}\left[\sqrt{2} \cdot \sqrt{3}\right] & \rightarrow \{1.41421, 1.73205\}
\end{align*}
\]

approx(List1) \Rightarrow \text{list}  \\
approx(Matrix1) \Rightarrow \text{matrix}
Returns a list or matrix where each element has been evaluated to a decimal value, when possible.

\text{approxFraction()} \hspace{1cm} \text{Catalog > keys}

\text{Expr \textbf{approxFraction}([Tol]) \rightarrow expression} \\
\text{List \textbf{approxFraction}([Tol]) \rightarrow list}  \\
\text{Matrix \textbf{approxFraction}([Tol]) \rightarrow matrix}
Returns the input as a fraction, using a tolerance of \( \text{Tol} \). If \( \text{Tol} \) is omitted, a tolerance of 5.\text{E}-14 is used.

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

\[
\begin{align*}
\frac{1}{2} + \frac{1}{3} + \tan(\pi) & \rightarrow 0.833333 \\
0.83333333333333 & \cdot \text{approxFraction}(5.\text{E}-14) \\
5 & 6 \\
\{\pi, 1.5\} & \cdot \text{approxFraction}(5.\text{E}-14) \\
\frac{5419351}{3} & \rightarrow \frac{17250339}{2}
\end{align*}
\]

approxRational() \hspace{1cm} \text{Catalog > keys}

\text{approxRational(Expr, Tol) \rightarrow expression} \\
\text{approxRational(List, Tol) \rightarrow list}  \\
\text{approxRational(Matrix, Tol) \rightarrow matrix}
Returns the argument as a fraction using a tolerance of \( \text{Tol} \). If \( \text{Tol} \) is omitted, a tolerance of 5.\text{E}-14 is used.

\[
\begin{align*}
\text{approxRational}(0.333, 5 \cdot 10^{-5}) & \rightarrow \frac{333}{1000} \\
\text{approxRational}(\{0.2, 0.33, 4.125\}, 5 \cdot 10^{-14}) & \rightarrow \left\{\frac{1}{5}, \frac{33}{100}, \frac{33}{8}\right\}
\end{align*}
\]

arccos()\hspace{1cm} \text{See cos}^-1, \text{page 23.}
arccosh()  See cosh\(^{-1}\), page 24.

arccot()  See cot\(^{-1}\), page 25.

arccoth()  See coth\(^{-1}\), page 25.

arccsc()  See csc\(^{-1}\), page 27.

arccsch()  See csch\(^{-1}\), page 27.

arcLen()  

\[
\text{arcLen}(\text{Expr1}, \text{Var}, \text{Start}, \text{End}) \Rightarrow \text{expression}
\]

Returns the arc length of \text{Expr1} from \text{Start} to \text{End} with respect to variable \text{Var}.

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

\[
\text{arcLen}(\text{List1}, \text{Var}, \text{Start}, \text{End}) \Rightarrow \text{list}
\]

Returns a list of the arc lengths of each element of \text{List1} from \text{Start} to \text{End} with respect to \text{Var}.

arcsec()  See sec\(^{-1}\), page 106.

arcsech()  See sech\(^{-1}\), page 107.

arcsin()  See sin\(^{-1}\), page 113.

arcsinh()  See sinh\(^{-1}\), page 114.

arctan()  See tan\(^{-1}\), page 125.

arctanh()  See tanh\(^{-1}\), page 126.

augment()  

\[
\text{augment}(\text{List1}, \text{List2}) \Rightarrow \text{list}
\]

Returns a new list that is \text{List2} appended to the end of \text{List1}.
**augment()**

```
augment(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix}
```

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

```
\begin{pmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
```

```
\text{augment}(m1,m2)
```

```
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
```

**avgRC()**

```
avgRC(\text{Expr1}, \text{Var}=\text{Value}, \text{Step}) \Rightarrow \text{expression}
```

```
\begin{pmatrix}
\frac{f(x+h)-f(x)}{h} \\
\frac{\sin(h+2)-\sin(2)}{h} \\
2\cdot(x-0.4995) \\
2\cdot(x-0.45) \\
2\cdot(x+1)
\end{pmatrix}
```

```
avgRC(\sin(x), x, h)
```

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

\text{Expr1} can be a user-defined function name (see \text{func}).

When \text{Value} is specified, it overrides any prior variable assignment or any current "[" substitution for the variable.

\text{Step} is the step value. If \text{Step} is omitted, it defaults to 0.001.

Note that the similar function \text{centralDiff()} uses the central-difference quotient.

**Bal()**

```
\text{bal}(\text{NPmt}, \text{N}, \text{I}, \text{PV}, \text{Pmt}, \text{FV}, \text{PpY}, \text{CpY}, \text{PmtAt}, \text{roundValue}) \Rightarrow \text{value}
```

```
\text{bal}(5,6,5,75,5000,,12,12)
```

```
833.11
```

```
\text{bal}(4,\text{tbl})
```

```
1674.27
```

Amortization function that calculates schedule balance after a specified payment.

\text{N}, \text{I}, \text{PV}, \text{Pmt}, \text{FV}, \text{PpY}, \text{CpY}, \text{PmtAt} are described in the table of TVM arguments, page 132.

\text{NPmt} specifies the payment number after which you want the data calculated.

\text{N}, \text{I}, \text{PV}, \text{Pmt}, \text{FV}, \text{PpY}, \text{CpY}, \text{PmtAt} are described in the table of TVM arguments, page 132.

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

\text{roundValue} specifies the number of decimal places for rounding. Default=2.

\text{bal}(\text{NPmt}, \text{amortTable}) \text{ calculates the balance after payment number } \text{NPmt}, \text{based on amortization table } \text{amortTable}. \text{The amortTable argument must be a matrix in the form described under amortTbl()}, page 7.

\textbf{Note:} See also \text{gInt()} and \text{gPrn()}, page 154.
**Base2**

*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.

- 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

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

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

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

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

- \(-2^{63} - 1\) becomes \(2^{63} - 1\) and is displayed as
  0h7FFFFFFFFFFFF 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.

- Zero, not the letter O, followed by b or h.
- A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal. The result is displayed in decimal, regardless of the Base mode.
**Base16**

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 14.

**binomCdf()**

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 ≤ X ≤ upBound) ⇒ number if upBound is a number, list if upBound is a list

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

For P(X ≤ upBound), set lowBound=0

**binomPdf()**

binomPdf(n,p) → number

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

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

**ceiling()**

ceiling(Expr1) → integer

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.
centralDiff()

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

Returns the numerical derivative using the central difference quotient formula.

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

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

When using List1 or Matrix1, the operation gets mapped across the values in the list or across the matrix elements.

Note: See also avgRC() and d()).

cFactor()

cFactor(Expr1, Var) ⇒ expression
cFactor(List1, Var) ⇒ list
cFactor(Matrix1, Var) ⇒ matrix

cFactor(Expr1) returns Expr1 factored with respect to all of its variables over a common denominator.

Expr1 is factored as much as possible toward linear rational factors even if this introduces new non-real numbers. This alternative is appropriate if you want factorization with respect to more than one variable.

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

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

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

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

Note: See also factor().
char() \hspace{1cm} \text{Catalog > \hspace{1cm} $\left[\begin{array}{c} 38 \\ 65 \end{array}\right]$}

char(Integer) \Rightarrow \text{character}

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

\begin{align*}
\text{char}(38) &= "&E" \\
\text{char}(65) &= "A"
\end{align*}

charPoly() \hspace{1cm} \text{Catalog > \hspace{1cm} $\left[\begin{array}{c} 38 \\ 65 \end{array}\right]$}

charPoly(squareMatrix, Var) \Rightarrow \text{polynomial expression}

charPoly(squareMatrix, Expr) \Rightarrow \text{polynomial expression}

charPoly(squareMatrix1, Matrix2) \Rightarrow \text{polynomial expression}

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

\[ p_A(\lambda) = \det(\lambda \cdot I - A) \]

where \( I \) denotes the \( n \times n \) identity matrix.

squareMatrix1 and squareMatrix2 must have the equal dimensions.

\[
\begin{pmatrix}
1 & 3 & 0 \\
2 & -1 & 0 \\
-2 & 2 & 5 \\
\end{pmatrix}
\]

\[
\begin{array}{c}
\text{charPoly}(m,x) = -x^3 + 5 \cdot x^2 + 7 \cdot x - 35 \\
\text{charPoly}(m,x^2 + 1) = -x^6 + 2 \cdot x^4 + 14 \cdot x^2 - 24 \\
\text{charPoly}(m,m) = 0
\end{array}
\]

\( \chi^2 \)2way \hspace{1cm} \text{Catalog > \hspace{1cm} $\left[\begin{array}{c} 38 \\ 65 \end{array}\right]$}

\( \chi^2 \)2way obsMatrix

chi22way obsMatrix

Computes a \( \chi^2 \) 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 120.)

For information on the effect of empty elements in a matrix, see “Empty (Void) Elements” on page 162.

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

\( \chi^2 \)Cdf() \hspace{1cm} \text{Catalog > \hspace{1cm} $\left[\begin{array}{c} 38 \\ 65 \end{array}\right]$}

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

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

Computes the \( \chi^2 \) 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 162.
\( \chi^2 \text{GOF} \)

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

\[ \text{chi2GOF} \ obsList, \ expList, \ df \]

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 \( \text{stat.results} \) variable. (See page 120.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

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

\( \chi^2 \text{Pdf()} \)

\[ \chi^2 \text{Pdf}(XVal, df) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list} \]

\[ \text{chi2Pdf}(XVal, df) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list} \]

Computes the probability density function (pdf) for the \( \chi^2 \) 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 162.

**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 \text{unLock}, page 135.
**ClrErr**

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

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

**Note:** See also `PassErr`, page 88, and `Try`, page 130.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `@` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter` instead of `[enter]` at the end of each line. For an example of `ClErr`, See Example 2 under the `Try` command, page 130.

### `colAugment()`

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`.

### `colDim()`

Returns the number of columns contained in `Matrix`.

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

### `colNorm()`

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()`.

### `comDenom()`

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

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

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

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

**completeSquare()**

`completeSquare(ExprEqn, Var)` \( \Rightarrow \) expression or equation

`completeSquare(ExprEqn, Var^Power)` \( \Rightarrow \) expression or equation

`completeSquare(ExprEqn, Var1 Var2 [...] )` \( \Rightarrow \) expression or equation

`completeSquare(ExprEqn, {Var1 Var2 [...]})` \( \Rightarrow \) expression or equation

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

- 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 [... ]`. 
**conj()**

Catalog >

\[
\text{conj(expression)} \quad \text{conj(List1)} \quad \text{conj(Matrix1)}
\]

Returns the complex conjugate of the argument.

**Note:** All undefined variables are treated as real variables.

**conj\(1+2\cdot i\)**

\[1-2\cdot i\]

**conj\(\begin{array}{ccc}
2 & 1 & -3\cdot i \\
-1 & i & 7
\end{array}\)**

\[\begin{array}{ccc}
2 & 1 & 3\cdot i \\
i & i & 7
\end{array}\]

**conj\(z\)**

\[z\]

**conj\(x+i\cdot y\)**

\[x-y\cdot i\]

**constructMat()**

Catalog >

\[
\text{constructMat(Expr,Var1,Var2,numRows,numCols)} \quad \Rightarrow \text{matrix}
\]

Returns a matrix based on the arguments.

**Note:**

\(Expr\) 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 \(numCols\).

\[
\text{constructMat}\left(\frac{1}{i+j}, i, j, 3, 4\right) = \begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 3 & 4 & 5 \\
1 & 1 & 1 & 1 \\
3 & 4 & 5 & 6 \\
1 & 1 & 1 & 1 \\
4 & 5 & 6 & 7
\end{array}
\]

**CopyVar**

Catalog >

\[
\text{Var1, Var2}
\]

**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 indirect expression that simplifies to a variable name meeting the requirements.

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

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

**Define** \(a(x)=\frac{1}{x}\)

Done

**Define** \(b(x)=x^2\)

Done

**CopyVar** \(a,c::c(4)\)

\[1\]

**CopyVar** \(b,c::c(4)\)

\[16\]

**aa.a::=45**

45

**aa.b::=6.78**

6.78

**aa.c::=8.9**

8.9

**getVarInfo()**

\[
\begin{array}{m}{llll}
\text{aa.a} & "NUM" & \text{"n"} & \text{"n"} \\
\text{aa.b} & "NUM" & \text{"n"} & \text{"n"} \\
\text{aa.c} & "NUM" & \text{"n"} & \text{"n"}
\end{array}
\]

**CopyVar** \(aa.,bb.\)

Done

**getVarInfo()**

\[
\begin{array}{m}{llll}
\text{aa.a} & "NUM" & \text{"n"} & \text{"n"} \\
\text{aa.b} & "NUM" & \text{"n"} & \text{"n"} \\
\text{aa.c} & "NUM" & \text{"n"} & \text{"n"} \\
\text{bb.a} & "NUM" & \text{"n"} & \text{"n"} \\
\text{bb.b} & "NUM" & \text{"n"} & \text{"n"} \\
\text{bb.c} & "NUM" & \text{"n"} & \text{"n"}
\end{array}
\]

**TI-Nspire™ CAS Reference Guide**

21
\textbf{corrMat()}

\texttt{corrMat(List1,List2,...,[List20])}

Computes the correlation matrix for the augmented matrix \([\text{List1},\text{List2},...\text{List20}]\).

\textbf{\texttt{\textgreater\textgreater cos}}

\textit{Expr} \texttt{\textgreater\textgreater cos}

\textbf{Note:} You can insert this operator from the computer keyboard by typing @>cos.

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

\textbf{\textgreater\textgreater cos} reduces all powers of 
\[ \sin(...) \mod 1 - \cos(...)^2 \]
so that any remaining powers of \(\cos(...)\) have exponents in the range \((0, 2)\). Thus, the result will be free of \(\sin(...)\) if and only if \(\sin(...)\) occurs in the given expression only to even powers.

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

\textbf{\texttt{cos()}}

\texttt{cos(Expr1) \Rightarrow expression}

\texttt{cos(List1) \Rightarrow list}

\texttt{cos(Expr1)} returns the cosine of the argument as an expression.

\texttt{cos(List1)} returns a list of the cosines of all elements in \textit{List1}.

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

\begin{tabular}{|c|c|}
\hline
\text{In Degree angle mode:} & \\
\hline
\text{\texttt{cos(\pi/4)}} & \frac{\sqrt{2}}{2} \\
\text{\texttt{cos(45)}} & \frac{\sqrt{2}}{2} \\
\text{\texttt{cos\{\{0,60,90\}\}}} & \left\{1,\frac{1}{2},0\right\} \\
\hline
\text{In Gradian angle mode:} & \\
\hline
\text{\texttt{cos\{\{0,50,100\}\}}} & \left\{1,\frac{\sqrt{2}}{2},0\right\} \\
\hline
\text{In Radian angle mode:} & \\
\hline
\text{\texttt{cos(\pi/4)}} & \frac{\sqrt{2}}{2} \\
\text{\texttt{cos(45\circ)}} & \frac{\sqrt{2}}{2} \\
\hline
\end{tabular}
cos(__)

\( \cos(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

Returns the matrix cosine of \( \text{squareMatrix1} \). This is not the same as calculating the cosine of each element.

When a scalar function \( f(A) \) operates on \( \text{squareMatrix1} (A) \), the result is calculated by the algorithm:

Compute the eigenvalues \( (\lambda_i) \) and eigenvectors \( (V_i) \) of \( A \).

\( \text{squareMatrix1} \) must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

Form the matrices:

\[
B = \begin{bmatrix}
\lambda_1 & 0 & \ldots & 0 \\
0 & \lambda_2 & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & \lambda_n
\end{bmatrix}
\]

and \( X = [V_1, V_2, \ldots, V_n] \).

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

\[
\cos(B) = 
\begin{bmatrix}
\cos(\lambda_1) & 0 & \ldots & 0 \\
0 & \cos(\lambda_2) & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & \cos(\lambda_n)
\end{bmatrix}
\]

All computations are performed using floating-point arithmetic.

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

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

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

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

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

**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 \( \arccos(\ldots) \).

\( \cos^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

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

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.
cosh()

\[ \text{cosh}(\text{Expr1}) \Rightarrow \text{expression} \]
\[ \text{cosh}(\text{List1}) \Rightarrow \text{list} \]
\[ \text{cosh}(\text{Expr1}) \text{ returns the hyperbolic cosine of the argument as an expression.} \]
\[ \text{cosh}(\text{List1}) \text{ returns a list of the hyperbolic cosines of each element of List1.} \]
\[ \text{cosh}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]
\[ \text{Returns the matrix hyperbolic cosine of squareMatrix1. This is not the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to cos().} \]
\[ \text{squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.} \]

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1 \\
\end{bmatrix}
\]
\[
\begin{bmatrix}
421.255 & 253.909 & 216.905 \\
327.635 & 255.301 & 202.958 \\
226.297 & 216.623 & 167.628 \\
\end{bmatrix}
\]

\[ \text{In Radian angle mode:} \]

\[ \text{cosh}^{-1}() \]

\[ \text{cosh}^{-1}(\text{Expr1}) \Rightarrow \text{expression} \]
\[ \text{cosh}^{-1}(\text{List1}) \Rightarrow \text{list} \]
\[ \text{cosh}^{-1}(\text{Expr1}) \text{ returns the inverse hyperbolic cosine of the argument as an expression.} \]
\[ \text{cosh}^{-1}(\text{List1}) \text{ returns a list of the inverse hyperbolic cosines of each element of List1.} \]
\[ \text{Note:} \text{ You can insert this function from the keyboard by typing } \texttt{arccosh(...).} \]
\[ \text{cosh}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]
\[ \text{Returns the matrix inverse hyperbolic cosine of squareMatrix1. This is not the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to cos().} \]
\[ \text{squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.} \]

\[ \begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1 \\
\end{bmatrix}
\]
\[
\begin{bmatrix}
2.52503+1.73485i & 0.009241-1.49086i \\
0.489696-0.725533i & 1.66262+0.623491i \\
-0.322354-2.08316i & 1.26707+1.79018i \\
\end{bmatrix}
\]
\[ \text{To see the entire result, press } \mathbf{\Delta} \text{ and then use } \mathbf{\langle} \text{ and } \mathbf{\rangle} \text{ to move the cursor.} \]

cot()

\[ \text{cot}(\text{Expr1}) \Rightarrow \text{expression} \]
\[ \text{cot}(\text{List1}) \Rightarrow \text{list} \]
\[ \text{Returns the cotangent of } \text{Expr1} \text{ or returns a list of the cotangents of all elements in List1.} \]
\[ \text{Note:} \text{ The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use } ^\circ, _\text{G}, \text{ or } \text{i} \text{ to override the angle mode temporarily.} \]

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1 \\
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 0,1.37286,\text{cosh}^{-1}(3) \\
0 & 2.52503+1.73485i & 0.009241-1.49086i \\
1,5 & 0.489696-0.725533i & 1.66262+0.623491i \\
3,6 & -0.322354-2.08316i & 1.26707+1.79018i \\
\end{bmatrix}
\]
\[ \text{In Degree angle mode:} \]
\[ \text{In Gradian angle mode:} \]
\[ \text{In Radian angle mode:} \]

\[
\begin{bmatrix}
\frac{1}{\tan(1)} & 0.584848 & \frac{1}{\tan(3)} \\
\end{bmatrix}
\]

24  \textit{TI-Nspire™ CAS Reference Guide}
\( \cot^{-1}(\text{Expr1}) \Rightarrow \text{expression} \)

\( \cot^{-1}(\text{List1}) \Rightarrow \text{list} \)

Returns the angle whose cotangent is \( \text{Expr1} \) or returns a list containing the inverse cotangents of each element of \( \text{List1} \).

**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 \( \text{arccot}(\ldots) \).

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

In Degree angle mode:

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

In Gradian angle mode:

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

In Radian angle mode:

\( \cot^{-1}(1) \quad \pi/4 \)

\( \coth() \)

\( \coth(\text{Expr1}) \Rightarrow \text{expression} \)

\( \coth(\text{List1}) \Rightarrow \text{list} \)

Returns the hyperbolic cotangent of \( \text{Expr1} \) or returns a list of the hyperbolic cotangents of all elements of \( \text{List1} \).

\[
\begin{align*}
\coth(1.2) & \quad 1.19954 \\
\coth\left\{1,3,2\right\} & \quad \frac{1}{\tanh(1)} \quad 1.00333 \\
\end{align*}
\]

\( \coth^{-1}(\text{Expr1}) \Rightarrow \text{expression} \)

\( \coth^{-1}(\text{List1}) \Rightarrow \text{list} \)

Returns the inverse hyperbolic cotangent of \( \text{Expr1} \) or returns a list containing the inverse hyperbolic cotangents of each element of \( \text{List1} \).

**Note:** You can insert this function from the keyboard by typing \( \text{arccoth}(\ldots) \).

\[
\begin{align*}
\coth^{-1}(3.5) & \quad 0.293893 \\
\coth^{-1}\left\{-2,2,1,6\right\} & \quad \left\{\frac{-\ln(3)}{2},0.518046,\frac{\ln(7)}{5}\right\} \\
\end{align*}
\]

\( \text{count()} \)

\( \text{count}(\text{Value1orList1[,Value2orList2[...]]}) \Rightarrow \text{value} \)

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 162.

\[
\begin{align*}
\text{count}(2,4,6) & \quad 3 \\
\text{count}(\{2,4,6\}) & \quad 3 \\
\text{count}(\{2,\{4,6\},[8,10],[12,14]\}) & \quad 7 \\
\text{count}\left\{\frac{1}{2},3+4i,\text{undef},"hello",x+5.,\text{sign}(0)\right\} & \quad 2 \\
\end{align*}
\]

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

\[
\text{countif(List,Criteria) } \Rightarrow \text{ value}
\]

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 162.

**Note:** See also sumIf(), page 123, and frequency(), page 51.

\[\begin{align*}
\text{countIf}\left(\{1,3,"abc",\text{undef},3,1\}\right),3\right) & \Rightarrow 2 \\
\text{Counts the number of elements equal to 3.} \\
\text{countIf}\left(\{"abc","def","abc",3\},"def"\right) & \Rightarrow 1 \\
\text{Counts the number of elements equal to "def."} \\
\text{countIf}\left(\{x^{-2},x^{-1},1,x,x^2\},x\right) & \Rightarrow 1 \\
\text{Counts the number of elements equal to } x; \text{ this example assumes the variable } x \text{ is undefined.} \\
\text{countIf}\left(\{1,3,5,7,9\},?<5\right) & \Rightarrow 2 \\
\text{Counts } 1 \text{ and } 3. \\
\text{countIf}\left(\{1,3,5,7,9\},2<?<8\right) & \Rightarrow 3 \\
\text{Counts } 3, 5, \text{ and } 7. \\
\text{countIf}\left(\{1,3,5,7,9\},?<4 \text{ or } ?>6\right) & \Rightarrow 4 \\
\text{Counts } 1, 3, 7, \text{ and } 9.
\end{align*}\]

cPolyRoots()

\[\begin{align*}
\text{cPolyRoots(Poly,Var) } & \Rightarrow \text{ list} \\
\text{cPolyRoots(ListOfCoeffs) } & \Rightarrow \text{ list}
\end{align*}\]

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

\(\text{Poly}\) must be a polynomial in one variable.

The second syntax, \(\text{cPolyRoots(ListOfCoeffs)}\), returns a list of complex roots for the coefficients in \(\text{ListOfCoeffs}\).

**Note:** See also polyRoots(), page 91.

\[\begin{align*}
\text{cPolyRoots}\left(x^3+1,\text{y}\right) & \Rightarrow \{-1\} \\
\text{cPolyRoots}\left(x^3+1,\text{y}\right) & \Rightarrow \{-1, \frac{1}{2} + \frac{\sqrt{3}}{2} \text{i}, \frac{1}{2} - \frac{\sqrt{3}}{2} \text{i}\} \\
\text{cPolyRoots}\left(x^2+2x+1,\text{y}\right) & \Rightarrow \{-1,-1\} \\
\text{cPolyRoots}\left\{1,2,1\right\} & \Rightarrow \{-1,-1\}
\end{align*}\]

crossP()

\[\begin{align*}
\text{crossP(List1, List2) } & \Rightarrow \text{ list} \\
\text{crossP(Vector1, Vector2) } & \Rightarrow \text{ vector}
\end{align*}\]

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.

\[\begin{align*}
\text{crossP}\left(\{a1,b1\},\{a2,b2\}\right) & \Rightarrow \{0,0,a1\cdot b2-a2\cdot b1\} \\
\text{crossP}\left(\{0.1,2.2,\text{5}\},\{1,-0.5,0\}\right) & \Rightarrow \{-2.5,-\text{5},2.25\}
\end{align*}\]

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

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

\[ \text{csc}(\text{Expr}_1) \Rightarrow \text{expression} \]
\[ \text{csc}(\text{List}_1) \Rightarrow \text{list} \]

Returns the cosecant of \(\text{Expr}_1\) or returns a list containing the cosecants of all elements in \(\text{List}_1\).

In Degree angle mode:
\[ \text{csc}(45) \rightarrow \frac{1}{\sin(45)} \]
\[ \sqrt{2} \]

In Gradian angle mode:
\[ \text{csc}(50) \rightarrow \frac{1}{\sin(50)} \]
\[ \sqrt{2} \]

In Radian angle mode:
\[ \text{csc}\left\{\frac{\pi}{2}, \frac{\pi}{3}\right\} \rightarrow \left\{\frac{1}{\sin(1)}, \frac{2}{\sqrt{3}}\right\} \]

\[ \text{csch}() \]

\[ \text{csch}(\text{Expr}_1) \Rightarrow \text{expression} \]
\[ \text{csch}(\text{List}_1) \Rightarrow \text{list} \]

Returns the hyperbolic cosecant of \(\text{Expr}_1\) or returns a list of the hyperbolic cosecants of all elements of \(\text{List}_1\).

\[ \text{csch}(3) \rightarrow \frac{1}{\sinh(3)} \]

\[ \text{csch}\left\{\frac{\pi}{2}, \frac{\pi}{3}\right\} \rightarrow \left\{\frac{1}{\sinh(1)}, 0.248641, \frac{1}{\sinh(4)}\right\} \]

\[ \text{csch}^{-1}() \]

\[ \text{csch}^{-1}(\text{Expr}_1) \Rightarrow \text{expression} \]
\[ \text{csch}^{-1}(\text{List}_1) \Rightarrow \text{list} \]

Returns the inverse hyperbolic cosecant of \(\text{Expr}_1\) or returns a list containing the inverse hyperbolic cosecants of each element of \(\text{List}_1\).

\[ \text{csch}^{-1}(1) \rightarrow \sinh^{-1}(1) \]
\[ \frac{1}{3} \]

\[ \text{csch}^{-1}\left\{\frac{\pi}{2}, \frac{\pi}{3}\right\} \rightarrow \left\{\sinh^{-1}(1), 0.459815, \sinh^{-1}\left(\frac{1}{3}\right)\right\} \]
cSolve() is a function that returns candidate complex solutions of an equation or inequality for \( Var \). The goal is to produce candidates for all real and non-real solutions. Even if \( Equation \) is real, \( \text{cSolve()} \) allows non-real results in Real result Complex Format.

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

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

\( \text{cSolve()} \) starts with exact symbolic methods. \( \text{cSolve()} \) also uses iterative approximate complex polynomial factoring, if necessary.

**Note:** See also \( \text{cZeros()} \), \( \text{solve()} \), and \( \text{zeros()} \).

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

In Display Digits mode of Fix 2:

\[
\text{cSolve}(\text{Eqn1 and Eqn2 [and ...]}, \ VarOrGuess1, \ VarOrGuess2 [and ...]) \Rightarrow \text{Boolean expression}
\]

Returns candidate complex solutions to the simultaneous algebraic equations, where each \( \text{varOrGuess} \) specifies a variable that you want to solve for.

Optionally, you can specify an initial guess for a variable. Each \( \text{varOrGuess} \) must have the form:

- \( \text{variable} \)
- \( \text{variable} = \text{real or non-real number} \)

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

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

**Note:** The following examples use an underscore (press \( /_\)) so that the variables will be treated as complex.
Complex solutions can include both real and non-real solutions, as in the example to the right.

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

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

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

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

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

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

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

Computes the cubic polynomial regression \( y = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

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

\( \text{Category} \) is a list of category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) 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 162.

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

**cumulativeSum()**

\( \text{cumulativeSum} \) \( \text{List1} \) \( \Rightarrow \) list

Returns a list of the cumulative sums of the elements in \( \text{List1} \), starting at element 1.

\( \text{cumulativeSum} \) \( \text{Matrix1} \) \( \Rightarrow \) matrix

Returns a matrix of the cumulative sums of the elements in \( \text{Matrix1} \). Each element is the cumulative sum of the column from top to bottom.

An empty (void) element in \( \text{List1} \) or \( \text{Matrix1} \) produces a void element in the resulting list or matrix. For more information on empty elements, see page 162.

| \( \text{cumulativeSum} \) \( \{1,2,3,4\} \) | \( \{1,3,6,10\} \) |
| \( \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \) | \( \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \) |

**cumulativeSum**

\( \text{cumulativeSum} \) \( m1 \) \( \Rightarrow \) list

Cumulative sums of the elements in \( m1 \). Each element is the cumulative sum of the column from top to bottom.
Cycle

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 Enter 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 \( g() = \frac{\text{Func}}{\text{Done}} \)

\[
\text{Local } temp, i \\
0 \to temp \\
\text{For } i, 1, 100, 1 \\
\text{If } i = 50 \\
\text{Cycle} \\
temp + i \to temp \\
\text{EndFor} \\
\text{Return } temp \\
\text{EndFunc}
\]

\( g() \) 5000

\( \text{Cylind} \)

\( \text{Vector} \text{Cylind} \)

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

Displays the row or column vector in cylindrical form \([r, \pm \theta, z]\).

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

\( \text{cZeros()} \)

\( \text{cZeros}(\text{Expr}, \text{Var}) \Rightarrow \text{list} \)

Returns a list of candidate real and non-real values of \( \text{Var} \) that make \( \text{Expr}=0 \). \( \text{cZeros()} \) does this by computing \( \text{expList} (\text{cSolve}(\text{Expr}=0, \text{Var}) , \text{Var}) \). Otherwise, \( \text{cZeros()} \) is similar to\( \text{zeros()} \).

Note: See also \( \text{cSolve()} \), \( \text{solve()} \), and \( \text{zeros()} \).

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

You should also use \( \text{var}_- \) for any other variables in \( \text{Expr} \) that might have unreal values. Otherwise, you may receive unexpected results.

\( \text{cZeros}((\text{Expr1}, \text{Expr2} [, ... ]), \{\text{VarOrGuess1}, \text{VarOrGuess2} [, ... ]\}) \Rightarrow \text{matrix} \)

Returns candidate positions where the expressions are zero simultaneously. Each \( \text{VarOrGuess} \) specifies an unknown whose value you seek.

Optionally, you can specify an initial guess for a variable. Each \( \text{VarOrGuess} \) must have the form:

\( \text{variable} \\
= \text{or} \ \\
\text{variable} = \text{real or non-real number} \)

For example, \( x \) is valid and so is \( x=3+i \).
\textbf{cZeros()}

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

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

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

\begin{align*}
\text{Extract row 2:} & \quad \begin{pmatrix}
0 & 0 \\
1 - \sqrt{3} & 1 + \sqrt{3} \\
2 & 2 \\
1 + \sqrt{3} & 1 - \sqrt{3} \\
2 & 2
\end{pmatrix}
\end{align*}

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

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

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

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

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

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

\begin{align*}
\text{cZeros}\left( u_-, v_-, v_-^2 + u_-, \{u_-, v_-\} \right) & \quad \begin{pmatrix}
0 \\
1 - \sqrt{3} \\
2 \\
1 + \sqrt{3} \\
2
\end{pmatrix} \\
\text{cZeros}\left( u_-, v_- - c_- v_- v_-^2 + u_-, \{u_-, v_-\} \right) & \quad \begin{pmatrix}
0 \\
\sqrt{1 - 4 c_-} - 1 \\
4 \\
\sqrt{1 - 4 c_-} + 1 \\
4
\end{pmatrix} \\
\text{cZeros}\left( u_-, v_- - v_-^2 + u_-, \{u_-, v_- v_-\} \right) & \quad \begin{pmatrix}
0 & c4 \\
1 - \sqrt{3} & c4 \\
2 & 2 \\
1 + \sqrt{3} & c4 \\
2 & 2
\end{pmatrix} \\
\text{cZeros}\left( u_+ + v_- e^{w_-} - u_- v_- i, \{u_+, v_-\} \right) & \quad \begin{pmatrix}
e^{w_- + i} & e^{w_- - i} \\
2 & 2
\end{pmatrix} \\
\text{cZeros}\left( e^z - w_- w_- z_-^2, \{w_- z_-\} \right) & \quad \begin{pmatrix}
0.494866 & 0.703467 \\
1.58805 & 1.54022 i
\end{pmatrix} \\
\text{cZeros}\left( e^z - w_- w_- z_-^2, \{w_- z_- = 1 + i\} \right) & \quad \begin{pmatrix}
0.149606 + 4.8919 i & 1.58805 + 1.54022 i
\end{pmatrix}
\end{align*}
\textit{dbd()} \quad \text{Catalog >}

\texttt{dbd(date1,date2) \Rightarrow value}

Returns the number of days between \texttt{date1} and \texttt{date2} using the actual-day-count method.

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

\texttt{date1} and \texttt{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.

\texttt{MM.DDYY} (format used commonly in the United States)
\texttt{DDMM.YY} (format use commonly in Europe)

\texttt{Expr1} \texttt{\textbackslash DD} \Rightarrow value
\texttt{List1} \texttt{\textbackslash DD} \Rightarrow list
\texttt{Matrix1} \texttt{\textbackslash DD} \Rightarrow matrix

\textbf{Note:} You can insert this operator from the computer keyboard by typing \texttt{\textbackslash 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.

\texttt{\frac{1}{3}} \texttt{\textbackslash Decimal} \Rightarrow 0.333333

\texttt{\textbackslash Decimal} \texttt{\Rightarrow expression}
\texttt{List1} \texttt{\textbackslash Decimal} \Rightarrow \texttt{expression}
\texttt{Matrix1} \texttt{\textbackslash Decimal} \Rightarrow \texttt{expression}

\textbf{Note:} You can insert this operator from the computer keyboard by typing \texttt{\textbackslash Decimal}.

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

Define \( V = \text{Expression} \)

Define \( \text{Function}(\text{Param1}, \text{Param2}, \ldots) = \text{Expression} \)

Defines the variable \( V \) or the user-defined function \( \text{Function} \).

Parameters, such as \( \text{Param1} \), 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 \( \text{Expression} \) using the supplied arguments.

\( V \) and \( \text{Function} \) 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:

\[
\text{expression} \rightarrow \text{Function}(\text{Param1, Param2}).
\]

Define \( \text{Function}(\text{Param1}, \text{Param2}, \ldots) = \text{Func} \)

\text{EndFunc}

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

\text{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 \( \text{If, Then, Else, and For} \)).

**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 34, and **Define LibPub**, page 35.

Define LibPriv

Define LibPriv \( V = \text{Expression} \)

Define LibPriv \( \text{Function}(\text{Param1}, \text{Param2}, \ldots) = \text{Expression} \)

Define LibPriv \( \text{Function}(\text{Param1}, \text{Param2}, \ldots) = \text{Func} \)

\text{EndFunc}

Define LibPriv \( \text{Program}(\text{Param1}, \text{Param2}, \ldots) = \text{Prgm} \)

\text{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 34, and **Define LibPub**, page 35.
Define LibPub

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 34, and Define LibPriv, page 34.

deltaList()  See ΔList(), page 67.

deltaTmpCnv()  See ΔtmpCnv(), page 129.

DelVar

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 135.

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 → a
(a+2)² → b
DelVar a
(a+2)² → c

aa.a:=45
aa.b:=5.67
aa.c:=78.9

aa.a "NUM" "\[\text{45}\]"
aa.b "NUM" "\[\text{5.67}\]"
aa.c "NUM" "\[\text{78.9}\]"

getVarInfo()
|
| "aa.a"
| "aa.b"
| "aa.c"

DelVar aa.

getVarInfo()
| "NONE"
```

delVoid()

delVoid(List1) ⇒ list

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

For more information on empty elements, see page 162.

derivative()

See d(), page 150.
\textbf{deSolve()}

\textbf{deSolve(1stOr2ndOrderODE, Var, depVar)}

\(\Rightarrow\) \text{a general solution}

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

- Use a prime symbol (press \(\text{[alt]+}\)) to denote the 1st derivative of the dependent variable with respect to the independent variable.
- Use two prime symbols to denote the corresponding second derivative.

The prime symbol is used for derivatives within \texttt{deSolve()}. In other cases, use \texttt{d}().

The general solution of a 1st-order equation contains an arbitrary constant of the form \(c_k\), where \(k\) is an integer suffix from 1 through 255. The solution of a 2nd-order equation contains two such constants.

Apply \texttt{solve()} to an implicit solution if you want to try to convert it to one or more equivalent explicit solutions.

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

\begin{align*}
\text{deSolve(1stOrderODE and initCond, Var, depVar)} & \Rightarrow \text{a particular solution} \\
\text{deSolve(2ndOrderODE and initCond1 and initCond2, Var, depVar)} & \Rightarrow \text{a particular solution}
\end{align*}

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

\(y=(c_3\cdot x+c_4)\cdot e^{-x}+x^2-4\cdot x+6\)

\textservice{right(Ans) \rightarrow temp}

\(\begin{align*}
\frac{d^2}{dx^2}(\text{temp})+2\cdot \frac{d}{dx}(\text{temp})+\text{temp} \cdot x^2 &= 0 \\
\text{DelVar temp} & \text{ Done}
\end{align*}\)

\texttt{deSolve(y'=(\cos(y))^2\cdot x,x,y)}

\(\tan(y) \frac{x^2}{2} + c_4\)

\texttt{solve(Ans,y)}

\(y=\tan\left(\frac{x^2+2\cdot c_4}{2}\right)+n\cdot 3\cdot \pi\)

\texttt{Ans|c4=c-1 and n3=0}

\(y=\tan\left(\frac{x^2+2\cdot (c-1)}{2}\right)\)

\texttt{sin(y)=(y\cdot e^x+cos(y))\cdot y' \rightarrow ode}

\(\sin(y)=(e^x\cdot y+\cos(y))\cdot y'\)

\texttt{deSolve\(ode\ and \ y(0)=0,x,y) \rightarrow soln\)}

\(\frac{2\cdot \sin(y) \cdot y^2}{2} = (e^x-1)\cdot e^{-x}\cdot \sin(y)\)

\texttt{solve(ode | x=0 and y=0) true}

\texttt{ode|y'='impDiff(soln,x,y) true}

\texttt{DelVar ode,soln} \text{ Done}

\texttt{deSolve\(y''=y^2\ and \ y(0)=0\ and \ y'(0)=0,t,y\)}

\(\frac{-1}{3} = t\)

\texttt{solve(Ans,y)}

\(\frac{2}{2}\cdot \left(\frac{3\cdot t}{3}\right)^{\frac{4}{3}} \text{ and } t \geq 0\)
deSolve(2ndOrderODE and bndCond1 and bndCond2, Var, depVar) ⇒ particular solution

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

det()

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 \[ \text{ctrl enter} \] 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:

\[ 5 \times 10^{-14} \cdot \max(\text{dim}(\text{squareMatrix})) \cdot \text{rowNorm}(\text{squareMatrix}) \]

diag()

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

diag(squareMatrix) ⇒ rowMatrix

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

dim()

Returns the dimension of List.

dim(Matrix) ⇒ list

Returns the dimensions of matrix as a two-element list (rows, columns).

dim(String) ⇒ integer

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

Disp [exprOrString1] [, exprOrString2] ...
Displays the arguments in the Calculator history. The arguments are displayed in succession, with thin spaces as separators.
Useful mainly in programs and functions to ensure the display of intermediate calculations.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \( \text{[alt]} \) instead of \( \text{[enter]} \) at the end of each line. On the computer keyboard, hold down Alt and press Enter.

\( \text{Disp} \)

\( \text{Disp} \) \( \text{[start,end]} \) \( \text{= Prgm} \)
For \( i, \text{start}, \text{end} \)
\( \text{Disp i, } '' \text{,char(i)} \)
EndFor
EndPrgm

\( \text{Done} \)

\( \text{chars(240,243)} \)

\( 240 \delta \)
\( 241 \text{fn} \)
\( 242 \delta \)
\( 243 \delta \)

\( \text{Done} \)

\( \text{DMS} \)

\( \text{Expr} \) \( \text{DMS} \)
\( \text{List} \) \( \text{DMS} \)
\( \text{Matrix} \) \( \text{DMS} \)

Note: You can insert this operator from the computer keyboard by typing \( \text{[alt]} \text{DMS} \).
Interprets the argument as an angle and displays the equivalent DMS \((\text{DDDDDD} \angle \text{MM}' \text{SS.ss}''')\) number. See \( 0', ' \) on page 156 for DMS (degree, minutes, seconds) format.

Note: \( \text{DMS} \) 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 \( \text{DMS} \) only at the end of an entry line.

\( \text{domain()} \)

\( \text{domain(Expr1, Var)} \) \( \Rightarrow \) \text{expression}
Returns the domain of \( \text{Expr1} \) with respect to \( \text{Var} \).
\( \text{domain() \rangle} \) can be used to examine domains of functions. It is restricted to real and finite domain.
This functionality has limitations due to shortcomings of computer algebra simplification and solver algorithms.
Certain functions cannot be used as arguments for \( \text{domain()} \), regardless of whether they appear explicitly or within user-defined variables and functions. In the following example, the expression cannot be simplified because \( f(t) \) is a disallowed function.

\[ \text{domain} \left( \begin{align*}
\frac{x}{t} \\
\frac{1}{t}
\end{align*} \right) \]
\( \text{domain} \left( \begin{align*}
\int \frac{1}{t} \\
1
\end{align*} \right) \)

\( \text{Catalog} > \frac{x}{t} \)

\( \left[ \begin{array}{c}
\frac{x}{t} \\
\frac{1}{t}
\end{array} \right] \frac{1}{t} \)

\( \text{domain(} x^2, x \text{)} \) \( \Rightarrow \) \( x \in (-\infty, \infty) \)
\( \text{domain(} \frac{x+1}{x^2+2}, x \text{)} \) \( \Rightarrow \) \( x \neq -2 \text{ and } x \neq 0 \)
\( \text{domain(} \sqrt{x}, x \text{)} \) \( \Rightarrow \) \( 0 \leq x \leq \infty \)
\( \text{domain(} \frac{1}{x+y}, y \text{)} \) \( \Rightarrow \) \( y \neq x \)
**dominantTerm()**

**Catalog >**

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

**dominantTerm(Expr₁, Var [, Point]) | Var=Point**

⇒ expression

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

⇒ expression

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

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

**dominantTerm(…)** returns "dominantTerm(…)", if it is unable to determine such a representation, such as for essential singularities such as \( \sin\left(\frac{1}{z}\right) \) at \( z=0 \), \( e^{-\frac{1}{z}} \) at \( z=0 \), or \( e^z \) at \( z = \infty \) or \( -\infty \).

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

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

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

**Note:** See also **series()**, page 109.

**dotP()**

**Catalog >**

**dotP(List₁, List₂) ⇒ expression**

Returns the "dot" product of two lists.

**dotP(Vector₁, Vector₂) ⇒ expression**

Returns the "dot" product of two vectors.

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

---

**Examples:**

\[
\begin{align*}
\text{dominantTerm}\left(\tan\sin(x) - \sin(\tan(x)), x\right) & \Rightarrow \frac{x}{2}\left(1 - \frac{7}{30}\right)
\\
dominantTerm\left(\frac{1 - \cos(x - 1)}{(x - 1)^3}, x, 1\right) & \Rightarrow \frac{1}{2(1 - x)}
\\
dominantTerm\left(x^{-2} \cdot \tan(x^3), x\right) & \Rightarrow \frac{5}{x^3}
\\
dominantTerm\left(\ln(x^{1/2} - x^{-2}), x\right) & \Rightarrow \frac{\ln(x \cdot \ln(x))}{x^2}
\\
dominantTerm\left(e^{z_1 \cdot z_2}, z_1, z_2\right) & \Rightarrow e^{-1}
\\
dominantTerm\left(1 + \frac{1}{n}, n, \infty\right) & \Rightarrow e
\\
dominantTerm\left(\tan\left(\frac{1}{x}\right), x, 0\right) & \Rightarrow \frac{\pi \cdot \text{sign}(x)}{2}
\\
dominantTerm\left(\tan\left(\frac{1}{x}\right), x > 0\right) & \Rightarrow \frac{\pi}{2}
\\
\text{dotP}\left\{\{a, b, c\}, \{d, e, f\}\right\} & \Rightarrow a \cdot d + b \cdot e + c \cdot f
\\
\text{dotP}\left\{\{1, 2\}, \{5, 6\}\right\} & \Rightarrow 17
\\
\text{dotP}\left\{\{a \ b \ c\}, \{d \ e \ f\}\right\} & \Rightarrow a \cdot d + b \cdot e + c \cdot f
\\
\text{dotP}\left\{\{1 \ 2 \ 3\}, \{4 \ 5 \ 6\}\right\} & \Rightarrow 32
\end{align*}
\]
E

\(e^x()\)

\(e^x(Expr1) \rightarrow expression\)

Returns \(e\) raised to the \(Expr1\) power.

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

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

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

\(e^x(List1) \rightarrow list\)

Returns \(e\) raised to the power of each element in \(List1\).

\(e^x(squareMatrix1) \rightarrow squareMatrix\)

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 \(\cos()\).

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

\(eff()\)

\(eff(nominalRate,CpY) \rightarrow value\)

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 82.

\(eigVc()\)

\(eigVc(squareMatrix) \rightarrow matrix\)

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 \(V = [x_1, x_2, \ldots, x_n]\), then:

\(x_1^2 + x_2^2 + \ldots + x_n^2 = 1\)

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

In Rectangular Complex Format:

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

\[
\begin{bmatrix}
-1 & 2 & 5 \\
3 & 6 & 9 \\
2 & 5 & 7 \\
\end{bmatrix}
\]

\(eigVc(m1)\)

\[
\begin{bmatrix}
-0.800906 & 0.767947 & ( \\
0.484029 & 0.573804 & +0.052258\cdot i \\
0.352512 & 0.262687 & +0.096286\cdot i \\
\end{bmatrix}
\]

To see the entire result, press \(\downarrow\) and then use \(\leftarrow\) and \(\rightarrow\) to move the cursor.
eigVl(

\text{eigVl}(\text{squareMatrix}) \Rightarrow \text{list}

Returns a list of the eigenvalues of a real or complex \text{squareMatrix}. \text{squareMatrix} is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The \text{squareMatrix} is then reduced to upper Hessenberg form and the eigenvalues are computed from the upper Hessenberg matrix.

\begin{align*}
\begin{bmatrix}
-1 & 2 & 5 \\
3 & -6 & 9 \\
2 & -5 & 7
\end{bmatrix} & \Rightarrow m1 \\
\begin{bmatrix}
-1 & 2 & 5 \\
3 & -6 & 9 \\
2 & -5 & 7
\end{bmatrix}
\end{align*}

eigVl(m1)
\{ -4.40941, 2.20471 + 0.763006\cdot i, 2.20471 - 0.\}

To see the entire result, press $\downarrow$ and then use $\leftarrow$ and $\rightarrow$ to move the cursor.

Else

See If, page 57.

ElseIf

If \( \text{BooleanExpr1} \) Then
Block1
ElseIf \( \text{BooleanExpr2} \) Then
Block2
ElseIf \( \text{BooleanExprN} \) Then
BlockN
EndIf

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

EndFor

See For, page 49.

EndFunc

See Func, page 52.

EndIf

See If, page 57.

EndLoop

See Loop, page 73.

EndPrgm

See Prgm, page 93.

EndTry

See Try, page 130.
euler()  

**Catalog > euler()**  
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  
\[ \frac{d \text{depVar}}{d \text{Var}} = \text{Expr}(\text{Var}, \text{depVar}) \]  

with \(\text{depVar}(\text{Var0})=\text{depVar0}\) on the interval \([\text{Var0}, \text{VarMax}]\). Returns a matrix whose first row defines the \text{Var} output values and whose second row defines the value of the first solution component at the corresponding \text{Var} values, and so on.  

**Expr** is the right-hand side that defines the ordinary differential equation (ODE).  
**SystemOfExpr** is the system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in **ListOfDepVars**).  
**ListOfExpr** is a list of right-hand sides that define the system of ODEs (corresponds to the order of dependent variables in **ListOfDepVars**).  
**Var** is the independent variable.  
**ListOfDepVars** is a list of dependent variables.  
\{\text{Var0}, \text{VarMax}\} is a two-element list that tells the function to integrate from \text{Var0} to \text{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 \(i=0,1,2,\ldots\) such that \(\text{Var0}+i\cdot\text{VarStep}\) is in \([\text{Var0}, \text{VarMax}]\) (there may not be a solution value at \text{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:  
\[ y' = 0.001 \cdot y \cdot (100-y) \]  
and \(y(0)=10\)  

euler\{0.001 \cdot y \cdot (100-y), y(0)=10\} \Rightarrow  
\[ \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 10 & 10.9 & 11.8712 & 12.9174 & 14.0427 \end{bmatrix} \]  

To see the entire result, press \(\trianglelefteq\) and then use \(\downarrow\) and \(\uparrow\) to move the cursor.  

Compare above result with CAS exact solution obtained using deSolve() and seqGen():  
deSolve\{y' = -0.001 \cdot y \cdot (100-y) \text{ and } y(0) = 10\} \Rightarrow  
\[ \begin{align*}  
y &= \frac{100 \cdot (1.10517)^t}{(1.10517)^t + 9} \end{align*} \]  

seqGen\{100 \cdot (1.10517)^t, \text{, }\{0,100\} \Rightarrow  
\[ \begin{bmatrix} 10, 10.9367, 11.9494, 13.0423, 14.2189 \end{bmatrix} \]  

System of equations:  
\[ \begin{align*}  
y' &= -y \cdot 0.1 \cdot y \cdot y2 \\
y2' &= 3 \cdot y2 - y \cdot y2 \\
\text{with } y(0) &= 2 \text{ and } y2(0) = 5 \end{align*} \]  

euler\{y', y2' = 3 \cdot y2 - y \cdot y2, y(0) = 0.5, y2(0) = 2.5\} \Rightarrow  
\[ \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 27 & 243 \end{bmatrix} \]  
\[ \begin{bmatrix} 5 & 10 & 30 & 90 & 270 & -2070 \end{bmatrix} \]  

exact()  

**Catalog > exact()**  
exact(Expr \[, \text{Tolerance}\]) \Rightarrow expression  
exact(List \[, \text{Tolerance}\]) \Rightarrow list  
exact(Matrix \[, \text{Tolerance}\]) \Rightarrow matrix  

Uses Exact mode arithmetic to return, when possible, the rational-number equivalent of the argument.  
**Tolerance** specifies the tolerance for the conversion; the default is 0 (zero).  

| exact(0.25) | 1 \[4] |
| exact(0.333333) | 333333 \[1000000] |
| exact(0.333333, 0.001) | 1 \[3] |
| exact(3.5 \cdot x + y) | 7 \cdot x \[2] + y |
| exact(\{0.2, 0.33, 4.125\}) | \begin{bmatrix} 1 \[5] & 33 \[100] & 33 \[8] \end{bmatrix} |
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 [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Function listing:

\[ \text{Define } g() = \text{Func.} \]
\[ \text{Local temp1} \]
\[ 0 \rightarrow \text{temp} \]
\[ \text{For } i, 1, 100, 1 \]
\[ \text{temp} + i \rightarrow \text{temp} \]
\[ \text{If } \text{temp} > 20 \text{ Then} \]
\[ \text{Exit} \]
\[ \text{EndIf} \]
\[ \text{EndFor} \]
\[ \text{EndFunc} \]

\[ g() \]

\[ \text{exp} \]

\[ \text{Expr} \exp \]

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

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

\[ \text{exp}() \]

\[ \text{exp(Expr1)} \Rightarrow \text{expression} \]

Returns \( e \) raised to the \( \text{Expr1} \) power.

Note: See also e exponent template, page 2.

You can enter a complex number in \( r \text{e}^\theta \) polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

\[ \text{exp(List1)} \Rightarrow \text{list} \]

Returns \( e \) raised to the power of each element in \( \text{List1} \).

\[ \text{exp(squareMatrix1)} \Rightarrow \text{squareMatrix} \]

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

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating point numbers.
exp\text{list}(Expr,Var) \Rightarrow list

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

\textbf{Note:} \text{exp\text{list}()} is not necessary with the \text{zeros()} and \text{cZeros()} functions because they return a list of solution values directly.

You can insert this function from the keyboard by typing \text{exp\text{list}(...)}. 

\text{expand()} \Rightarrow \text{expression, list, matrix}

\text{expand(Expr \{,Var\}) \Rightarrow expression, list, matrix}

\text{expand(List \{,Var\}) \Rightarrow list, matrix}

\text{expand(Matrix \{,Var\}) \Rightarrow matrix}

\text{expand(Expr \{,Var\}) returns} \text{Expr} \text{expanded with respect to all its variables. The expansion is polynomial expansion for polynomials and partial fraction expansion for rational expressions.}

The goal of \text{expand()} is to transform \text{Expr} into a sum and/or difference of simple terms. In contrast, the goal of \text{factor()} is to transform \text{Expr} into a product and/or quotient of simple factors.

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

\begin{align*}
\text{expand}\left(\frac{x^2+x+y^2+y}{x^2+y^2}\right) &=\frac{1}{x+1} + \frac{1}{y-1} \\
\text{expand}\left(\frac{x^2+y^2+y}{x^2+y^2}\right) &=\frac{1}{x-1} + \frac{1}{y-1} \\
\text{expand}\left(\frac{x^2+x^2}{x^2}\right) &=\frac{2x}{x^2} + \frac{1}{x+1} \\
\text{expand}\left(\frac{x^2}{x^2}\right) &=\frac{1}{x-\sqrt{2}} + \frac{1}{x+\sqrt{2}} + \frac{1}{x+1}
\end{align*}

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

Hint: For rational expressions, \text{propFrac()} is a faster but less extreme alternative to \text{expand()}. 

\textbf{Note:} See also \text{comDenom()} for an expanded numerator over an expanded denominator.
**expand()**

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

\[ \text{expand}(\text{Ans}, \text{[Var]} \text{)} \]

\[ \text{sign}(\text{x} \cdot \text{y}) + e^{2 \cdot x + y} + \ln(2 \cdot x + y) \]

\[ \text{expand}(\text{Ans}) \]

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

---

**expr()**

\[ \text{expr} \text{(String)} \rightarrow \text{expression} \]

Returns the character string contained in \( \text{String} \) as an expression and immediately executes it.

---

**ExpReg**

**ExpReg** \( X, Y \), \{Freq\} \{, Category, Include\}

Computes the exponential regression \( y = a \cdot (b)^x \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \). A summary of results is stored in the \text{stat.results} variable. (See page 120.)

All the lists must have equal dimension except for \text{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) 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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a \cdot (b)^x )</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient for transformed data (x, ln(y))</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals associated with the exponential model</td>
</tr>
<tr>
<td>stat.ResidTrans</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

**factor()**

\[
\text{factor}(\text{Expr1}, \text{Var}) \Rightarrow \text{expression}
\]

\[
\text{factor}(\text{List1}, \text{Var}) \Rightarrow \text{list}
\]

\[
\text{factor}(\text{Matrix1}, \text{Var}) \Rightarrow \text{matrix}
\]

\[
\text{factor}(\text{Expr1}) \text{ returns Expr1 factored with respect to all of its variables over a common denominator.}
\]

\[
\text{factor}(\text{Expr1}, \text{Var}) \text{ returns Expr1 factored with respect to variable Var.}
\]

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

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

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

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

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

\[
\text{factor}(a^3 \cdot x^2 - a \cdot x^2 - a^3 + a)
\]

\[
\text{factor}(x^2 + 1)
\]

\[
\text{factor}(x^2 - 4)
\]

\[
\text{factor}(x^2 - 3)
\]

\[
\text{factor}(x^2 - a)
\]

\[
\text{factor}(a^3 \cdot x^2 - a \cdot x^2 - a^3 + a, x)
\]

\[
\text{factor}(x^2 - 3, x)
\]

\[
\text{factor}(x^2 - a, x)
\]

\[
\text{factor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x^3 - 3)
\]

\[
\text{factor}(x^5 + 4 \cdot x^4 + 5 \cdot x^3 - 6 \cdot x^3 - 3, x)
\]

\[
(x - 0.964673) \cdot (x + 0.611649) \cdot (x + 2.12543) \cdot (x^4)
\]

46  **TI-Nspire™ CAS Reference Guide**
factor() 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.

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

For P(X ≤ upBound), set lowBound = 0.

Fill Replaces each element in variable matrixVar with Expr.

Fill Replaces each element in variable listVar with Expr.

---

Cdf() 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.

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

For P(X ≤ upBound), set lowBound = 0.

Fill Replaces each element in variable matrixVar with Expr.

Fill Replaces each element in variable listVar with Expr.

---

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

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

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

For P(X ≤ upBound), set lowBound = 0.

Fill

Fill Expr, matrixVar ⇒ matrix
Replaces each element in variable matrixVar with Expr.
matrixVar must already exist.

Fill 1,2,\{1,2,3,4,5\} → matrix

Fill 1.01,amatrix ⇒ list
Replaces each element in variable listVar with Expr.
listVar must already exist.

Fill 1.01,alist ⇒ list

---

TI-Nspire™ CAS Reference Guide 47
FiveNumSummary

FiveNumSummary \( X \), \([Freq], \) \([\)Category\), \([Include]\)\]

Provides an abbreviated version of the 1-variable statistics on list \( X \).
A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

\( X \) 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.

\( \text{Category} \) is a list of numeric category codes for the corresponding \( X \)
data.

\( \text{Include} \) 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 \( \text{Category} \)
results in a void for the corresponding element of all those lists. For
more information on empty elements, see page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.MinX} )</td>
<td>Minimum of ( x ) values.</td>
</tr>
<tr>
<td>( \text{stat.Q}_1X )</td>
<td>1st Quartile of ( x ).</td>
</tr>
<tr>
<td>( \text{stat.MedianX} )</td>
<td>Median of ( x ).</td>
</tr>
<tr>
<td>( \text{stat.Q}_3X )</td>
<td>3rd Quartile of ( x ).</td>
</tr>
<tr>
<td>( \text{stat.MaxX} )</td>
<td>Maximum of ( x ) values.</td>
</tr>
</tbody>
</table>

floor()

floor(Expr1) \( \Rightarrow \) integer

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

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().

fMax()

fMax(Expr, Var) \( \Rightarrow \) Boolean expression

fMax(Expr, Var, lowBound)

fMax(Expr, Var, lowBound, upBound)

fMax(Expr, Var) \mid lowBound \leq Var \leq supBound

Returns a Boolean expression specifying candidate values of \( Var \) that
maximize \( Expr \) or locate its least upper bound.
fMax()

You can use the constraint ("|") operator to restrict the solution interval and/or specify other constraints.

For the Approximate setting of the Auto or Approximate mode, fMax() iteratively searches for one approximate local maximum. This is often faster, particularly if you use the "|" operator to constrain the search to a relatively small interval that contains exactly one local maximum.

Note: See also fMin() and max().

fMin()

fMin(Expr, Var) → Boolean expression
fMin(Expr, Var, lowBound)
fMin(Expr, Var, lowBound, upBound)
fMin(Expr, Var) | lowBound ≤ Var ≤ upBound

Returns a Boolean expression specifying candidate values of Var that minimize Expr or locate its greatest lower bound.

You can use the constraint ("|") operator to restrict the solution interval and/or specify other constraints.

For the Approximate setting of the Auto or Approximate mode, fMin() iteratively searches for one approximate local minimum. This is often faster, particularly if you use the "|" operator to constrain the search to a relatively small interval that contains exactly one local minimum.

Note: See also fMax() and min().

For

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 [to instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.
format()  

format(Expr[, formatString]) \rightarrow string  

Returns Expr as a character string based on the format template. Expr must simplify to a number.

formatString is a string and must be in the form: "F[n]", "S[n]", "E[n]", "G[n][c]", where [ ] indicate optional portions.

F[n]: Fixed format. n is the number of digits to display after the decimal point.

S[n]: Scientific format. n is the number of digits to display after the decimal point.

E[n]: Engineering format. n is the number of digits after the first significant digit. The exponent is adjusted to a multiple of three, and the decimal point is moved to the right by zero, one, or two digits.

G[n][c]: Same as fixed format but also separates digits to the left of the radix into groups of three. c specifies the group separator character and defaults to a comma. If c is a period, the radix will be shown as a comma.

[Rc]: Any of the above specifiers may be suffixed with the Rc radix flag, where c is a single character that specifies what to substitute for the radix point.

fPart()  

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.

FPdf()  

FPdf(XVal, dfNumer, dfDenom) \rightarrow number if XVal is a number, list if XVal is a list  

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

freqTable\{list()  

freqTable\{List1,freqIntegerList\} \rightarrow list  

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

List1 can be any valid list.

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

Note: You can insert this function from the computer keyboard by typing freqTable\{list( ... )\}.

Empty (void) elements are ignored. For more information on empty elements, see page 162.
frequency() Returns a list containing counts of the elements in \( \text{List1} \). The counts are based on ranges (bins) that you define in \( \text{binsList} \).

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

Each element of the result corresponds to the number of elements from \( \text{List1} \) that are in the range of that bin. Expressed in terms of the \( \text{countIf()} \) function, the result is \( \{\text{countIf(list, } ? \leq b(1))\), \text{countIf(list, } b(1) < ? \leq b(2))\), \ldots, \text{countIf(list, } b(n-1) < ? \leq b(n))\), \text{countIf(list, } b(n) > ?\}\).

Elements of \( \text{List1} \) that cannot be "placed in a bin" are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 162.

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

\textbf{Note}: See also \( \text{countIf()} \), page 26.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Output variable} & \textbf{Description} \\
\hline
\text{stat}\_F & Calculated \( F \) statistic for the data sequence \\
\hline
\text{stat.PVal} & Smallest level of significance at which the null hypothesis can be rejected \\
\hline
\text{stat.dfNumer} & Numerator degrees of freedom = \( n1 - 1 \) \\
\hline
\text{stat.dfDenom} & Denominator degrees of freedom = \( n2 - 1 \) \\
\hline
\text{stat.sx1, stat.sx2} & Sample standard deviations of the data sequences in \( \text{List 1 and List 2} \) \\
\hline
\text{stat.x1\_bar, stat.x2\_bar} & Sample means of the data sequences in \( \text{List 1 and List 2} \) \\
\hline
\text{stat.n1, stat.n2} & Size of the samples \\
\hline
\end{tabular}
\end{table}

FTest\_2Samp

\begin{table}[h]
\centering
\begin{tabular}{|c|}
\hline
\text{FTest\_2Samp} \( \{\text{List1, List2}, Freq1, Freq2, Hypoth\} \) \\
\hline
\text{FTest\_2Samp} \( \{\text{List1, List2}, Freq1, Freq2, Hypoth\} \) \\
\hline
\end{tabular}
\end{table}

\text{FTest\_2Samp} \( \{\text{List1, List2}, Freq1, Freq2, Hypoth\} \) \( \text{(Data list input)} \)

\begin{table}[h]
\centering
\begin{tabular}{|c|}
\hline
\text{FTest\_2Samp} \( \{\text{sx1, n1, sx2, n2}, Hypoth\} \) \\
\hline
\text{FTest\_2Samp} \( \{\text{sx1, n1, sx2, n2}, Hypoth\} \) \( \text{(Summary stats input)} \)
\hline
\end{tabular}
\end{table}

Performs a two-sample \( F \) test. A summary of results is stored in the \text{stat.results} variable. (See page 120.)

For \( H_2: \sigma1 > \sigma2 \), set \( \text{Hypoth} = 0 \)
For \( H_2: \sigma1 \neq \sigma2 \) (default), set \( \text{Hypoth} = 0 \)
For \( H_2: \sigma1 < \sigma2 \), set \( \text{Hypoth} = 0 \)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.
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 \(\text{\texttt{\textbackslash}}\) instead of \(\text{\texttt{\textbackslash\texttt{\textbackslash}}}\) at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Define a piecewise function:

\[
g(x) = \begin{cases} 
3 \cdot \cos(x) & \text{if } x < 0 \\
3 - x & \text{otherwise}
\end{cases}
\]

Result of graphing \(g(x)\):

\[f(x) = g(x)\]

G

gcd()

gcd(Number1, 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.

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

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

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

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

geomCdf()

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

geomCdf(p,upBound) for P(1<X\leq upBound) \Rightarrow \text{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 \(P(X \leq \text{upBound})\), set \(\text{lowBound} = 1\).
**geomPdf()**

**geomPdf(p,XVal)** → number if XVal is a number, list if XVal is a list

Computes a probability at XVal, the number of the trial on which the first success occurs, for the discrete geometric distribution with the specified probability of success p.

---

**getDenom()**

**getDenom(Expr1)** → expression

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{x+2}{y-3} )</td>
<td>7</td>
</tr>
<tr>
<td>( \frac{2}{7} )</td>
<td>7</td>
</tr>
<tr>
<td>( \frac{1+y^2+y}{x+y^2} )</td>
<td>7</td>
</tr>
</tbody>
</table>

---

**getLangInfo()**

**getLangInfo()** → string

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

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

---

**getLockInfo()**

**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 70, and **unLock**, page 135.

---

```
ti:nspire™ cas reference guide 53
```

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a := 65</td>
<td>65</td>
</tr>
<tr>
<td>Lock a</td>
<td>Done</td>
</tr>
<tr>
<td>getLockInfo(a)</td>
<td>1</td>
</tr>
<tr>
<td>a := 75</td>
<td>&quot;Error: Variable is locked.&quot;</td>
</tr>
<tr>
<td>DelVar a</td>
<td>&quot;Error: Variable is locked.&quot;</td>
</tr>
<tr>
<td>Unlock a</td>
<td>Done</td>
</tr>
<tr>
<td>a := 75</td>
<td>75</td>
</tr>
<tr>
<td>DelVar a</td>
<td>Done</td>
</tr>
</tbody>
</table>
```

---

**TI-Nspire™ CAS Reference Guide** 53
**getMode()**

- **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 110.

### Mode Name and Integer Setting Integers

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td>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</td>
</tr>
<tr>
<td>Angle</td>
<td>2</td>
<td>1=Radian, 2=Degree, 3=Gradian</td>
</tr>
<tr>
<td>Exponential Format</td>
<td>3</td>
<td>1=Normal, 2=Scientific, 3=Engineering</td>
</tr>
<tr>
<td>Real or Complex</td>
<td>4</td>
<td>1=Real, 2=Rectangular, 3=Polar</td>
</tr>
<tr>
<td>Auto or Approx.</td>
<td>5</td>
<td>1=Auto, 2=Approximate, 3=Exact</td>
</tr>
<tr>
<td>Vector Format</td>
<td>6</td>
<td>1=Rectangular, 2=Cylindrical, 3=Spherical</td>
</tr>
<tr>
<td>Base</td>
<td>7</td>
<td>1=Decimal, 2=Hex, 3=Binary</td>
</tr>
<tr>
<td>Unit system</td>
<td>8</td>
<td>1=SI, 2=Eng/US</td>
</tr>
</tbody>
</table>

**getNum()**

- **getNum(Expr1) ⇒ expression**

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

- `getNum\left(\frac{x+2}{y-3}\right)`
  
  \[ x+2 \]

- `getNum\left(\frac{2}{7}\right)`
  
  \[ 2 \]

- `getNum\left(\frac{1}{x} + \frac{1}{y}\right)`
  
  \[ x+y \]
**getType()**

getType(var) ⇒ string

Returns a string that indicates the data type of variable var. If var has not been defined, returns the string "NONE".

**getVarInfo()**

getVarInfo() ⇒ matrix or string

getVarInfo(LibNameString) ⇒ matrix or string

getVarInfo() returns a matrix of information (variable name, type, library accessibility, and locked/unlocked state) for all variables and library objects defined in the current problem. If no variables are defined, `getVarInfo()` returns the string "NONE".

getVarInfo(LibNameString) returns a matrix of information for all library objects defined in library LibNameString. LibNameString must be a string (text enclosed in quotation marks) or a string variable. If the library LibNameString does not exist, an error occurs.

Note the example, in which the result of `getVarInfo()` is assigned to variable vs. Attempting to display row 2 or row 3 of vs returns an "Invalid list or matrix" error because at least one of elements in those rows (variable b, for example) reevaluates to a matrix.

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

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

**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`.

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

### Grad

**Expr1 ▶ Grad → expression**

Converts `Expr1` to gradian angle measure.

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

In Degree angle mode:

\[(1.5) \rightarrowGrad \Rightarrow (1.66667)^g\]

In Radian angle mode:

\[(1.5) \rightarrowGrad \Rightarrow (95.493)^g\]

### identity()

**identity(Integer) → matrix**

Returns 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
```
If

\[ \text{If BooleanExpr Then} \]
\[ \text{Block} \]
\[ \text{EndIf} \]

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

\[ \text{If BooleanExpr Then} \]
\[ \text{Block1} \]
\[ \text{Else} \]
\[ \text{Block2} \]
\[ \text{EndIf} \]

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.

\[ \text{If BooleanExpr1 Then} \]
\[ \text{Block1} \]
\[ \text{ElseIf BooleanExpr2 Then} \]
\[ \text{Block2} \]
\[ \text{ElseIf BooleanExprN Then} \]
\[ \text{BlockN} \]
\[ \text{EndIf} \]

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

Define \( g(x) = \text{Func} \)
\[ \text{If } x < 0 \text{ Then} \]
\[ \text{Return } x^2 \]
\[ \text{EndIf} \]
\[ \text{EndFunc} \]

\( g(-2) \)

4

Define \( g(x) = \text{Func} \)
\[ \text{If } x < 0 \text{ Then} \]
\[ \text{Return } -x \]
\[ \text{Else} \]
\[ \text{Return } x \]
\[ \text{EndIf} \]
\[ \text{EndFunc} \]

\( g(12) \)

12

\( g(-12) \)

12

Define \( g(x) = \text{Func} \)
\[ \text{If } x < 5 \text{ Then} \]
\[ \text{Return } 5 \]
\[ \text{ElseIf } x > 5 \text{ and } x < 0 \text{ Then} \]
\[ \text{Return } -x \]
\[ \text{ElseIf } x \geq 0 \text{ and } x \neq 10 \text{ Then} \]
\[ \text{Return } x \]
\[ \text{ElseIf } x = 10 \text{ Then} \]
\[ \text{Return } 3 \]
\[ \text{EndIf} \]
\[ \text{EndFunc} \]

\( g(-4) \)

4

\( g(10) \)

3
ifFn() \[\text{ifFn}(\text{BooleanExpr}, \text{Value}_\text{If_true}, \text{Value}_\text{If_false}, \text{Value}_\text{If_unknown}) \Rightarrow \text{expression, list, or matrix}\]

Evaluates the boolean expression \(\text{BooleanExpr}\) (or each element from \(\text{BooleanExpr}\)) and produces a result based on the following rules:

- \(\text{BooleanExpr}\) can test a single value, a list, or a matrix.
- If an element of \(\text{BooleanExpr}\) evaluates to true, returns the corresponding element from \(\text{Value}_\text{If_true}\).
- If an element of \(\text{BooleanExpr}\) evaluates to false, returns the corresponding element from \(\text{Value}_\text{If_false}\). If you omit \(\text{Value}_\text{If_false}\), returns undf.
- If an element of \(\text{BooleanExpr}\) is neither true nor false, returns the corresponding element \(\text{Value}_\text{If_unknown}\). If you omit \(\text{Value}_\text{If_unknown}\), returns undf.
- If the second, third, or fourth argument of the \text{ifFn()} function is a single expression, the Boolean test is applied to every position in \(\text{BooleanExpr}\).

Note: If the simplified \(\text{BooleanExpr}\) 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).

<table>
<thead>
<tr>
<th>ifFn()</th>
<th>[\text{ifFn}\left({1,2,3} &lt; 2.5, {5,6,7}, {8,9,10}\right) \Rightarrow {5,6,10}]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{Test value of } 1) is less than 2.5, so its corresponding (\text{Value}_\text{If_true}) element of (5) is copied to the result list.</td>
<td></td>
</tr>
<tr>
<td>(\text{Test value of } 2) is less than 2.5, so its corresponding (\text{Value}_\text{If_true}) element of (6) is copied to the result list.</td>
<td></td>
</tr>
<tr>
<td>(\text{Test value of } 3) is not less than 2.5, so its corresponding (\text{Value}_\text{If_false}) element of (10) is copied to the result list.</td>
<td></td>
</tr>
</tbody>
</table>

imag() \[\text{imag}(\text{Expr1}) \Rightarrow \text{expression}\]

Returns the imaginary part of the argument.

Note: All undefined variables are treated as real variables. See also \text{real()}, page 99

<table>
<thead>
<tr>
<th>imag()</th>
<th>[\text{imag}{1+2\cdot i} \Rightarrow 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{imag}{z} \Rightarrow 0)</td>
<td></td>
</tr>
<tr>
<td>(\text{imag}{x+i\cdot y} \Rightarrow y)</td>
<td></td>
</tr>
</tbody>
</table>

imag() \[\text{imag}(\text{List1}) \Rightarrow \text{list}\]

Returns a list of the imaginary parts of the elements.

<table>
<thead>
<tr>
<th>imag()</th>
<th>[\text{imag}{-3,4-i, i} \Rightarrow {0, 1, 1}]</th>
</tr>
</thead>
</table>

imag() \[\text{imag}(\text{Matrix1}) \Rightarrow \text{matrix}\]

Returns a matrix of the imaginary parts of the elements.

<table>
<thead>
<tr>
<th>imag()</th>
<th>[\begin{bmatrix} a &amp; b \ i\cdot c &amp; i\cdot d \end{bmatrix} \Rightarrow \begin{bmatrix} 0 &amp; 0 \ c &amp; d \end{bmatrix}]</th>
</tr>
</thead>
</table>

impDif() \[\text{impDif}(\text{Equation}, \text{Var}, \text{dependVar}, \text{Ord}) \Rightarrow \text{expression}\]

where the order \(\text{Ord}\) defaults to 1.

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

<table>
<thead>
<tr>
<th>impDif()</th>
<th>[\text{impDif}\left(x^2+y^2=100, x, y\right) \Rightarrow \begin{bmatrix} -x \ y \end{bmatrix}]</th>
</tr>
</thead>
</table>

Indirection \[\text{Indirection} \Rightarrow \text{See } \#(), \text{page } 155.\]
**inString()**

**inString**\( (src\text{String}, \text{subString}, \text{Start}) \Rightarrow \text{integer} \)

Returns the character position in string \( src\text{String} \) at which the first occurrence of string \( \text{subString} \) begins.

\( Start, \) if included, specifies the character position within \( src\text{String} \) where the search begins. Default = 1 (the first character of \( src\text{String} \)).

If \( src\text{String} \) does not contain \( \text{subString} \) or \( \text{Start} \) is > the length of \( src\text{String} \), returns zero.

**inString**\( ("Hello there","the") \) \( \Rightarrow \) 7

**inString**\( ("ABCEFG","D") \) \( \Rightarrow \) 0

**int()**

**int**\( (Expr) \Rightarrow \text{integer} \)

**int**\( (List1) \Rightarrow \text{list} \)

**int**\( (Matrix1) \Rightarrow \text{matrix} \)

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.

<table>
<thead>
<tr>
<th>expression</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>int((-2.5))</td>
<td>-3.</td>
</tr>
<tr>
<td>int([-1.234\ 0.37])</td>
<td>([-2.\ 0\ 0.])</td>
</tr>
</tbody>
</table>

**intDiv()**

**intDiv**\( (Number1, Number2) \Rightarrow \text{integer} \)

**intDiv**\( (List1, List2) \Rightarrow \text{list} \)

**intDiv**\( (Matrix1, Matrix2) \Rightarrow \text{matrix} \)

Returns the signed integer part of \( (Number1 \div Number2) \).

For lists and matrices, returns the signed integer part of \( (\text{argument 1} \div \text{argument 2}) \) for each element pair.

<table>
<thead>
<tr>
<th>expression</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>intDiv((-7,2))</td>
<td>-3</td>
</tr>
<tr>
<td>intDiv((4,5))</td>
<td>0</td>
</tr>
<tr>
<td>intDiv({12,14,16},{5,4,3})</td>
<td>({2,3,5})</td>
</tr>
</tbody>
</table>

**integral**

See \( \int \), page 151.
interpolate()

**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 undefined.

\(xList, yList,\) and \(yPrimeList\) must be of equal dimension \(\geq 2\) and contain expressions that simplify to numbers.

\(xValue\) can be an undefined variable, a number, or a list of numbers.

**Differential equation:**

\[y' = -3y + 6t + 5\] and \(y(0) = 5\)

\[
\begin{bmatrix}
 0 & 1 & 2 & 3 & 4 \\
 5 & 3.19499 & 5.00394 & 6.99957 & 9.00593 & 10
\end{bmatrix}
\]

To see the entire result, press \(\uparrow\) and then use \(\leftarrow\) and \(\rightarrow\) to move the cursor.

Use the interpolate() function to calculate the function values for the \(xValueList:\)

\[
\begin{align*}
xValueList &= \text{seq}(i, i, 0, 10, 0.5) \\
&= \{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5, 10\}
\end{align*}
\]

\[
\begin{align*}
xList &= \text{mat} \triangleright \text{list}(rk\{1\}) \\
&= \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}
\end{align*}
\]

\[
\begin{align*}
yList &= \text{mat} \triangleright \text{list}(rk\{2\}) \\
&= \{5, 3.19499, 5.00394, 6.99957, 9.00593, 10.997\}
\end{align*}
\]

\[
\text{yPrimeList} = -3y + 6t + 5\] and \(t = xList\)

\[
\begin{align*}
&= \{-10.1, 41503.1, 98819.2, 200129.1, 982221.2, 2000\}
\end{align*}
\]

\[
\text{interpolate}(xValueList, xList, yList, yPrimeList) \\
= \{5, 2.67062, 3.19499, 4.02782, 5.00394, 6.0001\}
\]
### iPart()

**iPart(Number)** ⇒ integer

**iPart(List)** ⇒ list

**iPart(Matrix)** ⇒ 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.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPart(1.234)</td>
<td>-1</td>
</tr>
<tr>
<td>iPart(\left{ \frac{3}{2},-2.3,7.003 \right} )</td>
<td>(\left{ 1, _2,.7. \right} )</td>
</tr>
</tbody>
</table>

### irr()

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

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

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

**Note:** See also **mirr()**, page 77.

<table>
<thead>
<tr>
<th>List 1</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\left{ 6000,8000,2000,_3000 \right} )</td>
<td>(\left{ 6000,8000,2000,_3000 \right} )</td>
</tr>
<tr>
<td>(\left{ 2,2,2,1 \right} )</td>
<td>(\left{ 2,2,2,1 \right} )</td>
</tr>
<tr>
<td>irr(\left( 5000,\left{ list1,_list2 \right} \right) )</td>
<td>-4.64484</td>
</tr>
</tbody>
</table>

### isPrime()

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

Returns true or false to indicate if Number is a whole number ≥ 2 that is evenly divisible only by itself and 1.

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

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

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

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPrime(5)</td>
<td>true</td>
</tr>
<tr>
<td>isPrime(6)</td>
<td>false</td>
</tr>
</tbody>
</table>

Function to find the next prime after a specified number:

```
Define nextprim(n)=Func
  n+1 \rightarrow n
  If isPrime(n)
    Return n
  EndLoop
EndFunc
```

```
nextprim(7) \rightarrow 11
```

### isVoid()

**isVoid(Var)** ⇒ Boolean constant expression

**isVoid(Expr)** ⇒ Boolean constant expression

**isVoid(List)** ⇒ list of Boolean constant expressions

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

For more information on void elements, see page 162.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isVoid(a)</td>
<td>true</td>
</tr>
<tr>
<td>isVoid(\left{ 1,_3 \right} )</td>
<td>(\left{ \text{false,true,false} \right} )</td>
</tr>
</tbody>
</table>
**Lbl**

**Lbl labelName**

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

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

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

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

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

55

---

**lcm()**

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.

```plaintext
lcm(Number1, Number2) → expression
lcm(List1, List2) → list
lcm(Matrix1, Matrix2) → matrix
```

<table>
<thead>
<tr>
<th>lcm(6,9)</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcm([1/3, 14, 16], [2/15, 7, 5], [2/3, 14, 80])</td>
<td></td>
</tr>
</tbody>
</table>

---

**left()**

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

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

```plaintext
left(sourceString, Num) → string
```

<table>
<thead>
<tr>
<th>left(&quot;Hello&quot;, 2)</th>
<th>&quot;He&quot;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>left({1, 3, 2, 4}, 3)</th>
<th>{1, 3, 2}</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>left(x &lt; 3)</th>
<th>x</th>
</tr>
</thead>
</table>
libShortcut()

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

Create 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 21.
To delete a variable group, see DelVar on page 35.

limit() or lim()

\[ \text{limit(Expr1, Var, Point [Direction])} \Rightarrow \text{expression} \]
\[ \text{limit(List1, Var, Point [Direction])} \Rightarrow \text{list} \]
\[ \text{limit(Matrix1, Var, Point [Direction])} \Rightarrow \text{matrix} \]

Returns the limit requested.

Note: See also Limit template, page 6.

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

Limits at positive \(\infty\) and at negative \(-\infty\) are always converted to one-sided limits from the finite side.

Depending on the circumstances, limit() returns itself or undef when it cannot determine a unique limit. This does not necessarily mean that a unique limit does not exist. undef means that the result is either an unknown number with finite or infinite magnitude, or it is the entire set of such numbers.

limit() uses methods such as L’Hospital’s rule, so there are unique limits that it cannot determine. If Expr1 contains undefined variables other than Var, you might have to constrain them to obtain a more concise result.

Limits can be very sensitive to rounding error. When possible, avoid the Approximate setting of the Auto or Approximate mode and approximate numbers when computing limits. Otherwise, limits that should be zero or have infinite magnitude probably will not, and limits that should have finite non-zero magnitude might not.
LinRegBx

LinRegBx XₜYₜ, [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 120.)

All the lists must have equal dimension except for Include.

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

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

$Category$ is a list of 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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $a + b \cdot x$</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified $X$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified $Y$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to $stat.XReg$ and $stat.YReg$</td>
</tr>
</tbody>
</table>

LinRegMx

LinRegMx XₜYₜ, [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 120.)

All the lists must have equal dimension except for Include.

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

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

$Category$ is a list of 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 162.
### Output variable | Description
--- | ---
stat.RegEqn | Regression Equation: \( y = m \cdot x + b \)
stat.m, stat.b | Regression coefficients
stat.\( r^2 \) | Coefficient of determination
stat.r | Correlation coefficient
stat.Resid | Residuals from the regression
stat.XReg | List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories
stat.YReg | List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories
stat.FreqReg | List of frequencies corresponding to stat.XReg and stat.YReg

### LinRegtIntervals
**LinRegtIntervals** \( X, Y, F[\text{[0]}], [CLev][\text{]}][\text{]}\]
For Slope. Computes a level C confidence interval for the slope.

**LinRegtIntervals** \( X, Y, F[\text{[1]}], Xval[\text{[1]}], [CLev][\text{]}][\text{]}\]
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 120.)

All the lists must have equal dimension.

\( X \) and \( Y \) are lists of independent and dependent variables.

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

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.
For Slope type only

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[stat.CLower, stat.CUpper]</td>
<td>Confidence interval for the slope</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td>stat.SESlope</td>
<td>Standard error of slope</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard error about the line</td>
</tr>
</tbody>
</table>

For Response type only

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[stat.CLower, stat.CUpper]</td>
<td>Confidence interval for the mean response</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td>stat.SE</td>
<td>Standard error of mean response</td>
</tr>
<tr>
<td>[stat.LowerPred, stat.UpperPred]</td>
<td>Prediction interval for a single observation</td>
</tr>
<tr>
<td>stat.MEPred</td>
<td>Prediction interval margin of error</td>
</tr>
<tr>
<td>stat.SEPred</td>
<td>Standard error for prediction</td>
</tr>
<tr>
<td>stat(y) = a + b \cdot XVal</td>
<td></td>
</tr>
</tbody>
</table>

**LinRegtTest**

\textbf{LinRegtTest} \(X, Y, Freq[, Hypoth]\)

Computes a linear regression on the \(X\) and \(Y\) lists and a \(r\) test on the value of slope \(\beta\) and the correlation coefficient \(\rho\) 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 \(\geq 0\).

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

For \(H_0: \beta \neq 0\) and \(\rho \neq 0\) (default), set \(Hypoth = 0\)

For \(H_0: \beta < 0\) and \(\rho < 0\), set \(Hypoth = 0\)

For \(H_0: \beta > 0\) and \(\rho > 0\), set \(Hypoth > 0\)

A summary of results is stored in the \(stat.results\) variable. (See page 120.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: $a + b \cdot x$</td>
</tr>
<tr>
<td>stat.t</td>
<td>$t$-Statistic for significance test</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard error about the line</td>
</tr>
<tr>
<td>stat.SESlope</td>
<td>Standard error of slope</td>
</tr>
<tr>
<td>stat.$r^2$</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

**linSolve()**

```
linSolve( SystemOfLinearEqns, Var1, Var2, ... ) → list
linSolve( LinearEqn1 and LinearEqn2 and ..., Var1, Var2, ... ) → list
linSolve( SystemOfLinearEqns, { Var1, Var2, ... } ) → list
linSolve( LinearEqn1 and LinearEqn2 and ..., { 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.
```

**\( \Delta \text{List}() \)**

```
\( \Delta \text{List}(List1) \rightarrow \text{list} \)

\textbf{Catalog > \( \Delta \)}

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

\textbf{Note:} You can insert this function from the keyboard by typing `deltaList(...)`. Returns a list containing the differences between consecutive elements in `List1`. Each element of `List1` is subtracted from the next element of `List1`. The resulting list is always one element shorter than the original `List1`.\)
**list→mat()**

**Catalog > list→mat**

\[ \text{list→mat}(\text{List 1, elementsPerRow}) \Rightarrow \text{matrix} \]

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(...)`.

### Examples

<table>
<thead>
<tr>
<th><strong>list→mat(1,2,3)</strong></th>
<th><strong>list→mat(1,2,3,4,5,2)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2 3 4 5 0</td>
</tr>
</tbody>
</table>

**ln()**

**Catalog > ln**

\[ \text{ln}(\text{Expr}) \Rightarrow \text{expression} \]

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

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

### Examples

\[ \log_{10}(10) \Rightarrow \text{ln} \]

\[ \text{ln}(x) \Rightarrow \text{ln}(10) \]

**ln()**

\[ \text{ln}(\text{List}) \Rightarrow \text{list} \]

Returns the natural logarithm of the argument.

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

### Example

\[ \text{ln}([-3,1,2,5]) \]

"Error: Non-real calculation"

**ln()**

\[ \text{ln}((\text{squareMatrix1})) \Rightarrow \text{squareMatrix} \]

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

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

In Radian angle mode and Rectangular complex format:

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

\[ \begin{bmatrix} 1.83145+1.73485\cdot i \\ 0.448761-0.72553\cdot i \\ -0.266891-2.08316\cdot i \end{bmatrix} \]

To see the entire result, press `▲` and then use `◄` and `►` to move the cursor.
**LnReg**  $X, Y[, Freq[, Category, Include]]$

Computes the logarithmic regression $y = a + b \cdot \ln(x)$ on lists $X$ and $Y$ with frequency $Freq$. A summary of results is stored in the $stat.results$ variable. (See page 120.)

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

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

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

$Category$ is a list of 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 162.

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

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

```
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 `tvn`.

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

See `unlock`, page 135, and `getlockinfo()`, page 53.

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

```

![Catalog](image)

```
Lock `Var`.

```

![Catalog](image)

```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
**log()**

\[ \text{log(Expr1,Expr2)} \Rightarrow \text{expression} \]
\[ \text{log(List1,Expr2)} \Rightarrow \text{list} \]

Returns the base-Expr2 logarithm of the first argument.

**Note:** See also Log template, page 2.

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

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

\[
\begin{array}{c|c}
\text{log} \left( \frac{2}{4} \right) & 0.5 \\
\text{log} \left( \frac{2}{3} \right) & 0.30103 \\
\text{log} \left( \frac{10}{3} - \log \left( \frac{5}{3} \right) \right) & \log \left( \frac{2}{3} \right)
\end{array}
\]

If complex format mode is Real:

\[
\text{log} \left( \begin{bmatrix} -3 & 1 & 2 & 5 \end{bmatrix} \right)
\]

Non-real result

If complex format mode is Rectangular:

\[
\text{log} \left( \begin{bmatrix} -3 & 1 & 2 & 5 \end{bmatrix} \right) = \log_{10} \begin{bmatrix} 3 \log_{10} (3) + 1.36438 \cdot i, 0.079181, \log_{10} (5) \end{bmatrix}
\]

In Radian angle mode and Rectangular complex format:

\[
\text{log} \left( \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{bmatrix} \right) = \begin{bmatrix} 0.795387 + 0.753438 \cdot i \\ 0.003993 - 0.64746 \cdot i \\ 0.194895 - 0.315095 \cdot i \\ 0.462485 + 0.2707 \cdot i \\ -0.115909 - 0.904706 \cdot i \end{bmatrix}
\]

To see the entire result, press \( \uparrow \) and then use \( \leftarrow \) and \( \rightarrow \) to move the cursor.

**logbase**

\[ \text{Expr} \logbase(Expr1) \Rightarrow \text{expression} \]

Causes the input Expression to be simplified to an expression using base \( \text{Expr1} \).

**Note:** You can insert this operator from the computer keyboard by typing \( \& \logbase(\ldots) \).
Logistic

Logistic \( X, Y, [\text{Freq}] [, \text{Category}, \text{Include}] \)

Computes the logistic regression \( y = \frac{c}{1 + a \cdot e^{bx}} \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \). A summary of results is stored in the \text{stat.results} variable. (See page 120.)

All the lists must have equal dimension except for \text{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) 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 162.

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

LogisticD

LogisticD \( X, Y, [\text{Iterations}], [\text{Freq}] [, \text{Category}, \text{Include}] \)

Computes the logistic regression \( y = \frac{c}{1 + a \cdot e^{bx} + d} \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \), using a specified number of \text{Iterations}. A summary of results is stored in the \text{stat.results} variable. (See page 120.)

All the lists must have equal dimension except for \text{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) 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 162.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( c/(1+a \cdot e^{b \cdot x}+d) )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XrReg</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq ), Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq ), Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XrReg and stat.YReg</td>
</tr>
</tbody>
</table>

**Loop**

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

**Catalog >**

Define \( \text{rollcount() = Func} \)

\[
\begin{align*}
\text{Local } & i \\
1 & \rightarrow i \\
\text{Loop} \\
\text{If } \text{randInt}(1,6) & = \text{randInt}(1,6) \\
\text{Goto end} \\
i+1 & \rightarrow i \\
\text{EndLoop} \\
\text{Lbl end} \\
\text{Return } i \\
\text{EndFunc}
\end{align*}
\]

Done

\( \text{rollcount()} \)

16

\( \text{rollcount()} \)

3
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 \( pMatrix \).

\( lMatrix \cdot uMatrix = pMatrix \cdot \text{matrix} \)

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

- If you use \( Ctrl+Enter \) or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If \( Tol \) is omitted or not used, the default tolerance is calculated as:
  \[ 5\cdot10^{-14} \cdot \max(\text{dim}(\text{Matrix})) \cdot \text{rowNorm}(\text{Matrix}) \]

The LU factorization algorithm uses partial pivoting with row interchanges.

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

\[
\text{LU } m1, \text{lower, upper, perm} \quad \text{Done}
\]

\[
\begin{array}{ccc}
\text{lower} & & \\
1 & 0 & 0 \\
\frac{5}{6} & 1 & 0 \\
1 & \frac{1}{2} & \frac{1}{2} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{upper} & & \\
6 & 12 & 18 \\
0 & 4 & 16 \\
0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{perm} & & \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{array}
\]

\[
\begin{bmatrix}
m & n \\
o & p
\end{bmatrix} \rightarrow m1
\]

\[
\text{LU } m1, \text{lower, upper, perm} \quad \text{Done}
\]

\[
\begin{array}{ccc}
\text{lower} & & \\
1 & 0 & 0 \\
0 & m & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{upper} & & \\
o & p \\
0 & n-m:p & o \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{perm} & & \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{array}
\]

\textbf{M}

\texttt{mat\#list()}

\texttt{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 \texttt{mat@>list(...)}. 

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix} \rightarrow m1
\]

\[
\text{mat\#list(m1)} \Rightarrow \{1, 2, 3, 4, 5, 6\} 
\]
\textbf{max()} \\
\textbf{max(Expr1}, \textit{Expr2}) \Rightarrow \textbf{expression} \\
\textbf{max(List1}, \textit{List2}) \Rightarrow \textbf{list} \\
\textbf{max(Matrix1}, \textit{Matrix2}) \Rightarrow \textbf{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. \\

\textbf{max(List)} \Rightarrow \textbf{expression} \\

Returns the maximum element in \textit{list}. \\

\textbf{max(Matrix1)} \Rightarrow \textbf{matrix} \\

Returns a row vector containing the maximum element of each column in \textit{Matrix1}. \\

Empty (void) elements are ignored. For more information on empty elements, see page 162. \\

Note: See also \textbf{fMax()} and \textbf{min>().} \\

\textbf{mean()} \\
\textbf{mean(List1}, \textit{freqList}) \Rightarrow \textbf{expression} \\

Returns the mean of the elements in \textit{List}. \\
Each \textit{freqList} element counts the number of consecutive occurrences of the corresponding element in \textit{List}. \\

\textbf{mean(Matrix1}, \textit{freqMatrix}) \Rightarrow \textbf{matrix} \\

Returns a row vector of the means of all the columns in \textit{Matrix1}. \\
Each \textit{freqMatrix} element counts the number of consecutive occurrences of the corresponding element in \textit{Matrix1}. \\

Empty (void) elements are ignored. For more information on empty elements, see page 162. \\

In Rectangular vector format: \\

\textbf{median()} \\
\textbf{median(List1}, \textit{freqList}) \Rightarrow \textbf{expression} \\

Returns the median of the elements in \textit{List}. \\
Each \textit{freqList} element counts the number of consecutive occurrences of the corresponding element in \textit{List}. \\

\textbf{mean}\left(\{0.2,0,1,-0.3,0.4\}\right) \quad 0.26 \\
mean\left(\{1,2,3\},\{3,2,1\}\right) \quad \frac{5}{3} \\

\begin{bmatrix} 0.2 & 0 \\ -1 & 3 \\ 0.4 & -0.5 \end{bmatrix} \\
\begin{bmatrix} -0.133333 & 0.833333 \\ \end{bmatrix} \\
\begin{bmatrix} 1 & 0 \\ 5 & 3 \\ \end{bmatrix} \\
\begin{bmatrix} -2 & 5 \\ 15 & 6 \\ \end{bmatrix} \\
\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 4 & 1 & 6 \\ 5 & 6 & 2 \end{bmatrix} \\
\begin{bmatrix} 47 & 11 \\ 15 & 3 \end{bmatrix} \\

\textbf{median}\left(\{0.2,0,1,-0.3,0.4\}\right) \quad 0.2
median()  

\[ \text{median}(\text{Matrix1}, \text{freqMatrix}) \Rightarrow \text{matrix} \]

Returns a row vector containing the medians of the columns in Matrix1. Each freqMatrix element counts the number of consecutive occurrences of the corresponding element in Matrix1.

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 162.

MedMed

\[ \text{MedMed} \ X, \ Y, \ Freq, \ [\ \text{Category}, \ \text{Include}] \]

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 \text{stat.results} variable. (See page 120.)

All the lists must have equal dimension except for \text{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

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

\( \text{Category} \) is a list of category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) 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 162.

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

\( \text{mid}(\text{sourceString}, \text{Start}, \text{Count}) \Rightarrow \text{string} \)

Returns \( \text{Count} \) characters from character string \text{sourceString}, beginning with character number \text{Start}.

If \text{Count} is omitted or is greater than the dimension of \text{sourceString}, returns all characters from \text{sourceString}, beginning with character number \text{Start}.

\( \text{Count} \) must be \( \geq 0 \). If \text{Count} = 0, returns an empty string.
mid()  

\[ \text{mid}(\text{sourceList}, \text{Start}, \text{Count}) \Rightarrow \text{list} \]

Returns \( \text{Count} \) elements from \( \text{sourceList} \), beginning with element number \( \text{Start} \).

If \( \text{Count} \) is omitted or is greater than the dimension of \( \text{sourceList} \), returns all elements from \( \text{sourceList} \), beginning with element number \( \text{Start} \).

\( \text{Count} \) must be \( \geq 0 \). If \( \text{Count} = 0 \), returns an empty list.

\[
\begin{align*}
\text{mid}\{9,8,7,6\},3 & \Rightarrow \{7,6\} \\
\text{mid}\{9,8,7,6\},2,2 & \Rightarrow \{8,7\} \\
\text{mid}\{9,8,7,6\},1,2 & \Rightarrow \{9,8\} \\
\text{mid}\{9,8,7,6\},1,0 & \Rightarrow \{\{\}\} \\
\text{mid}\{"A","B","C","D\},2,2 & \Rightarrow \{"B","C\}
\end{align*}
\]

min()  

\[ \text{min}(\text{Expr1}, \text{Expr2}) \Rightarrow \text{expression} \]

\[ \text{min}(\text{List1}, \text{List2}) \Rightarrow \text{list} \]

\[ \text{min}(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix} \]

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.

\[ \text{min}(\text{List}) \Rightarrow \text{expression} \]

Returns the minimum element of \( \text{List} \).

\[ \text{min}(\text{Matrix1}) \Rightarrow \text{matrix} \]

Returns a row vector containing the minimum element of each column in \( \text{Matrix1} \).

Note: See also \text{fMin()} and \text{max()}.

mirr()  

\[ \text{mirr}(\text{financeRate}, \text{reinvestRate}, \text{CF0}, \text{CFList}, \text{CFFreq}) \]

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

\( \text{financeRate} \) is the interest rate that you pay on the cash flow amounts.

\( \text{reinvestRate} \) is the interest rate at which the cash flows are reinvested.

\( \text{CF0} \) is the initial cash flow at time 0; it must be a real number.

\( \text{CFList} \) is a list of cash flow amounts after the initial cash flow \( \text{CF0} \).

\( \text{CFFreq} \) 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 \( \text{CFList} \). The default is 1; if you enter values, they must be positive integers < 10,000.

Note: See also \text{irr()}, page 61.
mod(
Expr1, Expr2) ⇒ expression
mod(List1, List2) ⇒ list
mod(Matrix1, Matrix2) ⇒ matrix

Returns the first argument modulo the second argument as defined by the identities:
mod(x,0) = x
mod(x,y) = x - y floor(x/y)

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

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

Note: See also remain(), page 100

mRow(

mRow(Expr, Matrix1, Index) ⇒ matrix

Returns a copy of Matrix1 with each element in row Index of Matrix1 multiplied by Expr.

mRowAdd(

mRowAdd(Expr, Matrix1, Index1, Index2) ⇒ matrix

Returns a copy of Matrix1 with each element in row Index2 of Matrix1 replaced with:
Expr · row Index1 + row Index2

MultReg

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 120.)

All the lists must have equal dimension.
For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: b0+b1 · x1+b2 · x2 + ...</td>
</tr>
<tr>
<td>stat.b0, stat.b1, ...</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.yList</td>
<td>y List = b0+b1 · x1+ ...</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>
### MultRegIntervals

**Description**

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 120.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

#### Output variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.RegEqn</code></td>
<td>Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ...$</td>
</tr>
<tr>
<td><code>stat.\hat{y}</code></td>
<td>A point estimate: $\hat{y} = b_0 + b_1 \cdot x_1 + ...$ for <code>XValList</code></td>
</tr>
<tr>
<td><code>stat.dError</code></td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td><code>stat.CLower, stat.CUpper</code></td>
<td>Confidence interval for a mean response</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td><code>stat.SE</code></td>
<td>Standard error of mean response</td>
</tr>
<tr>
<td><code>stat.LowerPred, stat.UpperPred</code></td>
<td>Prediction interval for a single observation</td>
</tr>
<tr>
<td><code>stat.MEPred</code></td>
<td>Prediction interval margin of error</td>
</tr>
<tr>
<td><code>stat.SEPred</code></td>
<td>Standard error for prediction</td>
</tr>
<tr>
<td><code>stat.bList</code></td>
<td>List of regression coefficients, {b_0, b_1, b_2, ...}</td>
</tr>
<tr>
<td><code>stat.Resid</code></td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

### MultRegTests

**Description**

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 120.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

#### Outputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.RegEqn</code></td>
<td>Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ...$</td>
</tr>
<tr>
<td><code>stat.F</code></td>
<td>Global $F$ test statistic</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>$P$-value associated with global $F$ statistic</td>
</tr>
<tr>
<td><code>stat.R^2</code></td>
<td>Coefficient of multiple determination</td>
</tr>
</tbody>
</table>
### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.AdjR²</td>
<td>Adjusted coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
<tr>
<td>stat.DW</td>
<td>Durbin-Watson statistic; used to determine whether first-order auto correlation is present in the model</td>
</tr>
<tr>
<td>stat.dfReg</td>
<td>Regression degrees of freedom</td>
</tr>
<tr>
<td>stat.SSReg</td>
<td>Regression sum of squares</td>
</tr>
<tr>
<td>stat.MSReg</td>
<td>Regression mean square</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td>stat.SSErr</td>
<td>Error sum of squares</td>
</tr>
<tr>
<td>stat.MSEx</td>
<td>Error mean square</td>
</tr>
<tr>
<td>stat.bList</td>
<td>{b₀,b₁,\ldots} List of coefficients</td>
</tr>
<tr>
<td>stat.tList</td>
<td>List of t statistics, one for each coefficient in the bList</td>
</tr>
<tr>
<td>stat.PList</td>
<td>List P-values for each t statistic</td>
</tr>
<tr>
<td>stat.SEList</td>
<td>List of standard errors for coefficients in bList</td>
</tr>
<tr>
<td>stat.yList</td>
<td>( \hat{y} = b₀ + b₁ \cdot x₁ + \ldots )</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.sResid</td>
<td>Standardized residuals; obtained by dividing a residual by its standard deviation</td>
</tr>
<tr>
<td>stat.CookDist</td>
<td>Cook's distance; measure of the influence of an observation based on the residual and leverage</td>
</tr>
<tr>
<td>stat.Leverage</td>
<td>Measure of how far the values of the independent variable are from their mean values</td>
</tr>
</tbody>
</table>

#### nand

Returns the negation of a logical and operation on the two arguments. Returns true, false, or a simplified form of the equation. For lists and matrices, returns comparisons element by element.

**Integer1 nand Integer2 \( \Rightarrow \) integer**

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

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

For integer Expr1 and Expr2 with Expr1 \( \geq \) Expr2 \( \geq 0 \), nCr() is the number of combinations of Expr1 things taken Expr2 at a time. (This is also known as a binomial coefficient.) Both arguments can be integers or symbolic expressions.

\[
\text{nCr(Expr, 0) } \Rightarrow 1
\]

\[
\text{nCr(Expr, negInteger) } \Rightarrow 0
\]

\[
\text{nCr(Expr, posInteger) } \Rightarrow \text{Expr} \cdot (\text{Expr}-1)\ldots (\text{Expr}-\text{posInteger}+1)/\text{posInteger}!
\]

\[
\text{nCr(Expr, nonInteger) } \Rightarrow \text{expression}!
\]

\[
\text{nCr(List1, List2) } \Rightarrow \text{list}
\]

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

\[
\text{nCr(Matrix1, Matrix2) } \Rightarrow \text{matrix}
\]

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

nDerivative()  

\[
\text{nDerivative(Expr1, Var=Value, Order) } \Rightarrow \text{value}
\]

\[
\text{nDerivative(Expr1, Var[Order]) | Var=Value } \Rightarrow \text{value}
\]

Returns the numerical derivative calculated using auto differentiation methods.

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

Order of the derivative must be 1 or 2.

newList()  

\[
\text{newList(numElements) } \Rightarrow \text{list}
\]

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

newMat()  

\[
\text{newMat(numRows, numColumns) } \Rightarrow \text{matrix}
\]

Returns a matrix of zeros with the dimension numRows by numColumns.
nfMax()

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 in the closed interval [lowBound, upBound] for the local maximum.

Note: See also fMax() and d().

nfMin()

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

Returns a candidate numerical value of variable Var where the local minimum of Expr occurs.

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

Note: See also fMin() and d().

nInt()

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 \( \int_{Lower}^{Upper} \) (Expr1, Var, Lower, 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.

Note: See also \( \int \), page 151.

nom()

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 40.
nor

BooleanExpr1 nor BooleanExpr2 returns Boolean expression
BooleanList1 nor BooleanList2 returns Boolean list
BooleanMatrix1 nor BooleanMatrix2 returns Boolean matrix

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

For lists and matrices, returns comparisons element by element.

Integer1 nor Integer2 ⇒ integer

Compares two real integers bit-by-bit using a nor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

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

\[
3 \text{ or } 4 \quad 7 \\
3 \text{ nor } 4 \quad -8 \\
\{1,2,3\} \text{ or } \{3,2,1\} \quad \{3,2,3\} \\
\{1,2,3\} \text{ nor } \{3,2,1\} \quad \{-4,-3,-4\}
\]

\[
\begin{array}{c}
\text{norm}() \\
\text{norm}() \\
\text{norm}() \\
\text{norm}() \\
\end{array}
\]

Catalog > 

\[
\text{norm}(\text{Matrix}) \Rightarrow \text{expression} \\
\text{norm}(\text{Vector}) \Rightarrow \text{expression}
\]

Returns the Frobenius norm.

\[
\text{norm}\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \sqrt{a^2+b^2+c^2+d^2} \\
\text{norm}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \sqrt{30} \\
\text{norm}\begin{pmatrix} 1 & 2 \end{pmatrix} = \sqrt{5} \\
\text{norm}\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \sqrt{5}
\]

\[
\begin{array}{c}
\text{normalLine() } \\
\text{normalLine() } \\
\text{normalLine() } \\
\text{normalLine() } \\
\end{array}
\]

Catalog > 

\[
\text{normalLine}(\text{Expr1},\text{Var},\text{Point}) \Rightarrow \text{expression} \\
\text{normalLine}(\text{Expr1},\text{Var}=\text{Point}) \Rightarrow \text{expression}
\]

Returns the normal line to the curve represented by \(\text{Expr1}\) at the point specified in \(\text{Var}=\text{Point}\).

Make sure that the independent variable is not defined. For example, if \(f1(x):=5\) and \(x:=3\), then \(\text{normalLine}(f1(x),x,2)\) returns "false."

\[
\text{normalLine}(x^2,x,1) = \frac{3-x}{2} \\
\text{normalLine}((x-3)^2-4,x,3) = x=3 \\
\text{normalLine}\left(\frac{1}{x^3},x=0\right) = 0 \\
\text{normalLine}(\sqrt{|x|},x=0) = \text{undef}
\]

\[
\begin{array}{c}
\text{normCdf()} \\
\text{normCdf()} \\
\end{array}
\]

Catalog > 

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

Computes the normal distribution probability between \(\text{lowBound}\) and \(\text{upBound}\) for the specified \(\mu\) (default=0) and \(\sigma\) (default=1).

For \(P(X \leq \text{upBound})\), set \(\text{lowBound} = -\infty\).
normPdf() \[\text{Catalog} > \]

\[\text{normPdf}(XVal; \mu, \sigma) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list}\]

Computes the probability density function for the normal distribution at a specified \(XVal\) value for the specified \(\mu\) and \(\sigma\).

not \[\text{Catalog} > \]

\[\text{not } \text{BooleanExpr} \Rightarrow \text{Boolean expression}\]

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

\[\text{not } \text{Integer1} \Rightarrow \text{integer}\]

Returns the one’s complement of a real integer. Internally, \(\text{Integer1}\) 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 format, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see \(\text{Base2, page 14.}\)

In Hex base mode:

\[\text{not } 0h7AC36 \Rightarrow 0hFFFFFFFFF853C9\]

In Bin base mode:

\[\begin{align*}
\text{not } 0b100101 & \Rightarrow \text{Base10} \quad 37 \\
\text{not } 0b100101 & \Rightarrow \text{Base10} \quad -38
\end{align*}\]

To see the entire result, press \(\text{£}\) and then use \(\text{¡}\) and \(\text{¢}\) 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() \[\text{Catalog} > \]

\[\text{nPr}(Expr1, \text{Expr2}) \Rightarrow \text{expression}\]

For integer \(\text{Expr1}\) and \(\text{Expr2}\) with \(\text{Expr1} \geq \text{Expr2} \geq 0\), \(\text{nPr()}\) is the number of permutations of \(\text{Expr1}\) things taken \(\text{Expr2}\) at a time. Both arguments can be integers or symbolic expressions.

\[\text{nPr}(\text{Expr}, 0) \Rightarrow 1\]

\[\text{nPr}(\text{Expr}, \text{negInteger}) \Rightarrow \frac{1}{((\text{Expr}+1) \cdot (\text{Expr}+2) \ldots (\text{expression}+\text{negInteger}))}\]

\[\text{nPr}(\text{Expr}, \text{posInteger}) \Rightarrow \text{Expr} \cdot (\text{Expr}+1) \ldots (\text{Expr}+\text{posInteger}+1)\]

\[\text{nPr}(\text{Expr}, \text{nonInteger}) \Rightarrow \text{Expr}! / (\text{Expr}+\text{nonInteger})!\]

\[\text{nPr}(\text{List1}, \text{List2}) \Rightarrow \text{list}\]

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

\[\text{nPr}(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix}\]

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

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.

**nSolve()**

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

- *variable*
- or -
- *variable*= real number

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

*nSolve()* is often much faster than *solve()* or *zeros()* , particularly if the "|" operator is used to constrain the search to a small interval containing exactly one simple solution.

*nSolve()* attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string "no solution found."

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

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the `stat.results` variable. (See page 120.)

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

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

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

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

An empty (void) element in any of the lists `X`, `Freq`, or `Category` results in a void for the corresponding element of all those lists. An empty element in any of the lists `X1` through `X20` results in a void for the corresponding element of all those lists. For more information on empty elements, see page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.(\bar{X})</td>
<td>Mean of x values</td>
</tr>
<tr>
<td>stat.(\Sigma x)</td>
<td>Sum of x values</td>
</tr>
<tr>
<td>stat.(\Sigma x^2)</td>
<td>Sum of (x^2) values</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of x</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Population standard deviation of x</td>
</tr>
<tr>
<td>stat.n</td>
<td>Number of data points</td>
</tr>
<tr>
<td>stat.MinX</td>
<td>Minimum of x values</td>
</tr>
<tr>
<td>stat.Q1X</td>
<td>1st Quartile of x</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of x</td>
</tr>
<tr>
<td>stat.Q3X</td>
<td>3rd Quartile of x</td>
</tr>
<tr>
<td>stat.MaxX</td>
<td>Maximum of x values</td>
</tr>
<tr>
<td>stat.SSX</td>
<td>Sum of squares of deviations from the mean of x</td>
</tr>
</tbody>
</table>
BooleanExpr1 or BooleanExpr2 returns Boolean expression
BooleanList1 or BooleanList2 returns Boolean list
BooleanMatrix1 or BooleanMatrix2 returns Boolean matrix

Returns true or false or a simplified form of the original entry.
Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See xor.

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

Integer1 or Integer2 \( \Rightarrow \) 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 \( \text{Base2}, \) page 14.

Note: See xor.

In Hex base mode:
\[ \text{0b7AC36 or 0b3D5F} \]

Important: Zero, not the letter O.

In Bin base mode:
\[ \text{0b100101 or 0b100} \]

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() \( \Rightarrow \) integer

Returns the numeric code of the first character in character string \( \text{String} \), or a list of the first characters of each list element.

In Radian angle mode:
\[
\begin{align*}
\text{P} \cdot \text{Rx}(r, \theta) & \Rightarrow \cos(\theta) \cdot r \\
\text{P} \cdot \text{Rx}(4,60^\circ) & \Rightarrow 2 \\
\text{P} \cdot \text{Rx}\left(-3,10,1.3\right), \left[\frac{\pi}{3}, \frac{\pi}{4}, 0\right] & \Rightarrow \left[\frac{-3}{2}, 5\sqrt{2}, 1.3\right]
\end{align*}
\]

\( x \geq 3 \) or \( x \geq 4 \)

\[
g(x) = \begin{cases} 
\text{Func} & \text{Done} \\
\text{If } x \leq 0 \text{ or } x \geq 5 \\
\text{Goto end} \\
\text{Return } x \cdot 3 \\
\text{Lbl end} \\
\text{EndFunc}
\end{cases}
\]

\[ g(3) \]

A function did not return a value
**P•Ry()**

**P•Ry(expression, θ)** \[\Rightarrow \text{expression}\]

**P•Ry(list, θ)** \[\Rightarrow \text{list}\]

**P•Ry(matrix, θ)** \[\Rightarrow \text{matrix}\]

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

**Note:** The θ argument is interpreted as either a degree, radian or gradian angle, according to the current angle mode. If the argument is an expression, you can use 0°, G or ꞌ to override the angle mode setting temporarily.

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

---

**PassErr**

**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 19, and Try, page 130.

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

---

**piecewise()**

**piecewise(Expr1 [, Cond1 [, Expr2 [, Cond2 [, ... ]]]])**

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

**Note:** See also Piecewise template, page 2.

---

**poissCdf()**

**poissCdf(λ, lowBound, upBound) \[\Rightarrow \text{number}\] if lowBound and upBound are numbers, list if lowBound and upBound are lists**

**poissCdf(λ, upBound) for P(0 ≤ X ≤ upBound) \[\Rightarrow \text{number}\] if upBound is a number, list if upBound is a list**

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

For P(X ≤ upBound), set lowBound=0

---

**poissPdf()**

**poissPdf(λ, XVal) \[\Rightarrow \text{number}\] if XVal is a number, list if XVal is a list**

Computes a probability for the discrete Poisson distribution with the specified mean λ.
Vector \textbf{Polar}

\textbf{Note:} You can insert this operator from the computer keyboard by typing @Polar.

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

\textbf{Note:} \textbf{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.

\textbf{Note:} See also \textbf{Rect}, page 99.

\textbf{complexValue} \textbf{Polar}

Displays complexVector in polar form.

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

\textbf{complexValue} can have any complex form. However, an \(re^{i\theta}\) entry causes an error in Degree angle mode.

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

\textbf{polyCoeffs()}\n
\textbf{polyCoeffs}(\textbf{Poly} \{\textbf{Var}\}) \Rightarrow \textbf{list}

Returns a list of the coefficients of polynomial \textbf{Poly} with respect to variable \textbf{Var}.

\textbf{Poly} must be a polynomial expression in \textbf{Var}. We recommend that you do not omit \textbf{Var} unless \textbf{Poly} is an expression in a single variable.
**polyDegree()**

**polyDegree(Poly [,Var]) ⇒ value**

Returns the degree of polynomial expression Poly with respect to variable Var. If you omit Var, the polyDegree() function selects a default from the variables contained in the polynomial Poly.

Poly must be a polynomial expression in Var. We recommend that you do not omit Var unless Poly is an expression in a single variable.

```plaintext
polyDegree(5) 0
polyDegree(ln(2)+π*x) 0

Constant polynomials
polyDegree(4*x^2-3*x+2,x) 2
polyDegree((x-1)^2*(x+2)^3) 5

polyDegree((x+y^2+z^3)^2,x) 2
polyDegree((x+y^2+z^3)^2,y) 4

polyDegree((x-1)^10000,x) 10000
```

The degree can be extracted even though the coefficients cannot. This is because the degree can be extracted without expanding the polynomial.

---

**polyEval()**

**polyEval(List1, Expr1) ⇒ expression**

**polyEval(List1, List2) ⇒ expression**

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

```plaintext
polyEval({a,b,c},x) a*x^2+b*x+c
polyEval({1,2,3,4},2) 26
polyEval({1,2,3,4},{2,-7}) {26,-262}
```

---

**polyGcd()**

**polyGcd(Expr1,Expr2) ⇒ expression**

Returns greatest common divisor of the two arguments. Expr1 and Expr2 must be polynomial expressions.

List, matrix, and Boolean arguments are not allowed.

```plaintext
polyGcd(100,30) 10
polyGcd(x^2-1,x-1) x-1
polyGcd(x^3-6*x^2+11*x-6,x^2-6*x+8) x-2
```
polyQuotient()

polyQuotient(\textit{Poly1}, \textit{Poly2} [\textit{Var}]) \Rightarrow \text{expression}

Returns the quotient of polynomial \textit{Poly1} divided by polynomial \textit{Poly2} with respect to the specified variable \textit{Var}.

\textit{Poly1} and \textit{Poly2} must be polynomial expressions in \textit{Var}. We recommend that you do not omit \textit{Var} unless \textit{Poly1} and \textit{Poly2} are expressions in the same single variable.

\begin{align*}
\text{polyQuotient}(x-1, x-3) &= 1 \\
\text{polyQuotient}(x-1, x^2-1) &= 0 \\
\text{polyQuotient}(x^2 - 1, x-1) &= x+1 \\
\text{polyQuotient}(x^3 - 6x^2 + 11x - 6, x^2 - 6x + 8) &= x \\
\text{polyQuotient}((x-y)(y-z), x+y+z, x) &= y-z \\
\text{polyQuotient}((x-y)(y-z), x+y+z, y) &= 2x-y+2z \\
\text{polyQuotient}((x-y)(y-z), x+y+z, z) &= -(x-y)
\end{align*}

polyRemainder()

polyRemainder(\textit{Poly1}, \textit{Poly2} [\textit{Var}]) \Rightarrow \text{expression}

Returns the remainder of polynomial \textit{Poly1} divided by polynomial \textit{Poly2} with respect to the specified variable \textit{Var}.

\textit{Poly1} and \textit{Poly2} must be polynomial expressions in \textit{Var}. We recommend that you do not omit \textit{Var} unless \textit{Poly1} and \textit{Poly2} are expressions in the same single variable.

\begin{align*}
\text{polyRemainder}(x-1, x-3) &= 2 \\
\text{polyRemainder}(x-1, x^2-1) &= x-1 \\
\text{polyRemainder}(x^2 - 1, x-1) &= 0 \\
\text{polyRemainder}((x-y)(y-z), x+y+z, x) &= (y-z)(2y+z) \\
\text{polyRemainder}((x-y)(y-z), x+y+z, y) &= -2x^2-5xz-2z^2 \\
\text{polyRemainder}((x-y)(y-z), x+y+z, z) &= (x-y)(x+2y)
\end{align*}

polyRoots()

polyRoots(\textit{Poly}, \textit{Var}) \Rightarrow \text{list}

polyRoots(\textit{ListOfCoeffs}) \Rightarrow \text{list}

The first syntax, \texttt{polyRoots(\textit{Poly}, \textit{Var})}, returns a list of real roots of polynomial \textit{Poly} with respect to variable \textit{Var}. If no real roots exist, returns an empty list: \texttt{\{\}}.

\textit{Poly} must be a polynomial in one variable.

The second syntax, \texttt{polyRoots(\textit{ListOfCoeffs})}, returns a list of real roots for the coefficients in \textit{ListOfCoeffs}.

\textbf{Note:} See also \texttt{cPolyRoots()}, page 26.

\begin{align*}
\text{polyRoots}(y^3+1, y) &= \{-1\} \\
\text{cPolyRoots}(y^3+1, y) &= \left\{-1, \frac{1}{2}, \frac{\sqrt{3}}{2} - \frac{\sqrt{3}}{2}, \frac{1}{2} + \frac{\sqrt{3}}{2}, \frac{\sqrt{3}}{2} + \frac{\sqrt{3}}{2}\right\} \\
\text{polyRoots}(x^2+2x+1, x) &= \{-1, -1\} \\
\text{polyRoots}(\{1, 2, 1\}) &= \{-1, -1\}
\end{align*}
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 120.)

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

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

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

$Category$ is a list of 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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$stat.RegEqn$</td>
<td>Regression equation: $a \cdot (x)^b$</td>
</tr>
<tr>
<td>$stat.a$, $stat.b$</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>$stat.r^2$</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>$stat.r$</td>
<td>Correlation coefficient for transformed data ($\ln(x)$, $\ln(y)$)</td>
</tr>
<tr>
<td>$stat.Resid$</td>
<td>Residuals associated with the power model</td>
</tr>
<tr>
<td>$stat.ResidTrans$</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>$stat.XReg$</td>
<td>List of data points in the modified $X$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories</td>
</tr>
<tr>
<td>$stat.YReg$</td>
<td>List of data points in the modified $Y$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and $Include$ Categories</td>
</tr>
<tr>
<td>$stat.FreqReg$</td>
<td>List of frequencies corresponding to $stat.XReg$ and $stat.YReg$</td>
</tr>
</tbody>
</table>
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 \[\text{enter}\] 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 \(b\neq 0\)
    \(d:=\text{mod}(a,b)\)
    \(a:=b\)
    \(b:=d\)
    Disp a," ",b
  EndWhile
  Disp "GCD=",a
EndPrgm
```

Done

```
proggcd(4560,450)
```

|  450 |
|  60  |
|  60  |
|  30  |
| GCD=30 |

Done

prodSeq()

See \(\prod\), page 152.

Product (\(\Pi\))

See \(\prod\), page 152.

```
prod(a,b)
```

product()

```
product(List, Start, End) \Rightarrow expression
```

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

| product\(\{1,2,3,4\}\) | 24 |
| product\(\{2,x,y\}\) | \(2 \times x \times y\) |
| product\(\{4,5,8,9\},2,3\) | 40 |

product()

```
product(Matrix1, Start, End) \Rightarrow matrix
```

Returns a row vector containing the products of the elements in the columns of Matrix1. Start and End are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 162.

| \[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\] | 28 | 80 | 162 |
| \[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\] | 4 | 10 | 18 |
propFrac()

\[ \text{propFrac}(\text{Expr1, Var}) \rightarrow \text{expression} \]

\[ \text{propFrac} \left( \frac{4}{3} \right) \rightarrow 1 + \frac{1}{3} \]

\[ \text{propFrac} \left( -\frac{4}{3} \right) \rightarrow -1 - \frac{1}{3} \]

\[ \text{propFrac} \left( \frac{x^2 + x + 1}{x + 1}, y + 1 \right) \rightarrow \frac{1}{x + 1} + \frac{y^2 + y + 1}{y + 1} \]

\[ \text{propFrac}(\text{Ans}) \rightarrow \frac{1}{x + 1} + \frac{1}{y + 1} \]

\[ \text{propFrac} \left( \frac{11}{7} \right) \rightarrow 1 + \frac{4}{7} \]

\[ \text{propFrac} \left( \frac{3 + \frac{1}{11} + \frac{3}{4}}{11} \right) \rightarrow 8 + \frac{37}{44} \]

\[ \text{propFrac} \left( \frac{3 + \frac{1}{11} + \frac{3}{4}}{11} \right) \rightarrow -2 + \frac{29}{44} \]

QR

\[ \text{QR Matrix, qMatrix, rMatrix[, Tol]} \]

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 \[ \text{ctrl} \text{ enter} \] 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:
  \[ 5E^{-14} \cdot \max(\text{dim}(\text{Matrix})) \cdot \text{rowNorm}(\text{Matrix}) \]

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\rightarrow m1
\]

QR m1,qm,rm

\[
\begin{bmatrix}
0.123091 & 0.904534 & 0.408248 \\
0.492366 & 0.301511 & 0.816497 \\
0.86164 & -0.301511 & 0.408248
\end{bmatrix}
\]

\[
\begin{bmatrix}
8.12404 & 9.60114 & 11.0782 \\
0 & 0.904534 & 1.80907 \\
0 & 0 & 0
\end{bmatrix}
\]

ClearAZ

Done
The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in qMatName are the orthonormal basis vectors that span the space defined by matrix.

QuadReg

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 \( \text{stat.results} \) variable. (See page 120.)

All the lists must have equal dimension except for Include.

\( X \) and \( Y \) are lists of independent and dependent variables.

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

\( Category \) is a list of 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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression equation: ( a \cdot x^2 + b \cdot x + c )</td>
</tr>
<tr>
<td>( \text{stat.a, stat.b, stat.c} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.R}^2 )</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>( \text{stat.Resid} )</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>( \text{stat.XReg} )</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, \ Category \text{List}, \text{and Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.YReg} )</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, \ Category \text{List}, \text{and Include Categories} )</td>
</tr>
<tr>
<td>( \text{stat.FreqReg} )</td>
<td>List of frequencies corresponding to ( \text{stat.XReg} ) and ( \text{stat.YReg} )</td>
</tr>
</tbody>
</table>
QuartReg

Computes the quartic polynomial regression

\[ y = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \]
on lists \( \text{X} \) and \( \text{Y} \) with frequency \( \text{Freq} \).

A summary of results is stored in the \textit{stat.results} variable. (See page 120.)

All the lists must have equal dimension except for \textit{Include}.

\( \text{X} \) and \( \text{Y} \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( \text{X} \) and \( \text{Y} \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\textit{Category} is a list of category codes for the corresponding \( \text{X} \) and \( \text{Y} \) data.

\textit{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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.RegEqn}</td>
<td>Regression equation: ( a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e )</td>
</tr>
<tr>
<td>\text{stat.a}, \text{stat.b}, \text{stat.c}, \text{stat.d}, \text{stat.e}</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>\text{stat.R}^2</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>\text{stat.Resid}</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>\text{stat.XReg}</td>
<td>List of data points in the modified \textit{X List} actually used in the regression based on restrictions of \textit{Freq}, \textit{Category List}, and \textit{Include Categories}</td>
</tr>
<tr>
<td>\text{stat.YReg}</td>
<td>List of data points in the modified \textit{Y List} actually used in the regression based on restrictions of \textit{Freq}, \textit{Category List}, and \textit{Include Categories}</td>
</tr>
<tr>
<td>\text{stat.FreqReg}</td>
<td>List of frequencies corresponding to \text{stat.XReg} and \text{stat.YReg}</td>
</tr>
</tbody>
</table>
### R

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(\theta)()</td>
<td>[R(\theta)(xExpr, yExpr) \Rightarrow expression] Returns the equivalent (\theta)-coordinate of the ((x,y)) pair arguments. <strong>Note:</strong> The result is returned as a degree, gradian or radian angle, according to the current angle mode setting. <strong>Note:</strong> You can insert this function from the computer keyboard by typing (R\theta) ((\theta)).</td>
</tr>
<tr>
<td>R(\rho)()</td>
<td>[R(\rho)(xList, yList) \Rightarrow list] [R(\rho)(xMatrix, yMatrix) \Rightarrow matrix] Returns the equivalent (r)-coordinate of the ((x,y)) pair arguments. <strong>Note:</strong> You can insert this function from the computer keyboard by typing (R\rho) ((\rho)).</td>
</tr>
<tr>
<td>(\xrightarrow{\text{Rad}})</td>
<td>[\text{Expr} \xrightarrow{\text{Rad}} \Rightarrow expression] Converts the argument to radian angle measure. <strong>Note:</strong> You can insert this operator from the computer keyboard by typing (\xrightarrow{\text{Rad}}).</td>
</tr>
<tr>
<td>rand()</td>
<td>[\text{rand}() \Rightarrow expression] [\text{rand}(#\text{Trials}) \Rightarrow list] (\text{rand}()) returns a random value between 0 and 1. (\text{rand}(#\text{Trials})) returns a list containing #Trials random values between 0 and 1. Sets the random-number seed.</td>
</tr>
</tbody>
</table>
**randBin()**

\[
\text{randBin}(n, p) \Rightarrow \text{expression}
\]

\[
\text{randBin}(n, p, \#\text{Trials}) \Rightarrow \text{list}
\]

`randBin(n, p)` returns a random real number from a specified Binomial distribution.

`randBin(n, p, \#\text{Trials})` returns a list containing \#Trials random real numbers from a specified Binomial distribution.

**randInt()**

\[
\text{randInt}(\text{lowBound}, \text{upBound}) \Rightarrow \text{expression}
\]

\[
\text{randInt}(\text{lowBound}, \text{upBound}, \#\text{Trials}) \Rightarrow \text{list}
\]

`randInt(\text{lowBound}, \text{upBound})` returns a random integer within the range specified by \text{lowBound} and \text{upBound} integer bounds.

`randInt(\text{lowBound}, \text{upBound}, \#\text{Trials})` returns a list containing \#Trials random integers within the specified range.

**randMat()**

\[
\text{randMat}(\text{numRows}, \text{numColumns}) \Rightarrow \text{matrix}
\]

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

Both arguments must simplify to integers.

**randNorm()**

\[
\text{randNorm}(\mu, \sigma) \Rightarrow \text{expression}
\]

\[
\text{randNorm}(\mu, \sigma, \#\text{Trials}) \Rightarrow \text{list}
\]

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

**randPoly()**

\[
\text{randPoly}(\text{Var}, \text{Order}) \Rightarrow \text{expression}
\]

Returns a polynomial in \text{Var} of the specified \text{Order}. The coefficients are random integers in the range -9 through 9. The leading coefficient will not be zero.

\text{Order} must be 0–99.

**randSamp()**

\[
\text{randSamp}(\text{List, \#Trials, noRepl}) \Rightarrow \text{list}
\]

Returns a list containing a random sample of \#Trials trials from \text{List} with an option for sample replacement (noRepl=0), or no sample replacement (noRepl=1). The default is with sample replacement.
**RandSeed**

### 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`.

### real()

#### real(Expr) ⇒ expression

Returns the real part of the argument.

**Note:** All undefined variables are treated as real variables. See also `imag()`, page 58.

#### real(List) ⇒ list

Returns the real parts of all elements.

#### real(Matrix) ⇒ matrix

Returns the real parts of all elements.

### Rect

#### Vector Rect

**Note:** You can insert this operator from the computer keyboard by typing `@>Rect`. Displays `Vector` in rectangular form `[x, y, z]`. The vector must be of dimension 2 or 3 and can be a row or a column.

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

**Note:** See also `Polar`, page 89.

#### complexValue Rect

Displays `complexValue` in rectangular form `a+bi`. The `complexValue` can have any complex form. However, an `r±q` entry causes an error in Degree angle mode.

**Note:** You must use parentheses for an `(r<θ)` polar entry.
ref() \[ \text{ref}(\text{Matrix1}[, \text{Tol}]) \Rightarrow \text{matrix} \]

Returns the row echelon form of Matrix1.

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

- If you use \[\text{ctrl}+\text{enter}\] or set the \textbf{Auto} or \textbf{Approximate} mode to \textbf{Approximate}, computations are done using floating-point arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as:
  \[5\times 10^{-14} \cdot \max(\text{dim}(\text{Matrix1})) \cdot \text{rowNorm}(\text{Matrix1})\]

Avoid undefined elements in Matrix1. They can lead to unexpected results.

For example, if \(a\) is undefined in the following expression, a warning message appears and the result is shown as:

\[
\begin{bmatrix}
a & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
1 & \frac{1}{a} & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

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 constraint (\(|\) operator to substitute a value, as shown in the following example.

\[
\begin{bmatrix}
\alpha & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\textbf{Note:} See also \textbf{rref()}, page 105.

remain() \[ \text{remain}(\text{Expr1}, \text{Expr2}) \Rightarrow \text{expression} \]
\[ \text{remain}(\text{List1}, \text{List2}) \Rightarrow \text{list} \]
\[ \text{remain}(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix} \]

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

\[
\text{remain}(x,0) = x
\]
\[
\text{remain}(x,y) = x - y \cdot \text{iPart}(x/y)
\]

As a consequence, note that \textbf{remain}(-x,y) = - \textbf{remain}(x,y). The result is either zero or it has the same sign as the first argument.

\textbf{Note:} See also \textbf{mod()}, page 78.
Request

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

Programming command: Pauses the program and displays a dialog box containing the message `promptString` and an input box for the user's response.

When the user types a response and clicks OK, the contents of the input box are assigned to variable `var`.

If the user clicks Cancel, the program proceeds without accepting any input. The program uses the previous value of `var` if `var` was already defined.

The optional `DispFlag` argument can be any expression.
- If `DispFlag` is omitted or evaluates to 1, the prompt message and user's response are displayed in the Calculator history.
- If `DispFlag` evaluates to 0, the prompt and response are not displayed in the history.

The optional `statusVar` argument gives the program a way to determine how the user dismissed the dialog box. Note that `statusVar` requires the `DispFlag` argument.
- If the user clicked OK or pressed Enter or Ctrl+Enter, variable `statusVar` is set to a value of 1.
- Otherwise, variable `statusVar` is set to a value of 0.

The `func()` argument allows a program to store the user's response as a function definition. This syntax operates as if the user executed the command:

```
Define func(arg1, ...argn) = user's response
```

The program can then use the defined function `func()`. The `promptString` should guide the user to enter an appropriate user's response that completes the function definition.

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

To stop a program that contains a `Request` command inside an infinite loop:
- **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 `enter` repeatedly.

**Note:** See also `RequestStr`, page 102.
RequestStr

RequestStr promptString, var[, DispFlag]

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

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

To stop a program that contains a RequestStr command inside an infinite loop:
• 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 [enter] repeatedly.

Note: See also Request, page 101.

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

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 [enter] at the end of each line. On the computer keyboard, hold down [Alt] and press [Enter].

Define factorial(nn)=Func
Local answer,count
1 → answer
For count,1,nn
answer·count → answer
EndFor
Return answer
EndFunc

Done

factorial(3) 6

right()

right(List1[, Num]) ⇒ list

Returns the rightmost Num elements contained in List1.

If you omit Num, returns all of List1.

right(sourceString[, Num]) ⇒ string

Returns the rightmost Num characters contained in character string sourceString.

If you omit Num, returns all of sourceString.

right(Comparison) ⇒ expression

Returns the right side of an equation or inequality.
**rk23()**

\[
\text{rk23}(\text{Expr}, \text{Var}, \text{depVar}, \{\text{Var0}, \text{VarMax}\}, \text{depVar0}, \text{VarStep} \{, \text{diftol} \}) \Rightarrow \text{matrix}
\]

\[
\text{rk23}(\text{SystemOfExpr}, \text{Var}, \text{ListOfDepVars}, \{\text{Var0}, \text{VarMax}\}, \text{ListOfDepVar0}, \text{VarStep} \{, \text{diftol} \}) \Rightarrow \text{matrix}
\]

\[
\text{rk23}(\text{ListOfExpr}, \text{Var}, \text{ListOfDepVars}, \{\text{Var0}, \text{VarMax}\}, \text{ListOfDepVar0}, \text{VarStep} \{, \text{diftol} \}) \Rightarrow \text{matrix}
\]

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 \(\text{Var}\) output values as defined by \(\text{VarStep}\). The second row defines the value of the first solution component at the corresponding \(\text{Var}\) values, and so on.

\(\text{Expr}\) is the right hand side that defines the ordinary differential equation (ODE).

\(\text{SystemOfExpr}\) is a system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in \(\text{ListOfDepVar}\)).

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

\(\text{Var}\) is the independent variable.

\(\text{ListOfDepVars}\) is a list of dependent variables.

\(\{\text{Var0}, \text{VarMax}\}\) is a two-element list that tells the function to integrate from \(\text{Var0}\) to \(\text{VarMax}\).

\(\text{ToListOfDepVar0}\) is a list of initial values for dependent variables.

If \(\text{VarStep}\) evaluates to a nonzero number: \(\text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax}-\text{Var0})\) for all \(i=0,1,2,\ldots\) such that \(\text{Var0}+i*\text{VarStep}\) is in \([\text{Var0}, \text{VarMax}]\) (may not get a solution value at \(\text{VarMax}\)).

If \(\text{VarStep}\) evaluates to zero, solutions are returned at the "Runge-Kutta" \(\text{Var}\) values.

\(\text{diftol}\) is the error tolerance (defaults to 0.001).

Differential equation:

\[
y'=0.001*y*(100-y)\text{ and } y(0)=10
\]

\[
\begin{array}{cccc}
0. & 1. & 2. & 3. \\
10. & 10.9367 & 11.9493 & 13.0424
\end{array}
\]

To see the entire result, press \(\text{and then use } \) and \(\) to move the cursor.

Same equation with \(\text{diftol} \) set to \(1E\times6\)

\[
\begin{array}{cccc}
0. & 1. & 2. & 3. & 4. \\
\end{array}
\]

Compare above result with CAS exact solution obtained using \(\text{deSolve()}\) and \(\text{seqGen()}\):

\[
\text{deSolve}\left(y'=0.001*y*(100-y)\text{ and } y(0)=10, t, y\right)
\]

\[
\begin{array}{cccc}
0. \text{ to } 1.10517 \\
1.10517 \text{ to } 1.10517+9, \\
10.109367, 11.9496, 13.0424, 14.2189, 15.49
\end{array}
\]

System of equations:

\[
y_1'=y_1+0.1*y_1*y_2 \\
y_2'=3*y_2-y_1*y_2
\]

with \(y_1(0)=2\) and \(y_2(0)=5\)

\[
\begin{array}{cccc}
0. & 1. & 2. & 3. \\
2. & 1.94103 & 4.78694 & 3.25253 & 1.82848 \\
\end{array}
\]

**root()**

\[
\text{root}(\text{Expr}) \Rightarrow \text{root}
\]

\[
\text{root}(\text{Expr1}, \text{Expr2}) \Rightarrow \text{root}
\]

\(\text{root}(\text{Expr})\) returns the square root of \(\text{Expr}\).

\(\text{root}(\text{Expr1}, \text{Expr2})\) returns the \(\text{Expr2}\) root of \(\text{Expr1}\). \(\text{Expr1}\) can be a real or complex floating point constant, an integer or complex rational constant, or a general symbolic expression.

**Note:** See also \(\text{Nth root template}, \) page 1.
\textbf{rotate()}

\texttt{rotate(Integer1[#ofRotations]) \Rightarrow integer}

Rotates the bits in a binary integer. You can enter \texttt{Integer1} in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of \texttt{Integer1} is too large for this form, a symmetric modulo operation brings it within the range. For more information, see \textit{Base2}, page 14.

If \texttt{#ofRotations} is positive, the rotation is to the left. If \texttt{#ofRotations} is negative, the rotation is to the right. The default is \texttt{-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.

\texttt{rotate(List1[#ofRotations]) \Rightarrow list}

Returns a copy of \texttt{List1} rotated right or left by \texttt{#ofRotations} elements. Does not alter \texttt{List1}.

If \texttt{#ofRotations} is positive, the rotation is to the left. If \texttt{#ofRotations} is negative, the rotation is to the right. The default is \texttt{-1} (rotate right one element).

\texttt{rotate(String1[#ofRotations]) \Rightarrow string}

Returns a copy of \texttt{String1} rotated right or left by \texttt{#ofRotations} characters. Does not alter \texttt{String1}.

If \texttt{#ofRotations} is positive, the rotation is to the left. If \texttt{#ofRotations} is negative, the rotation is to the right. The default is \texttt{-1} (rotate right one character).

\textbf{round()}

\texttt{round(Expr1[digits]) \Rightarrow expression}

Returns the argument rounded to the specified number of digits after the decimal point.

\texttt{digits} must be an integer in the range 0–12. If \texttt{digits} is not included, returns the argument rounded to 12 significant digits.

\textbf{Note:} Display digits mode may affect how this is displayed.

\texttt{round(List1[digits]) \Rightarrow list}

Returns a list of the elements rounded to the specified number of digits.

\texttt{round(Matrix1[digits]) \Rightarrow matrix}

Returns a matrix of the elements rounded to the specified number of digits.
rowAdd()  

\[
\text{rowAdd}(\text{Matrix1}, rIndex1, rIndex2) \Rightarrow \text{matrix}
\]

Returns a copy of Matrix1 with row rIndex2 replaced by the sum of rows rIndex1 and rIndex2.

\[
\begin{bmatrix}
3 & 4 \\
-3 & -2
\end{bmatrix}_{1,2} +
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}_{1,2} =
\begin{bmatrix}
a+c & b+d \\
0 & 2
\end{bmatrix}
\]

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

\[
\text{rowAdd}\left(\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}, 1,2\right) =
\begin{bmatrix}
a+c & b+d
\end{bmatrix}
\]

rowDim()  

\[
\text{rowDim}(\text{Matrix}) \Rightarrow \text{expression}
\]

Returns the number of rows in Matrix.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} \Rightarrow m1
\]

\[
\text{rowDim}(m1) = 3
\]

rowNorm()  

\[
\text{rowNorm}(\text{Matrix}) \Rightarrow \text{expression}
\]

Returns the maximum of the sums of the absolute values of the elements in the rows in Matrix.

\[
\begin{bmatrix}
-5 & 6 & -7 \\
3 & 4 & 9 \\
9 & -9 & -7
\end{bmatrix}
\]

\[
\text{rowNorm}\left(\begin{bmatrix}
-5 & 6 & -7 \\
3 & 4 & 9 \\
9 & -9 & -7
\end{bmatrix}\right) = 25
\]

Note: All matrix elements must simplify to numbers. See also colNorm(), page 19.

rowSwap()  

\[
\text{rowSwap}(\text{Matrix1}, rIndex1, rIndex2) \Rightarrow \text{matrix}
\]

Returns Matrix1 with rows rIndex1 and rIndex2 exchanged.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} \Rightarrow \text{mat}
\]

\[
\text{rowSwap}(\text{mat}, 1, 3) \Rightarrow
\begin{bmatrix}
5 & 6 \\
3 & 4 \\
1 & 2
\end{bmatrix}
\]

rref()  

\[
\text{rref}(\text{Matrix1}, \text{Tol}) \Rightarrow \text{matrix}
\]

Returns the reduced row echelon form of Matrix1.

\[
\begin{bmatrix}
-2 & -2 & 0 & -6 \\
1 & 1 & 9 & -9 \\
5 & 2 & 4 & -4
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & \frac{66}{71} \\
0 & 1 & 0 & \frac{147}{71} \\
0 & 0 & 1 & \frac{-62}{71}
\end{bmatrix}
\]
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:

$$5 \times 10^{-14} \cdot \max(\text{dim}(	ext{Matrix1})) \cdot \text{rowNorm}(	ext{Matrix1})$$

Note: See also ref(), page 100.

**sec()**

sec(Expr1) ⇒ expression
sec(List1) ⇒ list

Returns the secant of Expr1 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 °, ′, or ″ to override the angle mode temporarily.

**sec^−1()**

sec^−1(Expr1) ⇒ expression
sec^−1(List1) ⇒ list

Returns the angle whose secant is Expr1 or returns a list containing the inverse secants of each element of List1.

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

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

**sech()**

sech(Expr1) ⇒ expression
sech(List1) ⇒ list

Returns the hyperbolic secant of Expr1 or returns a list containing the hyperbolic secants of the List1 elements.
\(\text{sech}^{-1}(\text{Expr1}) \Rightarrow \text{expression}\)

\(\text{sech}^{-1}(\text{List1}) \Rightarrow \text{list}\)

Returns the inverse hyperbolic secant of \(\text{Expr1}\) or returns a list containing the inverse hyperbolic secants of each element of \(\text{List1}\).

Note: You can insert this function from the keyboard by typing \(\text{arcsech}(\ldots)\).

In Radian angle and Rectangular complex mode:

\[
\begin{align*}
\text{sech}^{-1}(1) & \Rightarrow 0 \\
\text{sech}^{-1}(\{1, -2, 2.1\}) & \Rightarrow \left\{0, \frac{2\pi}{3}, i8, e^{-15} + 1.07448i\right\}
\end{align*}
\]

\(\text{seq}(\text{Expr}, \text{Var}, \text{Low}, \text{High}, \text{Step}) \Rightarrow \text{list}\)

Increments \(\text{Var}\) from \(\text{Low}\) through \(\text{High}\) by an increment of \(\text{Step}\), evaluates \(\text{Expr}\), and returns the results as a list. The original contents of \(\text{Var}\) are still there after \(\text{seq()}\) is completed.

The default value for \(\text{Step} = 1\).

\[
\begin{align*}
\text{seq}(n^2, n, 1, 6) & \Rightarrow \{1, 4, 9, 16, 25, 36\} \\
\text{seq}(\frac{1}{n}, n, 1, 10, 2) & \Rightarrow \left\{\frac{1}{1}, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}\right\} \\
\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right) & \Rightarrow 1968329 \\
\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right) & \Rightarrow 1270080
\end{align*}
\]

Press \(\text{Ctrl}+\text{Enter} \(\text{ctrl}+\text{enter}\) \(\text{Macintosh}: +\text{Enter}\) \(\text{Xor}: +\text{Enter}\)\) to evaluate:

\[
\begin{align*}
\text{sum}\left(\text{seq}\left(\frac{1}{n^2}, n, 1, 10, 1\right)\right) & \Rightarrow 1.54977
\end{align*}
\]
seqGen()

Generates a list of terms for a sequence \( \text{depVar}(\text{Var})=\text{Expr} \) as follows:
Increments \( \text{Var} \) from \( \text{Var0} \) through \( \text{VarMax} \) by \( \text{VarStep} \), evaluates \( \text{depVar}(\text{Var}) \) for corresponding values of \( \text{Var} \) using the \( \text{Expr} \) formula and \( \text{ListOfInitTerms} \), and returns the results as a list.

seqGen(ListOfDepVars(Var)=ListOrSystemOfExpr, \text{Var}, \text{MatrixOfInitTerms} [, \text{VarStep}, \text{CeilingValue}]) \Rightarrow \text{matrix}

Generates a matrix of terms for a system (or list) of sequences \( \text{ListOfDepVars}(\text{Var})=\text{ListOrSystemOfExpr} \) as follows: Increments \( \text{Var} \) from \( \text{Var0} \) through \( \text{VarMax} \) by \( \text{VarStep} \), evaluates \( \text{ListOfDepVars}(\text{Var}) \) for corresponding values of \( \text{Var} \) using \( \text{ListOrSystemOfExpr} \) formula and \( \text{MatrixOfInitTerms} \), and returns the results as a matrix.

The original contents of \( \text{Var} \) are unchanged after \( \text{seqGen()} \) is completed.

The default value for \( \text{VarStep} = 1 \).

seqn()

Generates a list of terms for a sequence \( u(n)=\text{Expr}(u, n) \) as follows:
Increments \( n \) from 1 through \( nMax \) by 1, evaluates \( u(n) \) for corresponding values of \( n \) using the \( \text{Expr}(u, n) \) formula and \( \text{ListOfInitTerms} \), and returns the results as a list.

seqn(\text{Expr}(n), \text{nMax}, \text{CeilingValue})) \Rightarrow \text{list}

Generates a list of terms for a non-recursive sequence \( u(n)=\text{Expr}(n) \) as follows: Increments \( n \) from 1 through \( nMax \) by 1, evaluates \( u(n) \) for corresponding values of \( n \) using the \( \text{Expr}(n) \) 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: \( \text{seqn() calls seqGen()} \) with \( \text{nterms}=1 \) and \( nstep = 1 \)

Generate the first 5 terms of the sequence \( u(n) = u(n-1)^2/2 \), with \( u(1)=2 \) and \( \text{VarStep}=1 \).

\[
\text{seqGen}\left(\frac{(u(n-1))^2}{n}, u, \{1,5\}, \{2\}\right) = \begin{array}{c}
2, 2, 4, 4, 16 \\
3, 3, 9, 405
\end{array}
\]

Example in which \( \text{Var0}=2 \):

\[
\text{seqGen}\left(\frac{u(n+1)+1}{n}, u, \{2,5\}, \{3\}\right) = \begin{array}{c}
3, 4, 7, 19 \\
3, 3, 12, 60
\end{array}
\]

Example in which initial term is symbolic:

\[
\text{seqGen}(u(n+1)+2+n, u, \{1,5\}, \{a\}) = \{a, a+2, a+4, a+6, a+8\}
\]

System of two sequences:

\[
\text{seqGen}\left(\begin{pmatrix}
\frac{1}{n} & \frac{u(n-1)}{2} \\
\end{pmatrix}, \text{Var0}, \{1,5\}, \{2\}\right) = \begin{array}{cc}
1 & 2 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
5 & 6 \\
6 & 7 \\
7 & 8
\end{array}
\]

Note: The Void (_) in the initial term matrix above is used to indicate that the initial term for \( u1(n) \) is calculated using the explicit sequence formula \( u1(n)=1/n \).
**series()**

series(Expr1, Var, Order [, Point]) \( \Rightarrow \) expression

series(Expr1, Var, Order [, Point]) | Var \( \Rightarrow \) Point \( \Rightarrow \) expression

Returns a generalized truncated power series representation of Expr1 expanded about Point through degree Order. Order can be any rational number. The resulting powers of (Var \( \Rightarrow \) Point) can include negative and/or fractional exponents. The coefficients of these powers can include logarithms of (Var \( \Rightarrow \) Point) and other functions of Var that are dominated by all powers of (Var \( \Rightarrow \) Point) having the same exponent sign.

Point defaults to 0. Point can be \( \infty \) or \(-\infty\), in which cases the expansion is through degree Order in \( 1/(Var \Rightarrow Point) \).

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

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

series() can provide symbolic approximations to indefinite integrals and definite integrals for which symbolic solutions otherwise can’t be obtained.

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

series() is a generalized version of taylor().

As illustrated by the last example to the right, the display routines downstream of the result produced by series(...) might rearrange terms so that the dominant term is not the leftmost one.

**Note:** See also dominantTerm(), page 39.
setMode()

setMode(modeNameInteger, settingInteger) ⇒ integer

setMode(list) ⇒ integer list

Valid only within a function or program.

setMode(modeNameInteger, settingInteger) temporarily sets mode modeNameInteger to the new setting settingInteger, and
returns an integer corresponding to the original setting of that
mode. The change is limited to the duration of the program/
function’s execution.

modeNameInteger specifies which mode you want to set. It must
be one of the mode integers from the table below.

settingInteger specifies the new setting for the mode. It must be
one of the setting integers listed below for the specific mode you
are setting.

setMode(list) lets you change multiple settings. list contains
pairs of mode integers and setting integers. setMode(list)
returns a similar list whose integer pairs represent the original
modes and settings.

If you have saved all mode settings with getMode(0) → var,
you can use setMode(var) to restore those settings until the
function or program exits. See getMode(), page 54.

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 [alt] instead of [enter] at the end of each line. On the
computer keyboard, hold down Alt and press Enter.

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td>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</td>
</tr>
<tr>
<td>Angle</td>
<td>2</td>
<td>1=Radian, 2=Degree, 3=Gradian</td>
</tr>
<tr>
<td>Exponential Format</td>
<td>3</td>
<td>1=Normal, 2=Scientific, 3=Engineering</td>
</tr>
<tr>
<td>Real or Complex</td>
<td>4</td>
<td>1=Real, 2=Rectangular, 3=Polar</td>
</tr>
<tr>
<td>Auto or Approx.</td>
<td>5</td>
<td>1=Auto, 2=Approximate, 3=Exact</td>
</tr>
<tr>
<td>Vector Format</td>
<td>6</td>
<td>1=Rectangular, 2=Cylindrical, 3=Spherical</td>
</tr>
<tr>
<td>Base</td>
<td>7</td>
<td>1=Decimal, 2=Hex, 3=Binary</td>
</tr>
<tr>
<td>Unit system</td>
<td>8</td>
<td>1=SI, 2=Eng/US</td>
</tr>
</tbody>
</table>

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
Disp approx(π)
setMode(1,16)
Disp approx(π)
EndPrgm

prog1()

3.14159
3.14

Done
shift()

\[ \text{shift}(\text{Integer1}, \text{#ofShifts}) \Rightarrow \text{integer} \]

Shifts the bits in a binary integer. You can enter \text{Integer1} in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of \text{Integer1} is too large for this form, a symmetric modulo operation brings it within the range. For more information, see \text{Base2}, page 14.

If \#of\text{Shifts} is positive, the shift is to the left. If \#of\text{Shifts} is negative, the shift is to the right. The default is \text{L1} (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.

\[ \text{0b0000000000000111101011000011010} \]

Inserts 0 if leftmost bit is 0, or 1 if leftmost bit is 1.

produces:

\[ \text{0b00000000000000111101011000011010} \]

The result is displayed according to the Base mode. Leading zeros are not shown.

\[ \text{shift}(\text{List1}, \text{#ofShifts}) \Rightarrow \text{list} \]

Returns a copy of \text{List1} shifted right or left by \#of\text{Shifts} elements. Does not alter \text{List1}.

If \#of\text{Shifts} is positive, the shift is to the left. If \#of\text{Shifts} is negative, the shift is to the right. The default is \text{L1} (shift right one element).

Elements introduced at the beginning or end of \text{list} by the shift are set to the symbol "undef".

\[ \text{shift}(\text{String1}, \text{#ofShifts}) \Rightarrow \text{string} \]

Returns a copy of \text{String1} shifted right or left by \#of\text{Shifts} characters. Does not alter \text{String1}.

If \#of\text{Shifts} is positive, the shift is to the left. If \#of\text{Shifts} is negative, the shift is to the right. The default is \text{L1} (shift right one character).

Characters introduced at the beginning or end of \text{string} by the shift are set to a space.

sign()

\[ \text{sign}(\text{Expr1}) \Rightarrow \text{expression} \]

\[ \text{sign}(\text{List1}) \Rightarrow \text{list} \]

\[ \text{sign}(\text{Matrix1}) \Rightarrow \text{matrix} \]

For real and complex \text{Expr1}, returns \text{Expr1}/\text{abs}(\text{Expr1}) when \text{Expr1} \neq 0.

Returns \text{L1} if \text{Expr1} is positive.

Returns \text{L1} if \text{Expr1} is negative.

\text{sign}(0) \text{ returns } \pm 1 \text{ if the complex format mode is Real; otherwise, it returns itself.}

\text{sign}(0) \text{ represents the unit circle in the complex domain.}

For a list or matrix, returns the signs of all the elements.
simult()

\[ \text{simult}(\text{coeffMatrix}, \text{constVector}, \text{Tol}) \rightarrow \text{matrix} \]

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also \text{linSolve()}, page 67.

\text{coeffMatrix} must be a square matrix that contains the coefficients of the equations.

\text{constVector} must have the same number of rows (same dimension) as \text{coeffMatrix} and contain the constants.

Optionally, any matrix element is treated as zero if its absolute value is less than \text{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, \text{Tol} is ignored.

- If you set the \text{Auto} or \text{Approximate} mode to \text{Approximate}, computations are done using floating-point arithmetic.
- If \text{Tol} is omitted or not used, the default tolerance is calculated as:
  \[ 5 \times 10^{-14} \cdot \max(\text{dim(coeffMatrix)}) \cdot \text{rowNorm(coeffMatrix)} \]

\[ \text{simult(coeffsMatrix, constMatrix, Tol)} \rightarrow \text{matrix} \]

Solves multiple systems of linear equations, where each system has the same equation coefficients but different constants.

Each column in \text{constMatrix} must contain the constants for a system of equations. Each column in the resulting matrix contains the solution for the corresponding system.

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \text{matxl} \]

\[ \text{simult(matxl, \begin{bmatrix} 1 \\ 2 \end{bmatrix})} \rightarrow \begin{bmatrix} -2-b-d \\ 2a-c \frac{a-d-b \cdot c}{2} \end{bmatrix} \]

\[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \text{matrix} \]

\[ \text{simult(coeffsMatrix, constMatrix, Tol)} \rightarrow \text{matrix} \]

Solve for \( x \) and \( y \):

\[ \begin{align*}
  x + 2y &= 1 \\
  3x + 4y &= -1
\end{align*} \]

\[ \text{simult} \left( \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) \rightarrow \begin{bmatrix} -3 \\ 2 \end{bmatrix} \]

The solution is \( x=-3 \) and \( y=2 \).

Solve:

\[ \begin{align*}
  ax + by &= 1 \\
  cx + dy &= 2
\end{align*} \]

\[ \begin{align*}
  a & b \\
  c & d
\end{align*} \rightarrow \text{matxl} \]

\[ \text{simult(matxl, \begin{bmatrix} 1 \\ 2 \end{bmatrix})} \rightarrow \begin{bmatrix} -2-b-d \\ 2a-c \frac{a-d-b \cdot c}{2} \end{bmatrix} \]

\[ \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \text{matxl} \]

\[ \text{simult(coeffsMatrix, constMatrix, Tol)} \rightarrow \text{matrix} \]

\[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \text{matrix} \]

For the first system, \( x=-3 \) and \( y=2 \). For the second system, \( x=-7 \) and \( y=9/2 \).

\( \text{Expr} \rightarrow \text{sin} \)

Note: You can insert this operator from the computer keyboard by typing \text{@>sin}.

Represents \text{Expr} in terms of sine. This is a display conversion operator. It can be used only at the end of the entry line.

\( \text{Expr} \rightarrow \text{sin} \) reduces all powers of \( \cos(\ldots) \) modulo \( 1-\sin(\ldots)^2 \) so that any remaining powers of \( \sin(\ldots) \) have exponents in the range \((0, 2)\). Thus, the result will be free of \( \cos(\ldots) \) if and only if \( \cos(\ldots) \) occurs in the given expression only to even powers.

Note: This conversion operator is not supported in Degree or Gradian Angle modes. Before using it, make sure that the Angle mode is set to Radians and that \text{Expr} does not contain explicit references to degree or gradian angles.
\[ \sin() \]

\[ \sin(\text{Expr1}) \Rightarrow \text{expression} \]
\[ \sin(\text{List1}) \Rightarrow \text{list} \]

\[ \sin(\text{Expr1}) \] returns the sine of the argument as an expression.
\[ \sin(\text{List1}) \] returns a list of the sines of all elements in \text{List1}.

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use \( \circ \), \( \degree \), or \( \pi \) to override the angle mode setting temporarily.

\[ \sin(\text{Expr1}) \]

In Degree angle mode:

\[
\begin{align*}
\sin\left(\frac{\pi}{4}\right) &= \frac{\sqrt{2}}{2} \\
\sin(45) &= \frac{\sqrt{2}}{2} \\
\sin\left\{0,60,90\right\} &= \left\{0,\frac{\sqrt{3}}{2},1\right\}
\end{align*}
\]

In Gradian angle mode:

\[
\sin\left(50\right) = \frac{\sqrt{2}}{2}
\]

In Radian angle mode:

\[
\begin{align*}
\sin\left(\frac{\pi}{4}\right) &= \frac{\sqrt{2}}{2} \\
\sin(45^\circ) &= \frac{\sqrt{2}}{2}
\end{align*}
\]

\[ \sin(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

Returns the matrix sine of \text{squareMatrix1}. This is not the same as calculating the sine of each element. For information about the calculation method, refer to \( \cos() \).

\text{squareMatrix1} must be diagonalizable. The result always contains floating-point numbers.

\[ \sin^{-1}() \]

\[ \sin^{-1}(\text{Expr1}) \Rightarrow \text{expression} \]
\[ \sin^{-1}(\text{List1}) \Rightarrow \text{list} \]

\[ \sin^{-1}(\text{Expr1}) \] returns the angle whose sine is \text{Expr1} as an expression.
\[ \sin^{-1}(\text{List1}) \] returns a list of the inverse sines of each element of \text{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 \( \text{arcsin}(\ldots) \).
\(\text{sin}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\)

Returns the matrix inverse sine of \(\text{squareMatrix1}\). This is not the same as calculating the inverse sine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode and Rectangular complex format mode:

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

\[
\begin{bmatrix}
-0.164058 & -0.064606 \cdot i \\
0.725533 & -1.51594 \cdot i \\
-2.08316 & -2.63205 \cdot i
\end{bmatrix} = \sin^{-1}(\begin{bmatrix}1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1\end{bmatrix})
\]

To see the entire result, press \(\uparrow\) and then use \(\downarrow\) and \(\uparrow\) to move the cursor.

\(\text{sinh()}\)

\[\text{sinh}(\text{Expr1}) \Rightarrow \text{expression}\]

\[\text{sinh}(\text{List1}) \Rightarrow \text{list}\]

\(\text{sinh}(\text{Expr1})\) returns the hyperbolic sine of the argument as an expression.

\(\text{sinh}(\text{List1})\) returns a list of the hyperbolic sines of each element of \(\text{List1}\).

\(\text{sinh}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\)

Returns the matrix hyperbolic sine of \(\text{squareMatrix1}\). This is not the same as calculating the hyperbolic sine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

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

\[
\begin{bmatrix}
360.954 & 305.708 & 239.604 \\
352.912 & 233.495 & 193.564 \\
298.632 & 154.599 & 140.251
\end{bmatrix} = \sinh(\begin{bmatrix}1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1\end{bmatrix})
\]

\(\text{sinh}^{-1}()\)

\[\text{sinh}^{-1}(\text{Expr1}) \Rightarrow \text{expression}\]

\[\text{sinh}^{-1}(\text{List1}) \Rightarrow \text{list}\]

\(\text{sinh}^{-1}(\text{Expr1})\) returns the inverse hyperbolic sine of the argument as an expression.

\(\text{sinh}^{-1}(\text{List1})\) returns a list of the inverse hyperbolic sines of each element of \(\text{List1}\).

\(\text{Note:} \) You can insert this function from the keyboard by typing \text{arcsinh}(\ldots)\).

\(\text{sinh}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\)

Returns the matrix inverse hyperbolic sine of \(\text{squareMatrix1}\). This is not the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

In Radian angle mode:

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

\[
\begin{bmatrix}
0.041751 & 2.15557 & 1.1582 \\
1.46382 & 0.926568 & 0.112557 \\
2.75079 & -1.5283 & 0.57268
\end{bmatrix} = \sinh^{-1}(\begin{bmatrix}1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1\end{bmatrix})
\]
**SinReg**

**SinReg** $X, Y [, Iterations], [, Period], [, Category, Include]$

Computes the sinusoidal regression on lists $X$ and $Y$. A summary of results is stored in the `stat.results` variable. (See page 120.)

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

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

*Iterations* is a value that specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

*Period* specifies an estimated period. If omitted, the difference between values in $X$ should be equal and in sequential order. If you specify *Period*, the differences between $x$ values can be unequal.

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

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

The output of **SinReg** is always in radians, regardless of the angle mode setting.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 162.

---

**Output variable**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>stat.RegEqn</strong></td>
</tr>
<tr>
<td><strong>stat.a, stat.b, stat.c, stat.d</strong></td>
</tr>
<tr>
<td><strong>stat.Resid</strong></td>
</tr>
<tr>
<td><strong>stat.XReg</strong></td>
</tr>
<tr>
<td><strong>stat.YReg</strong></td>
</tr>
<tr>
<td><strong>stat.FreqReg</strong></td>
</tr>
</tbody>
</table>

**solve()**

| **solve**($Equation$, $Var$) | $\Rightarrow$ Boolean expression |
| **solve**($Equation$, $Var$=$Guess$) | $\Rightarrow$ Boolean expression |
| **solve**($Inequality$, $Var$) | $\Rightarrow$ Boolean expression |

Returns candidate real solutions of an equation or an inequality for $Var$. The goal is to return candidates for all solutions. However, there might be equations or inequalities for which the number of solutions is infinite.

Solution candidates might not be real finite solutions for some combinations of values for undefined variables.

---

**Example**

\[
solve\left(a \cdot x^2 + b \cdot x + c = 0, x\right)\\
\]

\[
x = \frac{\sqrt{b^2 - 4 \cdot a \cdot c} - b}{2 \cdot a} \text{ or } x = \frac{\sqrt{b^2 - 4 \cdot a \cdot c} + b}{2 \cdot a}\\
\]

Ans if $a = 1$ and $b = 1$ and $c = 1$

\[
x = \frac{-1 + \sqrt{3}}{2} i \text{ or } x = \frac{-1 - \sqrt{3}}{2} i\\
\]
For the Auto setting of the Auto or Approximate mode, the goal is to produce exact solutions when they are concise, and supplemented by iterative searches with approximate arithmetic when exact solutions are impractical.

Due to default cancellation of the greatest common divisor from the numerator and denominator of ratios, solutions might be solutions only in the limit from one or both sides.

For inequalities of types $\geq$, $\leq$, $<$, or $>$, explicit solutions are unlikely unless the inequality is linear and contains only $\text{Var}$.

For the Exact mode, portions that cannot be solved are returned as an implicit equation or inequality.

Use the constraint ("|") operator to restrict the solution interval and/or other variables that occur in the equation or inequality. When you find a solution in one interval, you can use the inequality operators to exclude that interval from subsequent searches.

In Radian angle mode:

false is returned when no real solutions are found. true is returned if solve() can determine that any finite real value of $\text{Var}$ satisfies the equation or inequality.

Since solve() always returns a Boolean result, you can use "and," "or," and "not" to combine results from solve() with each other or with other Boolean expressions.

Solutions might contain a unique new undefined constant of the form $n_j$ with $j$ being an integer in the interval 1–255. Such variables designate an arbitrary integer.

In Real mode, fractional powers having odd denominators denote only the real branch. Otherwise, multiple branched expressions such as fractional powers, logarithms, and inverse trigonometric functions denote only the principal branch. Consequently, solve() produces only solutions corresponding to that one real or principal branch.

Note: See also cSolve(), cZeros(), nSolve(), and zeros().
If all of the equations are polynomials and if you do NOT specify any initial guesses, \texttt{solve()} uses the lexical Gröbner/Buchberger elimination method to attempt to determine all real solutions.

For example, suppose you have a circle of radius \( r \) at the origin and another circle of radius \( r \) centered where the first circle crosses the positive x-axis. Use \texttt{solve()} to find the intersections.

As illustrated by \( r \) in the example to the right, simultaneous polynomial equations can have extra variables that have no values, but represent given numeric values that could be substituted later.

You can also (or instead) include solution variables that do not appear in the equations. For example, you can include \( z \) as a solution variable to extend the previous example to two parallel intersecting cylinders of radius \( r \).

The cylinder solutions illustrate how families of solutions might contain arbitrary constants of the form \( ck \), where \( k \) is an integer suffix from 1 through 255.

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

If you do not include any guesses and if any equation is non-polynomial in any variable but all equations are linear in the solution variables, \texttt{solve()} uses Gaussian elimination to attempt to determine all real solutions.

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

Each solution variable starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional solutions one by one. For convergence, a guess may have to be rather close to a solution.
**SortA**

\[ \text{SortA } \{ \text{List1, List2, List3} \} \ldots \]

\[ \text{SortA } \{ \text{Vector1, Vector2, Vector3} \} \ldots \]

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 162.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,1,4,3} → list1</td>
<td>{2,1,4,3}</td>
</tr>
<tr>
<td>{4,3,2,1} → list2</td>
<td>{4,3,2,1}</td>
</tr>
</tbody>
</table>

**SortD**

\[ \text{SortD } \{ \text{List1, List2, List3} \} \ldots \]

\[ \text{SortD } \{ \text{Vector1, Vector2, Vector3} \} \ldots \]

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 162.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,1,4,3} → list1</td>
<td>{2,1,4,3}</td>
</tr>
<tr>
<td>{1,2,3,4} → list2</td>
<td>{1,2,3,4}</td>
</tr>
</tbody>
</table>

SortD \{list1, list2\} \text{ Done}
**Sphere**

**Vector Sphere**

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

Displays the row or column vector in spherical form \( [\rho \angle \theta \angle \phi] \).

**Vector** must be of dimension 3 and can be either a row or a column vector.

**Note:** `Sphere` is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

Press `Ctrl + Enter` (Macintosh®: `⌘ + Enter`) to evaluate:

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix} \cdot \text{Sphere}
\]

\[
\begin{bmatrix}
3.74166 & \angle 1.10715 & \angle 0.640522
\end{bmatrix}
\]

Press `Ctrl + Enter` (Macintosh®: `⌘ + Enter`) to evaluate:

\[
\begin{bmatrix}
2 & \angle \frac{\pi}{4} & 3
\end{bmatrix} \cdot \text{Sphere}
\]

\[
\begin{bmatrix}
3.60555 & \angle 0.785398 & \angle 0.588003
\end{bmatrix}
\]

\[
\begin{align*}
\text{sqrt}(\text{Expr1}) & \Rightarrow \text{expression} \\
\text{sqrt}(\text{List1}) & \Rightarrow \text{list}
\end{align*}
\]

Returns the square root of the argument.

For a list, returns the square roots of all the elements in `List1`.

**Note:** See also Square root template, page 1.
stat.results

Displays results from a statistics calculation.

The results are displayed as a set of 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.

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.

<table>
<thead>
<tr>
<th>stat</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.a</td>
<td>stat.dfDenom</td>
</tr>
<tr>
<td>stat.b</td>
<td>stat.dfBlock</td>
</tr>
<tr>
<td>stat.b0</td>
<td>stat.dfCol</td>
</tr>
<tr>
<td>stat.b1</td>
<td>stat.dfError</td>
</tr>
<tr>
<td>stat.b2</td>
<td>stat.dfReg</td>
</tr>
<tr>
<td>stat.b3</td>
<td>stat.dfNumer</td>
</tr>
<tr>
<td>stat.b4</td>
<td>stat.dfRow</td>
</tr>
<tr>
<td>stat.b5</td>
<td>stat.DW</td>
</tr>
<tr>
<td>stat.b6</td>
<td>stat.e</td>
</tr>
<tr>
<td>stat.b7</td>
<td>stat.ExpMatrix</td>
</tr>
<tr>
<td>stat.b8</td>
<td>stat.F</td>
</tr>
<tr>
<td>stat.b9</td>
<td>stat.FBlock</td>
</tr>
<tr>
<td>stat.b10</td>
<td>stat.FCol</td>
</tr>
<tr>
<td>stat.blst</td>
<td>stat.FInteract</td>
</tr>
<tr>
<td>stat.χ²</td>
<td>stat.FreqReg</td>
</tr>
<tr>
<td>stat.c</td>
<td>stat.Frow</td>
</tr>
<tr>
<td>stat.Clower</td>
<td>stat.Leverage</td>
</tr>
<tr>
<td>stat.ClowerList</td>
<td>stat.LowerPred</td>
</tr>
<tr>
<td>stat.ComplMatrix</td>
<td>stat.LowerVal</td>
</tr>
<tr>
<td>stat.CookDist</td>
<td>stat.m</td>
</tr>
<tr>
<td>stat.CUpper</td>
<td>stat.MaxX</td>
</tr>
<tr>
<td>stat.CUpperList</td>
<td>stat.MaxY</td>
</tr>
<tr>
<td>stat.d</td>
<td>stat.ME</td>
</tr>
<tr>
<td>stat.medianX</td>
<td>stat.MedianX</td>
</tr>
<tr>
<td>stat.dMedianY</td>
<td>stat.Q1X</td>
</tr>
<tr>
<td>stat.dMEPred</td>
<td>stat.Q3X</td>
</tr>
<tr>
<td>stat.dMinPred</td>
<td>stat.Q3Y</td>
</tr>
<tr>
<td>stat.dMinX</td>
<td>stat.r</td>
</tr>
<tr>
<td>stat.dr</td>
<td>stat.RegEqn</td>
</tr>
<tr>
<td>stat.dResid</td>
<td>stat.Resid</td>
</tr>
<tr>
<td>stat.dResidTrans</td>
<td>stat.ResidTrans</td>
</tr>
<tr>
<td>stat.dox</td>
<td>stat.c</td>
</tr>
<tr>
<td>stat.dox1</td>
<td>stat.Ç1</td>
</tr>
<tr>
<td>stat.dox2</td>
<td>stat.Ç2</td>
</tr>
<tr>
<td>stat.dξ</td>
<td>stat.ξ</td>
</tr>
<tr>
<td>stat.dξ1</td>
<td>stat.ξ1</td>
</tr>
<tr>
<td>stat.dξ2</td>
<td>stat.ξ2</td>
</tr>
<tr>
<td>stat.dξy</td>
<td>stat.ξy</td>
</tr>
<tr>
<td>stat.dξy²</td>
<td>stat.ξy²</td>
</tr>
<tr>
<td>stat.dξ₂</td>
<td>stat.ξ₂</td>
</tr>
<tr>
<td>stat.dξ</td>
<td>stat.ξ</td>
</tr>
<tr>
<td>stat.dv</td>
<td>stat.v</td>
</tr>
<tr>
<td>stat.dv1</td>
<td>stat.v1</td>
</tr>
<tr>
<td>stat.dv2</td>
<td>stat.v2</td>
</tr>
<tr>
<td>stat.dvList</td>
<td>stat.vDiff</td>
</tr>
<tr>
<td>stat.dXReg</td>
<td>stat.X</td>
</tr>
<tr>
<td>stat.dX</td>
<td>stat.X</td>
</tr>
<tr>
<td>stat.dXdiff</td>
<td>stat.XList</td>
</tr>
<tr>
<td>stat.dXlist</td>
<td>stat.XReg</td>
</tr>
<tr>
<td>stat.dXval</td>
<td>stat.XVal</td>
</tr>
<tr>
<td>stat.dy</td>
<td>stat.y</td>
</tr>
<tr>
<td>stat.dyl</td>
<td>stat.y</td>
</tr>
<tr>
<td>stat.dYReg</td>
<td>stat.YReg</td>
</tr>
</tbody>
</table>
stat.values

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike stat.results, stat.values omits the names associated with the values.

You can copy a value and paste it into other locations.

stDevPop()

\[ \text{stDevPop}(List\{, freqList\}) \Rightarrow \text{expression} \]

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 162.

\[
\begin{align*}
\text{stDevPop}\{\{a,b,c\}\} & = \sqrt{2\cdot(a^2-a\cdot(b+c)+b^2-b\cdot c+c^2)} \\
& = \frac{3}{6}
\end{align*}
\]

\[
\begin{align*}
\text{stDevPop}\{\{1,2,5,6,3,2\}\} & = \sqrt{465} \\
& = \frac{6}{6}
\end{align*}
\]

\[
\begin{align*}
\text{stDevPop}\{\{1,3,2,5,6,4\},\{3,2,5\}\} & = 4.11107
\end{align*}
\]

\[
\begin{align*}
\text{stDevPop}\{\begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 1 \\ 5 & 7 & 3 \end{bmatrix}, \begin{bmatrix} 4 & \sqrt{6} & \sqrt{78} \\ 3 & 3 & 3 \end{bmatrix}\} & = \begin{bmatrix} 2.52608 & 5.21506 \end{bmatrix}
\end{align*}
\]

stDevSamp()

\[ \text{stDevSamp}(List\{, freqList\}) \Rightarrow \text{expression} \]

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 162.

\[
\begin{align*}
\text{stDevSamp}\{\{a,b,c\}\} & = \sqrt{3\cdot(a^2-a\cdot(b+c)+b^2-b\cdot c+c^2)} \\
& = \frac{3}{6}
\end{align*}
\]

\[
\begin{align*}
\text{stDevSamp}\{\{1,2,5,6,3,2\}\} & = \sqrt{62} \\
& = \frac{2}{2}
\end{align*}
\]

\[
\begin{align*}
\text{stDevSamp}\{\{1,3,2,5,6,4\},\{3,2,5\}\} & = 4.33345
\end{align*}
\]
\section*{stDevSamp()}

\textbf{stDevSamp}($\text{Matrix1}$, $\text{freqMatrix}$) $\Rightarrow$ \textit{matrix}

Returns a row vector of the sample standard deviations of the columns in $\text{Matrix1}$.

Each $\text{freqMatrix}$ element counts the number of consecutive occurrences of the corresponding element in $\text{Matrix1}$.

\textbf{Note}: $\text{Matrix1}$ must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 162.

\begin{verbatim}
stDevSamp\left(\begin{bmatrix} 1 & 2 & 5 \\ 3 & 0 & 1 \\ 5 & 7 & 3 \end{bmatrix}, \begin{bmatrix} 3.26599 \\ 2.94392 \\ 1.63299 \end{bmatrix}\right)
\end{verbatim}

\begin{verbatim}
stDevPop\left(\begin{bmatrix} -1.2 & 5.3 \\ 2.5 & 7.3 \\ 6 & -4 \end{bmatrix}, \begin{bmatrix} 4 & 2 \\ 3 & 3 \\ 1 & 7 \end{bmatrix}\right)
\end{verbatim}

\begin{verbatim}
\begin{bmatrix} 2.52608 \\ 5.21506 \end{bmatrix}
\end{verbatim}

\section*{Stop}

\textbf{Stop}

Programming command: Terminates the program.

\textbf{Stop} is not allowed in functions.

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

\begin{verbatim}
\texttt{prog1()} = \texttt{Prgm}
\texttt{Done}
\texttt{For i=0,1,10,1}
\texttt{If i=5}
\texttt{Stop}
\texttt{EndFor}
\texttt{EndPrgm}
\end{verbatim}

\begin{verbatim}
\begin{array}{c}
p\texttt{rog1()}
\texttt{i=5}
\end{array}
\end{verbatim}

\section*{Store}

See $\rightarrow$ (\textbf{store}), page 160.

\section*{string()}

\textbf{string}($\textbf{Expr}$) $\Rightarrow$ \textit{string}

Simplifies $\textbf{Expr}$ and returns the result as a character string.

\begin{verbatim}
string(1.2345) = "1.2345"
string(1+2) = "3"
string(cos(x)+\sqrt{3}) = "cos(x)+\sqrt{3}"
\end{verbatim}

\section*{subMat()}

\textbf{subMat}($\textbf{Matrix1}$, \textit{startRow} $\begin{bmatrix} \text{startCol} \end{bmatrix}$, \textit{endRow} $\begin{bmatrix} \text{endCol} \end{bmatrix}$) $\Rightarrow$ \textit{matrix}

Returns the specified submatrix of $\textbf{Matrix1}$.

Defaults: $\textit{startRow}=1$, $\textit{startCol}=1$, $\textit{endRow}=$last row, $\textit{endCol}=$last column.

\begin{verbatim}
\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow m1
\end{verbatim}

\begin{verbatim}
\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \Rightarrow m1,2,1,3,2
\end{verbatim}

\begin{verbatim}
\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \Rightarrow m1,2,2
\end{verbatim}

\section*{Sum (Sigma)}

See $\Sigma()$, page 153.
sum() \[sum(\text{List}, \text{Start}, \text{End})] \Rightarrow \text{expression}

Returns the sum of all elements in \text{List}. \text{Start} and \text{End} are optional. They specify a range of elements.

Any void argument produces a void result. Empty (void) elements in \text{List} are ignored. For more information on empty elements, see page 162.

\[\begin{align*}
\text{sum}\{1,2,3,4,5\} &\Rightarrow 15 \\
\text{sum}\{a,2\cdot a,3\cdot a\} &\Rightarrow 6\cdot a \\
\text{sum}\{\text{seq}[n,n,1,10]\} &\Rightarrow 55 \\
\text{sum}\{1,3,5,7,9\},3 &\Rightarrow 21 \\
\text{sum}\{1,2,3\},4 &\Rightarrow 5 \quad 7 \quad 9 \\
\text{sum}\{1,2,3\},4 &\Rightarrow 12 \quad 15 \quad 18 \\
\text{sum}\{1,2,3\},4 &\Rightarrow 11 \quad 13 \quad 15 \\
\end{align*}\]

sum() \[\sum(\text{Matrix}, \text{Start}, \text{End})] \Rightarrow \text{matrix}

Returns a row vector containing the sums of all elements in the columns in \text{Matrix}.

\text{Start} and \text{End} are optional. They specify a range of rows.

Any void argument produces a void result. Empty (void) elements in \text{Matrix} are ignored. For more information on empty elements, see page 162.

sumIf() \[\text{sumIf}(\text{List}, \text{Criteria}, \text{SumList})] \Rightarrow \text{value}

Returns the accumulated sum of all elements in \text{List} that meet the specified \text{Criteria}. Optionally, you can specify an alternate list, \text{sumList}, to supply the elements to accumulate.

\text{List} can be an expression, list, or matrix. \text{SumList}, if specified, must have the same dimension(s) as \text{List}.

\text{Criteria} can be:

- A value, expression, or string. For example, 34 accumulates only those elements in \text{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 \text{List} that are less than 10.

When a \text{List} element meets the \text{Criteria}, the element is added to the accumulating sum. If you include \text{sumList}, the corresponding element from \text{sumList} is added to the sum instead.

Within the Lists & Spreadsheet application, you can use a range of cells in place of \text{List} and \text{sumList}.

Empty (void) elements are ignored. For more information on empty elements, see page 162.

Note: See also \text{countIf()}, page 26.

sumSeq() \[\text{See } \Sigma, \text{ page } 153.\]

system() \[\text{system}(\text{Eqn}1[, \text{Eqn}2[, \text{Eqn}3[, ...]])]

Returns a system of equations, formatted as a list. You can also create a system by using a template.

Note: See also \text{System of equations}, page 3.
T (transpose)

Matrix\(^T\) ⇒ matrix

Returns the complex conjugate transpose of Matrix\(_1\).

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

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}^T = \begin{bmatrix}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b \\
c & d \\
\end{bmatrix}^T = \begin{bmatrix}
a & c \\
b & d \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1+i & 2+i \\
3+i & 4+i \\
\end{bmatrix}^T = \begin{bmatrix}
1-i & 3-i \\
2-i & 4-i \\
\end{bmatrix}
\]

\[\tan()\]

\(\tan(Expr1) \Rightarrow \text{expression}\)

\(\tan(List1) \Rightarrow \text{list}\)

\(\tan(Expr1)\) returns the tangent of the argument as an expression.

\(\tan(List1)\) returns a list of the tangents of all elements in \(List1\).

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

\[
\tan\left(\frac{\pi}{4}\right) = 1
\]

\[
\tan(45) = 1
\]

\[
\tan\left\{\{0,60,90\}\right\} = \{0,\sqrt{3},\text{undef}\}
\]

In Degree angle mode:

In Gradian angle mode:

In Radian angle mode:

\[
\tan\left(\frac{\pi}{3}\right)
\]

\[
\tan(\frac{\pi}{3}, \frac{\pi}{4})
\]

\[
\tan\left\{\frac{\pi}{3}, \frac{\pi}{4}\right\} = \{0,\sqrt{3},0,1\}
\]

\[
\tan\left(\frac{\pi}{3}, \frac{\pi}{4}\right)
\]

\[
\tan\left\{\frac{\pi}{3}, \frac{\pi}{4}\right\} = \{0,\sqrt{3},0,1\}
\]

\[\tan(squareMatrix1) \Rightarrow squareMatrix\]

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.

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-28.2912 & 26.0887 & 11.1142 \\
12.1171 & -7.83536 & -5.48138 \\
36.8181 & -32.8063 & -10.4594 \\
\end{bmatrix}
\]
tan⁻¹() \[\text{tan}^{-1}(\text{Expr1}) \Rightarrow \text{expression}\]
\[\text{tan}^{-1}(\text{List1}) \Rightarrow \text{list}\]
\[\text{tan}^{-1}(\text{Expr1})\] returns the angle whose tangent is \(\text{Expr1}\) as an expression.
\[\text{tan}^{-1}(\text{List1})\] returns a list of the inverse tangents of each element of \(\text{List1}\).

**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 \(\text{arctan}(...)\).

\[\text{tan}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\]

Returns the matrix inverse tangent of \(\text{squareMatrix1}\). This is not the same as calculating the inverse tangent of each element. For information about the calculation method, refer to \(\text{cos}()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

\[\text{tangentLine()}\]
\[\text{tangentLine}(\text{Expr1}, \text{Var}, \text{Point}) \Rightarrow \text{expression}\]
\[\text{tangentLine}(\text{Expr1}, \text{Var} = \text{Point}) \Rightarrow \text{expression}\]

Returns the tangent line to the curve represented by \(\text{Expr1}\) at the point specified in \(\text{Var} = \text{Point}\).

Make sure that the independent variable is not defined. For example, if \(f1(x):=5\) and \(x:=3\), then \(\text{tangentLine}(f1(x),x,2)\) returns "false."

\[\text{tanh()}\]
\[\text{tanh}(\text{Expr1}) \Rightarrow \text{expression}\]
\[\text{tanh}(\text{List1}) \Rightarrow \text{list}\]
\[\text{tanh}(\text{Expr1})\] returns the hyperbolic tangent of the argument as an expression.
\[\text{tanh}(\text{List1})\] returns a list of the hyperbolic tangents of each element of \(\text{List1}\).

\[\text{tanh}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}\]

Returns the matrix hyperbolic tangent of \(\text{squareMatrix1}\). This is not the same as calculating the hyperbolic tangent of each element. For information about the calculation method, refer to \(\text{cos}()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.
tanh^(-1)()  

tanh^(-1)(Expr) \Rightarrow expression  
tanh^(-1)(List) \Rightarrow list  

**tanh^(-1)(Expr)** returns the inverse hyperbolic tangent of the argument as an expression.

**tanh^(-1)(List)** returns a list of the inverse hyperbolic tangents of each element of List.

**Note:** You can insert this function from the keyboard by typing \texttt{arctanh(...)}.

**tanh^(-1)(squareMatrix)** \Rightarrow squareMatrix  

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

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

\[  \begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix} \]

\[ \begin{pmatrix} -0.099353 + 0.164058 i \\ 0.267834 - 1.4908 i \\ -0.087596 - 0.725533 i \\ 0.479679 - 0.94730 i \\ 0.511463 - 2.08316 i \\ -0.878563 + 1.7901 i \end{pmatrix} \]

To see the entire result, press \( \downarrow \) and then use \( \leftarrow \) and \( \rightarrow \) to move the cursor.

\[
\text{taylor()}  \\
\text{taylor(Expr, Var, Order, Point)} \Rightarrow expression  
\]

Returns the requested Taylor polynomial. The polynomial includes non-zero terms of integer degrees from zero through Order in (Var minus Point). \texttt{taylor()} returns itself if there is no truncated power series of this order, or if it would require negative or fractional exponents. Use substitution and/or temporary multiplication by a power of (Var minus Point) to determine more general power series.

\textit{Point} defaults to zero and is the expansion point.

As illustrated by the last example to the right, the display routines downstream of the result produced by \texttt{taylor(...)} might rearrange terms so that the dominant term is not the leftmost one.

\[
\text{tcdf()}  \\
\text{tcdf(lowBound, upBound, df)} \Rightarrow \text{number if lowBound and upBound are numbers, list if lowBound and upBound are lists}  
\]

Computes the Student-\( \mathcal{t} \) distribution probability between \texttt{lowBound} and \texttt{upBound} for the specified degrees of freedom \texttt{df}.

For \( P(X \leq \text{upBound}) \), set \texttt{lowBound} = \( -\infty \).
tCollect()  

**tCollect(Expr1)** ⇒ **expression**

Returns an expression in which products and integer powers of sines and cosines are converted to a linear combination of sines and cosines of multiple angles, angle sums, and angle differences. The transformation converts trigonometric polynomials into a linear combination of their harmonics.

Sometimes **tCollect()** will accomplish your goals when the default trigonometric simplification does not. **tCollect()** tends to reverse transformations done by **tExpand()**. Sometimes applying **tExpand()** to a result from **tCollect()**, or vice versa, in two separate steps simplifies an expression.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Simplified Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cos(a)^2 )</td>
<td>( \frac{\cos(2a) + 1}{2} )</td>
</tr>
<tr>
<td>( \sin(a) \cdot \cos(\beta) )</td>
<td>( \frac{\sin(\alpha - \beta) + \sin(\alpha + \beta)}{2} )</td>
</tr>
</tbody>
</table>

**tExpand()**

**tExpand(Expr1)** ⇒ **expression**

Returns an expression in which sines and cosines of integer-multiple angles, angle sums, and angle differences are expanded. Because of the identity \((\sin(x))^2 + (\cos(x))^2 = 1\), there are many possible equivalent results. Consequently, a result might differ from a result shown in other publications.

Sometimes **tExpand()** will accomplish your goals when the default trigonometric simplification does not. **tExpand()** tends to reverse transformations done by **tCollect()**. Sometimes applying **tCollect()** to a result from **tExpand()**, or vice versa, in two separate steps simplifies an expression.

**Note:** Degree-mode scaling by \( \frac{\pi}{180} \) interferes with the ability of **tExpand()** to recognize expandable forms. For best results, **tExpand()** should be used in Radian mode.

**Text**

**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 101, or **RequestStr**, page 102.

**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 \( \text{Alt-Enter} \) instead of \( \cdot \). On the computer keyboard, hold down **Alt** and press **Enter**.

Define **text_demo()**=Prgm

For \( i, 1, 5 \)

```
strinfo:="Random number “ & string(rand(i))
Text strinfo
EndFor
EndPrgm
```

Run the program:

**text_demo()**

Sample of one dialog box:

![Sample dialog box](image)

**Then**

See **If**, page 57.
`tInterval` **Catalog >**

`tInterval` `List, Freq[, CLevel]`

(Data list input)

Computes a $t$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 120.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower, stat.CUpper</code></td>
<td>Confidence interval for an unknown population mean</td>
</tr>
<tr>
<td><code>stat.x</code></td>
<td>Sample mean of the data sequence from the normal random distribution</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>stat.df</code></td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td><code>stat.sx</code></td>
<td>Sample standard deviation</td>
</tr>
<tr>
<td><code>stat.n</code></td>
<td>Length of the data sequence with sample mean</td>
</tr>
</tbody>
</table>

`tInterval_2Samp` **Catalog >**

`tInterval_2Samp` `List1, List2, Freq1[, Freq2[, CLevel[, Pooled]]]`

(Data list input)

Computes a two-sample $t$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 120.)

`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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower, stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>stat.x1-x2</code></td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>stat.df</code></td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td><code>stat.sx1, stat.sx2</code></td>
<td>Sample standard deviations for <code>List 1</code> and <code>List 2</code></td>
</tr>
<tr>
<td><code>stat.n1, stat.n2</code></td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td><code>stat.sp</code></td>
<td>The pooled standard deviation. Calculated when <code>Pooled = YES</code></td>
</tr>
</tbody>
</table>
tmpCnv()  

\[ \text{tmpCnv}(\text{Expr}_1, \text{tempUnit}_1, \text{tempUnit}_2) \Rightarrow \text{expression}_2 \text{tempUnit}_2 \]

Converts a temperature value specified by \( \text{Expr} \) from one unit to another. Valid temperature units are:

- \( ^\circ\text{C} \) Celsius
- \( ^\circ\text{F} \) Fahrenheit
- \( ^\circ\text{K} \) Kelvin
- \( ^\circ\text{R} \) Rankine

To type \( ^\circ \), select it from the Catalog symbols.

To type \( _ \), press [ctrl] [0].

For example, 100 \( ^\circ\text{C} \) converts to 212 \( ^\circ\text{F} \).

To convert a temperature range, use \( \Delta\text{tmpCnv}() \) instead.

\( \Delta\text{tmpCnv}() \)

\[ \Delta\text{tmpCnv}(\text{Expr}_1, \text{tempUnit}_1, \text{tempUnit}_2) \Rightarrow \text{expression}_2 \text{tempUnit}_2 \]

\( \Delta\text{tmpCnv}() \)

\[ \Delta\text{tmpCnv}(\text{Expr}_1, \text{tempUnit}_1, \text{tempUnit}_2) \Rightarrow \text{expression}_2 \text{tempUnit}_2 \]

\( \Delta\text{tmpCnv}(100\_\text{C},\_\text{F}) \) 212\_\text{F}
\( \Delta\text{tmpCnv}(32\_\text{F},\_\text{C}) \) 0\_\text{C}
\( \Delta\text{tmpCnv}(0\_\text{C},\_\text{K}) \) 273.15\_\text{K}
\( \Delta\text{tmpCnv}(0\_\text{F},\_\text{R}) \) 459.67\_\text{R}

Note: You can use the Catalog to select temperature units.

\( \text{tPdf}() \)

\[ \text{tPdf}(\text{XVal}, \text{df}) \Rightarrow \text{number} \]

Computes the probability density function (pdf) for the Student-\( t \) distribution at a specified \( x \) value with specified degrees of freedom \( df \).
trace() \[ \text{Catalog >} \]

\( \text{trace}(\text{squareMatrix}) \Rightarrow \text{expression} \)

Returns the trace (sum of all the elements on the main diagonal) of \( \text{squareMatrix} \).

| \( \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \) | 15 |
| \( \begin{pmatrix} a & 0 \\ 1 & a \end{pmatrix} \) | 2 \cdot a |

Try

Try \( \text{block1} \)
Else \( \text{block2} \)
EndTry

Executes \( \text{block1} \) unless an error occurs. Program execution transfers to \( \text{block2} \) if an error occurs in \( \text{block1} \). System variable \( \text{errCode} \) contains the error code to allow the program to perform error recovery. For a list of error codes, see "Error codes and messages," page 168.

\( \text{block1} \) and \( \text{block2} \) 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 \( \text{\textasciitilde} \) instead of \( \text{\cdot} \) at the end of each line. On the computer keyboard, hold down \( \text{Alt} \) and press \( \text{Enter} \).

Example 2

To see the commands Try, \( \text{ClrErr} \), and \( \text{PassErr} \) in operation, enter the eigenvals() program shown at the right. Run the program by executing each of the following expressions.

\[ \begin{align*}
\text{eigenvals} & = \begin{bmatrix} -3 \\ 41 \\ 5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3.1 \end{bmatrix} \\
\text{eigenvals} & = \begin{bmatrix} 1 & 2 & 3 \\ 1 \end{bmatrix} \\
\end{align*} \]

Note: See also \( \text{ClrErr} \), page 19, and \( \text{PassErr} \), page 88.
**tTest**

**(Data list input)**

$$tTest\,\mu_0, List[, Freq[, Hypoth]]$$

**(Summary stats input)**

Performs a hypothesis test for a single unknown population mean $$\mu$$ when the population standard deviation $$\sigma$$ is unknown. A summary of results is stored in the $stat.results$ variable. (See page 120.)

Test H$_0$: $$\mu = \mu_0$$, against one of the following:

- For Ha: $$\mu < \mu_0$$, set $Hypoth<0$
- For Ha: $$\mu \neq \mu_0$$ (default), set $Hypoth=0$
- For Ha: $$\mu > \mu_0$$, set $Hypoth>0$

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$stat.t$</td>
<td>$$(\bar{X} - \mu_0) / (\text{stddev} / \sqrt{n})$$</td>
</tr>
<tr>
<td>$stat.PVal$</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>$stat.df$</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>$stat.\bar{X}$</td>
<td>Sample mean of the data sequence in $List$</td>
</tr>
<tr>
<td>$stat.sx$</td>
<td>Sample standard deviation of the data sequence</td>
</tr>
<tr>
<td>$stat.n$</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

**tTest_2Samp**

**(Data list input)**

$$tTest_2Samp\, List1, List2[, Freq1[, Freq2[, Hypoth[, Pooled]]]]$$

**(Summary stats input)**

Computes a two-sample t test. A summary of results is stored in the $stat.results$ variable. (See page 120.)

Test H$_0$: $$\mu_1 = \mu_2$$, against one of the following:

- For Ha: $$\mu_1 < \mu_2$$, set $Hypoth<0$
- For Ha: $$\mu_1 \neq \mu_2$$ (default), set $Hypoth=0$
- For Ha: $$\mu_1 > \mu_2$$, 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 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$stat.t$</td>
<td>Standard normal value computed for the difference of means</td>
</tr>
<tr>
<td><strong>Output variable</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom for the t-statistic</td>
</tr>
<tr>
<td>stat.(\bar{x}_1), stat.(\bar{x}_2)</td>
<td>Sample means of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Size of the samples</td>
</tr>
<tr>
<td>stat.sp</td>
<td>The pooled standard deviation. Calculated when Pooled=1.</td>
</tr>
</tbody>
</table>

### tvmFV()

**tvmFV**

\[
tvmFV(N, I, PV, Pmt, [PpY], [CpY], [PmtAt]) \Rightarrow value
\]

Financial function that calculates the future value of money.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 132. See also `amortTbl()`, page 7.

\[
tvmFV(120, 5, 0, -500, 12, 12) \quad 77641.1
\]

### tvmI()

**tvmI**

\[
tvmI(N, PV, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow value
\]

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 132. See also `amortTbl()`, page 7.

\[
tvmI(240, 100000, -1000, 0, 12, 12) \quad 10.5241
\]

### tvmN()

**tvmN**

\[
tvmN(I, PV, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow value
\]

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 132. See also `amortTbl()`, page 7.

\[
tvmN(5, 0, -500, 77641, 12, 12) \quad 120.
\]

### tvmPmt()

**tvmPmt**

\[
tvmPmt(N, I, PV, FV, [PpY], [CpY], [PmtAt]) \Rightarrow value
\]

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 132. See also `amortTbl()`, page 7.

\[
tvmPmt(60, 4, 30000, 0, 12, 12) \quad -552.496
\]

### tvmPV()

**tvmPV**

\[
tvmPV(N, I, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow value
\]

Financial function that calculates the present value.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 132. See also `amortTbl()`, page 7.

\[
tvmPV(48, 4, -500, 30000, 12, 12) \quad -3426.7
\]

<table>
<thead>
<tr>
<th>TVM argument*</th>
<th><strong>Description</strong></th>
<th><strong>Data type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>Number of payment periods</td>
<td>real number</td>
</tr>
<tr>
<td>(I)</td>
<td>Annual interest rate</td>
<td>real number</td>
</tr>
</tbody>
</table>
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

Calculates the TwoVar statistics. A summary of results is stored in the stat.results variable. (See page 120.)

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 162.

Output variable Description
stat.X Mean of x values
stat.Σx Sum of x values
stat.Σx2 Sum of x2 values
stat.sx Sample standard deviation of x
stat.ox Population standard deviation of x
stat.n Number of data points
stat.Y Mean of y values
stat.Σy Sum of y values
stat.Σy2 Sum of y2 values
stat.sy Sample standard deviation of y
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.σy</td>
<td>Population standard deviation of y</td>
</tr>
<tr>
<td>stat.Σxy</td>
<td>Sum of x·y values</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.MinX</td>
<td>Minimum of x values</td>
</tr>
<tr>
<td>stat.Q₁x</td>
<td>1st Quartile of x</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of x</td>
</tr>
<tr>
<td>stat.Q₃x</td>
<td>3rd Quartile of x</td>
</tr>
<tr>
<td>stat.MaxX</td>
<td>Maximum of x values</td>
</tr>
<tr>
<td>stat.MinY</td>
<td>Minimum of y values</td>
</tr>
<tr>
<td>stat.Q₁y</td>
<td>1st Quartile of y</td>
</tr>
<tr>
<td>stat.MedY</td>
<td>Median of y</td>
</tr>
<tr>
<td>stat.Q₃y</td>
<td>3rd Quartile of y</td>
</tr>
<tr>
<td>stat.MaxY</td>
<td>Maximum of y values</td>
</tr>
<tr>
<td>stat.Σ(x-x̄)²</td>
<td>Sum of squares of deviations from the mean of x</td>
</tr>
<tr>
<td>stat.Σ(y-ȳ)²</td>
<td>Sum of squares of deviations from the mean of y</td>
</tr>
</tbody>
</table>

### unitV()

**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.

<table>
<thead>
<tr>
<th><code>unitV([[a b c]])</code></th>
<th><code>unitV([[a b c]])</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{bmatrix} a \ b \ c \end{bmatrix} ]</td>
<td>[ \begin{bmatrix} \sqrt{a^2+b^2+c^2} \ \sqrt{a^2+b^2+c^2} \ \sqrt{a^2+b^2+c^2} \end{bmatrix} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>unitV([[1 2 1]])</code></th>
<th><code>unitV([[1 2 1]])</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{bmatrix} 1 \ 2 \ 3 \end{bmatrix} ]</td>
<td>[ \begin{bmatrix} \sqrt{14} \ 14 \ 14 \end{bmatrix} ]</td>
</tr>
</tbody>
</table>

To see the entire result, press `£` and then use `<` and `>` to move the cursor.
unLock

unLock \( \text{Var1, Var2} \), \( \text{Var3} \), ...

unLock \( \text{Var} \).

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

See Lock, page 70, and getLockInfo(), page 53.

\[ a := 65 \]

\[ \text{Lock } a \] Done

\[ \text{getLockInfo}(a) \]

\[ a := 75 \]

"Error: Variable is locked."

\[ \text{DelVar } a \]

"Error: Variable is locked."

\[ \text{Unlock } a \] Done

\[ a := 75 \]

\[ \text{Done} \]

\[ \text{varPop()} \]

\[ \text{varPop}(\text{List}, \text{freqList}) \Rightarrow \text{expression} \]

Returns the population variance of \( \text{List} \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List} \).

Note: \( \text{List} \) 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 162.

\[ \text{varPop}\{5, 10, 15, 20, 25, 30\} \]

\[ 875 \]

\[ 12 \]

\[ \text{Ans}\cdot1. \]

\[ 72.9167 \]

\[ \text{varSamp()} \]

\[ \text{varSamp}(\text{List}, \text{freqList}) \Rightarrow \text{expression} \]

Returns the sample variance of \( \text{List} \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List} \).

Note: \( \text{List} \) 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 162.

\[ \text{varSamp}\{1, 2, 5, 6, 3, 2\} \]

\[ 31 \]

\[ 2 \]

\[ \text{varSamp}\{1, 3, 5\}, \{4, 6, 2\} \]

\[ 68 \]

\[ 33 \]

\[ \text{varSamp}(\text{Matrix1}, \text{freqMatrix}) \Rightarrow \text{matrix} \]

Returns a row vector containing the sample variance of each column in \( \text{Matrix1} \).

Each \( \text{freqMatrix} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{Matrix1} \).

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 162.

Note: \( \text{Matrix1} \) must contain at least two rows.

\[ \text{varSamp}\{1, 2, 3, 5\} \]

\[ 4.75 \]

\[ 1.03 \]

\[ 4 \]

\[ \begin{bmatrix}
1 & 2 & 5 \\
-3 & 0 & 1 \\
.5 & .7 & 3
\end{bmatrix} \]

\[ \begin{bmatrix}
1.1 & 2.2 & 6 & 3 \\
3.4 & 5.1 & 2 & 4 \\
-2.3 & 4.3 & 5 & 1
\end{bmatrix} \]

\[ 3.91731 \]

\[ 2.08411 \]
**warnCodes()**

**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 174.

**when()**

**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.

**While**

**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.
BooleanExpr1 xor BooleanExpr2 returns Boolean expression
BooleanList1 xor BooleanList2 returns Boolean list
BooleanMatrix1 xor BooleanMatrix2 returns Boolean matrix

Returns true if BooleanExpr1 is true and BooleanExpr2 is false, or vice versa.

Returns false if both arguments are true or if both are false. Returns a simplified Boolean expression if either of the arguments cannot be resolved to true or false.

Note: See or, page 87.

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 14.

Note: See or, page 87.

zeros()

zeros(Expr, Var) => list
zeros(Expr, Var=Guess) => list

Returns a list of candidate real values of Var that make Expr=0.
zeros() does this by computing exp(list(solve(Expr=0, Var), Var).
zeros(
[Expr1, Expr2],
{VarOrGuess1, VarOrGuess2 [ , ... ]}) ⇒ matrix

Returns candidate real zeros of the simultaneous algebraic expressions, where each VarOrGuess specifies an unknown whose value you seek.

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

variable
− or −
variable = real or non-real number

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

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

For example, suppose you have a circle of radius r at the origin and another circle of radius r centered where the first circle crosses the positive x-axis. Use zeros() to find the intersections.

As illustrated by r in the example to the right, simultaneous polynomial expressions can have extra variables that have no values, but represent given numeric values that could be substituted later.

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

Extract row 2:

You can also (or instead) include unknowns that do not appear in the expressions. For example, you can include z as an unknown to extend the previous example to two parallel intersecting cylinders of radius r. The cylinder zeros illustrate how families of zeros might contain arbitrary constants in the form ck, where k is an integer suffix from 1 through 255.

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

If you do not include any guesses and if any expression is non-polynomial in any variable but all expressions are linear in the unknowns, zeros() uses Gaussian elimination to attempt to determine all real zeros.
If a system is neither polynomial in all of its variables nor linear in its unknowns, `zeros()` determines at most one zero using an approximate iterative method. To do so, the number of unknowns must equal the number of expressions, and all other variables in the expressions must simplify to numbers.

Each unknown starts at its guessed value if there is one; otherwise, it starts at 0.0.

Use guesses to seek additional zeros one by one. For convergence, a guess may have to be rather close to a zero.

### `zInterval`

`zInterval σ,List[,Freq[,CLevel]]`

(Data list input)

`zInterval σ,\bar{x},n [,CLevel]`

(Summary stats input)

Computes a \( z \) confidence interval. A summary of results is stored in the `stat.results` variable. (See page 120.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval for an unknown population mean</td>
</tr>
<tr>
<td><code>stat.\bar{x}</code></td>
<td>Sample mean of the data sequence from the normal random distribution</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>stat.sx</code></td>
<td>Sample standard deviation</td>
</tr>
<tr>
<td><code>stat.n</code></td>
<td>Length of the data sequence with sample mean</td>
</tr>
<tr>
<td><code>stat.\sigma</code></td>
<td>Known population standard deviation for data sequence <code>List</code></td>
</tr>
</tbody>
</table>

### `zInterval_1Prop`

`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 120.)

\( x \) is a non-negative integer.

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>stat.\hat{p}</code></td>
<td>The calculated proportion of successes</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
</tbody>
</table>
Computes a two-proportion z confidence interval. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

\( x_1 \) and \( x_2 \) are non-negative integers.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.n} )</td>
<td>Number of samples in data sequence</td>
</tr>
</tbody>
</table>

### zInterval_2Prop

\[ \text{zInterval}_2\text{Prop } x_1, n_1, x_2, n_2[, \text{CLevel}] \]

Computes a two-proportion z confidence interval. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

\( x_1 \) and \( x_2 \) are non-negative integers.

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.CLower}, \text{stat.CUpper} )</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>( \hat{\theta} )</td>
<td>The calculated difference between proportions</td>
</tr>
<tr>
<td>( \text{stat.ME} )</td>
<td>Margin of error</td>
</tr>
<tr>
<td>( \hat{\theta}_1 )</td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td>( \hat{\theta}_2 )</td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td>( n_1 )</td>
<td>Sample size in data sequence one</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>Sample size in data sequence two</td>
</tr>
</tbody>
</table>

### zInterval_2Samp

\[ \text{zInterval}_2\text{Samp } \sigma_1, \sigma_2, \text{List1}, \text{List2}, \text{Freq1}, \text{Freq2}, [\text{CLevel}] \]

(Data list input)

\[ \text{zInterval}_2\text{Samp } \sigma_1, \sigma_2, \bar{x}_1, n_1, \bar{x}_2, n_2, [\text{CLevel}] \]

(Summary stats input)

Computes a two-sample z confidence interval. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.CLower}, \text{stat.CUpper} )</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>( \bar{x}_1 - \bar{x}_2 )</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>( \text{stat.ME} )</td>
<td>Margin of error</td>
</tr>
<tr>
<td>( \bar{x}_1, \bar{x}_2 )</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>( \sigma_1, \sigma_2 )</td>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>( n_1, n_2 )</td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>( \sigma_1, \sigma_2 )</td>
<td>Known population standard deviations for data sequence List 1 and List 2</td>
</tr>
</tbody>
</table>
zTest

zTest \( \mu, \sigma, \text{List} \{ \text{Freq}, \text{Hypoth} \} \)

(Data list input)

zTest \( \mu, \sigma, \bar{x}, n \{ \text{Hypoth} \} \)

(Summary stats input)

Performs a \( z \) test with frequency \( \text{FreqList} \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

Test \( H_0: \mu = \mu_0 \), against one of the following:

For \( H_a: \mu < \mu_0 \), set \( \text{Hypoth} < 0 \)

For \( H_a: \mu \neq \mu_0 \) (default), set \( \text{Hypoth} = 0 \)

For \( H_a: \mu > \mu_0 \), set \( \text{Hypoth} > 0 \)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.z} )</td>
<td>( (\bar{x} - \mu_0) / (\sigma / \sqrt{n}) )</td>
</tr>
<tr>
<td>( \text{stat.PVal} )</td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>( \text{stat.\bar{x}} )</td>
<td>Sample mean of the data sequence in ( \text{List} )</td>
</tr>
<tr>
<td>( \text{stat.sx} )</td>
<td>Sample standard deviation of the data sequence. Only returned for ( \text{Data} ) input.</td>
</tr>
<tr>
<td>( \text{stat.n} )</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

zTest_1Prop

zTest_1Prop \( p_0, x, n \{ \text{Hypoth} \} \)

Computes a one-proportion \( z \) test. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 120.)

\( x \) is a non-negative integer.

Test \( H_0: p = p_0 \) against one of the following:

For \( H_a: p > p_0 \), set \( \text{Hypoth} > 0 \)

For \( H_a: p \neq p_0 \) (default), set \( \text{Hypoth} = 0 \)

For \( H_a: p < p_0 \), set \( \text{Hypoth} < 0 \)

For information on the effect of empty elements in a list, see "Empty (Void) Elements" on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.p0} )</td>
<td>Hypothesized population proportion</td>
</tr>
<tr>
<td>( \text{stat.z} )</td>
<td>Standard normal value computed for the proportion</td>
</tr>
<tr>
<td>( \text{stat.PVal} )</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>( \hat{p} )</td>
<td>Estimated sample proportion</td>
</tr>
<tr>
<td>( \text{stat.n} )</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>
zTest_2Prop

Computes a two-proportion $z$ test. A summary of results is stored in the `stat.results` variable. (See page 120.)

$x_1$ and $x_2$ are non-negative integers.

Test $H_0: p_1 = p_2$, against one of the following:

For $H_a: p_1 > p_2$, set `Hypoth` > 0
For $H_a: p_1 \neq p_2$ (default), set `Hypoth` = 0
For $H_a: p_1 < p_2$, set `Hypoth` < 0

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.z</code></td>
<td>Standard normal value computed for the difference of proportions</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td><code>stat.p1</code></td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.p2</code></td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.p</code></td>
<td>Pooled sample proportion estimate</td>
</tr>
<tr>
<td><code>stat.n1, stat.n2</code></td>
<td>Number of samples taken in trials 1 and 2</td>
</tr>
</tbody>
</table>

zTest_2Samp

(Data list input)

(zTest_2Samp $\sigma_1, \sigma_2, List1, List2, [Freq1, Freq2], [Hypoth])$

(Summary stats input)

Computes a two-sample $z$ test. A summary of results is stored in the `stat.results` variable. (See page 120.)

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements” on page 162.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.z</code></td>
<td>Standard normal value computed for the difference of means</td>
</tr>
<tr>
<td><code>stat.PVal</code></td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td><code>stat.x1, stat.x2</code></td>
<td>Sample means of the data sequences in <code>List1</code> and <code>List2</code></td>
</tr>
<tr>
<td><code>stat.sx1, stat.sx2</code></td>
<td>Sample standard deviations of the data sequences in <code>List1</code> and <code>List2</code></td>
</tr>
<tr>
<td><code>stat.n1, stat.n2</code></td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>
### Symbols

#### + (add)

<table>
<thead>
<tr>
<th>Expr1 + Expr2 ⇒ expression</th>
<th>+ key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the sum of the two arguments.</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>56+4</td>
<td>60</td>
</tr>
<tr>
<td>60+4</td>
<td>64</td>
</tr>
<tr>
<td>64+4</td>
<td>68</td>
</tr>
<tr>
<td>68+4</td>
<td>72</td>
</tr>
</tbody>
</table>

List1 + List2 ⇒ list
Matrix1 + Matrix2 ⇒ matrix

Returns a list (or matrix) containing the sums of corresponding elements in List1 and List2 (or Matrix1 and Matrix2).

Dimensions of the arguments must be equal.

### Note:

Use .+ (dot plus) to add an expression to each element.

#### - (subtract)

<table>
<thead>
<tr>
<th>Expr1 - Expr2 ⇒ expression</th>
<th>- key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns Expr1 minus Expr2.</td>
<td></td>
</tr>
<tr>
<td>6-2</td>
<td>4</td>
</tr>
<tr>
<td>$\pi$ - $\pi$</td>
<td>5-$\pi$</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

List1 - List2 ⇒ list
Matrix1 - Matrix2 ⇒ matrix

Subtracts each element in List2 (or Matrix2) from the corresponding element in List1 (or Matrix1), and returns the results.

Dimensions of the arguments must be equal.

### Note:

Subtracts each List1 element from Expr or subtracts Expr from each List1 element, and returns a list of the results.
- **(subtract)**

\[ \text{Expr} - \text{Matrix1} \Rightarrow \text{matrix} \]
\[ \text{Matrix1} - \text{Expr} \Rightarrow \text{matrix} \]

\[ \text{Expr} - \text{Matrix1} \text{ returns a matrix of Expr times the identity matrix minus Matrix1. Matrix1 must be square.} \]
\[ \text{Matrix1} - \text{Expr} \text{ returns a matrix of Expr times the identity matrix subtracted from Matrix1. Matrix1 must be square.} \]

**Note:** Use \( - \) (dot minus) to subtract an expression from each element.

- **(multiply)**

\[ \text{Expr1} \cdot \text{Expr2} \Rightarrow \text{expression} \]

Returns the product of the two arguments.

\[ 2 \cdot 3,45 \]
\[ x \cdot y \cdot x \]

\[ \{1,2,3\} \cdot \{4,5,6\} \]
\[ \{2,3\} \cdot \{a,2\} \]

\[ \text{List1} \cdot \text{List2} \Rightarrow \text{list} \]

Returns a list containing the products of the corresponding elements in List1 and List2.

**Note:** Use \( \cdot \) (dot multiply) to multiply an expression by each element.

\[ \text{Matrix1} \cdot \text{Matrix2} \Rightarrow \text{matrix} \]

Returns the matrix product of Matrix1 and Matrix2.

The number of columns in Matrix1 must equal the number of rows in Matrix2.

\[ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \]
\[ = \begin{bmatrix} a+2b+3c \\ 4a+5b+6c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} \]
\[ = \begin{bmatrix} d+2e+3f \\ 4d+5e+6f \end{bmatrix} \]

\[ \pi \cdot \{4,5,6\} \]
\[ = \{4\pi,5\pi,6\pi\} \]

\[ \text{Expr} \cdot \text{List1} \Rightarrow \text{list} \]
\[ \text{List1} \cdot \text{Expr} \Rightarrow \text{list} \]

Returns a list containing the products of Expr and each element in List1.

\[ \text{Expr} \cdot \text{Matrix1} \Rightarrow \text{matrix} \]
\[ \text{Matrix1} \cdot \text{Expr} \Rightarrow \text{matrix} \]

Returns a matrix containing the products of Expr and each element in Matrix1.

**Note:** Use \( \cdot \) (dot multiply) to multiply an expression by each element.

- **(divide)**

\[ \text{Expr1} / \text{Expr2} \Rightarrow \text{expression} \]

Returns the quotient of Expr1 divided by Expr2.

**Note:** See also Fraction template, page 1.
/ (divide)

List1 / List2 ⇒ list

Returns a list containing the quotients of List1 divided by List2.

Dimensions of the lists must be equal.

Expr / List1 ⇒ list
List1 / Expr ⇒ list

Returns a list containing the quotients of Expr divided by List1 or List1 divided by Expr.

Matrix1 / Expr ⇒ matrix

Returns a matrix containing the quotients of Matrix1 / Expr.

Note: Use . ÷ (dot divide) to divide an expression by each element.

^ (power)

Expr1 * Expr2 ⇒ expression
List1 ^ List2 ⇒ list

Returns the first argument raised to the power of the second argument.

Note: See also Exponent template, page 1.

For a list, returns the elements in List1 raised to the power of the corresponding elements in List2.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

Expr ^ List1 ⇒ list

Returns Expr raised to the power of the elements in List1.

List1 ^ Expr ⇒ list

Returns the elements in List1 raised to the power of Expr.

squareMatrix1 ^ integer ⇒ 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.
\[ x^2 \text{ (square)} \]

Returns the square of the argument.

\[ \text{List}^2 \Rightarrow \text{list} \]

Returns a list containing the squares of the elements in \( \text{List}1 \).

\[ \text{squareMatrix}^2 \Rightarrow \text{matrix} \]

Returns the matrix square of \( \text{squareMatrix}1 \). This is not the same as calculating the square of each element. Use \(^.\text{^2}\) to calculate the square of each element.

\[ \cdot^+ \text{(dot add)} \]

\[ \text{Matrix1} \cdot^+ \text{Matrix2} \Rightarrow \text{matrix} \]

\[ \text{Expr} \cdot^+ \text{Matrix1} \Rightarrow \text{matrix} \]

\( \text{Matrix1} \cdot^+ \text{Matrix2} \) returns a matrix that is the sum of each pair of corresponding elements in \( \text{Matrix1} \) and \( \text{Matrix2} \).

\( \text{Expr} \cdot^+ \text{Matrix1} \) returns a matrix that is the sum of \( \text{Expr} \) and each element in \( \text{Matrix1} \).

\[ \cdot^- \text{(dot subt.)} \]

\[ \text{Matrix1} \cdot^- \text{Matrix2} \Rightarrow \text{matrix} \]

\[ \text{Expr} \cdot^- \text{Matrix1} \Rightarrow \text{matrix} \]

\( \text{Matrix1} \cdot^- \text{Matrix2} \) returns a matrix that is the difference between each pair of corresponding elements in \( \text{Matrix1} \) and \( \text{Matrix2} \).

\( \text{Expr} \cdot^- \text{Matrix1} \) returns a matrix that is the difference of \( \text{Expr} \) and each element in \( \text{Matrix1} \).

\[ .\cdot \text{(dot mult.)} \]

\[ \text{Matrix1} .\cdot \text{Matrix2} \Rightarrow \text{matrix} \]

\[ \text{Expr} .\cdot \text{Matrix1} \Rightarrow \text{matrix} \]

\( \text{Matrix1} .\cdot \text{Matrix2} \) returns a matrix that is the product of each pair of corresponding elements in \( \text{Matrix1} \) and \( \text{Matrix2} \).

\( \text{Expr} .\cdot \text{Matrix1} \) returns a matrix containing the products of \( \text{Expr} \) and each element in \( \text{Matrix1} \).
/. (dot divide)

Matrix1 . J Matrix2 ⇒ matrix
Expr . J Matrix1 ⇒ matrix

Matrix1 . J Matrix2 returns a matrix that is the quotient of each pair of corresponding elements in Matrix1 and Matrix2.
Expr . J Matrix1 returns a matrix that is the quotient of Expr and each element in Matrix1.

\[
\begin{bmatrix}
a & 2 \\
b & 3
\end{bmatrix}
\div
\begin{bmatrix}
c & 4 \\
d & 5
\end{bmatrix}
= \begin{bmatrix}
a & \frac{1}{2} \\
b & \frac{3}{5}
\end{bmatrix}
\]

\[
x \div \begin{bmatrix}
c & 4 \\
d & 5
\end{bmatrix} = \begin{bmatrix}
x \div c \\
x \div d
\end{bmatrix}
\]

^ (dot power)

Matrix1 ^ Matrix2 ⇒ matrix
Expr ^ Matrix1 ⇒ matrix

Matrix1 ^ Matrix2 returns a matrix where each element in Matrix2 is the exponent for the corresponding element in Matrix1.
Expr ^ Matrix1 returns a matrix where each element in Matrix1 is the exponent for Expr.

\[
\begin{bmatrix}
a & 2 \\
b & 3
\end{bmatrix}
\div
\begin{bmatrix}
c & 4 \\
d & 5
\end{bmatrix}
= \begin{bmatrix}
a^c & 16 \\
b^5 & 3^d
\end{bmatrix}
\]

\[
x ^ \begin{bmatrix}
c & 4 \\
d & 5
\end{bmatrix} = \begin{bmatrix}
x^c & x^4 \\
x^5 & x^d
\end{bmatrix}
\]

¬ (negate)

¬ Expr1 ⇒ expression
¬ List1 ⇒ list
¬ Matrix1 ⇒ matrix

Returns the negation of the argument.
For a list or matrix, returns all the elements negated.
If the argument is a binary or hexadecimal integer, the negation gives the two’s complement.

-2.43

\{-1,0,4,1.2e19\} → \{0b1,-0.4,1.2e19\}

\[-a \cdot b\]

In Bin base mode:

Important: Zero, not the letter O

0b10010111→Dec → 37
-0b100101
-0b111111111111111111111111111111
Ans→Dec → -37

To see the entire result, press ▲, and then use ◊ and ▼ to move the cursor.

% (percent)

Expr1 % ⇒ expression
List1 % ⇒ list
Matrix1 % ⇒ matrix

Returns argument
100

For a list or matrix, returns a list or matrix with each element divided by 100.

13% → 0.13

\{\{1,10,100\}\} % → \{0.01,0.1,1.\}

Press Ctrl+Enter (Macintosh®: ⌘+Enter) to evaluate:
= (equal)

expr1 = expr2 \rightarrow Boolean expression
list1 = list2 \rightarrow Boolean list
matrix1 = matrix2 \rightarrow Boolean matrix

Returns true if expr1 is determined to be equal to expr2.
Returns false if expr1 is determined to not be equal to expr2.
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

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

Example function that uses math test symbols: =, ≠, <, ≤, >, ≥

Define \( g(x) = \text{Func} \)
If \( x \leq -5 \) Then
Return 5
ElseIf \( x > -5 \) and \( x < 0 \) Then
Return \( -x \)
ElseIf \( x \geq 0 \) and \( x \neq 10 \) Then
Return \( x \)
ElseIf \( x = 10 \) Then
Return 3
EndIf
EndFunc

Done

\[ f_2(x) = g(x) \]

≠ (not equal)

expr1 ≠ expr2 \rightarrow Boolean expression
list1 ≠ list2 \rightarrow Boolean list
matrix1 ≠ matrix2 \rightarrow Boolean matrix

Returns true if expr1 is determined to be not equal to expr2.
Returns false if expr1 is determined to be equal to expr2.
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing /=

< (less than)

expr1 < expr2 \rightarrow Boolean expression
list1 < list2 \rightarrow Boolean list
matrix1 < matrix2 \rightarrow Boolean matrix

Returns true if expr1 is determined to be less than expr2.
Returns false if expr1 is determined to be greater than or equal to expr2.
Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.
(less or equal)

\[ \text{Expr1} \leq \text{Expr2} \Rightarrow \text{Boolean expression} \]

List1 \leq \text{List2} \Rightarrow \text{Boolean list}

Matrix1 \leq \text{Matrix2} \Rightarrow \text{Boolean matrix}

Returns true if \text{Expr1} is determined to be less than or equal to \text{Expr2}.

Returns false if \text{Expr1} is determined to be greater than \text{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 \(\leq\).

(greater than)

\[ \text{Expr1} > \text{Expr2} \Rightarrow \text{Boolean expression} \]

List1 > List2 \Rightarrow \text{Boolean list}

Matrix1 > Matrix2 \Rightarrow \text{Boolean matrix}

Returns true if \text{Expr1} is determined to be greater than \text{Expr2}.

Returns false if \text{Expr1} is determined to be less than or equal to \text{Expr2}.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

(greater or equal)

\[ \text{Expr1} \geq \text{Expr2} \Rightarrow \text{Boolean expression} \]

List1 \geq \text{List2} \Rightarrow \text{Boolean list}

Matrix1 \geq \text{Matrix2} \Rightarrow \text{Boolean matrix}

Returns true if \text{Expr1} is determined to be greater than or equal to \text{Expr2}.

Returns false if \text{Expr1} is determined to be less than \text{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 \(\geq\).

(logical implication)

\[ \text{BooleanExpr1} \Rightarrow \text{BooleanExpr2} \Rightarrow \text{Boolean expression} \]

\[ \text{BooleanList1} \Rightarrow \text{BooleanList2} \Rightarrow \text{Boolean list} \]

\[ \text{BooleanMatrix1} \Rightarrow \text{BooleanMatrix2} \Rightarrow \text{Boolean matrix} \]

\[ \text{Integer1} \Rightarrow \text{Integer2} \Rightarrow \text{Integer} \]

Evaluates the expression \text{not} <\text{argument1}> \text{or} <\text{argument2}> and returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing \(\Rightarrow\).
\( \Leftrightarrow \) (logical double implication, XNOR)

BooleanExpr1 \( \Leftrightarrow \) BooleanExpr2 returns Boolean expression
BooleanList1 \( \Leftrightarrow \) BooleanList2 returns Boolean list
BooleanMatrix1 \( \Leftrightarrow \) BooleanMatrix2 returns Boolean matrix
Integer1 \( \Leftrightarrow \) Integer2 returns Integer

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

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing \( \Leftrightarrow \).

\[ 5 > 3 \text{ xor } 3 > 5 \]
\[ 5 > 3 \Leftrightarrow 3 > 5 \]
\[ 3 \text{ xor } 4 \]
\[ 3 \Leftrightarrow 4 \]
\[ \{1,2,3\} \text{ xor } \{3,2,1\} \]
\[ \{2,0,2\} \]
\[ \{1,2,3\} \Leftrightarrow \{3,2,1\} \]
\[ \{-3,-1,-3\} \]

! (factorial)

Expr1 \( ! \) \Rightarrow expression
List1 \( ! \) \Rightarrow list
Matrix1 \( ! \) \Rightarrow matrix

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

String1 & String2 \Rightarrow string

Returns a text string that is String2 appended to String1.

\& (append)

String1 \& String2 \Rightarrow string

Returns a text string that is String2 appended to String1.

\( d() \) (derivative)

\[ \text{d(Expr1, Var1, Order)} \Rightarrow \text{expression} \]
\[ \text{d(List1, Var1, Order)} \Rightarrow \text{list} \]
\[ \text{d(Matrix1, Var1, Order)} \Rightarrow \text{matrix} \]

Returns the first derivative of the first argument with respect to variable Var.

Order, if included, must be an integer. If the order is less than zero, the result will be an anti-derivative.

Note: You can insert this function from the keyboard by typing \( \text{d(...)} \).

\( d() \) does not follow the normal evaluation mechanism of fully simplifying its arguments and then applying the function definition to these fully simplified arguments. Instead, \( d() \) performs the following steps:

1. Simplify the second argument only to the extent that it does not lead to a non-variable.
2. Simplify the first argument only to the extent that it does recall any stored value for the variable determined by step 1.
3. Determine the symbolic derivative of the result of step 2 with respect to the variable from step 1.

If the variable from step 1 has a stored value or a value specified by the constraint ("|") operator, substitute that value into the result from step 3.

Note: See also First derivative, page 5; Second derivative, page 5; or Nth derivative, page 5.
\( \int (\text{integral}) \)

\[ \int (\text{Expr1, Var, Lower, Upper}) \Rightarrow \text{expression} \]

\[ \int (\text{Expr1, Var, Constant}) \Rightarrow \text{expression} \]

Returns the integral of \( \text{Expr1} \) with respect to the variable \( \text{Var} \) from \( \text{Lower} \) to \( \text{Upper} \).

**Note:** See also **Definite** or **Indefinite integral template**, page 5.

**Note:** You can insert this function from the keyboard by typing \( \text{integral(} \ldots \text{)} \).

If \( \text{Lower} \) and \( \text{Upper} \) are omitted, returns an anti-derivative. A symbolic constant of integration is omitted unless you provide the \( \text{Constant} \) argument.

\[ \int x^2 \, dx \]

\[ \frac{x^3}{3} \]

\[ \int (a \cdot x^2, x, c) \]

\[ a \cdot \frac{x^3}{3} + c \]

Equally valid anti-derivatives might differ by a numeric constant. Such a constant might be disguised—particularly when an anti-derivative contains logarithms or inverse trigonometric functions. Moreover, piecewise constant expressions are sometimes added to make an anti-derivative valid over a larger interval than the usual formula.

\( \int \) returns itself for pieces of \( \text{Expr1} \) that it cannot determine as an explicit finite combination of its built-in functions and operators.

When you provide \( \text{Lower} \) and \( \text{Upper} \), an attempt is made to locate any discontinuities or discontinuous derivatives in the interval \( \text{Lower} < \text{Var} < \text{Upper} \) and to subdivide the interval at those places.

For the Auto setting of the **Auto or Approximate** mode, numerical integration is used where applicable when an anti-derivative or a limit cannot be determined.

For the Approximate setting, numerical integration is tried first, if applicable. Anti-derivatives are sought only where such numerical integration is inapplicable or fails.

\[ \int b \cdot e^{-x^2} + \frac{a}{x^2 + a^2} \, dx \]

\[ b \cdot \int e^{-x^2} \, dx + \tan^{-1} \left( \frac{x}{a} \right) \]

Press \( \text{Ctrl} + \text{Enter} \) (Macintosh®: \( \text{Alt} + \text{Enter} \)) to evaluate:

\[ \int 1 \cdot e^{x^2} \, dx \]

\[ \int 1.49365 \]

\[ \int x \cdot \ln(x+y) \, dy \, dx \]

\[ \int a^2 \cdot \ln(a) + a^2 \cdot \left( \ln(2) \cdot \frac{3}{4} \right) \]

\[ \frac{a^2 \cdot \ln(a)}{2} + a^2 \cdot \left( \ln(2) \cdot \frac{3}{4} \right) \]

**Note:** See also **nInt()**, page 82.
The product formulas used are derived from the following reference:

\[ \Sigma(\text{Expr}, \text{Var}, \text{Low}, \text{High}) \Rightarrow \text{expression} \]

**Note:** You can insert this function from the keyboard by typing `\text{sumSeq}(\ldots)`.

Evaluates `\text{Expr}` for each value of `\text{Var}` from `\text{Low}` to `\text{High}`, and returns the sum of the results.

**Note:** See also **Sum template**, page 4.

\[ \Sigma(\text{Expr}, \text{Var}, \text{Low}, \text{Low}-1) \Rightarrow 0 \]
\[ \Sigma(\text{Expr}, \text{Var}, \text{Low}, \text{High}) \]
\[ \Rightarrow -\Sigma(\text{Expr}, \text{Var}, \text{High}+1, \text{Low}-1) \text{ if } \text{High} < \text{Low}-1 \]

The summation formulas used are derived from the following reference:

\[ \text{Int}(NPmt1, NPmt2, N, I, PV, [Pmt], [FY], [PpY], [CpY], [PmtAt], \text{[roundValue]}) \Rightarrow \text{value} \]

Amortization function that calculates the sum of the interest during a specified range of payments.

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

\text{roundValue} specifies the number of decimal places for rounding.
Default=2.

\[ \text{Int}(NPmt1, NPmt2, \text{amortTable}) \Rightarrow \text{value} \]

\[ \text{Int}(1,3,12,4,75,20000.,12,12) = 213.48 \]

\[ tbl = \text{amortTbl}(12,12,4,75,20000.,12,12) \]

\begin{array}{cccc}
0 & 0 & 0 & 20000. \\
1 & -77.49 & -1632.43 & 18367.6 \\
2 & -71.17 & -1638.75 & 16728.8 \\
3 & -64.82 & -1645.1 & 15083.7 \\
4 & -58.44 & -1651.48 & 13432.2 \\
5 & -52.05 & -1657.87 & 11774.4 \\
6 & -45.62 & -1664.3 & 10110.1 \\
7 & -39.17 & -1670.75 & 8439.32 \\
8 & -32.7 & -1677.22 & 6762.1 \\
9 & -26.2 & -1683.72 & 5078.38 \\
10 & -19.68 & -1690.24 & 3388.14 \\
11 & -13.13 & -1696.79 & 1691.35 \\
12 & -6.55 & -1703.37 & -12.02 \\
\end{array}

Note: See also \( \text{Prn}() \), below, and \( \text{Bal}() \), page 13.

\[ \text{Prn}(NPmt1, NPmt2, N, I, PV, [Pmt], [FY], [PpY], [CpY], [PmtAt], \text{[roundValue]}) \Rightarrow \text{value} \]

Amortization function that calculates the sum of the principal during a specified range of payments.

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

\[ \text{roundValue} \] specifies the number of decimal places for rounding.
Default=2.

\[ \text{Prn}(NPmt1, NPmt2, \text{amortTable}) \Rightarrow \text{value} \]

\[ \text{Prn}(1,3,12,4,75,20000.,12,12) = -4916.28 \]

\[ tbl = \text{amortTbl}(12,12,4,75,20000.,12,12) \]

\begin{array}{cccc}
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 \\
\end{array}

Note: See also \( \text{Int}() \), above, and \( \text{Bal}() \), page 13.
### # (indirection)

**# varNameString**

Refers to the variable whose name is varNameString. This lets you use strings to create variable names from within a function.

```
#("x" & "y" & "z")               xyz
```

Creates or refers to the variable xyz.

```
10→r
"r"→sl
"f"

#sl
```

Returns the value of the variable (r) whose name is stored in variable s1.

### E (scientific notation)

**mantissaEexponent**

Enters a number in scientific notation. The number is interpreted as mantissa \( \times 10^{\text{exponent}} \).

**Hint:** If you want to enter a power of 10 without causing a decimal value result, use \( 10^\text{integer} \).

**Note:** You can insert this operator from the computer keyboard by typing @E. For example, type \( 2.3@E4 \) to enter \( 2.3 \times 10^4 \).

<table>
<thead>
<tr>
<th>mantissa</th>
<th>exponent</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>23000.</td>
<td>23000.</td>
<td>EE</td>
</tr>
<tr>
<td>2300000000. +4.1E15</td>
<td>4.1E15</td>
<td></td>
</tr>
<tr>
<td>3-10^4</td>
<td>30000</td>
<td></td>
</tr>
</tbody>
</table>

### g (gradian)

**Expr1g** ⇒ **expression**

**List1g** ⇒ **list**

**Matrix1g** ⇒ **matrix**

This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian mode, multiplies \( \text{Expr1} \) by \( \pi/200 \).

In Degree mode, multiplies \( \text{Expr1} \) by \( g/100 \).

In Gradian mode, returns \( \text{Expr1} \) unchanged.

**Note:** You can insert this symbol from the computer keyboard by typing @g.

<table>
<thead>
<tr>
<th>Expr1g</th>
<th>List1g</th>
<th>Matrix1g</th>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cos{50g}</td>
<td>( \frac{\sqrt{2}}{2} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cos{0,100g,200g}</td>
<td>( {1.0, -1 } )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### r (radian)

**Expr1r** ⇒ **expression**

**List1r** ⇒ **list**

**Matrix1r** ⇒ **matrix**

This function gives you a way to specify a radian angle while in the Degree or Gradian mode.

In Degree mode, multiplies the argument by \( 180/\pi \).

In Radian mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by \( 200/\pi \).

**Hint:** Use \( r \) if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

**Note:** You can insert this symbol from the computer keyboard by typing @r.
(degree)

$\text{Expr1}^{\circ} \Rightarrow \text{expression}$

$\text{List1}^{\circ} \Rightarrow \text{list}$

$\text{Matrix1}^{\circ} \Rightarrow \text{matrix}$

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.

(\degree/minute/second)

$dd^\circ mm^\prime ss^\prime\prime \Rightarrow \text{expression}$

$dd$ 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 (‘).
\( \angle (\text{angle}) \)

(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.

In Radian angle mode and Rectangular complex format:

\[
5 + 3 \cdot i \left(10 \angle \frac{\pi}{4}\right) \quad 5 \cdot 5 \cdot 2 + (3 \cdot 5 \cdot 2) \cdot i
\]

Press Ctrl\-Enter \( \text{ctrl enter} \) (Macintosh\®: \( \text{\textasciitilde + enter} \)) to evaluate:

\[
5 + 3 \cdot i \left(10 \angle \frac{\pi}{4}\right) \quad -2.07107407107407107 \cdot i
\]

\( \prime \) (prime)

variable \( \prime \)

variable \( \prime \prime \)

Enters a prime symbol in a differential equation. A single prime symbol denotes a 1st-order differential equation, two prime symbols denote a 2nd-order, and so on.

\[
\text{deSolve} y'' = y^2 \quad \text{and} \quad y(0) = 0 \quad \text{and} \quad y'(0) = 0, t, y
\]

\[
\frac{3}{2 \cdot y^4} \quad \frac{3}{3} = t
\]

\( _\) (underscore as an empty element)

See “Empty (Void) Elements”, page 162.

\( _\) (underscore as a unit designator)

Expr\_Unit

Designates the units for an Expr. All unit names must begin with an underscore.

You can use pre-defined units or create your own units. For a list of pre-defined units, open the Catalog and display the Unit Conversions tab. You can select unit names from the Catalog or type the unit names directly.

Variable\_

When Variable has no value, it is treated as though it represents a complex number. By default, without the \( _\), the variable is treated as real.

If Variable has a value, the \( _\) is ignored and Variable retains its original data type.

\textbf{Note:} You can store a complex number to a variable without using \texttt{\_}. However, for best results in calculations such as \texttt{cSolve()} and \texttt{cZeros()}, the \texttt{\_} is recommended.
(convert)

Expr._Unit1 \( \rightarrow \)_Unit2 \( \Rightarrow \) Expr._Unit2

Converting an expression from one unit to another.

The \( _\cdot \) underscore character designates the units. The units must be in the same category, such as Length or Area.

For a list of pre-defined units, open the Catalog and display the Unit Conversions tab:

- You can select a unit name from the list.
- You can select the conversion operator, \( \rightarrow \), from the top of the list.

You can also type unit names manually. To type \( _\cdot \) " when typing unit names on the handheld, press \( \text{ctrl} \) \( \rightarrow \).

Note: To convert temperature units, use tmpCnv() and \( \Delta \text{tmpCnv}() \). The \( \rightarrow \) conversion operator does not handle temperature units.

10^()

\[ 10^\{(\text{Expr1})\} \Rightarrow \text{expression} \]
\[ 10^\{(\text{List1})\} \Rightarrow \text{list} \]

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in List1.

\[ 10^\{(\text{squareMatrix1})\} \Rightarrow \text{squareMatrix} \]

Returns 10 raised to the power of squareMatrix1. This is not the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to cos().

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

\( ^{-1} \) (reciprocal)

\[ \text{Expr1} \rightarrow \text{expression} \]
\[ \text{List1} \rightarrow \text{list} \]

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in List1.

\[ \text{squareMatrix1} \rightarrow \text{squareMatrix} \]

Returns the inverse of squareMatrix1.

squareMatrix1 must be a non-singular square matrix.
### (constraint operator)

The constraint ("|") symbol serves as a binary operator. The operand to the left of | is an expression. The operand to the right of | specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after | must be joined by logical "and" or "or" operators.

The constraint operator provides three basic types of functionality:

- **Substitutions**
- **Interval constraints**
- **Exclusions**

Substitutions are in the form of an equality, such as \( x=3 \) or \( y=\sin(x) \). To be most effective, the left side should be a simple variable. \( \text{Expr} \mid \text{Variable} = \text{value} \) will substitute value for every occurrence of Variable in Expr.

Interval constraints take the form of one or more inequalities joined by logical "and" or "or" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

Exclusions use the "not equals" (\( /= \) or \( \neq \)) relational operator to exclude a specific value from consideration. They are used primarily to exclude an exact solution when using \( \text{cSolve()} \), \( \text{cZeros()} \), \( \text{fMax()} \), \( \text{fMin()} \), \( \text{solve()} \), \( \text{zeros()} \), and so on.
(store)

\[ \text{Expr} \rightarrow \text{Var} \]
\[ \text{List} \rightarrow \text{Var} \]
\[ \text{Matrix} \rightarrow \text{Var} \]
\[ \text{Expr} \rightarrow \text{Function}(\text{Param1},...) \]
\[ \text{List} \rightarrow \text{Function}(\text{Param1},...) \]
\[ \text{Matrix} \rightarrow \text{Function}(\text{Param1},...) \]

If the variable \( \text{Var} \) does not exist, creates it and initializes it to \( \text{Expr} \), \( \text{List} \), or \( \text{Matrix} \).

If the variable \( \text{Var} \) already exists and is not locked or protected, replaces its contents with \( \text{Expr} \), \( \text{List} \), or \( \text{Matrix} \).

Hint: If you plan to do symbolic computations using undefined variables, avoid storing anything into commonly used, one-letter variables such as \( a, b, c, x, y, z \), and so on.

Note: You can insert this operator from the keyboard by typing \( =: \) as a shortcut. For example, type \( \pi/4 =: \text{myvar} \).

(assign)

\[ \text{Var} \rightarrow \text{Expr} \]
\[ \text{Var} \rightarrow \text{List} \]
\[ \text{Var} \rightarrow \text{Matrix} \]
\[ \text{Function}(\text{Param1},...) := \text{Expr} \]
\[ \text{Function}(\text{Param1},...) := \text{List} \]
\[ \text{Function}(\text{Param1},...) := \text{Matrix} \]

If variable \( \text{Var} \) does not exist, creates \( \text{Var} \) and initializes it to \( \text{Expr} \), \( \text{List} \), or \( \text{Matrix} \).

If \( \text{Var} \) already exists and is not locked or protected, replaces its contents with \( \text{Expr} \), \( \text{List} \), or \( \text{Matrix} \).

Hint: If you plan to do symbolic computations using undefined variables, avoid storing anything into commonly used, one-letter variables such as \( a, b, c, x, y, z \), and so on.

(comment)

\[ \text{©} \text{ [text]} \]

\( \text{©} \) processes \text{text} as a comment line, allowing you to annotate functions and programs that you create.

\( \text{©} \) can be at the beginning or anywhere in the line. Everything to the right of \( \text{©} \), 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 \( \rightarrow \) instead of \text{enter} at the end of each line. On the computer keyboard, hold down \text{Alt} and press \text{Enter}.

Define \( g(n)=\text{Func} \)

\[ \text{© Declare variables} \]
\[ \text{Local } i, \text{result} \]
\[ \text{result} := 0 \]
\[ \text{For } i, 1, n, 1 \text{ ©Loop n times} \]
\[ \text{result} := \text{result} + i^2 \]
\[ \text{EndFor} \]
\[ \text{Return result} \]
\[ \text{EndFunc} \]

Done

\[ g(3) \]

14
### Ob, Oh

<table>
<thead>
<tr>
<th><strong>Ob</strong></th>
<th><strong>binaryNumber</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oh</strong></td>
<td><strong>hexadecimalNumber</strong></td>
</tr>
</tbody>
</table>

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the Ob or Oh 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:

- Ob10 + 0hF + 10

#### In Bin base mode:

- Ob10 + 0hF + 10

#### In Hex base mode:

- Ob10 + 0hF + 10
**Empty (Void) Elements**

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ CAS Software allows empty, or void, data elements so you can proceed with the nearly complete data rather than having to start over or discard the incomplete cases.

You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “Graphing spreadsheet data.”

The delVoid() function lets you remove empty elements from a list. The isVoid() function lets you test for an empty element. For details, see delVoid(), page 35, and isVoid(), page 61.

**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 \[\text{ctrl} \ \text{w}\).

### 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

```
sum({2, _, 3, 5, 6, 7})
median({1, 2, _, _, _, 3})
cumulativeSum({1, 2, _, _, _, 4, 5})
cumulativeSum[1, 2, 3, _, 4, _, 5, 6, _]
```

**SortA** and **SortD** move all void elements within the first argument to the bottom.

```
{5, 4, 3, _, 1} → list1
{5, 4, 3, 2, 1} → list2
SortA, list1, list2
list1
list2
```

```
{1, 2, 3, _, 5} → list1
{1, 2, 3, 4, 5} → list2
SortD list1, list2
list1
list2
```

162  **TI-Nspire™ CAS Reference Guide**
In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

\[ H := \{1,2,3,4,5\}; I := \{2,3,5,6,6\} \]

\[ \text{LinRegMx } H,I \]

\[ \text{stat.Resid} \]

\[ \{0.434286, -0.862857, -0.011429, 0.44\} \]

\[ \text{stat.XReg} \]

\[ \{1, 3.4, 5\} \]

\[ \text{stat.YReg} \]

\[ \{2, 3, 5, 6, 6\} \]

\[ \text{stat.FreqReg} \]

\[ \{1, 1, 1, 1, 1\} \]

An omitted category in regressions introduces a void for the corresponding element of the residual.

\[ H := \{1,3,4,5\}; I := \{2,3,5,6,6\} \]

\[ \text{cat} = \{"M", "M", "F", "F"\}; \text{incl} = \{"F"\} \]

\[ \{\star\} \]

\[ \text{LinRegMx } H,I,1,\text{cat,incl} \]

\[ \text{stat.Resid} \]

\[ \{\star, 0, 0\} \]

\[ \text{stat.XReg} \]

\[ \{\star, 4, 5\} \]

\[ \text{stat.YReg} \]

\[ \{\star, 5, 6, 6\} \]

\[ \text{stat.FreqReg} \]

\[ \{\star, 1, 1, 1\} \]

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

\[ H := \{1,3,4,5\}; I := \{2,3,5,6,6\} \]

\[ \text{LinRegMx } H,I,2,\{1,0,1,1\} \]

\[ \text{stat.Resid} \]

\[ \{0.069231, -0.276923, 0.207692\} \]

\[ \text{stat.XReg} \]

\[ \{1, 4.5\} \]

\[ \text{stat.YReg} \]

\[ \{2, 5, 6, 6\} \]

\[ \text{stat.FreqReg} \]

\[ \{1, 1, 1, 1\} \]
Shortcuts for Entering Math Expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type $\text{sqrt}(6)$ on the entry line. When you press [enter], the expression $\text{sqrt}(6)$ 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

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi$</td>
<td>pi</td>
</tr>
<tr>
<td>$\theta$</td>
<td>theta</td>
</tr>
<tr>
<td>$\infty$</td>
<td>infinity</td>
</tr>
<tr>
<td>$\leq$</td>
<td>&lt;=</td>
</tr>
<tr>
<td>$\geq$</td>
<td>&gt;=</td>
</tr>
<tr>
<td>$\neq$</td>
<td>/=</td>
</tr>
<tr>
<td>$\Rightarrow$ (logical implication)</td>
<td>=&gt;</td>
</tr>
<tr>
<td>$\Leftrightarrow$ (logical double implication, XNOR)</td>
<td>&lt;=&gt;</td>
</tr>
<tr>
<td>$\rightarrow$ (store operator)</td>
<td>=:</td>
</tr>
<tr>
<td>$\mid$ (absolute value)</td>
<td>abs(...)</td>
</tr>
<tr>
<td>$\sqrt{0}$</td>
<td>sqrt(...)</td>
</tr>
<tr>
<td>$d()$</td>
<td>derivative(...)</td>
</tr>
<tr>
<td>$\int{}$ (Sum template)</td>
<td>sumSeq(...)</td>
</tr>
<tr>
<td>$\Pi()$ (Product template)</td>
<td>prodSeq(...)</td>
</tr>
<tr>
<td>$\sin(), \cos(), \ldots$</td>
<td>arcsin(...), arccos(...), \ldots</td>
</tr>
<tr>
<td>$\Delta\text{List}()$</td>
<td>deltaList(...)</td>
</tr>
<tr>
<td>$\Delta\text{tmpCnv}()$</td>
<td>deltaTmpCnv(...)</td>
</tr>
</tbody>
</table>

From the Computer Keyboard

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1, c_2, \ldots$ (constants)</td>
<td>@c1, @c2, \ldots</td>
</tr>
<tr>
<td>$n_1, n_2, \ldots$ (integer constants)</td>
<td>@n1, @n2, \ldots</td>
</tr>
<tr>
<td>$i$ (imaginary constant)</td>
<td>@i</td>
</tr>
<tr>
<td>To enter this:</td>
<td>Type this shortcut:</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>$e$ (natural log base e)</td>
<td>$@e$</td>
</tr>
<tr>
<td>$E$ (scientific notation)</td>
<td>$@E$</td>
</tr>
<tr>
<td>$T$ (transpose)</td>
<td>$@t$</td>
</tr>
<tr>
<td>$r$ (radians)</td>
<td>$@r$</td>
</tr>
<tr>
<td>$^\circ$ (degrees)</td>
<td>$@d$</td>
</tr>
<tr>
<td>$g$ (gradians)</td>
<td>$@g$</td>
</tr>
<tr>
<td>$\angle$ (angle)</td>
<td>$@&lt;\angle$</td>
</tr>
<tr>
<td>$&gt;$ (conversion)</td>
<td>$@&gt;$</td>
</tr>
<tr>
<td>$\Rightarrow$Decimal, $\Rightarrow$approxFraction(), and so on.</td>
<td>$\Rightarrow$Decimal, $\Rightarrow$approxFraction(), and so on.</td>
</tr>
</tbody>
</table>
**EOS™ (Equation Operating System) Hierarchy**

This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire™ CAS math and science learning technology. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS™ software evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

**Order of Evaluation**

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses ( ), brackets [ ], braces { }</td>
</tr>
<tr>
<td>2</td>
<td>Indirection (#)</td>
</tr>
<tr>
<td>3</td>
<td>Function calls</td>
</tr>
<tr>
<td>4</td>
<td>Post operators: degrees-minutes-seconds (°, ′, ″), factorial (!), percentage (%), radian ( ′ ), subscript ([ ]), transpose (ᵀ)</td>
</tr>
<tr>
<td>5</td>
<td>Exponentiation, power operator (^)</td>
</tr>
<tr>
<td>6</td>
<td>Negation (⁻)</td>
</tr>
<tr>
<td>7</td>
<td>String concatenation (&amp;)</td>
</tr>
<tr>
<td>8</td>
<td>Multiplication (•), division (/)</td>
</tr>
<tr>
<td>9</td>
<td>Addition (+), subtraction (-)</td>
</tr>
<tr>
<td>10</td>
<td>Equality relations: equal (=), not equal (≠ or /=), less than (&lt;), less than or equal (≤ or &lt;=), greater than (&gt;), greater than or equal (≥ or &gt;=)</td>
</tr>
<tr>
<td>11</td>
<td>Logical <strong>not</strong></td>
</tr>
<tr>
<td>12</td>
<td>Logical <strong>and</strong></td>
</tr>
<tr>
<td>13</td>
<td>Logical <strong>or</strong></td>
</tr>
<tr>
<td>14</td>
<td><strong>xor, nor, nand</strong></td>
</tr>
<tr>
<td>15</td>
<td>Logical implication (⇒)</td>
</tr>
<tr>
<td>16</td>
<td>Logical double implication, XNOR (⇔)</td>
</tr>
<tr>
<td>17</td>
<td>Constraint operator (&quot;</td>
</tr>
<tr>
<td>18</td>
<td>Store (→)</td>
</tr>
</tbody>
</table>

**Parentheses, Brackets, and Braces**

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression 4(1+2), EOS™ software first evaluates the portion of the expression inside the parentheses, 1+2, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, (1+2)/(3+4 will display the error message “Missing ).”
Note: Because the TI-Nspire™ CAS software allows you to define your own functions, a variable name followed by an expression in parentheses is considered a “function call” instead of implied multiplication. For example, \( a(b+c) \) is the function \( a \) evaluated by \( b+c \). To multiply the expression \( b+c \) by the variable \( a \), use explicit multiplication: \( a \times (b+c) \).

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°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 \( [\rightarrow] \) followed by the number. Post operations and exponentiation are performed before negation. For example, the result of \( -x^2 \) is a negative number, and \( -9^2 = -81 \). Use parentheses to square a negative number such as \( (-9)^2 \) to produce 81.

Constraint ("|")

The argument following the constraint ("|") operator provides a set of constraints that affect the evaluation of the argument preceding the operator.
Error Codes and Messages

When an error occurs, its code is assigned to variable `errCode`. User-defined programs and functions can examine `errCode` to determine the cause of an error. For an example of using `errCode`, see Example 2 under the `Try` command, page 130.

Note: Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A function did not return a value</td>
</tr>
<tr>
<td>20</td>
<td>A test did not resolve to TRUE or FALSE. Generally, undefined variables cannot be compared. For example, the test if a&lt;b will cause this error if either a or b is undefined when the If statement is executed.</td>
</tr>
<tr>
<td>30</td>
<td>Argument cannot be a folder name.</td>
</tr>
<tr>
<td>40</td>
<td>Argument error</td>
</tr>
<tr>
<td>50</td>
<td>Argument mismatch Two or more arguments must be of the same type.</td>
</tr>
<tr>
<td>60</td>
<td>Argument must be a Boolean expression or integer</td>
</tr>
<tr>
<td>70</td>
<td>Argument must be a decimal number</td>
</tr>
<tr>
<td>90</td>
<td>Argument must be a list</td>
</tr>
<tr>
<td>100</td>
<td>Argument must be a matrix</td>
</tr>
<tr>
<td>130</td>
<td>Argument must be a string</td>
</tr>
<tr>
<td>140</td>
<td>Argument must be a variable name. Make sure that the name: • does not begin with a digit • does not contain spaces or special characters • does not use underscore or period in invalid manner • does not exceed the length limitations See the Calculator section in the documentation for more details.</td>
</tr>
<tr>
<td>160</td>
<td>Argument must be an expression</td>
</tr>
<tr>
<td>165</td>
<td>Batteries too low for sending or receiving Install new batteries before sending or receiving.</td>
</tr>
<tr>
<td>170</td>
<td>Bound The lower bound must be less than the upper bound to define the search interval.</td>
</tr>
<tr>
<td>180</td>
<td>Break The <code>esc</code> or <code>home</code> key was pressed during a long calculation or during program execution.</td>
</tr>
<tr>
<td>190</td>
<td>Circular definition This message is displayed to avoid running out of memory during infinite replacement of variable values during simplification. For example, a+1-&gt;a, where a is an undefined variable, will cause this error.</td>
</tr>
<tr>
<td>200</td>
<td>Constraint expression invalid For example, solve(3x^2-4=0,x)</td>
</tr>
<tr>
<td>210</td>
<td>Invalid Data type An argument is of the wrong data type.</td>
</tr>
<tr>
<td>220</td>
<td>Dependent limit</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>230</td>
<td>Dimension error. 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.</td>
</tr>
<tr>
<td>235</td>
<td>Dimension Error. Not enough elements in the lists.</td>
</tr>
<tr>
<td>240</td>
<td>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.</td>
</tr>
<tr>
<td>250</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>260</td>
<td>Domain error. An argument must be in a specified domain. For example, ( \text{rand}(0) ) is not valid.</td>
</tr>
<tr>
<td>270</td>
<td>Duplicate variable name</td>
</tr>
<tr>
<td>280</td>
<td>Else and ElseIf invalid outside of If...EndIf block</td>
</tr>
<tr>
<td>290</td>
<td>EndTry is missing the matching Else statement</td>
</tr>
<tr>
<td>295</td>
<td>Excessive iteration</td>
</tr>
<tr>
<td>300</td>
<td>Expected 2 or 3-element list or matrix</td>
</tr>
<tr>
<td>310</td>
<td>The first argument of ( \text{nSolve} ) must be an equation in a single variable. It cannot contain a non-valued variable other than the variable of interest.</td>
</tr>
<tr>
<td>320</td>
<td>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.</td>
</tr>
<tr>
<td>345</td>
<td>Inconsistent units</td>
</tr>
<tr>
<td>350</td>
<td>Index out of range</td>
</tr>
<tr>
<td>360</td>
<td>Indirection string is not a valid variable name</td>
</tr>
<tr>
<td>380</td>
<td>Undefined Ans. Either the previous calculation did not create Ans, or no previous calculation was entered.</td>
</tr>
<tr>
<td>390</td>
<td>Invalid assignment</td>
</tr>
<tr>
<td>400</td>
<td>Invalid assignment value</td>
</tr>
<tr>
<td>410</td>
<td>Invalid command</td>
</tr>
<tr>
<td>430</td>
<td>Invalid for the current mode settings</td>
</tr>
<tr>
<td>435</td>
<td>Invalid guess</td>
</tr>
<tr>
<td>440</td>
<td>Invalid implied multiply. For example, ( x(x+1) ) is invalid; whereas, ( x^2(x+1) ) is the correct syntax. This is to avoid confusion between implied multiplication and function calls.</td>
</tr>
<tr>
<td>450</td>
<td>Invalid in a function or current expression. Only certain commands are valid in a user-defined function.</td>
</tr>
<tr>
<td>490</td>
<td>Invalid in Try..EndTry block</td>
</tr>
<tr>
<td>510</td>
<td>Invalid list or matrix</td>
</tr>
<tr>
<td>550</td>
<td>Invalid outside function or program. A number of commands are not valid outside a function or program. For example, ( \text{Local} ) cannot be used unless it is in a function or program.</td>
</tr>
<tr>
<td>560</td>
<td>Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks. For example, the Exit command is valid only inside these loop blocks.</td>
</tr>
<tr>
<td>565</td>
<td>Invalid outside program</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 570        | Invalid pathname  
For example, \(i\)var is invalid. |
| 575        | Invalid polar complex |
| 580        | Invalid program reference  
Programs cannot be referenced within functions or expressions such as 1+p(x) where p is a program. |
| 600        | Invalid table |
| 605        | Invalid use of units |
| 610        | Invalid variable name in a Local statement |
| 620        | Invalid variable or function name |
| 630        | Invalid variable reference |
| 640        | Invalid vector syntax |
| 650        | Link transmission  
A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends. |
| 665        | Matrix not diagonalizable |
| 670        | Low Memory  
1. Delete some data in this document  
2. Save and close this document  
If 1 and 2 fail, pull out and re-insert batteries |
| 672        | Resource exhaustion |
| 673        | Resource exhaustion |
| 680        | Missing ( |
| 690        | Missing ) |
| 700        | Missing * |
| 710        | Missing ] |
| 720        | Missing ) |
| 730        | Missing start or end of block syntax |
| 740        | Missing Then in the If..Endif block |
| 750        | Name is not a function or program |
| 765        | No functions selected |
| 780        | No solution found |
| 800        | Non-real result  
For example, if the software is in the Real setting, \(\sqrt{-1}\) is invalid.  
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. |
<p>| 855        | Rand type functions not allowed in graphing |
| 860        | Recursion too deep |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>870</td>
<td>Reserved name or system variable</td>
</tr>
</tbody>
</table>
| 900        | Argument error  
Median-median model could not be applied to data set. |
| 910        | Syntax error |
| 920        | Text not found |
| 930        | Too few arguments  
The function or command is missing one or more arguments. |
| 940        | Too many arguments  
The expression or equation contains an excessive number of arguments and cannot be evaluated. |
| 950        | Too many subscripts |
| 955        | Too many undefined variables |
| 960        | Variable is not defined  
No value is assigned to variable. Use one of the following commands:  
• \( \text{sto} \)  
• \( \text{Define} \)  
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:  
• \( \text{Define} \)  
• \( \text{sto} \)  
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. |
<p>| 1110       | Invalid bounds |
| 1120       | No sign change |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1130</td>
<td>Argument cannot be a list or matrix</td>
</tr>
</tbody>
</table>
| 1140       | Argument error  
The first argument must be a polynomial expression in the second argument. If the second argument is omitted, the software attempts to select a default. |
| 1150       | Argument error  
The first two arguments must be polynomial expressions in the third argument. If the third argument is omitted, the software attempts to select a default. |
| 1160       | Invalid library pathname  
A pathname must be in the form `xxx\yyy`, where:  
- The `xxx` part can have 1 to 16 characters.  
- The `yyy` part can have 1 to 15 characters.  
See the Library section in the documentation for more details. |
| 1170       | Invalid use of library pathname  
- A value cannot be assigned to a pathname using `Define`, `:=`, or `sto ->`.  
- A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition. |
| 1180       | Invalid library variable name.  
Make sure that the name:  
- Does not contain a period  
- Does not begin with an underscore  
- Does not exceed 15 characters  
See the Library section in the documentation for more details. |
| 1190       | Library document not found:  
- Verify library is in the MyLib folder.  
- Refresh Libraries.  
See the Library section in the documentation for more details. |
| 1200       | Library variable not found:  
- Verify library variable exists in the first problem in the library.  
- Make sure library variable has been defined as LibPub or LibPriv.  
- Refresh Libraries.  
See the Library section in the documentation for more details. |
| 1210       | Invalid library shortcut name.  
Make sure that the name:  
- Does not contain a period  
- Does not begin with an underscore  
- Does not exceed 16 characters  
- Is not a reserved name  
See the Library section in the documentation for more details. |
| 1220       | Domain error:  
The tangentLine and normalLine functions support real-valued functions only. |
| 1230       | Domain error:  
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. |
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1290       | Argument Error  
Use a polynomial in one variable. |
| 1300       | Argument Error  
The coefficients of the polynomial must evaluate to numeric values. |
| 1310       | Argument error:  
A function could not be evaluated for one or more of its arguments. |
| 1380       | Argument error:  
Nested calls to domain() function are not allowed. |
### Warning Codes and Messages

You can use the `warnCodes()` function to store the codes of warnings generated by evaluating an expression. This table lists each numeric warning code and its associated message. For an example of storing warning codes, see `warnCodes()`, page 136.

<table>
<thead>
<tr>
<th>Warning code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Operation might introduce false solutions.</td>
</tr>
<tr>
<td>10001</td>
<td>Differentiating an equation may produce a false equation.</td>
</tr>
<tr>
<td>10002</td>
<td>Questionable solution</td>
</tr>
<tr>
<td>10003</td>
<td>Questionable accuracy</td>
</tr>
<tr>
<td>10004</td>
<td>Operation might lose solutions.</td>
</tr>
<tr>
<td>10005</td>
<td><code>cSolve</code> might specify more zeros.</td>
</tr>
<tr>
<td>10006</td>
<td><code>Solve</code> may specify more zeros.</td>
</tr>
</tbody>
</table>
| 10007        | More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess. Examples using `solve()`:
|              | • `solve(Equation, Var=Guess)|lowBound<Var<upBound` |
|              | • `solve(Equation, Var)|lowBound<Var<upBound` |
|              | • `solve(Equation, Var=Guess)` |
| 10008        | Domain of the result might be smaller than the domain of the input. |
| 10009        | Domain of the result might be larger than the domain of the input. |
| 10012        | Non-real calculation |
| 10013        | $\infty^0$ or `undef^0` replaced by 1 |
| 10014        | `undef^0` replaced by 1 |
| 10015        | $1^\infty$ or `1^undef` replaced by 1 |
| 10016        | `1^undef` replaced by 1 |
| 10017        | Overflow replaced by $\infty$ or \(-\infty\) |
| 10018        | Operation requires and returns 64 bit value. |
| 10019        | Resource requires and returns 64 bit value. |
| 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 "\(1\) `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 U.S. and Canada:

For General Information

Home Page: education.ti.com
KnowledgeBase and e-mail inquiries: education.ti.com/support

Phone: (800) TI-CARES / (800) 842-2737
For U.S., Canada, Mexico, Puerto Rico, and Virgin Islands only

International information: education.ti.com/international

For Technical Support

KnowledgeBase and support by e-mail: education.ti.com/support

Phone (not toll-free): (972) 917-8324

For Product (Hardware) Service

Customers in the U.S., Canada, Mexico, Puerto Rico and Virgin Islands: Always contact Texas Instruments Customer Support before returning a product for service.

For All Other Countries:

For general information

For more information about TI products and services, contact TI by e-mail or visit the TI Internet address.

E-mail inquiries: ti-cares@ti.com
Home Page: education.ti.com
Service and Warranty Information

For information about the length and terms of the warranty or about product service, refer to the warranty statement enclosed with this product or contact your local Texas Instruments retailer/distributor.
Index

Symbols
^, power 145
^-1, reciprocal 158
_, unit designation 157
:=, assign 160
!, factorial 150
., dot power 147
.*, dot multiplication 146
.+., dot addition 146
.-., dot subtraction 146
÷., dot division 147
\', minute notation 156
\', prime 157
", second notation 156
≤, less than or equal 149
©, comment 160
Δlist( ), list difference 67
°, degree notation 156
°, degrees/minutes/seconds 156
›, convert units 158
∫, integral 151
√, square root 152
≠, not equal 148
−, subtract 143
÷, divide 144
Π, product 152
Σ( ), sum 153
⇔, logical double implication 150
⇒, logical implication 149, 164
*, multiply 144
&, append 150
⇒, store 160
#, indirection 155
#, indirection operator 167
%, percent 147
+, add 143
<, less than 148
=, equal 148
>, greater than 149
|, constraint operator 159
≥, greater than or equal 149

Numerics
0b, binary indicator 161
0h, hexadecimal indicator 161
10^(), power of ten 158
2-sample F Test 51
approxFraction( ) 11

A

abs( ), absolute value 7
absolute value
  template for 3
add, + 143
amortization table, amortTbl( ) 7, 13
amortTbl( ), amortization table 7, 13
and, Boolean operator 7
angle, angle( ) 8
angle( ), angle 8
ANOVA, one-way variance analysis 8
ANOVA2way, two-way variance analysis 9
Ans, last answer 11
answer (last), Ans 11
append, & 150
approx( ), approximate 11, 12
approximate, approx( ) 11, 12
approxRational( ) 11
arc length, arcLen( ) 12
arccos() 11
arccosh() 12
arccosine, cos⁻¹() 23
arccot() 12
arccoth() 12
arccsc() 12
arccsch() 12
arcLen( ), arc length 12
arcsec() 12
arcsech() 12
arcsec() 12
arcsine, sin⁻¹() 113
arcsinh() 12
arctan() 12
arctangent, tan⁻¹() 125
arctanh() 12
arguments in TVM functions 132
augment( ), augment/concatenate 12
augment/concatenate, augment()  12
average rate of change, avgRC()  13
avgRC(), average rate of change  13

B
▷Base10, display as decimal integer  14
▷Base16, display as hexadecimal  15
▷Base2, display as binary  14
binary
display, ▷Base2  14
indicator, 0b 161
binomCdf()  15
binomPdf()  15
Boolean operators
and 7
nand 80
nor 83
not 84
or 87
⇔ 150
xor 137
⇒ 149, 164

c
χ²2way 17
χ²Cdf()  17
χ²GOF 18
χ²Pdf()  18
Cdf()  47
ceiling, ceiling()  15, 16, 26
celling(), ceiling  15
centralDiff()  16
cFactor(), complex factor 16
cchar(), character string 17
ccharacter string, char()  17
ccharacters
numeric code, ord()  87
string, char()  17
charPoly()  17
clear
error, ClrErr  19
ClearAZ  18
ClrErr, clear error  19
colAugment  19
colDim(), matrix column dimension  19
colNorm(), matrix column norm  19
combinations, nCr()  81
comDenom(), common denominator  19
coment, © 160
common denominator,
comDenom()  19
completeSquare(), complete square  20
complex
conjugate, conj()  21
factor, cFactor()  16
solve, cSolve()  28
zeros, cZeros()  31
conj(), complex conjugate  21
constant
in solve()  116
constants
in cSolve()  29
in cZeros()  32
in deSolve()  36
in solve()  117
in zeros()  138
shortcuts for  164
constraint operator "|"  159
constraint operator, order of evaluation  166
construct matrix, constructMat()  21
constructMat(), construct matrix  21
convert
▷Grad  56
▷Rad  97
units  158
copy variable or function, CopyVar  21
correlation matrix, corrMat()  22
corrMat(), correlation matrix  22
cos, display in terms of cosine  22
cos(), cosine  22
cos⁻¹, arccosine  23
cosh(), hyperbolic cosine  24
cosh⁻¹(), hyperbolic arccosine  24
cosine
display expression in terms of  22
cosine, cos()  22
cot(), cotangent  24
cot( ), arccotangent 25
cotangent, cot( ) 24
coth( ), hyperbolic cotangent 25
coth( ), hyperbolic arccotangent 25
count days between dates, dbd( ) 33
count items in a list conditionally, countif( ) 26
count items in a list, count( ) 25
count( ), count items in a list 25
countif( ), conditionally count items in a list 26
cPolyRoots( ) 26
cross product, crossP( ) 26
crossP( ), cross product 26
csc( ), cosecant 27
csc( ), inverse cosecant 27
csch( ), hyperbolic cosecant 27
csch( ), inverse hyperbolic cosecant 27
cSolve( ), complex solve 28
cubic regression, CubicReg 30
CubicReg, cubic regression 30
cumulative sum, cumulativeSum( ) 30
cumulativeSum( ), cumulative sum 30
Cycle, cycle 31
cycle, Cycle 31
CYlind, display as cylindrical vector 31
cylindrical vector display, CYlind 31
cZeros( ), complex zeros 31
define, Define 34
defining
private function or program 34
public function or program 35
definite integral
template for 5
degree notation, ° 156
degree/minute/second display, DMS 38
degree/minute/second notation 156
delete
void elements from list 35
deleting
variable, DelVar 35
deltaList( ) 35
deltaTmpCnv( ) 35
DelVar, delete variable 35
delVoid( ), remove void elements 35
denominator 19
derivative or nth derivative
template for 5
derivative( ) 35
derivatives
first derivative, d( ) 150
numeric derivative, nDeriv( ) 82
numeric derivative, nDerivative( ) 81
desolve( ), solution 36
det( ), matrix determinant 37
diag( ), matrix diagonal 37
dim( ), dimension 37
dimension, dim( ) 37
Disp, display data 38
display as
binary, BASE2 14
cylindrical vector, CYlind 31
decimal angle, DD 33
decimal integer, BASE10 14
degree/minute/second, DMS 38
hexadecimal, BASE16 15
polar vector, POLAR 89
rectangular vector, RECT 99
spherical vector, SPHERE 119
display data, Disp 38
distribution functions
binomCdf( ) 15
binomPdf( ) 15
χ² 2way( ) 17
\(\chi^2\text{Df}()\) 17
\(\chi^2\text{Gof}()\) 18
\(\chi^2\text{Pdf}()\) 18
inv\(\chi^2()\) 60
invNorm() 60
invt() 60
normCdf() 83
normPdf() 84
poissCdf() 88
poissPdf() 88
tCdf() 126
tPdf() 129
divide, ÷ 144
\(\text{DMS},\) display as degree/minute/second 38
domain function, domain() 38
domain(), domain function 38
dominant term, dominantTerm() 39
dominantTerm(), dominant term 39
dot
addition, .+ 146
division, .÷ 147
multiplication, .* 146
power, .^ 147
product, dotP() 39
subtraction, .- 146
dotP(), dot product 39

\textbf{E}

e exponent
  template for 2
e to a power, e^\() 40, 43
exp\()(), e to a power 40
e, display expression in terms of 43
E, exponent 155
eff()), convert nominal to effective rate 40
effective rate, eff() 40
eigenvalue, eigVl() 41
eigenvector, eigVc() 40
eigVc(), eigenvector 40
eigVl(), eigenvalue 41
else if, Elself 41
else, Else 57
Elself, else if 41
empty (void) elements 162
end

for, EndFor 49
function, EndFunc 52
if, EndIf 57
loop, EndLoop 73
program, EndPrgm 93
try, EndTry 130
while, EndWhile 136
end function, EndFunc 52
end if, EndIf 57
end loop, EndLoop 73
end while, EndWhile 136
EndTry, end try 130
EndWhile, end while 136
EOS (Equation Operating System) 166
equal, = 148
Equation Operating System (EOS) 166
error codes and messages 168
errors and troubleshooting
clear error, ClrErr 19
pass error, PassErr 88
euler\()(), Euler function 42
evaluate polynomial, polyEval() 90
evaluation, order of 166
exact, exact() 42
exact\(), exact 42
exclusion with "|" operator 159
Exit, exit 43
exit, Exit 43
\(\text{exp},\) display in terms of \(e\) 43
\exp()(), e to a power 43
\exp\text{list}(), expression to list 44
expand, expand() 44
expand\(), expand 44
exponent, \(E\) 155
exponential regression, ExpReg 45
exponents
  template for 1
expr\()(), string to expression 45, 71
ExpReg, exponential regression 45
expressions
  expression to list, \exp\text{list}() 44
  string to expression, expr\() 45, 71
F
factor, factor() 46
factor(), factor 46
decimal, ! 150
Fill, matrix fill 47
financial functions, tvmFV( ) 132
financial functions, tvmI( ) 132
financial functions, tvmN( ) 132
financial functions, tvmPmt( ) 132
financial functions, tvmPV( ) 132
first derivative
template for 5
FiveNumSummary 48
floor, floor( ) 48
floor(), floor 48
fMax(), function maximum 48
fMin(), function minimum 49
For 49
For, for 49
for, For 49
format string, format() 50
format(), format string 50
fpart(), function part 50
fractions
propFrac 94
template for 1
freqTable() 50
frequency( ) 51
Frobenius norm, norm( ) 83
Func, function 52
Func, program function 52
functions
maximum, fMax() 48
minimum, fMin() 49
part, fpart() 50
program function, Func 52
user-defined 34
functions and variables
copying 21

G
9, gradians 155
gcd(), greatest common divisor 52
geomCdf() 52
geomPdf() 53
get/return
denominator, getDenom() 53
number, getNum() 54
variables information,
getVarInfo() 53, 55
getDenom(), get/return
denominator 53
getLangInfo(), get/return language
information 53
getLockInfo(), tests lock status of
variable or variable group 53
getMode(), get mode settings 54
getNum(), get/return number 54
type(), get type of variable 55
getAddress(), get/return variables
information 55
go to, Goto 56
Goto, go to 56
\textasciitilde, convert to gradian angle 56
gradian notation, \textasciitilde 155
greater than or equal, \textasciitilde \geq 149
greater than, > 149
greatest common divisor, gcd() 52
groups, locking and unlocking 70, 135
groups, testing lock status 53

H
hexadecimal
display, \textasciitilde Base16 15
indicator, 0h 161
hyperbolic
arccosine, \textasciitilde \cos^{-1} 24
arcsine, \textasciitilde \sin^{-1} 114
arctangent, \textasciitilde \tan^{-1} 126
cosine, \textasciitilde \cos 24
sine, \textasciitilde \sin 114
tangent, \textasciitilde \tan 125

I
identity matrix, identity() 56
identity(), identity matrix 56
If, if 57
if, If 57
ifFn() 58
imag(), imaginary part 58
imaginary part, imag() 58
ImpDif(), implicit derivative 58
implicit derivative, Impdif() 58
indefinite integral
  template for 5
indirection operator (#) 167
indirection, # 155
Input, input 58
input, Input 58
inString( ), within string 59
int( ), integer 59
intDiv( ), integer divide 59
integer divide, intDiv( ) 59
integer part, iPart( ) 61
integer, int( ) 59
integral, \int 151
interpolate( ), interpolate 60
\text{Inv} \chi^2( ) 60
inverse cumulative normal distribution (invNorm( ) 60
inverse, \ ^{-1} 158
invF( ) 60
invNorm( ), inverse cumulative normal distribution 60
invt( ) 60
iPart( ), integer part 61
irr( ), internal rate of return
  internal rate of return, irr( ) 61
isPrime( ), prime test 61
isVoid( ), test for void 61

L
label, Lbl 62
language
  get language information 53
Lbl, label 62
lcm, least common multiple 62
least common multiple, lcm 62
left, left( ) 62
left( ), left 62
length of string 37
less than or equal, \leq 149
less than, < 148
LibPriv 34
LibPub 35
library
  create shortcuts to objects 63
libShortcut( ), create shortcuts to library objects 63
limit
  \text{lim}( ) 63
  \text{limit}( ) 63
  template for 6
limit( ) or \text{lim}( ), limit 63
linear regression, LinRegAx 64
linear regression, LinRegBx 64, 65
LinRegBx, linear regression 64
LinRegMx, linear regression 64
LinRegIntervals, linear regression 65
LinRegTTest 66
linSolve() 67
list to matrix, list\text{mat}( ) 68
list, conditionally count items in 26
list, count items in 25
list\text{mat}( ), list to matrix 68
lists
  augment/concatenate, augment( ) 12
cross product, crossP( ) 26
cumulative sum,
  cumulativeSum( ) 30
difference, \Delta \text{list}( ) 67
differences in a list, \Delta \text{list}( ) 67
dot product, dotP( ) 39
empty elements in 162
expression to list, exp\text{list}( ) 44
list to matrix, list\text{mat}( ) 68
matrix to list, mat\text{list}( ) 74
maximum, max( ) 75
mid-string, mid( ) 76
minimum, min( ) 77
new, newList( ) 81
product, product( ) 93
sort ascending, SortA 118
sort descending, SortD 118
summation, sum( ) 123
\ln( ), natural logarithm 68
LnReg, logarithmic regression 69
local variable, Local 70
local, Local 70
Local, local variable 70
Lock, lock variable or variable group 70
locking variables and variable groups 70
Log
  template for 2

182
logarithmic regression, LnReg 69
logarithms 68
logical double implication, ⇔ 150
logical implication, ⇒ 149, 164
logistic regression, Logistic 72
logistic regression, LogisticD 72
Logistic, logistic regression 72
LogisticD, logistic regression 72
Loop, loop 73
loop, Loop 73
LU, matrix lower-upper decomposition 74

M
mat\(\)list\(\), matrix to list 74
matrices
  augment/concatenate, augment\(\) 12
  column dimension, colDim\(\) 19
  column norm, colNorm\(\) 19
  cumulative sum, cumulativeSum\(\) 30
  determinant, det\(\) 37
diagonal, diag\(\) 37
dimension, dim\(\) 37
dot addition, .+ 146
dot division, .\(\) 147
dot multiplication, .* 146
dot power, .^ 147
dot subtraction, .- 146
eigenvalue, eigVl\(\) 41
eigenvector, eigVc\(\) 40
filling, Fill 47
identity, identity\(\) 56
list to matrix, list°mat\(\) 68
lower-upper decomposition, LU 74
matrix to list, mat\(\)list\(\) 74
maximum, max\(\) 75
minimum, min\(\) 77
new, newMat\(\) 81
product, product\(\) 93
QR factorization, QR 94
random, randMat\(\) 98
reduced row echelon form, rref\(\) 105
row addition, rowAdd\(\) 105
row dimension, rowDim\(\) 105
row echelon form, ref\(\) 100
row multiplication and addition, mRowAdd\(\) 78
row norm, rowNorm\(\) 105
row operation, mRow\(\) 78
row swap, rowSwap\(\) 105
submatrix, subMat\(\) 122, 123
summation, sum\(\) 123
transpose, \(\)T 124
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\(\) 74
max\(\), maximum 75
maximum, max\(\) 75
mean, mean\(\) 75
mean\(\), mean 75
median, median\(\) 75
median\(\), median 75
medium-medium line regression, MedMed 76
MedMed, medium-medium line regression 76
mid\(\), mid-string 76
mid-string, mid\(\) 76
min\(\), minimum 77
minimum, min\(\) 77
minute notation, ’ 156
mirr\(\), modified internal rate of return 77
mixed fractions, using propFrac\(\) with 94
mod\(\), modulo 78
mode settings, getMode\(\) 54
modes
  setting, setMode\(\) 110
modified internal rate of return, mirr\(\) 77
modulo, mod\(\) 78
mRow\(\), matrix row operation 78
mRowAdd\(\), matrix row multiplication and addition 78
Multiple linear regression t test 79
multiply, * 144
MultReg 78
MultRegIntervals( ) 79
MultRegTests( ) 79

N
nand, Boolean operator 80
natural logarithm, ln( ) 68
nCr( ), combinations 81
nDerivative( ), numeric derivative 81
negation, entering negative numbers 167
net present value, npv( ) 85
new
list, newList() 81
matrix, newMat( ) 81
newList(), new list 81
newMat( ), new matrix 81
nfMax( ), numeric function maximum 82
nfMin( ), numeric function minimum 82
nInt( ), numeric integral 82
nom( ), convert effective to nominal rate 82
nominal rate, nom( ) 82
nor, Boolean operator 83
norm( ), Frobenius norm 83
normal distribution probability, normCdf( ) 83
normal line, normalLine( ) 83
normalLine( ) 83
normCdf( ) 83
normPdf( ) 84
not equal, 148
not, Boolean operator 84
nPr( ), permutations 84
npv( ), net present value 85
nSolve( ), numeric solution 85
nth root
template for 1
numeric
derivative, nDeriv( ) 82
derivative, nDerivative( ) 81
integral, nInt( ) 82
solution, nSolve( ) 85

O
objects
create shortcuts to library 63
OneVar, one-variable statistics 86
one-variable statistics, OneVar 86
operators
order of evaluation 166
or (Boolean), or 87
or, Boolean operator 87
ord( ), numeric character code 87

P
PRx( ), rectangular x coordinate 87
PRy( ), rectangular y coordinate 88
pass error, PassErr 88
PassErr, pass error 88
Pdf( ) 50
percent, % 147
permutations, nPr( ) 84
piecewise function (2-piece)
template for 2
piecewise function (N-piece)
template for 2
piecewise( ) 88
poissCdf( ) 88
poissPdf( ) 88
Polar, display as polar vector 89
polar
coordinate, RvPθ( ) 97
coordinate, RvPr( ) 97
vector display, Polar 89
polyCoeff( ) 89
polyDegree( ) 90
polyEval( ), evaluate polynomial 90
polyGcd( ) 90, 91
polynomials
evaluate, polyEval( ) 90
random, randPoly( ) 98
PolyRoots() 91
power of ten, 10^( ) 158
power regression, PowerReg 91, 92, 101, 102, 127
power, ^ 145
PowerReg, power regression 92
Prgm, define program 93
prime number test, isPrime( ) 61
prime, ' 157
probability density, normPdf() 84
prodSeq() 93
product (Π)
  template for Π
product, Π() 152
product, product() 93
product(), product 93
programming
  define program, Prgm 93
display data, Disp 38
pass error, PassErr 88
programs
  defining private library 34
defining public library 35
programs and programming
  clear error, ClrErr 19
display I/O screen, Disp 38
end program, EndPrgm 93
d Bayer, EndTry 130
try, Try 130
proper fraction, propFrac 94
propFrac, proper fraction 94

Q
QR factorization, QR 94
QR, QR factorization 94
quadratic regression, QuadReg 95
QuadReg, quadratic regression 95
quartic regression, QuartReg 96
QuartReg, quartic regression 96

R
r, radian 155
Prθ(), polar coordinate 97
Pr(), polar coordinate 97
Rad, convert to radian angle 97
radian, r 155
rand(), random number 97
randBin, random number 98
randInt(), random integer 98
randMat(), random matrix 98
randNorm(), random norm 98
random
  matrix, randMat() 98
  norm, randNorm() 98
number seed, RandSeed 99
polynomial, randPoly() 98
random sample 98
randPoly(), random polynomial 98
randSamp() 98
RandSeed, random number seed 99
real, real() 99
real(), real 99
reciprocal, ^-1 158
Rect, display as rectangular vector 99
rectangular x coordinate, P Rx() 87
rectangular y coordinate, P Ry() 88
rectangular-vector display, Rect 99
reduced row echelon form, ref() 105
ref(), row echelon form 100
regressions
  cubic, CubicReg 30
  exponential, ExpReg 45
  linear regression, LinRegAx 64
  linear regression, LinRegBx 64, 65
  logarithmic, LnReg 69
  Logistic 72
  logistic, Logistic 72
  medium-medium line, MedMed 76
  MultReg 78
  power regression, PowerReg 91, 92, 101, 102, 127
  quadratic, QuadReg 95
  quartic, QuartReg 96
  sinusoidal, SinReg 115
remain(), remainder 100
remainder, remain() 100
remove
  void elements from list 35
Request 101
RequestStr 102
result
  display in terms of cosine 22
display in terms of e 43
display in terms of sine 112
result values, statistics 121
results, statistics 120
Return, return 102
return, Return 102
right, right() 120, 60, 102, 103, 136
right(), right 102
rk23(), Runge Kutta function 103
rotate, rotate( ) 103, 104
rotate( ), rotate 103, 104
round, round( ) 104
round( ), round 104
erow echelon form, ref( ) 100
rowAdd( ), matrix row addition 105
rowDim( ), matrix row dimension 105
rowNorm( ), matrix row norm 105
rowSwap( ), matrix row swap 105
rref( ), reduced row echelon form 105

sec( ), secant 106
sec( ), inverse secant 106
sech( ), hyperbolic secant 106
sech( ), inverse hyperbolic secant 107
second derivative
  template for 5
second notation, “ 156
seq( ), sequence 107
seqGen( ) 108
seqn( ) 108
sequence, seq( ) 107, 108
series, series( ) 109
series( ), series 109
set
  mode, setMode( ) 110
setMode( ), set mode 110
settings, get current 54
shift, shift( ) 111
shift( ), shift 111
sign, sign( ) 111
sign( ), sign 111
simult( ), simultaneous equations 112
simultaneous equations, simult( ) 112
sin( ), sine 113
sin( ), arcsine 113
sine
  display expression in terms of 112

sine, sin( ) 113
sinh( ), hyperbolic sine 114
sinh( ), hyperbolic arcsine 114
SinReg, sinusoidal regression 115
SinInt( ) 154
sinusoidal regression, SinReg 115
solution, deSolve( ) 36
solve, solve( ) 115
solve( ), solve 115
SortA, sort ascending 118
SortD, sort descending 118
sorting
  ascending, SortA 118
descending, SortD 118
Sphere, display as spherical vector 119
spherical vector display, Sphere 119
ΣPrn( ) 154
sqrt( ), square root 119
square root
  template for 1
square root, √( ) 119, 152
standard deviation, stdDev( ) 121, 135
stat.results 120
stat.values 121
statistics
  combinations, nCr( ) 81
  factorial, ! 150
  mean, mean( ) 75
  median, median( ) 75
  one-variable statistics, OneVar 86
  permutations, nPr( ) 84
random norm, randNorm( ) 98
random number seed, RandSeed 99
standard deviation, stdDev( ) 121, 135
two-variable results, TwoVar 133
variance, variance( ) 135
stdDevPop( ), population standard deviation 121
stdDevSamp( ), sample standard deviation 121
Stop command 122
storing
  symbol, → 160
string
dimension, \text{dim}( ) 37
length 37
\text{string( )}, \text{expression} to \text{string} 122
strings
append, \& 150
character code, \text{ord}( ) 87
character string, \text{char( )} 17
expression to string, \text{string( )} 122
format, \text{format( )} 50
formatting 50
indirection, \# 155
left, \text{left( )} 62
mid-string, \text{mid( )} 76
right, \text{right( )} 20, 42, 60, 102, 103, 136
rotate, \text{rotate( )} 103, 104
shift, \text{shift( )} 111
string to expression, \text{expr( )} 45, 71
using to create variable names 167
within, \text{InString} 59
\text{student-}\text{t} \text{distribution} probability, \text{tCdf( )} 126
\text{student-}\text{t} \text{probability} density, \text{tpdf( )} 129
\text{subMat( )}, \text{submatrix} 122, 123
\text{submatrix}, \text{subMat( )} 122, 123
\text{substitution} with "|" \text{operator} 159
subtract, \text{\text{-} } 143
\text{sum} (\Sigma)
\text{template} for 4
\text{sum} of \text{interest payments} 154
\text{sum} of \text{principal payments} 154
\text{sum}, \text{\text{\Sigma}( )} 153
\text{sum( )}, \text{\text{summa}tion} 123
\text{sumIf( )} 123
\text{\text{summa}tion}, \text{\text{\text{sum}( )} 123}
\text{\text{sumSeq()}} 123
\text{\text{system of equations (2-equation)}}
\text{\text{template} for 3}
\text{\text{system of equations (N-equation)}}
\text{\text{template} for 3}

\text{T}
\text{t test, tTest} 131
\text{T, transpose} 124
\text{tan( ), tangent} 124
\text{tan}^{-1}( ), \text{arctangent} 125
\text{tangent line, tangentLine( )} 125
\text{tangent, tan( )} 124
\text{tangentLine( )} 125
\text{tanh( ), hyperbolic tangent} 125
\text{tanh}^{-1}( ), \text{hyperbolic arctangent} 126
\text{Taylor polynomial, taylor( )} 126
\text{taylor( ), Taylor polynomial} 126
\text{tCdf( ), student-}\text{t} \text{distribution probability} 126
\text{tCollect( ), trigonometric collection} 127
\text{templates}
\text{absolute value} 3
\text{definite integral} 5
\text{derivative or nth derivative} 5
\text{e exponent} 2
\text{exponent} 1
\text{first derivative} 5
\text{fraction} 1
\text{indefinite integral} 5
\text{limit} 6
\text{Log} 2
\text{matrix} (1 \times 2) 4
\text{matrix} (2 \times 1) 4
\text{matrix} (2 \times 2) 3
\text{matrix} (m \times n) 4
\text{nth root} 1
\text{piecewise function} (2-piece) 2
\text{piecewise function} (N-piece) 2
\text{product} (\prod) 4
\text{second derivative} 5
\text{square root} 1
\text{sum} (\Sigma) 4
\text{system of equations (2-equation)} 3
\text{system of equations (N-equation)} 3
\text{test for void, isVoid( )} 61
\text{Test_2S, 2-sample F test} 51
\text{tExpand( ), trigonometric expansion} 127
\text{Text command} 127
\text{time value of money, Future Value} 132
\text{time value of money, Interest} 132
time value of money, number of payments 132
time value of money, payment amount 132
time value of money, present value 132
tInterval_2Samp, two-sample $t$ confidence interval 128
tInterval, $t$ confidence interval 128
$\text{tmpCnv}()$ 129
tPdf(), student-$t$ probability density 129
trace( ) 130
transpose, $^T$ 124
trigonometric collection, tCollect( ) 127
trigonometric expansion, tExpand( ) 127
Try, error handling command 130
tTest_2Samp, two-sample $t$ test 131
tTest, $t$ test 131
TVM arguments 132
tvmFV() 132
tvml() 132
tvmN() 132
tvmPmt() 132
tvmPV() 132
TwoVar, two-variable results 133
two-variable results, TwoVar 133

U
underscore, _ 157
unit vector, unitV( ) 134
units
convert 158
unitV( ), unit vector 134
unLock(), unit vector or variable group 135
unlocking variables and variable groups 135
user-defined functions 34
user-defined functions and programs 34, 35

V
variable

creating name from a character string 167
variable and functions
copying 21
variables
clear all single-letter 18
delete, DelVar 35
local, Local 70
variables, locking and unlocking 53, 70, 135
variance, variance() 135
varPop() 135
varSamp(), sample variance 135
vectors
cross product, crossP() 26
cylindrical vector display, $\text{Cylind}$ 31
dot product, dotP() 39
unit, unitV( ) 134
void elements 162
void elements, remove 35
void, test for 61

W
warnCodes(), Warning codes 136
warning codes and messages 174
when, when() 136
when(), when 136
While, while 136
while, While 136
with, | 159
within string, inString() 59

X
x2, square 146
XNOR 150
xor, Boolean exclusive or 137

Z
zeroes, zeroes() 137
zeroes(), zeroes 137
zInterval_1Prop, one-proportion $z$ confidence interval 139
zInterval_2Prop, two-proportion $z$ confidence interval 140
zInterval_2Samp, two-sample $z$ confidence interval 140
zInterval, $z$ confidence interval 139
zTest 141
zTest_1Prop, one-proportion $z$ test 141
zTest_2Prop, two-proportion $z$ test 142
zTest_2Samp, two-sample $z$ test 142