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.

© 2006 - 2019 Texas Instruments Incorporated
Contents

Expression Templates ................................................................. 1

Alphabetical Listing ................................................................. 7

A ................................................................. 7
B ................................................................. 15
C ................................................................. 19
D ................................................................. 35
E ................................................................. 43
F ................................................................. 51
G ................................................................. 58
I ................................................................. 68
L ................................................................. 76
M ................................................................. 91
N ................................................................. 99
O ................................................................. 107
P ................................................................. 109
Q ................................................................. 116
R ................................................................. 119
S ................................................................. 133
T ................................................................. 152
U ................................................................. 164
V ................................................................. 164
W ................................................................. 165
X ................................................................. 167
Z ................................................................. 168

Symbols ............................................................................. 174

TI-Nspire™ CX II - Draw Commands ......................................... 197
Graphics Programming .............................................................. 197
Graphics Screen ................................................................. 197
Default View and Settings ...................................................... 198
Graphics Screen Errors Messages ............................................. 199
Invalid Commands While in Graphics Mode ......................... 199
C ................................................................. 200
D ................................................................. 201
F ................................................................. 204
G ................................................................. 206
P ................................................................. 207
S ................................................................. 209
U ................................................................. 211
Empty (Void) Elements ................................................................. 212
Shortcuts for Entering Math Expressions ................................. 214
EOS™ (Equation Operating System) Hierarchy .......................... 216
TI-Nspire CX II - TI-Basic Programming Features ..................... 218
  Auto-indentation in Programming Editor ................................. 218
  Improved Error Messages for TI-Basic ................................. 218
Constants and Values ................................................................. 221
Error Codes and Messages .......................................................... 222
Warning Codes and Messages .................................................... 230
General Information ................................................................. 232
  Online Help ................................................................. 232
  Contact TI Support .......................................................... 232
  Service and Warranty Information ........................................ 232
Index ...................................................................................... 233
Expression Templates

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

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

<table>
<thead>
<tr>
<th>Fraction template</th>
<th>( \text{ctrl} ) (+) keys</th>
</tr>
</thead>
</table>
| ![Fraction template](image) | Example: \[
\begin{array}{c}
12 \\
8.2 \\
\end{array} \begin{array}{c}
3 \\
4 \\
\end{array}
\] |
| **Note:** See also \( \div (\text{divide}), \) page 176. |

<table>
<thead>
<tr>
<th>Exponent template</th>
<th>( \text{^} ) key</th>
</tr>
</thead>
</table>
| ![Exponent template](image) | Example: \[
2^3 \\
\] |
| **Note:** Type the first value, press \( \text{^} \), and then type the exponent. To return the cursor to the baseline, press right arrow \( \downarrow \). |
| **Note:** See also \^ (power), page 177. |

<table>
<thead>
<tr>
<th>Square root template</th>
<th>( \text{ctrl} ) ( \times^2 ) keys</th>
</tr>
</thead>
</table>
| ![Square root template](image) | Example: \[
\begin{array}{c}
\sqrt{4} \\
\sqrt{9,\alpha,4} \\
\sqrt{9,16,4} \\
\end{array} \begin{array}{c}
2 \\
\{3,\sqrt{\alpha},2\} \\
\{3,4,2\} \\
\end{array}
\] |
| **Note:** See also \( \sqrt{} (\text{square root}), \) page 186. |

<table>
<thead>
<tr>
<th>Nth root template</th>
<th>( \text{ctrl} ) ( \text{^} ) keys</th>
</tr>
</thead>
</table>
| ![Nth root template](image) | Example: \[
\sqrt[3]{2} \\
\sqrt[4]{9,\alpha,4} \\
\sqrt[4]{9,16,4} \\
\] |
| **Note:** See also root\( () \), page 130. |
**Nth root template**

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

**e exponent template**

\[ e^{\gamma} \]

Natural exponential \( e \) raised to a power

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

**Log template**

\[ \log_{\gamma}(\gamma) \]

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

*Note:* See also \( \log() \), page 87.

**Piecewise template (2-piece)**

\[ \left\{ \begin{array}{l} \gamma, \quad \gamma \\ \gamma, \quad \gamma \end{array} \right. \]

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

*Note:* See also \( \text{piecewise()} \), page 111.

**Piecewise template (N-piece)**

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

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

Create Piecewise Function

<table>
<thead>
<tr>
<th>Piecewise Function</th>
<th>Number of function pieces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: See also `piecewise()`, page 111.

### System of 2 equations template

\[
\begin{align*}
\text{Example:} & \\
& \text{solve} \begin{cases}
  x+y=0 \\
  x-y=5
\end{cases} \\
& x=\frac{5}{2} \text{ and } y=\frac{-5}{2}
\end{align*}
\]

\[
\begin{align*}
& \text{solve} \begin{cases}
  y=3x^2 - 2 \\
  x+2y=-1
\end{cases} \\
& x=\frac{-3}{2} \text{ and } y=\frac{1}{4} \text{ or } x=1 \text{ and } y=-1
\end{align*}
\]

Note: See also `system()`, page 151.

### System of N equations template

Create a System of Eq...

<table>
<thead>
<tr>
<th>System of Equations</th>
<th>Number of equations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Note: See also `system()`, page 151.

### Absolute value template

Note: See also `abs()`, page 7.

Example:

\[
\begin{align*}
& \{ 2, -3, 4, -4^3 \} \\
& \{ 2, 3, 4, 64 \}
\end{align*}
\]
**dd°mm's'ss'' template**

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

**Matrix template (2 x 2)**

```
[0 0]
[0 0]
```

Creates a 2 x 2 matrix.

**Matrix template (1 x 2)**

```
[0 0]
```

Example:

```
crossP([1 2], [3 4])
```

**Matrix template (2 x 1)**

```
[0 0]
```

Example:

```
[5 0.01]
```

**Matrix template (m x n)**

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

Example:

```
diag([4 2 6])
```

---

4  *Expression Templates*
Matrix template (m x n)  
Note: If you create a matrix with a large number of rows and columns, it may take a few moments to appear.

Sum template (Σ)  
\[ \sum_{i=0}^{\infty} (\cdot) \]  
Example:  
\[ \sum_{n=3}^{7} (n) = 25 \]  
Note: See also Σ() (sumSeq), page 187.

Product template (Π)  
\[ \prod_{i=0}^{\infty} (\cdot) \]  
Example:  
\[ \prod_{n=1}^{5} \left( \frac{1}{n} \right) = \frac{1}{120} \]  
Note: See also Π() (prodSeq), page 186.

First derivative template  
\[ \frac{d}{dx} (\cdot) \]  
The first derivative template can be used to calculate first derivative at a point numerically, using auto differentiation methods.  
Note: See also d() (derivative), page 185.

Second derivative template  
\[ \frac{d^2}{dx^2} (\cdot) \]  
Example:
Second derivative template

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

Note: See also `d()` (derivative), page 185.

| Example: |  
|-------------------------------------|---|
| \[
\frac{d^2}{dx^2}(x^3)_{x=3}
\] | 18 |

Definite integral template

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

Note: See also `nInt()`, page 102.

| Example: |  
|-------------------------------------|---|
| \[
\int_{0}^{10} x^2 \, dx
\] | 333.333 |
Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, page 174. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

**abs()**

*abs(Value1) ⇒ value*

*abs(List1) ⇒ list*

*abs(Matrix1) ⇒ matrix*

Returns the absolute value of the argument.

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

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

**amortTbl()**

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

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

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

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 162.

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

roundValue specifies the number of decimal places for rounding. Default=2.
amortTbl()
The columns in the result matrix are in this order: Payment number, amount paid to interest, amount paid to principal, and balance.

The balance displayed in row \( n \) is the balance after payment \( n \).

You can use the output matrix as input for the other amortization functions \( \Sigma \text{Int}() \) and \( \Sigma \text{Prn}() \), page 187, and \( \text{bal}() \), page 15.

and

\[
\text{BooleanExpr1 and BooleanExpr2 } \Rightarrow \text{ Boolean expression}
\]

\[
\text{BooleanList1 and BooleanList2 } \Rightarrow \text{ Boolean list}
\]

\[
\text{BooleanMatrix1 and BooleanMatrix2 } \Rightarrow \text{ Boolean matrix}
\]

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

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

Comparing 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).

angle()

\[
\text{angle(Value1)} \Rightarrow \text{ value}
\]

In Degree angle mode:

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0h7AC36</td>
<td>0h3D5F</td>
</tr>
<tr>
<td>0h2C16</td>
<td></td>
</tr>
</tbody>
</table>

Important: Zero, not the letter O.

In Hex base mode:

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b100101</td>
<td>0b100</td>
</tr>
</tbody>
</table>

In Bin base mode:

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>0b100</td>
</tr>
</tbody>
</table>

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()  

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

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

In Gradian angle mode:

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

In Radian angle mode:

\[
\begin{aligned}
\text{angle}(1+i) & \Rightarrow 0.785398 \\
\text{angle}\{1+2\cdot i,3+0\cdot i,0-4\cdot i\} & \Rightarrow \{1.10715,0,-1.5708\} \\
\text{angle}\{1+2\cdot i,3+0\cdot i,0-4\cdot i\} & \Rightarrow \left\{\frac{\pi}{2},\tan^{-1}\left(\frac{1}{2}\right),0,-\frac{\pi}{2}\right\}
\end{aligned}
\]

\[\text{angle}(List1) \Rightarrow list\]
\[\text{angle}(Matrix1) \Rightarrow matrix\]

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

ANOVA  

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

Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the stat.results variable. (page 146)

*Flag=0* for Data, *Flag=1* for Stats

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>Value of the F statistic</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 of the groups</td>
</tr>
<tr>
<td>stat.SS</td>
<td>Sum of squares of the groups</td>
</tr>
<tr>
<td>stat.MS</td>
<td>Mean squares for the groups</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSErr</td>
<td>Sum of squares of the errors</td>
</tr>
</tbody>
</table>
### ANOVA2way

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

Computes a two-way analysis of variance for comparing the means of two to 10 populations. A summary of results is stored in the `stat.results` variable. (See page 146.)

*LevRow* = 0 for Block

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

**Outputs:** Block Design

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.F</code></td>
<td>F statistic of the column factor</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.df</code></td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td><code>stat.SS</code></td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td><code>stat.MS</code></td>
<td>Mean squares for column factor</td>
</tr>
<tr>
<td><code>stat.FBlock</code></td>
<td>F statistic for factor</td>
</tr>
<tr>
<td><code>stat.PValBlock</code></td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td><code>stat.dfBlock</code></td>
<td>Degrees of freedom for factor</td>
</tr>
<tr>
<td><code>stat.SSBlock</code></td>
<td>Sum of squares for factor</td>
</tr>
<tr>
<td><code>stat.MSBlock</code></td>
<td>Mean squares for factor</td>
</tr>
<tr>
<td><code>stat.dfError</code></td>
<td>Degrees of freedom of the errors</td>
</tr>
</tbody>
</table>
### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>

### COLUMN FACTOR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.Fcol</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.FRow</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.FInteract</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**

<table>
<thead>
<tr>
<th>Ans ⇒ value</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>56+4</td>
<td>60</td>
</tr>
<tr>
<td>60+4</td>
<td>64</td>
</tr>
</tbody>
</table>

**approx()**

approx(\(Value\)) ⇒ number

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

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

approx(List\(l\)) ⇒ list
approx(Matrix\(l\)) ⇒ matrix

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

**approxFraction()**

\(Value\) \(\text{\(\downarrow\) apprxFraction([[\(Tol\)]]) ⇒ value}\)

\(List\) \(\text{\(\downarrow\) apprxFraction([[\(Tol\)]]) ⇒ list}\)

\(Matrix\) \(\text{\(\downarrow\) apprxFraction([[\(Tol\)]]) ⇒ matrix}\)

Returns the input as a fraction, using a tolerance of \(Tol\). If \(Tol\) is omitted, a tolerance of 5.E-14 is used.
Note: You can insert this function from the computer keyboard by typing @>approxFraction(...).

approxRational()

\[ \text{approxRational}(Value[, Tol]) \Rightarrow \text{value} \]
\[ \text{approxRational}(List[, Tol]) \Rightarrow \text{list} \]
\[ \text{approxRational}(Matrix[, Tol]) \Rightarrow \text{matrix} \]

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

arccos()  
See \( \cos^{-1}(\cdot) \), page 26.

arccosh()  
See \( \cosh^{-1}(\cdot) \), page 27.

arccot()  
See \( \cot^{-1}(\cdot) \), page 28.

arccoth()  
See \( \coth^{-1}(\cdot) \), page 29.

arccsc()  
See \( \csc^{-1}(\cdot) \), page 31.

arccsch()  
See \( \csch^{-1}(\cdot) \), page 32.
arcsec()  See sec⁻¹(), page 134.

arcsech()  See sech⁻¹(), page 134.

arcsin()  See sin⁻¹(), page 142.

arcsinh()  See sinh⁻¹(), page 143.

arctan()  See tan⁻¹(), page 153.

arctanh()  See tanh⁻¹(), page 154.

augment()  

augment(List1, List2) ⇒ list

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

augment(Matrix1, Matrix2) ⇒ matrix

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

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

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

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

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

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

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

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

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

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

**B**

**bal()**

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

bal(NPmt,amortTable) ⇒ value  

Amortization function that calculates schedule balance after a specified payment.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 162.

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

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 162.
If you omit \( Pmt \), it defaults to 
\[ Pmt = \text{tvmPmt}(N, I, PV, FV, PpY, CpY, PmtAt) \]
If you omit \( FV \), it defaults to \( FV = 0 \).
The defaults for \( PpY \), \( CpY \), and \( PmtAt \) are the same as for the TVM functions.

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

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

**Note:** See also \( \text{ΣInt}() \) and \( \text{ΣPrn}() \), page 187.

### Base2

**Integer1 \( \text{Base2} \Rightarrow \text{integer} **

**Note:** You can insert this operator from the computer keyboard by typing \( @>\text{Base2} \).

Converts \( \text{Integer1} \) to a binary number. Binary or hexadecimal numbers always have a \( 0b \) or \( 0h \) prefix, respectively. Use a zero, not the letter \( O \), followed by \( b \) or \( h \).

\( 0b \) \( \text{binaryNumber} \)
\( 0h \) \( \text{hexadecimalNumber} \)

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

Without a prefix, \( \text{Integer1} \) 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,

\( \text{1} \) is displayed as
\( 0hFFFFFFFFFFFFFF \) in Hex base mode
\( 0b111...111 \) (64 1’s) in Binary base mode

\( \text{2}^{63} \) 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
0h7FFFFFFF00000000 in Hex base mode
0b111...111 (64 1’s) in Binary base mode

**Base10**

**Integer1 ▶ Base10 ⇒ integer**

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

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

\(0b\) binaryNumber
\(0h\) hexadecimalNumber

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

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

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

**Base16**

**Integer1 ▶ Base16 ⇒ integer**

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

\(256\) Base16
\(0b11110001111\) Base16

\(0h100\)
\(0hF0F\)
Converts \textit{Integer1} to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

\begin{align*}
0b \text{ } & \text{binaryNumber} \\
0h \text{ } & \text{hexadecimalNumber}
\end{align*}

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, \textit{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 \textbf{\textit{Base2}}, page 16.

\textbf{\textit{binomCdf()}}

\textit{binomCdf(n,p) \Rightarrow} \text{list}

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

\textit{binomCdf(n,p,upBound) for P(0 \leq X \leq \text{upBound}) \Rightarrow} \text{number if upBound is a number, list if upBound is a list}

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

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

\textbf{\textit{binomPdf()}}

\textit{binomPdf(n,p) \Rightarrow} \text{list}

\textit{binomPdf(n,p,XVal) \Rightarrow} \text{number if \textit{XVal} is a number, list if \textit{XVal} is a list}
binomPdf() Computes a probability for the discrete binomial distribution with \( n \) number of trials and probability \( p \) of success on each trial.

C

ceiling(\( Value \)) \( \Rightarrow \) \( value \)

Returns the nearest integer that is \( \geq \) the argument.

The argument can be a real or a complex number.

Note: See also floor().

\[
\begin{align*}
\text{ceiling}(\{3.1,1.2,5\}) & \Rightarrow \{-3,1,3\} \\
\text{ceiling}\left(\begin{array}{cc}
0 & -3.2 \cdot i \\
1.3 & 4
\end{array}\right) & \Rightarrow \begin{array}{cc}
0 & -3 \cdot i \\
2 & 4
\end{array}
\end{align*}
\]

centralDiff()

\[
\begin{align*}
\text{centralDiff}(Expr,Var [=Value][,Step]) & \Rightarrow expression \\
\text{centralDiff}(Expr,Var [,Step]) & | Var=Value \Rightarrow expression \\
\text{centralDiff}(Expr,Var [=Value][,List]) & \Rightarrow list \\
\text{centralDiff}(List,Var [=Value][,Step]) & \Rightarrow list \\
\text{centralDiff}(Matrix,Var [=Value][,Step]) & \Rightarrow matrix
\end{align*}
\]

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.
**centralDiff()**

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

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

**char()**

\[ \text{char}(\text{Integer}) \Rightarrow \text{character} \]

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

**χ²2way**

\[ \chi^2 \text{2way} \text{ obsMatrix} \]

**chi22way obsMatrix**

Computes a χ² test for association on the two-way table of counts in the observed matrix *obsMatrix*. A summary of results is stored in the *stat.results* variable. (page 146)

For information on the effect of empty elements in a matrix, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.χ²</td>
<td>Chi square stat: sum (observed - expected)²/expected</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 chi square statistics</td>
</tr>
<tr>
<td>stat.ExpMat</td>
<td>Matrix of expected elemental count table, assuming null hypothesis</td>
</tr>
<tr>
<td>stat.CompMat</td>
<td>Matrix of elemental chi square statistic contributions</td>
</tr>
</tbody>
</table>

**χ²Cdf()**

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

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

Computes the $\chi^2$ distribution probability
between \text{lowBound} \text{ and } \text{upBound} \text{ for the}
specified degrees of freedom \text{df}.

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

For information on the effect of empty
elements in a list, see “Empty (Void)
Elements,” page 212.

$\chi^2 \text{GOF}$

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

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

Performs a test to confirm that sample data
is from a population that conforms to a
specified distribution. \text{obsList} is a list of
counts and must contain integers. A
summary of results is stored in the
\text{stat.results} variable. (See page 146.)

For information on the effect of empty
elements in a list, see “Empty (Void)
Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.}x^2</td>
<td>Chi square stat: $\sum((\text{observed} - \text{expected})^2/\text{expected}$</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 for the chi square statistics</td>
</tr>
<tr>
<td>\text{stat.ComplList}</td>
<td>Elemental chi square statistic contributions</td>
</tr>
</tbody>
</table>

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

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

\text{chi2Pdf}(XVal, df) \Rightarrow \text{number}$
if $XVal$ is a
number, \text{list} if $XVal$ is a list
**χ²Pdf()**

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,” page 212.

**ClearAZ**

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

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See **unLock**, page 164.

---

**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 110, and **Try**, page 158.

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

---

For an example of **ClrErr**, See Example 2 under the **Try** command, page 158.
colAugment()  

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

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

<table>
<thead>
<tr>
<th></th>
<th>1 2</th>
<th>3 4</th>
<th>5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>1 2</td>
<td>3 4</td>
<td>5 6</td>
</tr>
<tr>
<td>m2</td>
<td>1 2</td>
<td>3 4</td>
<td>5 6</td>
</tr>
</tbody>
</table>

colDim()  

**colDim**(*Matrix*) ⇒ expression

Returns the number of columns contained in *Matrix*.

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

<table>
<thead>
<tr>
<th></th>
<th>0 1 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 4 5</td>
</tr>
<tr>
<td>colDim</td>
<td>3</td>
</tr>
</tbody>
</table>

colNorm()  

**colNorm**(*Matrix*) ⇒ expression

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

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

<table>
<thead>
<tr>
<th></th>
<th>1 -2 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 5 -6</td>
</tr>
<tr>
<td>colNorm(mat)</td>
<td>9</td>
</tr>
</tbody>
</table>

conj()  

**conj**(*Value1*) ⇒ value

**conj**(*List1*) ⇒ list

**conj**(*Matrix1*) ⇒ matrix

Returns the complex conjugate of the argument.

| conj(1+2*i) | 1-2*i |
| conj(2 1-3*i) | 2 1+3*i |
| conj(i -7) | i -7 |
**constructMat()**

### constructMat

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

Returns a matrix based on the arguments.

- \( \text{Expr} \) is an expression in variables \( \text{Var1} \) and \( \text{Var2} \). Elements in the resulting matrix are formed by evaluating \( \text{Expr} \) for each incremented value of \( \text{Var1} \) and \( \text{Var2} \).

- \( \text{Var1} \) is automatically incremented from 1 through \( \text{numRows} \). Within each row, \( \text{Var2} \) is incremented from 1 through \( \text{numCols} \).

---

### CopyVar

#### CopyVar \( \text{Var1}, \text{Var2} \)

**Catalog >**

<table>
<thead>
<tr>
<th>( \text{Var1} )</th>
<th>( \text{Var2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(x) = \frac{1}{x} )</td>
<td>( \text{Done} )</td>
</tr>
<tr>
<td>( b(x) = x^2 )</td>
<td>( \text{Done} )</td>
</tr>
</tbody>
</table>

- \( \text{Var1} \) is the name of an existing user-defined function, copies the definition of that function to function \( \text{Var2} \). Function \( \text{Var1} \) must be defined.

- \( \text{Var1} \) must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.

- \( \text{Var2} \) must be the name of an existing variable group, such as the statistics \( \text{stat.nn} \) results, or variables created using the \( \text{LibShortcut()} \) function. If \( \text{Var2} \) 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 \( \text{Var2} \) are locked, all members of \( \text{Var2} \) are left unchanged.

---

**Calculated Values**

- \( \text{aa.a} := 45 \)
- \( \text{aa.b} := 6.78 \)
- \( \text{bb.a} := 0 \)
- \( \text{bb.b} := 0 \)

---

24  Alphabetic Listing
**corrMat()**

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

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

**cos()**

**cos(Value1) ⇒ value**

**cos(List1) ⇒ list**

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

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

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

**cos(squareMatrix1) ⇒ squareMatrix**

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

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

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

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

Form the matrices:

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

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

In **Degree angle mode**:

\[
\begin{align*}
\cos(\frac{\pi}{4}) & \Rightarrow 0.707107 \\
\cos(45) & \Rightarrow 0.707107 \\
\cos(\{0,60,90\}) & \Rightarrow \{1.0, 0.5, 0.0\}
\end{align*}
\]

In **Gradian angle mode**:

\[
\begin{align*}
\cos(\{0,50,100\}) & \Rightarrow \{1.0, 0.707107, 0.0\}
\end{align*}
\]

In **Radian angle mode**:

\[
\begin{align*}
\cos(\frac{\pi}{4}) & \Rightarrow 0.707107 \\
\cos(45°) & \Rightarrow 0.707107
\end{align*}
\]

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

\[
\begin{bmatrix}
0.212493 & 0.205064 & 0.121389 \\
0.160871 & 0.259042 & 0.037126 \\
0.248079 & -0.090153 & 0.218972 \\
\end{bmatrix}
\]
Then $A = X \cdot B \cdot X^{-1}$ and $f(A) = X \cdot f(B) \cdot X^{-1}$. For example, $\cos(A) = X \cdot \cos(B) \cdot X^{-1}$ 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()**

$\cos^{-1}(Value1) \Rightarrow value$

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

$\cos^{-1}(Value1)$ returns the angle whose cosine is $Value1$.

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

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

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

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

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

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

In Degree angle mode:

```
\cos^{-1}(1) 0.
```

In Gradian angle mode:

```
\cos^{-1}(0) 100.
```

In Radian angle mode:

```
\cos^{-1}\{(0,0.2,0.5)\}  \{1.5708,1.36944,1.0472\}
```

In Radian angle mode and Rectangular Complex Format:

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

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

To see the entire result, press **▲** and then use **◄** and **►** to move the cursor.
**cosh()**

\[
cosh(\text{Value1}) \Rightarrow \text{value} \\
cosh(\text{List1}) \Rightarrow \text{list}
\]

cosh(\text{Value1}) returns the hyperbolic cosine of the argument.

cosh(\text{List1}) returns a list of the hyperbolic cosines of each element of \text{List1}.

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

Returns the matrix hyperbolic cosine of \text{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.

**cosh⁻¹()**

\[
cosh⁻¹(\text{Value1}) \Rightarrow \text{value} \\
cosh⁻¹(\text{List1}) \Rightarrow \text{list}
\]

cosh⁻¹(\text{Value1}) returns the inverse hyperbolic cosine of the argument.

cosh⁻¹(\text{List1}) returns a list of the inverse hyperbolic cosines of each element of \text{List1}.

\textbf{Note:} You can insert this function from the keyboard by typing \texttt{arccosh(...)}. 

cosh⁻¹(\text{squareMatrix1}) \Rightarrow \text{squareMatrix}

Returns the matrix inverse hyperbolic cosine of \text{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.
**cot()**

\[ \cot(Value1) \Rightarrow value \]
\[ \cot(List1) \Rightarrow list \]

Returns the cotangent of \( Value1 \) or returns a list of the cotangents of all elements in \( List1 \).

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

- In Degree angle mode:
  \[ \cot(45) \Rightarrow 1. \]

- In Gradian angle mode:
  \[ \cot(50) \Rightarrow 1. \]

- In Radian angle mode:
  \[ \cot\{1,2,1,3\} \Rightarrow \{0.642093, 0.584848, 7.01525\} \]

**cot⁻¹()**

\[ \cot^{-1}(Value1) \Rightarrow value \]
\[ \cot^{-1}(List1) \Rightarrow list \]

Returns the angle whose cotangent is \( Value1 \) or returns a list containing the inverse cotangents of each element of \( List1 \).

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

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

- In Degree angle mode:
  \[ \cot^{-1}(1) \Rightarrow 45. \]

- In Gradian angle mode:
  \[ \cot^{-1}(1) \Rightarrow 50. \]

- In Radian angle mode:
  \[ \cot^{-1}(1) \Rightarrow .785398 \]

**coth()**

\[ \coth(Value1) \Rightarrow value \]
\[ \coth(List1) \Rightarrow list \]

Returns the hyperbolic cotangent of \( Value1 \) or returns a list of the hyperbolic cotangents of all elements of \( List1 \).
coth⁻¹()Catalog > \[\text{coth}^{-1}(\text{Value1}) \Rightarrow \text{value}\]
\[\text{coth}^{-1}(\text{List1}) \Rightarrow \text{list}\]

Returns the inverse hyperbolic cotangent of \text{Value1} or returns a list containing the inverse hyperbolic cotangents of each element of \text{List1}.

\textbf{Note:} You can insert this function from the keyboard by typing \texttt{arccoth} (...) .

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

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 212.

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

Returns the accumulated count of all elements in \text{List} that meet the specified \text{Criteria}.

\textbf{Criteria} can be:

- A value, expression, or string. For
**countif()**

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 212.

**Note:** See also sumIf(), page 150, and frequency(), page 56.

**Counts the number of elements equal to “def.”**

<table>
<thead>
<tr>
<th>countIf({1,3,5,7,9},?&lt;5)</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts 1 and 3.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>countIf({1,3,5,7,9},2&lt;?&lt;8)</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts 3, 5, and 7.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>countIf({1,3,5,7,9},?&lt;4 or ?&gt;6)</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts 1, 3, 7, and 9.</td>
<td></td>
</tr>
</tbody>
</table>

**cPolyRoots()**

**cPolyRoots(Poly,Var) ⇒ list**

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

Poly must be a polynomial in expanded form in one variable. Do not use unexpanded forms such as y²*y+1 or x*x+2*x+1

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

**Note:** See also polyRoots(), page 113.

**crossP()**

**crossP(List1, List2) ⇒ list**

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

<table>
<thead>
<tr>
<th>crossP({0,1,2,2, 5},{1,0,5,0})</th>
<th>{2,5,5,2}</th>
</tr>
</thead>
</table>
crossP()

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

crossP($Vector1$, $Vector2$) $\Rightarrow$ vector

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

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

csc()

\begin{align*}
csc(Value1) & \Rightarrow value \\
csc(List1) & \Rightarrow list
\end{align*}

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

\begin{align*}
\text{In Degree angle mode:} \\
csc(45) & \quad 1.41421
\end{align*}

\begin{align*}
\text{In Gradian angle mode:} \\
csc(50) & \quad 1.41421
\end{align*}

\begin{align*}
\text{In Radian angle mode:} \\
csc\left(1, \frac{\pi}{2}, \frac{\pi}{3}\right) & \quad \{1.18841,1.1547\}
\end{align*}

csc^{-1}()

\begin{align*}
csc^{-1}(Value1) & \Rightarrow value \\
csc^{-1}(List1) & \Rightarrow list
\end{align*}

Returns the angle whose cosecant is $Value1$ or returns a list containing the inverse cosecants of each element of $List1$.

\begin{align*}
\text{In Degree angle mode:} \\
csc^{-1}(1) & \quad 90.
\end{align*}

\begin{align*}
\text{In Gradian angle mode:} \\
csc^{-1}(1) & \quad 100.
\end{align*}

\begin{align*}
\text{In Radian angle mode:} \\
csc^{-1}\{1,4,6\} & \quad \{1.5708,0.25268,0.167448\}
\end{align*}
\[ \text{csch}(Value1) \Rightarrow value \]
\[ \text{csch}(List1) \Rightarrow list \]

Returns the hyperbolic cosecant of \( Value1 \) or returns a list of the hyperbolic cosecants of all elements of \( List1 \).

\[ \text{csch}^{-1}(Value) \Rightarrow value \]
\[ \text{csch}^{-1}(List1) \Rightarrow list \]

Returns the inverse hyperbolic cosecant of \( Value1 \) or returns a list containing the inverse hyperbolic cosecants of each element of \( List1 \).

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

---

**CubicReg**

\[ \text{CubicReg} \](\text{ \textit{X}}, \text{ \textit{Y}[\text{ \textit{\textit{\textit{Freq}}} \text{[]} \text{\textit{, \textit{\textit{\textit{Category}}} \text{[]} \text{\textit{, \textit{\textit{\textit{Include}}} \text{[]}}}})}])

Computes the cubic polynomial regression
\[ y = a \times x^3 + b \times x^2 + c \times x + d \]
on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the `stat.results` variable. (See page 146.)

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 numeric or string category codes for the corresponding \( X \) and \( Y \) data.
Include is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a \cdot x^3 + b \cdot x^2 + c \cdot x + d )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of determination</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>

**cumulativeSum()**

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

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

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

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

An empty (void) element in \( List1 \) or \( Matrix1 \) produces a void element in the resulting list or matrix. For more information on empty elements, see page 212.
Cycle

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

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

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

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

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

```
g() 5000
```

► Cylind

**Vector ► Cylind**

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

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

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

```
[2 2 3]►Cylind
```

```
2.82843 \angle 0.785398 3.
```
**dbd()**

\[ \text{dbd}(\text{date1}, \text{date2}) \Rightarrow \text{value} \]

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

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

\text{date1} and \text{date2} must be between the years 1950 through 2049.

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

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

DDMM.YY (format use commonly in Europe)

**DD**

\[ \text{Expr1} \uparrow \text{DD} \Rightarrow \text{valueList1} \]

\[ \uparrow \text{DD} \Rightarrow \text{listMatrix1} \]

\[ \uparrow \text{DD} \Rightarrow \text{matrix} \]

**Note:** You can insert this operator from the computer keyboard by typing \(\uparrow \text{DD}\).

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.

**Catalog >**

| \text{dbd}([12.3103,1.0104]) | 1 |
| \text{dbd}([1.0107,6.0107]) | 151 |
| \text{dbd}([31.12,0.3,101.04]) | 1 |
| \text{dbd}([101.07,106.07]) | 151 |

In Degree angle mode:

\[ \frac{1.5}{1.5°} \]

\[ \frac{45°22°14.3°}{45.3706°} \]

\[ \frac{\{45°22°14.3°,60°00°\}}{45.3706°,60°} \]

In Gradian angle mode:

\[ \frac{1.0°}{9.0°} \]

In Radian angle mode:

\[ \frac{1.5°}{85.9437°} \]
Decom

Number1 ▶ Decimal ⇒ value

List1 ▶ Decimal ⇒ value

Matrix1 ▶ Decimal ⇒ value

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

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

Define

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

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

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

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

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

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

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

Define g(x,y)=2·x–3·y
Define h(x)=when(x<2,2·x–3,2·x+3)

Define g(1,2)  Define h(-3)
Define g(3,4)  Define h(4)
Define

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

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

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

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

---

Define LibPriv

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

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

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

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

Note: See also Define, page 36, and Define LibPub, page 37.

---

Define LibPub

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

Define LibPub Function(Param1, Param2,
Define LibPub

```plaintext
(... = 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 36, and Define LibPriv, page 37.

deltaList()

See ΔList(), page 83.

DelVar

```plaintext
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 164.
**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.

### DelVoid()

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

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

For more information on empty elements, see page 212.

### det()

\[
\text{det(squareMatrix[, Tolerance])} \Rightarrow \text{expression}
\]

Returns the determinant of squareMatrix.

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

- If you use \[\text{ctrl+enter}\] 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: \[5E^{-14} \text{max(dim(squareMatrix))} \times \text{rowNorm(squareMatrix)}\]
**diag()**

\[
\text{diag}(\text{List}) \Rightarrow \text{matrix} \\
\text{diag}(\text{rowMatrix}) \Rightarrow \text{matrix} \\
\text{diag}(\text{columnMatrix}) \Rightarrow \text{matrix}
\]

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

\[
\text{diag}(\text{squareMatrix}) \Rightarrow \text{rowMatrix}
\]

Returns a row matrix containing the elements from the main diagonal of \(\text{squareMatrix}\).

\(\text{squareMatrix}\) must be square.

**dim()**

\[
\text{dim}(\text{List}) \Rightarrow \text{integer} \\
\text{dim}(\text{Matrix}) \Rightarrow \text{list} \\
\text{dim}(\text{String}) \Rightarrow \text{integer}
\]

Returns the dimension of \(\text{List}\).

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

Returns the number of characters contained in character string \(\text{String}\).

**Disp**

\textbf{Disp} \(\text{exprOrString1 [, exprOrString2] ...}\)

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.

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

DispAt \texttt{int, expr1 [,expr2 ...]} ...

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

The line number can be specified as an expression.

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

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

DispAt and Disp can be used within the same program.

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

\begin{tabular}{|l|l|}
\hline
\textbf{Illustrative examples:} & \textbf{Output} \\
\hline
\texttt{Define z()}= & \texttt{z()} \\
\texttt{Prgm} & \\
\texttt{For n,1,3} & \\
\texttt{DispAt 1,"N: ",n} & \textbf{Iteration 1:} \\
\texttt{Disp "Hello"} & \textbf{Line 1: N:1} \\
\texttt{EndFor} & \texttt{Line 2: Hello} \\
\texttt{EndPrgm} & \texttt{Line 3: Hello} \\
\hline
\texttt{Iteration 2:} & \\
\texttt{Line 1: N:2} & \\
\texttt{Line 2: Hello} & \\
\texttt{Line 3: Hello} & \\
\hline
\texttt{Iteration 3:} & \\
\texttt{Line 1: N:3} & \\
\hline
\end{tabular}
### Error conditions:

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

### °DMS

<table>
<thead>
<tr>
<th>Value °DMS</th>
<th>In Degree angle mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>List °DMS</td>
<td>(45.371)°DMS 45°22'15.6&quot;</td>
</tr>
<tr>
<td>Matrix °DMS</td>
<td>({45.371,60})°DMS {45°22'15.6°,60°}</td>
</tr>
</tbody>
</table>
Interprets the argument as an angle and displays the equivalent DMS (DDDDD°MM'SS.ss'') number. See °, ', '" on page 191 for DMS (degree, minutes, seconds) format.

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

---

**dotP()**

\[ \text{dotP}(List1, List2) \Rightarrow expression \]

Returns the “dot” product of two lists.

\[ \text{dotP}(Vector1, Vector2) \Rightarrow expression \]

Returns the “dot” product of two vectors.

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

See Also: TI-Nspire™ CX II - Draw Commands

---

**E**

\[ e^\text{}(Value1) \Rightarrow value \]

Returns \( e \) raised to the \( Value1 \) power.

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

**Note:** Pressing \( e^\text{} \) to display \( e^\text{} \) 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.

\[ e^A(\text{List1}) \Rightarrow \text{list} \]

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

\[ e^A(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

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 \( \cos() \).

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

\[ e^A() \]

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

\[ \begin{bmatrix} 782.209 & 559.617 & 456.509 \\ 680.546 & 488.795 & 396.521 \\ 524.929 & 371.222 & 307.879 \end{bmatrix} \]

\[ e^{\{1,1,0.5\}} \Rightarrow \{2.71828,2.71828,1.64872\} \]

\[ \text{eff()} \]

\[ \text{eff(} \text{nominalRate}, \text{CpY} \text{)} \Rightarrow \text{value} \]

Financial function that converts the nominal interest rate \( \text{nominalRate} \) to an annual effective rate, given \( \text{CpY} \) as the number of compounding periods per year.

\( \text{nominalRate} \) must be a real number, and \( \text{CpY} \) must be a real number \( > 0 \).

Note: See also \( \text{nom()} \), page 103.

\[ \text{eigVc()} \]

\[ \text{eigVc(} \text{squareMatrix} \text{)} \Rightarrow \text{matrix} \]

Returns a matrix containing the eigenvectors for a real or complex \( \text{squareMatrix} \), 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:

\[ V = [x_1, x_2, \ldots, x_n] \]

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

\[ \begin{bmatrix} -0.800906 & 0.767947 & \cdot \\ 0.484029 & 0.5738 & \cdot \\ 0.352512 & 0.262687 & \cdot \end{bmatrix} \]

\[ \text{eigVc(m1)} \]

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

\[ \begin{bmatrix} -0.800906 & 0.767947 & \cdot \\ 0.484029 & 0.5738 & \cdot \\ 0.352512 & 0.262687 & \cdot \end{bmatrix} \]
then \( x_1^2 + x_2^2 + \ldots + x_n^2 = 1 \)

\( \text{squareMatrix} \) 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 eigenvectors are computed via a Schur factorization.

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

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.

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

\[ \text{Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.} \]
See For, page 54.

See Func, page 57.

See If, page 68.

See Loop, page 90.

See Prgm, page 114.

See Try, page 158.

See While, page 167.

euler()  

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

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

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

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

euler(0.001*y-y*(100-y), t, y, {0,100}, 10, 1)

press ▲ and then use ◀ and ▶ to move the cursor.
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.

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

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

\text{ListOfExpr} is a list of right-hand sides that define the system of ODEs (corresponds to the 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{ListOfDepVars0} is a list of initial values for dependent variables.

\text{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}).

\text{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 \(\text{VarStep} / \text{eulerStep}\).
**eval**

**eval**(Expr) ⇒ string

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

The argument Expr must simplify to a real number.

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

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

Reset the blue element to OFF.

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

**eval**( ) argument must simplify to a real number.

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

Program to fade-in the red element

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

Execute the program.

```
fadein()
```

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

```
iostr.SendAns
niostr.GetAns
niostr.GetStrAns
```

**Note:** See also Get (page 59), GetStr (page 66), and Send (page 135).

---

**Exit**

**Catalog**
**Exit**

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

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

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

---

<table>
<thead>
<tr>
<th><strong>exp()</strong></th>
<th><strong>e^x</strong> key</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>exp(Value1) ⇒ value</strong></td>
<td></td>
</tr>
<tr>
<td>Returns <em>e</em> raised to the Value1 power.</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> See also <em>e</em> exponent template, page 2.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><strong>exp(List1) ⇒ list</strong></td>
<td></td>
</tr>
<tr>
<td>Returns <em>e</em> raised to the power of each element in List1.</td>
<td></td>
</tr>
<tr>
<td><strong>exp(squareMatrix1) ⇒ squareMatrix</strong></td>
<td></td>
</tr>
<tr>
<td>Returns the matrix exponential of squareMatrix1. This is not the same as calculating <em>e</em> raised to the power of each element. For information about the calculation method, refer to <em>cos()</em></td>
<td></td>
</tr>
<tr>
<td><em>squareMatrix1</em> must be diagonalizable. The result always contains floating-point numbers.</td>
<td></td>
</tr>
</tbody>
</table>
**expr()**

**expr(String) \Rightarrow expression**

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

<table>
<thead>
<tr>
<th><em>Define cube(x)=x^3</em></th>
<th><em>funcstr</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>cube(2)</em></td>
<td>8</td>
</tr>
</tbody>
</table>

**ExpReg**

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

Computes the exponential regression \( y = a \cdot (b)^x \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the *stat.results* variable. (See page 146.)

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

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

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

*Category* is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>stat.RegEqn</em></td>
<td>Regression equation: ( a \cdot (b)^x )</td>
</tr>
<tr>
<td><em>stat.a, stat.b</em></td>
<td>Regression coefficients</td>
</tr>
<tr>
<td><em>stat.r^2</em></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()**

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

To stop a calculation manually,

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

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

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

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

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

Fill

Fill Value, matrixVar \rightarrow matrix

Replaces each element in variable matrixVar with Value.

matrixVar must already exist.

Fill Value, listVar \rightarrow list

Replaces each element in variable listVar with Value.

listVar must already exist.

FiveNumSummary

FiveNumSummary X[,\{Freq\}[,\{Category,\{Include\}\}]]

Provides an abbreviated version of the 1-variable statistics on list X. A summary of results is stored in the stat.results variable. (See page 146.)

X represents a list containing the data.
**FiveNumSummary**

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

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

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

An empty (void) element in any of the lists *X*, *Freq*, or *Category* results in a void for the corresponding element of all those lists. For more information on empty elements, see page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.MinX</td>
<td>Minimum of <em>x</em> values.</td>
</tr>
<tr>
<td>stat.Q₁X</td>
<td>1st Quartile of <em>x</em>.</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of <em>x</em>.</td>
</tr>
<tr>
<td>stat.Q₃X</td>
<td>3rd Quartile of <em>x</em>.</td>
</tr>
<tr>
<td>stat.MaxX</td>
<td>Maximum of <em>x</em> values.</td>
</tr>
</tbody>
</table>

**floor()**

*floor*(*Value I*) ⇒ *integer*  

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

The argument can be a real or a complex number.

*floor*(*List I*) ⇒ *list*  
*floor*(*Matrix I*) ⇒ *matrix*  

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

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

**Catalog >**

For Var, Low, High [, Step]  
Block  
EndFor

Executes the statements in Block iteratively for each value of Var, from Low to High, in increments of Step.

Var must not be a system variable.

Step can be positive or negative. The default value is 1.

Block can be either a single statement or a series of statements separated with the “;” character.

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

**format()**

**Catalog >**

format(Value[, formatString]) ⇒ string

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

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

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

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

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

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()**

\[
\text{fPart}(Expr) \Rightarrow \text{expression} \\
\text{fPart}(List) \Rightarrow \text{list} \\
\text{fPart}(Matrix) \Rightarrow \text{matrix}
\]

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()**

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

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

**freqTable ► list()**

\[
\text{freqTable} \ ► \text{list}(List,freqIntegerList) \Rightarrow \text{list}
\]

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

\(List\) can be any valid list.
**freqTable►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 212.

**frequency()**

`frequency(List1,binsList) ⇒ list`

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

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

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

Elements of `List1` that cannot be “placed in a bin” are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 212.

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

**Note:** See also `countIf()`, page 29.
**F Test_2Samp**

**F Test_2Samp** List1,List2[,Freq1[,Freq2 [,Hypoth]]]

**F Test_2Samp** List1,List2[,Freq1[,Freq2 [,Hypoth]]]

(Data list input)

**F Test_2Samp** sx1,n1,sx2,n2[,Hypoth]

**F Test_2Samp** sx1,n1,sx2,n2[,Hypoth]

(Summary stats input)

Performs a two-sample F test. A summary of results is stored in the `stat.results` variable. (See page 146.)

For H₀: σ₁ > σ₂, set `Hypoth>0`
For H₀: σ₁ ≠ σ₂ (default), set `Hypoth=0`
For H₀: σ₁ < σ₂, set `Hypoth<0`

For information on the effect of empty elements in a list, see Empty (Void) Elements, page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>Calculated F statistic for the data sequence</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.dfNumer</td>
<td>Numerator degrees of freedom = n1-1</td>
</tr>
<tr>
<td>stat.dfDenom</td>
<td>Denominator degrees of freedom = n2-1</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.x1_bar</td>
<td>Sample means of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.x2_bar</td>
<td>Sample means 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>
</tbody>
</table>

**Func**

**Func**

**Block**

**EndFunc**

Template for creating a user-defined function.
**Func**

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

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

---

**g**

**gcd()**

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

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

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

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

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

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

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

---

**geomCdf()**

\[ \text{geomCdf}(p, \text{lowBound}, \text{upBound}) \Rightarrow \text{number} \]
**geomCdf()**

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

**geomCdf(p,upBound) for P(1 ≤ X ≤ upBound)**
⇒ number if `upBound` is a number, list if `upBound` is a list

Computes a cumulative geometric probability from `lowBound` to `upBound` with the specified probability of success `p`.

For P(X ≤ `upBound`), set `lowBound` = 1.

**geomPdf()**

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

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

**Get**

**Get [promptString,] var[, statusVar]**

**Get [promptString,] func(arg1, ...argn) [, statusVar]**

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

The value must be requested:

- In advance, through a Send "READ ..." command.
  — or —
- By embedding a "READ ..." request as the optional `promptString` argument. This method lets you use a single command to request the value and retrieve it.
Implicit simplification takes place. For example, a received string of "123" is interpreted as a numeric value. To preserve the string, use `GetStr` instead of `Get`.

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

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

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

The program can then use the defined function `func()`.

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

**Note:** See also `GetStr`, page 66 and `Send`, page 135.

### getDenom()

**getDenom(Fraction1) ⇒ value**

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x=5; y=6 )</td>
<td>6</td>
</tr>
<tr>
<td>( \text{getDenom}(\frac{x+2}{y-3}) )</td>
<td>3</td>
</tr>
<tr>
<td>( \text{getDenom}(\frac{2}{7}) )</td>
<td>7</td>
</tr>
<tr>
<td>( \text{getDenom}(\frac{1+y^2+y}{x-y^2}) )</td>
<td>30</td>
</tr>
</tbody>
</table>

### getKey()

**getKey([0|1]) ⇒ returnString**

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

**Example:**
### Example:

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

### Handling of key presses:

<table>
<thead>
<tr>
<th>Handheld Device/Emulator Key</th>
<th>Desktop</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc</td>
<td>Esc</td>
<td>&quot;esc&quot;</td>
</tr>
<tr>
<td>Touchpad - Top click</td>
<td>n/a</td>
<td>&quot;up&quot;</td>
</tr>
<tr>
<td>On</td>
<td>n/a</td>
<td>&quot;home&quot;</td>
</tr>
</tbody>
</table>

| Scratchapps                  | n/a     | "scratchpad"       |
| Touchpad - Left click        | n/a     | "left"             |
| Touchpad - Center click      | n/a     | "center"           |
| Touchpad - Right click       | n/a     | "right"            |
| Doc                          | n/a     | "doc"              |

| Tab                          | Tab     | "tab"              |
| Touchpad - Bottom click      | Down Arrow | "down"             |
| Menu                         | n/a     | "menu"             |

| Ctrl                         | Ctrl    | no return           |
| Shift                        | Shift   | no return           |
| Var                          | n/a     | "var"              |
| Del                          | n/a     | "del"              |

<p>| =                            | =       | &quot;=&quot;                |
| trig                         | n/a     | &quot;trig&quot;             |
| 0 through 9                  | 0-9     | &quot;0&quot; ... &quot;9&quot;        |</p>
<table>
<thead>
<tr>
<th>Handheld Device/Emulator Key</th>
<th>Desktop</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templates</td>
<td>n/a</td>
<td>&quot;template&quot;</td>
</tr>
<tr>
<td>Catalog</td>
<td>n/a</td>
<td>&quot;cat&quot;</td>
</tr>
<tr>
<td>^</td>
<td>^</td>
<td>&quot;^&quot;</td>
</tr>
<tr>
<td>X^2</td>
<td>n/a</td>
<td>&quot;square&quot;</td>
</tr>
<tr>
<td>/ (division key)</td>
<td>/</td>
<td>&quot;/&quot;</td>
</tr>
<tr>
<td>* (multiply key)</td>
<td>*</td>
<td>&quot;*&quot;</td>
</tr>
<tr>
<td>e^x</td>
<td>n/a</td>
<td>&quot;exp&quot;</td>
</tr>
<tr>
<td>10^x</td>
<td>n/a</td>
<td>&quot;10power&quot;</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>&quot;+&quot;</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>&quot;-&quot;</td>
</tr>
<tr>
<td>(</td>
<td>(</td>
<td>&quot;(&quot;</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>&quot;)&quot;</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>&quot;.&quot;</td>
</tr>
<tr>
<td>( )</td>
<td>n/a</td>
<td>&quot;.-&quot; (negate sign)</td>
</tr>
<tr>
<td>Enter</td>
<td>Enter</td>
<td>&quot;enter&quot;</td>
</tr>
<tr>
<td>ee</td>
<td>n/a</td>
<td>&quot;E&quot; (scientific notation E)</td>
</tr>
<tr>
<td>a - z</td>
<td>a-z</td>
<td>alpha = letter pressed (lower case) (&quot;a&quot; - &quot;z&quot;)</td>
</tr>
<tr>
<td>shift a-z</td>
<td>shift a-z</td>
<td>alpha = letter pressed &quot;A&quot; - &quot;Z&quot;</td>
</tr>
<tr>
<td>?!</td>
<td>n/a</td>
<td>&quot;?!!&quot;</td>
</tr>
<tr>
<td>pi</td>
<td>n/a</td>
<td>&quot;pi&quot;</td>
</tr>
<tr>
<td>Flag</td>
<td>n/a</td>
<td>no return</td>
</tr>
<tr>
<td>,</td>
<td>,</td>
<td>&quot;,&quot;</td>
</tr>
<tr>
<td>Return</td>
<td>n/a</td>
<td>&quot;return&quot;</td>
</tr>
<tr>
<td>Handheld Device/Emulator Key</td>
<td>Desktop</td>
<td>Return Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>Space</td>
<td>Space</td>
<td>&quot; &quot; (space)</td>
</tr>
<tr>
<td>Inaccessible</td>
<td>Special Character Keys like @,!,^, etc.</td>
<td>The character is returned</td>
</tr>
<tr>
<td>n/a</td>
<td>Function Keys</td>
<td>No returned character</td>
</tr>
<tr>
<td>n/a</td>
<td>Special desktop control keys</td>
<td>No returned character</td>
</tr>
<tr>
<td>Inaccessible</td>
<td>Other desktop keys that are not available on the calculator while getkey() is waiting for a keystroke. (,.&gt;;:;...)</td>
<td>Same character you get in Notes (not in a math box)</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Event</th>
<th>Device</th>
<th>Desktop - TI-Nspire™ Student Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Poll</td>
<td>Terminate program, handle event</td>
<td>Same as the handheld (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)</td>
</tr>
<tr>
<td>Remote file mgmt</td>
<td>Terminate program, handle event</td>
<td>Same as the handheld. (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)</td>
</tr>
<tr>
<td>End Class</td>
<td>Terminate program, handle event</td>
<td>Support (TI-Nspire™ Student Software, TI-Nspire™ Navigator™ NC Teacher Software-only)</td>
</tr>
<tr>
<td>TI-Innovator™ Hub connect/disconnect</td>
<td>Support - Can successfully issue commands to the TI-Innovator™ Hub. After you</td>
<td></td>
</tr>
</tbody>
</table>

**Alphabetical Listing** 63
exit the program the TI-Innovator™ Hub is still working with the handheld.

getLangInfo()  

getLangInfo() \Rightarrow string

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

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

getLockInfo()  

getLockInfo(Var) \Rightarrow value

Returns the current locked/unlocked state of variable Var.

value =0: Var is unlocked or does not exist.

value =1: Var is locked and cannot be modified or deleted.

See Lock, page 86, and unLock, page 164.
getMode()

getMode(\text{ModeNameInteger}) \Rightarrow \text{value}

\begin{align*}
\text{getMode}(0) & \Rightarrow \text{list} \\
\{1, 7, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1\} \\
\text{getMode}(1) & \Rightarrow 7 \\
\text{getMode}(7) & \Rightarrow 1
\end{align*}

getMode(\text{ModeNameInteger}) \text{ returns a value representing the current setting of the } \text{ModeNameInteger} \text{ mode.}

\text{getMode}(0) \text{ 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 \text{getMode}(0) \rightarrow \text{var}, you can use \text{setMode}(\text{var}) \text{ in a function or program to temporarily restore the settings within the execution of the function or program only. See setMode(), page 137.}

<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</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>
</tbody>
</table>
**getNum()**

*getNum(Fraction1) ⇒ value*

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x:=5; y:=6$</td>
<td>6</td>
</tr>
<tr>
<td>$\frac{x+2}{y-3}$</td>
<td>7</td>
</tr>
<tr>
<td>$\frac{2}{7}$</td>
<td>2</td>
</tr>
<tr>
<td>$\frac{1}{x} + \frac{1}{y}$</td>
<td>11</td>
</tr>
</tbody>
</table>

**GetStr**

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

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

**Note:** See also **Get**, page 59 and **Send**, page 135.

**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".

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1,2,3} \rightarrow temp$</td>
<td>${1,2,3}$</td>
</tr>
<tr>
<td>getType(temp)</td>
<td>&quot;LIST&quot;</td>
</tr>
<tr>
<td>$3 \cdot i \rightarrow temp$</td>
<td>$3 \cdot i$</td>
</tr>
<tr>
<td>getType(temp)</td>
<td>&quot;EXPR&quot;</td>
</tr>
<tr>
<td>DelVar temp</td>
<td>Done</td>
</tr>
<tr>
<td>getType(temp)</td>
<td>&quot;NONE&quot;</td>
</tr>
</tbody>
</table>
getVarInfo() \Rightarrow \text{matrix or string}

getVarInfo(LibNameString) \Rightarrow \text{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, \text{getVarInfo()} returns the string "NONE".

getVarInfo(LibNameString) returns a matrix of information for all library objects defined in library \text{LibNameString}. \text{LibNameString} must be a string (text enclosed in quotation marks) or a string variable.

If the library \text{LibNameString} does not exist, an error occurs.

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

This error could also occur when using \text{Ans} to reevaluate a \text{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:** For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

---

**►Grad**

**Expr1►Grad⇒** *expression*

Converts *Expr1* to gradian angle measure.

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

---

**identity()**

**identity**(Integer)⇒ *matrix*

Returns the identity matrix with a dimension of *Integer*.

*Integer* must be a positive integer.

---

**If**

**If** BooleanExpr

Statement

**If** BooleanExpr Then

Block

EndIf

---

**Catalog >**

Define $g(x)$=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

$g()$

**Done**

---

In Degree angle mode:

(1.5)►Grad

(1.66667)°

---

In Radian angle mode:

(1.5)►Grad

(95.493)°

---

Identity

identity(4)

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

Define $g(x)$=Func

If $x<0$ Then

Return $x^2$

EndIf

EndFunc

$g(-2)$

**Done**

---

**Catalog >**

**Catalog >**

**Catalog >**

---

68  Alphabetical Listing
If \( \text{BooleanExpr} \) evaluates to true, executes the single statement \( \text{Statement} \) or the block of statements \( \text{Block} \) before continuing execution.

If \( \text{BooleanExpr} \) evaluates to false, continues execution without executing the statement or block of statements.

\( \text{Block} \) can be either a single statement or a sequence of statements separated with the “;” character.

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

**If** \( \text{BooleanExpr} \) **Then**

**Else**

**EndIf**

If \( \text{BooleanExpr} \) evaluates to true, executes \( \text{Block1} \) and then skips \( \text{Block2} \).

If \( \text{BooleanExpr} \) evaluates to false, skips \( \text{Block1} \) but executes \( \text{Block2} \).

\( \text{Block1} \) and \( \text{Block2} \) can be a single statement.

**If** \( \text{BooleanExpr1} \) **Then**

**ElseIf** \( \text{BooleanExpr2} \) **Then**

⋮

**ElseIf** \( \text{BooleanExprN} \) **Then**

**EndIf**

Allows for branching. If \( \text{BooleanExpr1} \) evaluates to true, executes \( \text{Block1} \). If \( \text{BooleanExpr1} \) evaluates to false, evaluates \( \text{BooleanExpr2} \), and so on.

<table>
<thead>
<tr>
<th>Function definition</th>
<th>( g(x) )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(12) )</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>( g(-12) )</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function definition</th>
<th>( g(x) )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(4) )</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( g(10) )</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
ifFn(BooleanExpr, Value_If_true [, Value_If_false [, Value_If_unknown]]) ⇒ expression, list, or matrix

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

- **BooleanExpr** can test a single value, a list, or a matrix.
- If an element of **BooleanExpr** evaluates to true, returns the corresponding element from **Value_If_true**.
- If an element of **BooleanExpr** evaluates to false, returns the corresponding element from **Value_If_false**. If you omit **Value_If_false**, returns undef.
- If an element of **BooleanExpr** is neither true nor false, returns the corresponding element **Value_If_unknown**. If you omit **Value_If_unknown**, returns undef.
- If the second, third, or fourth argument of the ifFn() function is a single expression, the Boolean test is applied to every position in **BooleanExpr**.

**Note:** If the simplified **BooleanExpr** statement involves a list or matrix, all other list or matrix arguments must have the same dimension(s), and the result will have the same dimension(s).

<table>
<thead>
<tr>
<th>BooleanExpr</th>
<th>Value_If_true</th>
<th>Value_If_false</th>
<th>Value_If_unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3&lt;2.5,5,6,7,{8,9,10}</td>
<td>{5,6,10}</td>
<td>Test value of 1 is less than 2.5, so its corresponding <strong>Value_If_true</strong> element of 5 is copied to the result list.</td>
<td></td>
</tr>
<tr>
<td>1,2,3&lt;2.5,5,6,7</td>
<td>{4,4,10}</td>
<td>Test value of 2 is less than 2.5, so its corresponding <strong>Value_If_true</strong> element of 6 is copied to the result list.</td>
<td></td>
</tr>
<tr>
<td>1,2,3&lt;2.5,5,6,7</td>
<td>{5,6,undef}</td>
<td>Test value of 3 is not less than 2.5, so its corresponding <strong>Value_If_false</strong> element of 10 is copied to the result list.</td>
<td></td>
</tr>
</tbody>
</table>

imag()

**imag(Value1) ⇒ value**

Returns the imaginary part of the argument.

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2·i</td>
<td>2</td>
</tr>
</tbody>
</table>

imag(1+2·i) 2
imag() Returns a list of the imaginary parts of the elements.

imag(List I) ⇒ list

Returns a list of the imaginary parts of the elements.

imag(Matrix I) ⇒ matrix

Returns a matrix of the imaginary parts of the elements.

Indirection See #(), page 189.

inString() Returns the character position in string srcString at which the first occurrence of string subString begins.

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

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

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

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

int(Value) ⇒ integer

The argument can be a real or a complex number.

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

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

Returns the signed integer part of (Number1 ÷ Number2).

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

interpolate()  

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

This function does the following:

Given xList, yList=f(xList), and yPrimeList=f'(xList) for some unknown function f, a cubic interpolant is used to approximate the function f at xValue. It is assumed that xList is a list of monotonically increasing or decreasing numbers, but this function may return a value even when it is not. This function works 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 ≥ 2 and contain expressions that simplify to numbers.

xValue can be a number or a list of numbers.

inv\(\chi^2()\)  

inv\(\chi^2\)(Area,df)

invChi2(Area,df)

Computes the Inverse cumulative \(\chi^2\) (chi-square) probability function specified by degree of freedom, \(df\) for a given Area under the curve.
invF() computes the Inverse cumulative F distribution function specified by \(df_{\text{Numer}}\) and \(df_{\text{Denom}}\) for a given \(Area\) under the curve.

invBinom() is an Inverse binomial function. Given the number of trials (\(NumTrials\)) and the probability of success of each trial (\(Prob\)), this function returns the minimum number of successes, \(k\), such that the value, \(k\), is greater than or equal to the given cumulative probability (\(CumulativeProb\)).

\[
\text{invBinom}\left(CumulativeProb, NumTrials, Prob, OutputForm\right) \Rightarrow \text{scalar or matrix}
\]

Example: Mary and Kevin are playing a dice game. Mary has to guess the maximum number of times 6 shows up in 30 rolls. If the number 6 shows up that many times or less, Mary wins. Furthermore, the smaller the number that she guesses, the greater her winnings. What is the smallest number Mary can guess if she wants the probability of winning to be greater than 77%?

\[
\begin{align*}
\text{invBinom}\left(0.77, 30, \frac{1}{6}\right) & \Rightarrow 6 \\
\text{invBinom}\left(0.77, 30, \frac{1}{6}, 1\right) & \Rightarrow 5 \quad 0.616447 \\
& \quad 6 \quad 0.776537
\end{align*}
\]

invBinomN() is an Inverse binomial with respect to \(N\). Given the probability of success of each trial (\(Prob\)), and the number of successes (\(NumSuccess\)), this function returns the minimum number of trials, \(N\), such that the value, \(N\), is less than or equal to the given cumulative probability (\(CumulativeProb\)).

\[
\text{invBinomN}\left(CumulativeProb, Prob, NumSuccess, OutputForm\right) \Rightarrow \text{scalar or matrix}
\]

Example: Monique is practicing goal shots for netball. She knows from experience that her chance of making any one shot is 70%. She plans to practice until she scores 50 goals. How many shots must she attempt to ensure that the probability of making at least 50 goals is more than 0.99?

\[
\begin{align*}
\text{invBinomN}\left(0.01, 0.7, 49\right) & \Rightarrow 86 \\
\text{invBinomN}\left(0.01, 0.7, 49, 1\right) & \Rightarrow \begin{bmatrix} 85 & 0.010451 \\ 86 & 0.00709 \end{bmatrix}
\end{align*}
\]
**invNorm()**

`invNorm(Area, μ, σ)`

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

**invt()**

`invt(Area, df)`

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

**iPart()**

`iPart(Number) ⇒ integer`

`iPart(List1) ⇒ list`

`iPart(Matrix1) ⇒ matrix`

Returns the integer part of the argument.

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

The argument can be a real or a complex number.

**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`. 
irr()

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 95.

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.

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

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 212.
**LbL**

**LbL** *labelName*

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

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

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

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

---

**lcm()**

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

**lcm**(*List1, List2*) ⇒ *list*

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

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

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

---

**left()**

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

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

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

**left**(*List1*, *Num*) ⇒ *list*

---

---

---

---
left()

Returns the leftmost Num elements contained in List1.

If you omit Num, returns all of List1.

left(Comparison) ⇒ expression

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

libShortcut()

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

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

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

To copy a variable group, see CopyVar on page 24.
To delete a variable group, see DelVar on page 38.

LinRegBx

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

Computes the linear regression y = a+b•x on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable. (See page 146.)

All the lists must have equal dimension except for Include.

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

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

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

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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.$r^2$</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>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified <em>X List</em> actually used in the regression based on restrictions of <em>Freq</em>, <em>Category List</em>, and <em>Include Categories</em></td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified <em>Y List</em> actually used in the regression based on restrictions of <em>Freq</em>, <em>Category List</em>, and <em>Include Categories</em></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to <em>stat.XReg</em> and <em>stat.YReg</em></td>
</tr>
</tbody>
</table>

**LinRegMix**

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

Computes the linear regression $y = m \cdot x + b$ on lists *X* and *Y* with frequency *Freq*. A summary of results is stored in the *stat.results* variable. (See page 146.)

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

Catalog

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

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

$Category$ is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $y = m \times x + b$</td>
</tr>
<tr>
<td>stat.m, stat.b</td>
<td>Regression coefficients</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>
<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>

LinRegtIntervals

Catalog

$LinRegtIntervals \ X, Y[\cdot, F[\cdot, 0[, CLev]]]$

For Slope. Computes a level C confidence interval for the slope.
LinRegtIntervals \( X, Y, F, 1, Xval, \{CLev\} \]

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

A summary of results is stored in the \texttt{stat.results} variable. (See page 146.)

All the lists must have equal dimension.

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

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{stat.RegEqn}</td>
<td>Regression Equation: ( a + bx )</td>
</tr>
<tr>
<td>\texttt{stat.a}, \texttt{stat.b}</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>\texttt{stat.df}</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>\texttt{stat.r^2}</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>\texttt{stat.r}</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>\texttt{stat.Resid}</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

For Slope type only

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{[stat.CLower, stat.CUpper]}</td>
<td>Confidence interval for the slope</td>
</tr>
<tr>
<td>\texttt{stat.ME}</td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td>\texttt{stat.SESlope}</td>
<td>Standard error of slope</td>
</tr>
<tr>
<td>\texttt{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,</td>
<td>Prediction interval for a single observation</td>
</tr>
<tr>
<td>stat.UpperPred]</td>
<td></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.(\hat{y})</td>
<td>(a + b \cdot XVal)</td>
</tr>
</tbody>
</table>

**LinRegtTest**

**LinRegtTest** \(X,Y[,Freq[,Hypoth]]\)

Computes a linear regression on the \(X\) and \(Y\) lists and a \(t\) 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 \(H : \beta \neq 0\) and \(\rho \neq 0\) (default), set *Hypoth* = 0
For \(H^a : \beta < 0\) and \(\rho < 0\), set *Hypoth* < 0
For \(H^b : \beta > 0\) and \(\rho > 0\), set *Hypoth* > 0

A summary of results is stored in the *stat.results* variable. (See page 146.)
LinRegTest

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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.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>
</tbody>
</table>

linSolve()

\[
\text{linSolve}( \text{SystemOfLinearEqns}, \text{Var1}, \text{Var2}, ...) \Rightarrow \text{list}
\]

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

\[
\text{linSolve}(\{\text{LinearEqn1, LinearEqn2, ...}\}, \text{Var1}, \text{Var2}, ...) \Rightarrow \text{list}
\]

\[
\text{linSolve}(\text{SystemOfLinearEqns}, \{\text{Var1}, \text{Var2}, ...\}) \Rightarrow \text{list}
\]

\[
\text{linSolve}(\text{LinearEqn1} \text{ and LinearEqn2 and} \ldots, \{\text{Var1, Var2, } \ldots\}) \Rightarrow \text{list}
\]

\[
\text{linSolve}(\{\text{LinearEqn1, LinearEqn2, } \ldots\}, \{\text{Var1, Var2, } \ldots\}) \Rightarrow \text{list}
\]

Returns a list of solutions for the variables \( \text{Var1, Var2, ...} \).
linSolve()

The first argument must evaluate to a system of linear equations or a single linear equation. Otherwise, an argument error occurs.

For example, evaluating \texttt{linSolve(x=1 and x=2,x)} produces an “Argument Error” result.

\[ \Delta \text{List}() \]

\[ \Delta \text{List}(List1) \Rightarrow list \]

\textbf{Note:} You can insert this function from the keyboard by typing \texttt{deltaList(...)}.

Returns a list containing the differences between consecutive elements in \textit{List1}. Each element of \textit{List1} is subtracted from the next element of \textit{List1}. The resulting list is always one element shorter than the original \textit{List1}.

\[ \text{list\textgreater} \text{mat}() \]

\[ \text{list\textgreater} \text{mat}(List [, \textit{elementsPerRow}]) \Rightarrow matrix \]

Returns a matrix filled row-by-row with the elements from \textit{List}.

\textit{elementsPerRow}, if included, specifies the number of elements per row. Default is the number of elements in \textit{List} (one row).

If \textit{List} does not fill the resulting matrix, zeros are added.

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

\[ \text{ln()} \]

\[ \text{ln}(Value1) \Rightarrow value \]

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

Returns the natural logarithm of the argument.

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

\[ \ln(x) \Rightarrow x \]

If complex format mode is Real:

\[ \ln\{\{-3,1.2,5\}\} \]

"Error: Non–real calculation"

If complex format mode is Rectangular:

\[ \ln\{\{-3,1.2,5\}\} \]

\[ \{1.09861+3.14159\cdot i,0.182322,1.60944\} \]

\( \ln(\text{squareMatrix}) \Rightarrow \text{squareMatrix} \)

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

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

\( \ln(\text{squareMatrix}) \)

\[ \begin{bmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{bmatrix} \]

\[ \begin{bmatrix} 1.83145+1.73485\cdot i & 0.009193-1.49086 \\ 0.448761-0.725533\cdot i & 1.06491+0.623491 \\ -0.266891-2.08316\cdot i & 1.12436+1.79018 \end{bmatrix} \]

In Radian angle mode and Rectangular complex format:

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

\( \text{LnReg} \)

\[ \text{LnReg} \ X, \ Y[, \ Freq[, \ Category, \ Include]]] \]

Computes the logarithmic regression \( y = a+b\ln(x) \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \text{stat.results} variable. (See page 146.)

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 \( \geq 0 \).
\textit{Category} is a list of numeric or string category codes for the corresponding $X$ and $Y$ 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,” page 212.

<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

Local Var1[, Var2 [, Var3] ...}:

Declares the specified vars as local variables. Those variables exist only during evaluation of a function and are deleted when the function finishes execution.

**Note:** Local variables save memory because they only exist temporarily. Also, they do not disturb any existing global variable values. Local variables must be used for For loops and for temporarily saving values in a multi-line function since modifications on global variables are not allowed in a function.

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

---

Lock

LockVar1[, Var2 [, Var3] ...}:

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

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

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

See unLock, page 164, and getLockInfo(), page 64.
\[ \log(\text{Value1}, \text{Value2}) \Rightarrow \text{value} \]

\[ \log(\text{List1}, \text{Value2}) \Rightarrow \text{list} \]

Returns the base-\(\text{Value2}\) logarithm of the first argument.

**Note:** See also Log template, page 2.

For a list, returns the base-\(\text{Value2}\) logarithm of the elements.

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

\[ \log(\text{squareMatrix1}, \text{Value}) \Rightarrow \text{squareMatrix} \]

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

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

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

**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 146.)

All the lists must have equal dimension except for \(\text{Include}\).
The Logistic function computes the logistic regression \( y = \frac{c}{1 + a \cdot e^{-b \cdot x}} + d \) on lists \( X \) and \( Y \) with frequency \( Freq \), using a specified number of \( Iterations \). A summary of results is stored in the \( stat.results \) variable. (See page 146.)

### Logistic

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

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

**Category** is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( \frac{c}{1 + a \cdot e^{-b \cdot x}} )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c</td>
<td>Regression coefficients</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 <strong>Freq</strong>, <strong>Category List</strong>, and <strong>Include Categories</strong></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 <strong>Freq</strong>, <strong>Category List</strong>, and <strong>Include Categories</strong></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to <strong>stat.XReg</strong> and <strong>stat.YReg</strong></td>
</tr>
</tbody>
</table>

### LogisticD

**LogisticD** \( X, Y [, [Iterations] [, Freq] [, Category, Include] ] \)

Computes the logistic regression \( y = \frac{c}{(1+ae^{-bx})+d} \) on lists \( X \) and \( Y \) with frequency \( Freq \), using a specified number of \( Iterations \). A summary of results is stored in the \( stat.results \) variable. (See page 146.)
All the lists must have equal dimension except for Include.

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

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

$Category$ is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: $c/(1+a\cdot e^{-bx})+d)$</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.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>
Loop

**Block**

**EndLoop**

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

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

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

---

**LU**

**Matrix, lMatrix, uMatrix, pMatrix [.Tol]**

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

**lMatrix • uMatrix = pMatrix • matrix**

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

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

\[
5 \times 10^{-14} \times \max(\text{dim}(Matrix)) \times \text{rowNorm}(Matrix)
\]
The LU factorization algorithm uses partial pivoting with row interchanges.

\[\text{mat} \rightarrow \text{list()}\]

\text{mat} \rightarrow \text{list}(\text{Matrix}) \Rightarrow \text{list}

Returns a list filled with the elements in \text{Matrix}. The elements are copied from \text{Matrix} row by row.

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

\[\text{max()}\]

\text{max}(Value1, Value2) \Rightarrow \text{expression}
\text{max}(List1, List2) \Rightarrow \text{list}
\text{max}(Matrix1, Matrix2) \Rightarrow \text{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.

\text{max}(\text{List}) \Rightarrow \text{expression}

Returns the maximum element in \text{list}.

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

Returns a row vector containing the maximum element of each column in \text{Matrix1}.

Empty (void) elements are ignored. For more information on empty elements, see page 212.

\textbf{Note:} See also \texttt{min()}. 

Alphabetical Listing 91
mean() 

\[ \text{mean}(\text{List},\text{freqList}) \Rightarrow \text{expression} \]

Returns the mean of the elements in List.

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

**mean**(Matrix1[,freqMatrix]) ⇒ matrix

Returns a row vector of the means of all the columns in Matrix1.

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

Empty (void) elements are ignored. For more information on empty elements, see page 212.

---

median() 

\[ \text{median}(\text{List},\text{freqList}) \Rightarrow \text{expression} \]

Returns the median of the elements in List.

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

**median**(Matrix1[,freqMatrix]) ⇒ matrix

Returns a row vector containing the medians of the columns in Matrix1.

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

**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 212.
**MedMed X,Y [, Freq] [, Category, Include]**

Computes the median-median line $y = (m\times x + b)$ on lists $X$ and $Y$ with frequency $Freq$. A summary of results is stored in the `stat.results` variable. (See page 146.)

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

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

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

`Category` is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Median-median line equation: $m\times x + b$</td>
</tr>
<tr>
<td>stat.m,</td>
<td>Model coefficients</td>
</tr>
<tr>
<td>stat.b</td>
<td></td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the median-median line</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 <code>Freq</code>, <code>Category List</code>, and <code>Include Categories</code></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 <code>Freq</code>, <code>Category List</code>, and <code>Include Categories</code></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to <code>stat.XReg</code> and <code>stat.YReg</code></td>
</tr>
</tbody>
</table>
**mid()**

**mid(sourceString, Start[, Count]) ⇒ string**

Returns **Count** characters from character string **sourceString**, beginning with character number **Start**.

If **Count** is omitted or is greater than the dimension of **sourceString**, returns all characters from **sourceString**, beginning with character number **Start**.

**Count** must be ≥ 0. If **Count** = 0, returns an empty string.

**mid(sourceList, Start [, Count]) ⇒ list**

Returns **Count** elements from **sourceList**, beginning with element number **Start**.

If **Count** is omitted or is greater than the dimension of **sourceList**, returns all elements from **sourceList**, beginning with element number **Start**.

**Count** must be ≥ 0. If **Count** = 0, returns an empty list.

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

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

**min()**

**min(Value1, Value2) ⇒ expression**

**min(List1, List2) ⇒ list**

**min(Matrix1, Matrix2) ⇒ matrix**

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

**min(List) ⇒ expression**

Returns the minimum element of **List**.
\textbf{min()}

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

Returns a row vector containing the minimum element of each column in \textit{Matrix1}.

\textbf{Note:} See also \textbf{max()}.  

\textbf{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.

\textit{financeRate} is the interest rate that you pay on the cash flow amounts.

\textit{reinvestRate} is the interest rate at which the cash flows are reinvested.

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

\textit{CFList} is a list of cash flow amounts after the initial cash flow \textit{CF0}.

\textit{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 \textit{CFList}. The default is 1; if you enter values, they must be positive integers < 10,000.

\textbf{Note:} See also \textbf{irr()}, page 74.

\textbf{mod()}

\[ \text{mod}(\text{Value1}, \text{Value2}) \Rightarrow \text{expression} \]

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

\[ \text{mod}(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix} \]

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

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

\[
\text{mod}(x,0) = x \\
\text{mod}(x,y) = x - y \, \text{floor}(x/y)
\]

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

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

**Note:** See also `remain()`, page 126

**mRow()**

\[\text{mRow}(Value, Matrix1, Index) \Rightarrow \text{matrix}\]

Returns a copy of `Matrix1` with each element in row `Index` of `Matrix1` multiplied by `Value`.

**mRowAdd()**

\[\text{mRowAdd}(Value, Matrix1, Index1, Index2) \Rightarrow \text{matrix}\]

Returns a copy of `Matrix1` with each element in row `Index2` of `Matrix1` replaced with:

\[\text{Value} \times \text{row Index1} + \text{row Index2}\]

**MultReg**

\[\text{MultReg} \ \{Y, X1[,X2[,X3[,\ldots,X10]]]\}\]

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 146.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.
### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ...$</td>
</tr>
<tr>
<td>stat.b0, stat.b1, ...</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R^2</td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.( \hat{y} ) List</td>
<td>( \hat{y} ) List = $b_0 + b_1 \cdot x_1 + ...$</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

### MultRegIntervals

**MultRegIntervals** \( Y, X[1[, X2[, X3, ...[, X10]]], XValList[, CLevel] \)

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

A summary of results is stored in the `stat.results` variable. (See page 146.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: ( b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ... )</td>
</tr>
<tr>
<td>stat.( \hat{y} )</td>
<td>A point estimate: ( \hat{y} = b_0 + b_1 \cdot x_1 + ... ) for <code>XValList</code></td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval for a 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.bList</td>
<td>List of regression coefficients, {b0,b1,b2,...}</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>
**MultRegTests**  

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

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

A summary of results is stored in the `stat.results` variable. (See page 146.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

### Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.ReqEqn</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>
<tr>
<td><code>stat.AdjR^2</code></td>
<td>Adjusted coefficient of multiple determination</td>
</tr>
<tr>
<td><code>stat.s</code></td>
<td>Standard deviation of the error</td>
</tr>
<tr>
<td><code>stat.DW</code></td>
<td>Durbin-Watson statistic; used to determine whether first-order auto correlation is present in the model</td>
</tr>
<tr>
<td><code>stat.dfReg</code></td>
<td>Regression degrees of freedom</td>
</tr>
<tr>
<td><code>stat.SSReg</code></td>
<td>Regression sum of squares</td>
</tr>
<tr>
<td><code>stat.MSReg</code></td>
<td>Regression mean square</td>
</tr>
<tr>
<td><code>stat.dfError</code></td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td><code>stat.SSError</code></td>
<td>Error sum of squares</td>
</tr>
<tr>
<td><code>stat.MSError</code></td>
<td>Error mean square</td>
</tr>
<tr>
<td><code>stat.bList</code></td>
<td><code>{b_0,b_1,...}</code> List of coefficients</td>
</tr>
<tr>
<td><code>stat.tList</code></td>
<td>List of $t$ statistics, one for each coefficient in the <code>bList</code></td>
</tr>
<tr>
<td><code>stat.PList</code></td>
<td>List $P$-values for each $t$ statistic</td>
</tr>
<tr>
<td><code>stat.SEList</code></td>
<td>List of standard errors for coefficients in <code>bList</code></td>
</tr>
<tr>
<td><code>stat.yList</code></td>
<td>$y$ List = $b_0+b_1\cdot x_1+...$</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</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>

\[ N \]

**nand**

\[ \text{BooleanExpr1 nand BooleanExpr2 returns Boolean expression} \]
\[ \text{BooleanList1 nand BooleanList2 returns Boolean list} \]
\[ \text{BooleanMatrix1 nand BooleanMatrix2 returns Boolean matrix} \]

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

For lists and matrices, returns comparisons element by element.

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

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

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

<table>
<thead>
<tr>
<th>3 and 4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 nand 4</td>
<td>-1</td>
</tr>
<tr>
<td>{1,2,3} and {3,2,1}</td>
<td>{1,2,1}</td>
</tr>
<tr>
<td>{1,2,3} nand {3,2,1}</td>
<td>{-2,-3,-2}</td>
</tr>
</tbody>
</table>
\textbf{nCr()}

\texttt{nCr(Value1, Value2) \Rightarrow expression}

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

\texttt{nCr(Value, 0) \Rightarrow 1}

\texttt{nCr(Value, negInteger) \Rightarrow 0}

\texttt{nCr(Value, posInteger) \Rightarrow Value\cdot(\texttt{Value}−1)\ldots(\texttt{Value}−\texttt{posInteger}+1)/\texttt{posInteger}!}

\texttt{nCr(Value, nonInteger) \Rightarrow expression! / ((Value−nonInteger)!\cdot\texttt{nonInteger})!}

\texttt{nCr(List1, List2) \Rightarrow list}

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

\texttt{nCr(Matrix1, Matrix2) \Rightarrow matrix}

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

\textbf{nDerivative()}

\texttt{nDerivative(Expr1,Var=Value[,Order]) \Rightarrow value}

\texttt{nDerivative(Expr1,Var[,Order]) | Var=Value \Rightarrow value}

Returns the numerical derivative calculated using auto differentiation methods.

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

If the variable \texttt{Var} does not contain a numeric value, you must provide \texttt{Value}.

\textit{Order} of the derivative must be 1 or 2.
nDerivative()  

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

Consider the example on the right. The first derivative of \(x^3 (x^2 + x)^{(1/3)}\) at \(x=0\) is equal to 0. However, because the first derivative of the subexpression \((x^2 + x)^{(1/3)}\) is undefined at \(x=0\), and this value is used to calculate the derivative of the total expression, nDerivative() reports the result as undefined and displays a warning message.

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

newList()  

newList\(\text{numElements}\) \(\Rightarrow\) list

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

newMat()  

newMat\(\text{numRows, numColumns}\) \(\Rightarrow\) matrix

Returns a matrix of zeros with the dimension numRows by numColumns.

nfMax()  

nfMax\(\text{Expr, Var}\) \(\Rightarrow\) value

nfMax\(\text{Expr, Var, lowBound}\) \(\Rightarrow\) value

nfMax\(\text{Expr, Var, lowBound, upBound}\) \(\Rightarrow\) value

nfMax\(\text{Expr, Var} | \text{lowBound} \leq \text{Var} \leq \text{upBound}\) \(\Rightarrow\) value
**nfMax()**

Returns a candidate numerical value of variable \( Var \) where the local maximum of \( Expr \) occurs.

If you supply \( lowBound \) and \( upBound \), the function looks in the closed interval \([lowBound, upBound] \) for the local maximum.

**nfMin()**

\[
\textit{nfMin}(Expr, Var) \Rightarrow value \\
\textit{nfMin}(Expr, Var, lowBound) \Rightarrow value \\
\textit{nfMin}(Expr, Var, lowBound, upBound) \Rightarrow value \\
\textit{nfMin}(Expr, Var) | lowBound \leq Var \leq upBound \Rightarrow value
\]

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.

**nInt()**

\[
\textit{nInt}(Expr1, Var, Lower, Upper) \Rightarrow expression
\]

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

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.
nInt()  

A warning is displayed ("Questionable accuracy") when it seems that the goal has not been achieved.

Nest nInt() to do multiple numeric integration. Integration limits can depend on integration variables outside them.

\[
\text{nInt}\left(\text{nInt}\left(\frac{e^{-x^2-y^2}}{x^2+y^2},x,x,0,1\right),x,0,1\right) = 3.30423
\]

nom()  

\[
\text{nom}(\text{effectiveRate}, \text{CpY}) \Rightarrow \text{value}
\]

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 44.

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.

\[
\begin{array}{c|c}
\text{Integer1} \text{ nor } \text{Integer2} & \Rightarrow \text{integer} \\
3 \text{ or } 4 & 7 \\
3 \text{ nor } 4 & -8 \\
\{1,2,3\} \text{ or } \{3,2,1\} & \{3,2,3\} \\
\{1,2,3\} \text{ nor } \{3,2,1\} & \{-4,-3,-4\}
\end{array}
\]

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).

**norm()**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{norm}(\begin{bmatrix} 1 &amp; 2 \ 3 &amp; 4 \end{bmatrix}) )</td>
<td>5.47723</td>
</tr>
</tbody>
</table>

**normCdf()**

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 \( \text{lowBound} = \sim \text{E}999 \).

**normPdf()**

Computes the probability density function for the normal distribution at a specified \( XVal \) value for the specified \( \mu \) and \( \sigma \).

**not**

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

<table>
<thead>
<tr>
<th>BooleanExpr</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>not (2≥3)</td>
<td>true</td>
</tr>
<tr>
<td>not 0hB0 Base16</td>
<td>0FFFFFFFFFFFF4F</td>
</tr>
<tr>
<td>not not 2</td>
<td>2</td>
</tr>
</tbody>
</table>

In Hex base mode:
Returns the one’s complement of a real integer. Internally, Integer1 is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one’s complement. Results are displayed according to the Base mode.

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

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ▶ Base2, page 16.

Important: Zero, not the letter O.

In Bin base mode:

<table>
<thead>
<tr>
<th>not 0h7AC36</th>
<th>0hFFFFFFFFFFFF853C9</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Bin base mode:</td>
<td></td>
</tr>
<tr>
<td>0b100101 ▶ Base10</td>
<td>37</td>
</tr>
<tr>
<td>not 0b100101</td>
<td></td>
</tr>
<tr>
<td>0b11111111111111111111111111111111</td>
<td></td>
</tr>
<tr>
<td>not 0b100101 ▶ Base10</td>
<td>-38</td>
</tr>
</tbody>
</table>

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

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

\[
\text{nPr}()\]

\[
\text{nPr}(Value1, Value2) \Rightarrow \text{expression}
\]

For integer Value1 and Value2 with \(Value1 \geq Value2 \geq 0\), \(\text{nPr}()\) is the number of permutations of \(Value1\) things taken \(Value2\) at a time.

\[
\text{nPr}(Value, 0) \Rightarrow 1
\]

\[
\text{nPr}(Value, \text{negInteger}) \Rightarrow \frac{1}{((Value+1)\times...(Value-\text{negInteger}))}
\]

\[
\text{nPr}(Value, \text{posInteger}) \Rightarrow Value \times ...(Value-\text{posInteger}+1)
\]

\[
\text{nPr}(Value, \text{nonInteger}) \Rightarrow \frac{Value!}{(Value-\text{nonInteger})!}
\]

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

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

\[
\text{nPr}(Matrix1, Matrix2) \Rightarrow \text{matrix}
\]

\[
\begin{array}{c}
\text{nPr}(\{5,4,3\}, \{2,4,2\}) \\
\{20,24,6\}
\end{array}
\]
**nPr()**

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

---

**npv()**

\[ \text{npv}(\text{InterestRate}, \text{CFO}, \text{CFList}[, \text{CFFreq}]) \]

Financial function that calculates net present value; the sum of the present values for the cash inflows and outflows. A positive result for npv indicates a profitable investment.

- **InterestRate** is the rate by which to discount the cash flows (the cost of money) over one period.
- **CFO** is the initial cash flow at time 0; it must be a real number.
- **CFList** is a list of cash flow amounts after the initial cash flow **CFO**.
- **CFFreq** is a list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of **CFList**. The default is 1; if you enter values, they must be positive integers < 10,000.

---

**nSolve()**

\[ \text{nSolve}(\text{Equation}, \text{Var} [= \text{Guess}]) \Rightarrow \text{number or error_string} \]

\[ \text{nSolve}(\text{Equation}, \text{Var} [= \text{Guess}], \text{lowBound}) \Rightarrow \text{number or error_string} \]

\[ \text{nSolve}(\text{Equation}, \text{Var} [= \text{Guess}], \text{lowBound}, \text{upBound}) \Rightarrow \text{number or error_string} \]

\[ \text{nSolve}(\text{Equation}, \text{Var} [= \text{Guess}]) | \text{lowBound} \leq \text{Var} \leq \text{upBound} \Rightarrow \text{number or error_string} \]

**Note:** If there are multiple solutions, you can use a guess to help find a particular solution.
Iteratively searches for one approximate real numeric solution to Equation for its one variable. Specify the variable as:

variable
– or –
variable = real number

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

\[
\text{nSolve}() \text{ attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string “no solution found.”}
\]

\[
\begin{array}{ll}
\text{nSolve}(x^2+5\cdot x-25=9,x) & x<0 \quad -8.84429 \\
\text{nSolve}\left(\frac{(1+r)^{24}-1}{r} = 26, r \right) & r>0 \text{ and } r<0.25 \quad 0.006886 \\
\text{nSolve}(x^2-1,x) & \text{"No solution found"}
\end{array}
\]

\section{O}

\subsection{OneVar}

\textbf{OneVar} \[1,]X[,]\{Freq\}[[,]\text{Category,Include}]

\textbf{OneVar} \[n,]X1,X2[\ldots,X20]]

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the \texttt{stat.results} variable. (See page 146.)

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

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

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

\texttt{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.
An empty (void) element in any of the lists $X, Freq,$ or $Category$ results in a void for the corresponding element of all those lists. An empty element in any of the lists $X_1$ through $X_{20}$ results in a void for the corresponding element of all those lists. For more information on empty elements, see page 212.

<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.(\sum x)</td>
<td>Sum of (x) values</td>
</tr>
<tr>
<td>stat.(\sum 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.(\sigma x)</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.Q_1X</td>
<td>1st Quartile of (x)</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of (x)</td>
</tr>
<tr>
<td>stat.Q_3X</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>

**or**

`BooleanExpr1 or BooleanExpr2` returns

`Boolean expression`

`BooleanList1 or BooleanList2` returns

`Boolean list`

`BooleanMatrix1 or BooleanMatrix2` returns

`Boolean matrix`

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

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

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

Integer1 or Integer2 ⇒ integer

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

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

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ►Base2, page 16.

Note: See xor.

ord() in Hex base mode:

| 0h7AC36 or 0h3D5F | 0h7BD7F |

Important: Zero, not the letter O.

In Bin base mode:

| 0b100101 or 0b100 | 0b100101 |

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

ord(String) ⇒ integer

ord(List1) ⇒ list

Returns the numeric code of the first character in character string String, or a list of the first characters of each list element.

P

P►Rx() in Radian angle mode:

<table>
<thead>
<tr>
<th>P►Rx(rExpr, θExpr) ⇒ expression</th>
<th>In Radian angle mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>P►Rx(rList, θList) ⇒ list</td>
<td></td>
</tr>
<tr>
<td>P►Rx(rMatrix, θMatrix) ⇒ matrix</td>
<td></td>
</tr>
</tbody>
</table>
**P►Rx()**

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

**Note:** The \(\theta\) argument is interpreted as either a degree, gradian or radian angle, according to the current angle mode. If the argument is an expression, you can use °, ¹⁄₄, or \(r\) to override the angle mode setting temporarily.

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

---

**P►Ry()**

\[
P►Ry(rValue, \thetaValue) \Rightarrow value \\
P►Ry(rList, \thetaList) \Rightarrow list \\
P►Ry(rMatrix, \thetaMatrix) \Rightarrow matrix
\]

Returns the equivalent y-coordinate of the \((r, \theta)\) pair.

**Note:** The \(\theta\) argument is interpreted as either a degree, radian or gradian angle, according to the current angle mode.

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

---

**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 22, and **Try**, page 158.
Note for entering the example: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.

**piecewise()**

`piecewise(Expr1[, Cond1[, Expr2 [, Cond2 [,... ]]]])`

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

**Note:** See also Piecewise template, page 2.

**poissCdf()**

`poissCdf(λ,lowBound,upBound) ⇒ number if lowBound and upBound are numbers, list if lowBound and upBound are lists`

`poissCdf(λ,upBound) for P(0≤X≤upBound) ⇒ number if upBound is a number, list if upBound is a list`

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

For P(X ≤ upBound), set lowBound=0

**poissPdf()**

`poissPdf(λ,XVal) ⇒ number if XVal is a number, list if XVal is a list`

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

► Polar

Vector ► Polar

```
[1 3.]►Polar [3.16228 ∠ 71.5651]
```
**Polar**

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

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

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

**Note:** See also ► Rect, page 123.

**complexValue ► Polar**

Displays complexVector in polar form.

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

**complexValue** can have any complex form. However, an \(re^{i\theta}\) entry causes an error in Degree angle mode.

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

\[
\begin{align*}
\text{In Radian angle mode:} & \\
\left(3+4i\right)\text{► Polar} & \Rightarrow e^{927295\cdot i\cdot 5} \\
\left(4 \angle \frac{\pi}{3}\right)\text{► Polar} & \Rightarrow e^{1.0472\cdot i\cdot 4}.
\end{align*}
\]

\[
\begin{align*}
\text{In Gradian angle mode:} & \\
\left(4\cdot i\right)\text{► Polar} & \Rightarrow \left(4 \angle 100\right).
\end{align*}
\]

\[
\begin{align*}
\text{In Degree angle mode:} & \\
\left(3+4\cdot i\right)\text{► Polar} & \Rightarrow \left(5 \angle 53.1301\right).
\end{align*}
\]

**polyEval()**

\[
\begin{align*}
\text{polyEval(List1, Expr1)} & \Rightarrow \text{expression} \\
\text{polyEval(List1, List2)} & \Rightarrow \text{expression}
\end{align*}
\]

\[
\begin{align*}
\text{polyEval}\left(\{1,2,3,4\},2\right) & \Rightarrow 26 \\
\text{polyEval}\left(\{1,2,3,4\},\{2,7\}\right) & \Rightarrow \{26,262\}
\end{align*}
\]

112  Alphabetical Listing
**polyEval()**

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

**polyRoots()**

\[
polyRoots(Poly, Var) \Rightarrow list
\]

\[
polyRoots(ListOfCoeffs) \Rightarrow list
\]

The first syntax, \( polyRoots(Poly, Var) \), returns a list of real roots of polynomial \( Poly \) with respect to variable \( Var \). If no real roots exist, returns an empty list: \( \{ \} \).

\( Poly \) must be a polynomial in expanded form in one variable. Do not use unexpanded forms such as \( y^2 \cdot y + 1 \) or \( x \cdot x + 2 \cdot x + 1 \).

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

**Note:** See also \( cPolyRoots() \), page 30.

**PowerReg**

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

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 146.)

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

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

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

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

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: a*(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 <em>X List</em> actually used in the regression based on restrictions of <em>Freq, Category List, and Include Categories</em></td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified <em>Y List</em> actually used in the regression based on restrictions of <em>Freq, Category List, and Include Categories</em></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to <em>stat.XReg</em> and <em>stat.YReg</em></td>
</tr>
</tbody>
</table>

**Prgm**

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

---

<table>
<thead>
<tr>
<th>Define <code>proggcd(a,b)=Prgm</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Local <code>d</code></td>
</tr>
<tr>
<td>While <code>b&lt;&gt;0</code></td>
</tr>
<tr>
<td><code>d:=mod(a,b)</code></td>
</tr>
<tr>
<td><code>a:=b</code></td>
</tr>
<tr>
<td><code>b:=d</code></td>
</tr>
<tr>
<td>Disp <code>a&quot;,&quot; &quot;,b</code></td>
</tr>
<tr>
<td>EndWhile</td>
</tr>
<tr>
<td>Disp &quot;GCD=&quot;`,a</td>
</tr>
<tr>
<td>EndPrgm</td>
</tr>
</tbody>
</table>

Done

```
proggcd(4560,450)
```

| 450 60 |
| 60 30  |
| 30 0  |
| GCD=30 |

Done

---

**prodSeq()**  
See Π ( ), page 186.

**Product (Π)**  
See Π ( ), page 186.

**product()**

**product(List[, Start[, End]]) ⇒ expression**

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

**product(Matrix1[, Start[, End]]) ⇒ matrix**

Returns a row vector containing the products of the elements in the columns of `Matrix1`. `Start` and `end` are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 212.
propFrac()  

propFrac(Value1[, Var]) \Rightarrow \text{value}

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

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

If Var is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>propFrac(\frac{4}{3})</td>
<td>1\frac{1}{3}</td>
</tr>
<tr>
<td>propFrac(\frac{-4}{3})</td>
<td>-1\frac{1}{3}</td>
</tr>
</tbody>
</table>

Q

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.

The floating-point number (9.) in m1 causes results to be calculated in floating-point form.
• 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: $5 \times 10^{-14} \times \max(\text{dim}(\text{Matrix})) \times \text{rowNorm}(\text{Matrix})$

The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in `qMatName` are the orthonormal basis vectors that span the space defined by `matrix`.

QuadReg

QuadReg $X,Y[, \text{Freq}][, \text{Category}, \text{Include}]$

Computes the quadratic polynomial regression $y = a \cdot x^2 + b \cdot x + c$ on lists $X$ and $Y$ with frequency $Freq$. A summary of results is stored in the `stat.results` variable. (See page 146.)

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

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

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

`Category` is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.
**QuadReg**

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a \cdot x^2 + b \cdot x + c )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of determination</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>

**QuartReg**

*QuartReg \( X, Y[, Freq][, Category, Include] *

Computes the quartic polynomial regression \( y = a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( stat.results \) variable. (See page 146.)

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 \( \geq 0 \).
**QuartReg**

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

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

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>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>stat.a, stat.b, stat.c, stat.d, stat.e</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of determination</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>

**R**

\( R \mathbf{P}() \)

\( R \mathbf{P}(xValue, yValue) \Rightarrow value \)
\( R \mathbf{P}(xList, yList) \Rightarrow list \)
\( R \mathbf{P}(xMatrix, yMatrix) \Rightarrow matrix \)

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

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

**Note:** You can insert this function from the computer keyboard by typing \( R \mathbf{P} theta (...) \).

In Degree angle mode:

\[
R \mathbf{P}(2, 2) \quad 45.
\]

In Gradian angle mode:

\[
R \mathbf{P}(2, 2) \quad 50.
\]

In Radian angle mode:
R►Pθ()  

<table>
<thead>
<tr>
<th>R►Pθ(3,2)</th>
<th>0.588003</th>
</tr>
</thead>
<tbody>
<tr>
<td>R►Pθ([3 -4 2], [0 (\frac{\pi}{4}) 1.5])</td>
<td>[0. 2.94771 0.643501]</td>
</tr>
</tbody>
</table>

R►Pr()  

| R►Pr(xValue, yValue) ⇒ value |
| R►Pr(xList, yList) ⇒ list |
| R►Pr(xMatrix, yMatrix) ⇒ matrix |

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

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

►Rad  

| Value1►Rad ⇒ value |

In Radian angle mode:

<table>
<thead>
<tr>
<th>R►Pr(3,2)</th>
<th>3.60555</th>
</tr>
</thead>
<tbody>
<tr>
<td>R►Pr([3 -4 2], [0 (\frac{\pi}{4}) 1.5])</td>
<td>[3 4.07638 (\frac{5}{2})]</td>
</tr>
</tbody>
</table>

In Degree angle mode:

| (1.5)►Rad | \((0.02618)^r\) |

In Gradian angle mode:

| (1.5)►Rad | \((0.02356)^r\) |

rand()  

| rand() ⇒ expression |
| rand(#Trials) ⇒ list |

`rand()` returns a random value between 0 and 1.

`rand(#Trials)` returns a list containing \#Trials random values between 0 and 1.

randBin()  

| randBin(n, p) ⇒ expression |
| randBin(n, p, #Trials) ⇒ list |

```plaintext
randBin(80,0.5)  46.
randBin(80,0.5,3) \{43,.39,.41.\}
```
randBin()  

randBin(n, p) returns a random real number from a specified Binomial distribution.

randBin(n, p, #Trials) returns a list containing #Trials random real numbers from a specified Binomial distribution.

randInt()  

randInt(lowBound, upBound) ⇒ expression
randInt(lowBound, upBound, #Trials) ⇒ list

randInt(lowBound, upBound) returns a random integer within the range specified by lowBound and upBound integer bounds.

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

randMat()  

randMat(numRows, numColumns) ⇒ matrix

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

Both arguments must simplify to integers.

Note: The values in this matrix will change each time you press enter.
**randNorm()**

**randNorm(μ, σ)** \(\Rightarrow\) **expression**  
**randNorm(μ, σ, #Trials)** \(\Rightarrow\) **list**

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

**randNorm(μ, σ, #Trials)** returns a list containing #Trials decimal numbers from the specified normal distribution.

---

**randPoly()**

**randPoly(Var, Order)** \(\Rightarrow\) **expression**

Returns a polynomial in Var of the specified Order. The coefficients are random integers in the range −9 through 9. The leading coefficient will not be zero.

*Order* must be 0–99.

**randSamp()**

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

Returns a list containing a random sample of #Trials trials from List with an option for sample replacement (noRepl=0), or no sample replacement (noRepl=1). The default is with sample replacement.

---

**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(Value1)** \(\Rightarrow\) **value**

real(2+3·i) \(\Rightarrow\) 2
real()\n
Returns the real part of the argument.
\nreal(List1) \Rightarrow list

Returns the real parts of all elements.
\nreal(Matrix1) \Rightarrow 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 111.

**complexValue ▶ Rect**

Displays complexValue in rectangular form \(a+bi\). The complexValue can have any complex form. However, an \(re^{iθ}\) entry causes an error in Degree angle mode.

**Note:** You must use parentheses for an \((r \angle θ)\) polar entry.

In Radian angle mode:

\[
\begin{bmatrix}
3 & \frac{π}{4} & \frac{π}{6} \\
\end{bmatrix}
\]

\Rightarrow Rect

\[
\begin{bmatrix}
1.06066 & 1.06066 & 2.59808 \\
\end{bmatrix}
\]

In Gradian angle mode:

\[
\begin{bmatrix}
\frac{π}{3} \\
\end{bmatrix}
\]

\Rightarrow Rect

11.3986

\[
\begin{bmatrix}
4 & \frac{π}{3} \\
\end{bmatrix}
\]

\Rightarrow Rect

2.00 + 3.4641\cdot i

In Degree angle mode:

\[
\begin{bmatrix}
1 \angle 100 \\
\end{bmatrix}
\]

\Rightarrow Rect

i

\[
\begin{bmatrix}
4 \angle 60 \\
\end{bmatrix}
\]

\Rightarrow Rect

2.00 + 3.4641\cdot i

**Note:** To type \(\angle\), select it from the symbol list in the Catalog.
ref(Matrix1[, Tol]) \Rightarrow \textit{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 \(\texttt{ctr|enter}\) 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:
  \(5E-14 \times \max(\text{dim}(\text{Matrix1})) \times \text{rowNorm} (\text{Matrix1})\)

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:

\[
\begin{bmatrix}
  a & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & \frac{1}{a} & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

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.

\[
\begin{bmatrix}
  a & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix} | a=0
\]

\[
\begin{bmatrix}
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  0 & 0 & 0
\end{bmatrix}
\]

**Note:** See also \(\texttt{rref()\texttt{, page 133}}\).
**RefreshProbeVars**

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

<table>
<thead>
<tr>
<th>StatusVar Value</th>
<th>Status</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>statusVar =0</td>
<td>Normal (continue with the program)</td>
<td>The Vernier DataQuest™ application is in data collection mode.</td>
</tr>
<tr>
<td>statusVar =1</td>
<td>The Vernier DataQuest™ application must be in meter mode for this command to work.</td>
<td></td>
</tr>
<tr>
<td>statusVar =2</td>
<td>The Vernier DataQuest™ application is not launched.</td>
<td></td>
</tr>
<tr>
<td>statusVar =3</td>
<td>The Vernier DataQuest™ application is launched, but you have not connected any probes.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

Define temp()=
Prgm
© Check if system is ready
RefreshProbeVars status
If status=0 Then
Disp "ready"
For n,1,50
RefreshProbeVars status
temperature:=meter.temperature
Disp "Temperature: ",temperature
If temperature>30 Then
Disp "Too hot"
EndIf
© Wait for 1 second between samples
Wait 1
EndFor
Else
Disp "Not ready. Try again later"
EndIf
EndPrgm

Note: This can also be used with TI-Innovator™ Hub.
remain() \[\text{Catalog > } \]

\[\text{remain}(\text{Value1, Value2}) \Rightarrow \text{value} \]
\[\text{remain}(\text{List1, List2}) \Rightarrow \text{list} \]
\[\text{remain}(\text{Matrix1, Matrix2}) \Rightarrow \text{matrix} \]

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

\[\text{remain}(x,0) \equiv x \]
\[\text{remain}(x,y) \equiv x - y \cdot \text{iPart}(x/y)\]

As a consequence, note that \[\text{remain}(-x,y) = -\text{remain}(x,y)\]. The result is either zero or it has the same sign as the first argument.

**Note:** See also \(\text{mod}()\), page 95.

### Request

**Request** \(\text{promptString, var[, DispFlag [, statusVar]]}\)

**Request** \(\text{promptString, func(arg1, ...argn[, DispFlag [, statusVar]]}\)

Programming command: Pauses the program and displays a dialog box containing the message \(\text{promptString}\) 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 \(\text{var}\).

If the user clicks Cancel, the program proceeds without accepting any input. The program uses the previous value of \(\text{var}\) if \(\text{var}\) was already defined.

The optional \(\text{DispFlag}\) argument can be any expression.

- If \(\text{DispFlag}\) is omitted or evaluates to 1, the prompt message and user’s response are displayed in the Calculator history.
- If \(\text{DispFlag}\) evaluates to 0, the prompt and response are not displayed in the history.
The optional statusVar argument gives the program a way to determine how the user dismissed the dialog box. Note that statusVar requires the DispFlag argument.

- If the user clicked OK or pressed Enter or Ctrl+Enter, variable statusVar is set to a value of 1.
- Otherwise, variable statusVar is set to a value of 0.

The func() argument allows a program to store the user’s response as a function definition. This syntax operates as if the user executed the command:

Define func(arg1, ...argn) = user’s response

The program can then use the defined function func(). The promptString should guide the user to enter an appropriate user’s response that completes the function definition.

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

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

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

**Note:** See also RequestStr, page 127.

---

**RequestStr**

RequestStr promptString, var[, DispFlag]  
Define a program:
**RequestStr**

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

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

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

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

Note: See also Request, page 126.

**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: For instructions on entering multi-line program and function definitions, refer to the Calculator section of your product guidebook.
right()  

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()  

rk23(Expr, Var, depVar, {Var0, VarMax}, depVar0, VarStep [, , diftol]) ⇒ matrix

Uses the Runge-Kutta method to solve the system
g=Expr(Var,depVar)

with depVar(Var0)=depVar0 on the interval [Var0,VarMax]. Returns a matrix whose first row defines the Var output values as defined by VarStep. The second row defines the value of the first solution component at the corresponding Var values, and so on.

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

SystemOfExpr is a system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).
**rk23 ()**

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

*Var* is the independent variable.

*ListDepVars* is a list of dependent variables.

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

*ListDepVars0* is a list of initial values for dependent variables.

If *VarStep* evaluates to a nonzero number: 
\[ \text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax} - \text{Var0}) \]
and solutions are returned at \[ \text{Var0} + i \times \text{VarStep} \]
for all \( i = 0, 1, 2, \ldots \) such that \( \text{Var0} + i \times \text{VarStep} \) is in \( [\text{Var0}, \text{VarMax}] \) (may not get a solution value at *VarMax*).

if *VarStep* evaluates to zero, solutions are returned at the "Runge-Kutta" *Var* values.

*diftol* is the error tolerance (defaults to 0.001).

---

**root()**

**root(Value) ⇒ root**

**root(Value1, Value2) ⇒ root**

*root(Value)* returns the square root of *Value*.

*root(Value1, Value2)* returns the *Value2* root of *Value1*. *Value1* can be a real or complex floating point constant or an integer or complex rational constant.

**Note:** See also Nth *root* template, page 1.

---

**rotate()**

**rotate(Integer1[, #ofRotations]) ⇒ integer**  

In Bin base mode:
**rotate()**

Rotates the bits in a binary integer. You can enter Integer1 in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of Integer1 is too large for this form, a symmetric modulo operation brings it within the range. For more information, see ►Base2, page 16.

If #ofRotations is positive, the rotation is to the left. If #ofRotations is negative, the rotation is to the right. The default is −1 (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

\[0b00000000000001111010110000110101\]

Rightmost bit rotates to leftmost.

produces:

\[0b10000000000000111101011000011010\]

The result is displayed according to the Base mode.

\[\text{rotate}(\text{List1}, \#\text{ofRotations}) \Rightarrow \text{list}\]

Returns a copy of List1 rotated right or left by #ofRotations elements. Does not alter List1.

If #ofRotations is positive, the rotation is to the left. If #ofRotations is negative, the rotation is to the right. The default is −1 (rotate right one element).

\[\text{rotate}(\text{String1}, \#\text{ofRotations}) \Rightarrow \text{string}\]

Returns a copy of String1 rotated right or left by #ofRotations characters. Does not alter String1.

If #ofRotations is positive, the rotation is to the left. If #ofRotations is negative, the rotation is to the right. The default is −1 (rotate right one character).
round()

round(Value1[, digits]) ⇒ value

Returns the argument rounded to the specified number of digits after the decimal point.

digits must be an integer in the range 0–12. If digits is not included, returns the argument rounded to 12 significant digits.

Note: Display digits mode may affect how this is displayed.

round(List1[, digits]) ⇒ list

Returns a list of the elements rounded to the specified number of digits.

round(Matrix1[, digits]) ⇒ matrix

Returns a matrix of the elements rounded to the specified number of digits.

rowAdd()

rowAdd(Matrix1, rIndex1, rIndex2) ⇒ matrix

Returns a copy of Matrix1 with row rIndex2 replaced by the sum of rows rIndex1 and rIndex2.

rowDim()

rowDim(Matrix) ⇒ expression

Returns the number of rows in Matrix.

Note: See also colDim(), page 23.

rowNorm()

rowNorm(Matrix) ⇒ expression

Returns the maximum of the sums of the absolute values of the elements in the rows in Matrix.

Note: All matrix elements must simplify to numbers. See also colNorm(), page 23.
rowSwap()  

\[ \text{rowSwap}(\text{Matrix1}, \text{rIndex1}, \text{rIndex2}) \Rightarrow \text{matrix} \]

Returns \( \text{Matrix1} \) with rows \( \text{rIndex1} \) and \( \text{rIndex2} \) exchanged.

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

\[
\text{rowSwap}(\text{mat}, 1, 3)
\]

\[
\begin{pmatrix}
5 & 6 \\
3 & 4 \\
1 & 2
\end{pmatrix}
\]

\rref()  

\[ \text{rref}(\text{Matrix1}, \text{Tol}) \Rightarrow \text{matrix} \]

Returns the reduced row echelon form of \( \text{Matrix1} \).

\[
\begin{pmatrix}
2 & -2 & 0 & -6 \\
1 & -1 & 9 & -9 \\
-5 & 2 & 4 & -4
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
1 & 0 & 0 & \frac{66}{71} \\
0 & 1 & 0 & \frac{147}{71} \\
0 & 0 & 1 & \frac{62}{71}
\end{pmatrix}
\]

 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 use \[\text{ctrl enter}\] or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If \( \text{Tol} \) is omitted or not used, the default tolerance is calculated as:
\[
5 \times 10^{-14} \times \text{max(dim}(\text{Matrix1}) \times \text{rowNorm}(\text{Matrix1})
\]

Note: See also \( \text{ref()} \), page 124.

\( S \)

\sec()  

\[ \text{sec}(\text{Value1}) \Rightarrow \text{value} \]

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

Returns the secant of \( \text{Value1} \) or returns a list containing the secants of all elements in \( \text{List1} \).

\[
\text{sec}(45) \Rightarrow 1.41421
\]

\[
\text{sec}\{1,2,3,4\} \Rightarrow \{1.00015,1.00081,1.00244\}
\]

Alphabetical Listing  133
sec() \[\text{trig key}\]

**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() \[\text{trig key}\]

\[
\begin{align*}
\sec^{-1}(Value1) & \Rightarrow value \\
\sec^{-1}(List1) & \Rightarrow list
\end{align*}
\]

Returns the angle whose secant is \( Value1 \) or returns a list containing the inverse secants of each element of \( List1 \).

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

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

sech() \[\text{Catalog > } \text{Math}\]

\[
\begin{align*}
\text{sech}(Value1) & \Rightarrow value \\
\text{sech}(List1) & \Rightarrow list
\end{align*}
\]

Returns the hyperbolic secant of \( Value1 \) or returns a list containing the hyperbolic secants of the \( List1 \) elements.

sech⁻¹() \[\text{Catalog > } \text{Math}\]

\[
\begin{align*}
\text{sech}^{-1}(Value1) & \Rightarrow value \\
\text{sech}^{-1}(List1) & \Rightarrow list
\end{align*}
\]

Returns the inverse hyperbolic secant of \( Value1 \) or returns a list containing the inverse hyperbolic secants of each element of \( List1 \).

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

Send `exprOrString1 [, exprOrString2] ...`

Programming command: Sends one or more TI-Innovator™ Hub commands to a connected hub.

`exprOrString` must be a valid TI-Innovator™ Hub Command. Typically, `exprOrString` contains a "SET ..." command to control a device or a "READ ..." command to request data.

The arguments are sent to the hub in succession.

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

**Note:** See also `Get` (page 59), `GetStr` (page 66), and `eval()` (page 48).

**Example:** Turn on the blue element of the built-in RGB LED for 0.5 seconds.

```
Send "SET COLOR.BLUE ON TIME .5"
```

**Example:** Request the current value of the hub’s built-in light-level sensor. A `Get` command retrieves the value and assigns it to variable `lightval`.

```
Get lightval
```

```
lightval
```

```
0.347922
```

**Example:** Send a calculated frequency to the hub’s built-in speaker. Use special variable `iostr.SendAns` to show the hub command with the expression evaluated.

```
n:=50
m:=4
Send "SET SOUND eval(m·n)"
```

```
iostr.SendAns
```

```
"SET SOUND 200"
```

**seq()**

```
seq(Expr, Var, Low, High[, Step]) ⇒ list
```

Increments `Var` from `Low` through `High` by an increment of `Step`, evaluates `Expr`, and returns the results as a list. The original contents of `Var` are still there after `seq()` is completed.

The default value for `Step = 1`.

```
seq(n²,n,1,6) \{1,4,9,16,25,36\}
```

```
seq(\frac{1}{n},n,1,10,2) \{1,\frac{1}{3},\frac{1}{5},\frac{1}{7},\frac{1}{9}\}
```

```
sum(seq(\frac{1}{n²},n,1,10,1)) \Rightarrow 1968329/1270080
```

**Note:** To force an approximate result,

**Handheld:** Press `ctrl` enter.

**Windows®:** Press Ctrl+Enter.

**Macintosh®:** Press `Esc`+Enter.

**iPad®:** Hold enter, and select \( \approx \).

```
sum(seq(\frac{1}{n²},n,1,10,1)) \Rightarrow 1.54977
```

Alphabetical Listing 135
seqGen(Expr, Var, depVar, {Var0, VarMax}, [ListOfInitTerms, VarStep[, CeilingValue]]) ⇒ list

Generates a list of terms for sequence depVar(Var)=Expr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates depVar(Var) for corresponding values of Var using the Expr formula and ListOfInitTerms, and returns the results as a list.

seqGen(ListOrSystemOfExpr, Var, ListOfDepVars, {Var0, VarMax}, [MatrixOfInitTerms, VarStep[, CeilingValue]]) ⇒ matrix

Generates a matrix of terms for a system (or list) of sequences ListOrSystemOfExpr(Var) = ListOrSystemOfExpr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates ListOrSystemOfExpr(Var) for corresponding values of Var using ListOrSystemOfExpr formula and MatrixOfInitTerms, and returns the results as a matrix.

The original contents of Var are unchanged after seqGen() is completed.

The default value for VarStep = 1.

seqn(Expr[u, n], ListOfInitTerms[, nMax[, CeilingValue]]) ⇒ list

Generates a list of terms for a sequence u(n)=Expr(u,n) as follows: Increments n from 1 through nMax by 1, evaluates u(n) for corresponding values of n using the Expr(u,n) formula and ListOfInitTerms, and returns the results as a list.

seqn(Expr[n], nMax[, CeilingValue]) ⇒ list

Generate the first 5 terms of the sequence \( u(n) = u(n-1)^2/2 \), with \( u(1)=2 \) and VarStep=1.

\[
\text{seqGen}\left(\frac{(u(n-1))^2}{n}, u, \{1,5\}, \{2\}\right)
\]

\[
\left\{2, \frac{4}{3}, \frac{4}{9}, \frac{16}{9}, \frac{16}{405}\right\}
\]

Example in which Var0=2:

\[
\text{seqGen}\left(\frac{(u(n-1)+1)^2}{n}, u, \{2,5\}, \{3\}\right)
\]

\[
\left\{\frac{4}{3}, \frac{7}{12}, \frac{19}{60}\right\}
\]

System of two sequences:

\[
\text{seqGen}\left(\frac{u_2(n-1)}{n}, u_1(n), \{u_1, u_2\}, \{1,5\}\right)
\]

\[
\left[
\begin{array}{cccc}
1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\
2 & 2 & 3 & 13 & 19 \\
\end{array}
\right]
\]

Note: The Void (_) in the initial term matrix above is used to indicate that the initial term for \( u_1(n) \) is calculated using the explicit sequence formula \( u_1(n) = 1/n \).
Generates a list of terms for a non-recursive sequence \( u(n) = \text{Expr}(n) \) as follows: Increments \( n \) from 1 through \( n_{\text{Max}} \) by 1, evaluates \( u(n) \) for corresponding values of \( n \) using the \( \text{Expr}(n) \) formula, and returns the results as a list.

If \( n_{\text{Max}} \) is missing, \( n_{\text{Max}} \) is set to 2500

If \( n_{\text{Max}} = 0 \), \( n_{\text{Max}} \) is set to 2500

**Note:** \( \text{seqn}() \) calls \( \text{seqGen}() \) with \( n0 = 1 \) and \( n_{\text{step}} = 1 \)

**setMode()**

\[
\begin{align*}
\text{setMode}(\text{modeNameInteger}, \text{settingInteger}) & \Rightarrow \text{integer} \\
\text{setMode}(\text{list}) & \Rightarrow \text{integer list}
\end{align*}
\]

Valid only within a function or program.

\( \text{setMode}(\text{modeNameInteger}, \text{settingInteger}) \) temporarily sets mode \( \text{modeNameInteger} \) to the new setting \( \text{settingInteger} \), 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.

\( \text{modeNameInteger} \) specifies which mode you want to set. It must be one of the mode integers from the table below.

\( \text{settingInteger} \) specifies the new setting for the mode. It must be one of the setting integers listed below for the specific mode you are setting.

\( \text{setMode}(\text{list}) \) lets you change multiple settings. \( \text{list} \) contains pairs of mode integers and setting integers. \( \text{setMode}(\text{list}) \) returns a similar list whose integer pairs represent the original modes and settings.

If you have saved all mode settings with \( \text{getMode}(0) \rightarrow \text{var} \), you can use \( \text{setMode} \) \( (\text{var}) \) to restore those settings until the function or program exits. See \( \text{getMode}() \), page 65.
Note: The current mode settings are passed to called subroutines. If any subroutine changes a mode setting, the mode change will be lost when control returns to the calling routine.

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

<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</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>
</tbody>
</table>

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 ►Base2, page 16.
If \#ofShifts is positive, the shift is to the left. If \#ofShifts is negative, the shift is to the right. The default is −1 (shift right one bit).

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

For example, in a right shift:

Each bit shifts right.

\[0b000000000000111101011000011010\]

Inserts 0 if leftmost bit is 0, or 1 if leftmost bit is 1.

produces:

\[0b00000000000000111101011000011010\]

The result is displayed according to the Base mode. Leading zeros are not shown.

\[\text{shift}(\text{List1}[\#\text{ofShifts}]) \Rightarrow \text{list}\]

Returns a copy of \text{List1} shifted right or left by \#ofShifts elements. Does not alter \text{List1}.

If \#ofShifts is positive, the shift is to the left. If \#ofShifts is negative, the shift is to the right. The default is −1 (shift right one element).

Elements introduced at the beginning or end of \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 \#ofShifts characters. Does not alter \text{String1}.

If \#ofShifts is positive, the shift is to the left. If \#ofShifts is negative, the shift is to the right. The default is −1 (shift right one character).

**Important:** To enter a binary or hexadecimal number, always use the 0b or 0h prefix (zero, not the letter O).
**shift()**

Characters introduced at the beginning or end of string by the shift are set to a space.

**sign()**

\[
\begin{align*}
\text{sign}(Value1) & \Rightarrow value \\
\text{sign}(List1) & \Rightarrow list \\
\text{sign}(Matrix1) & \Rightarrow matrix
\end{align*}
\]

For real and complex Value1, returns \( Value1 / \text{abs}(Value1) \) when \( Value1 \neq 0 \).

Returns 1 if \( Value1 \) is positive. Returns \(-1\) if \( Value1 \) is negative. \( \text{sign}(0) \) returns \( \pm 1 \) if the complex format mode is Real; otherwise, it returns itself.

\( \text{sign}(0) \) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.

**simult()**

\[
\begin{align*}
\text{simult}(\text{coeffMatrix}, \text{constVector}[, \text{Tol}]) & \Rightarrow matrix
\end{align*}
\]

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also \( \text{linSolve()} \), page 82.

\( \text{coeffMatrix} \) 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 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:
  \(5 \times 10^{-14} \times \max(\text{dim}(\text{coeffMatrix}))\)
  \(\times \text{rowNorm}(\text{coeffMatrix})\)

**simult**(\(\text{coeffMatrix}, \text{constMatrix}[, \text{Tol}]\) ⇒ 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.

**sin()**

- **sin**(\(\text{Value}\)) ⇒ value
- **sin**(\(\text{List}\)) ⇒ list

\(\text{sin(Value)}\) returns the sine of the argument.

\(\text{sin(List)}\) returns a list of the sines of all elements in \(\text{List}\).

**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.

**sin(squareMatrix)** ⇒ squareMatrix

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

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

\[
\begin{bmatrix}
  1 & 5 & 3 \\
  4 & 2 & 1 \\
  6 & 2 & 1 
\end{bmatrix}
\]

\[
\begin{bmatrix}
  0.9424 & -0.04542 & -0.031999 \\
  -0.04542 & 0.949254 & -0.020274 \\
  -0.048739 & -0.00523 & 0.961051 
\end{bmatrix}
\]

**sin⁻¹()**

\[
\sin^{-1}(Value1) \Rightarrow value \\
\sin^{-1}(List1) \Rightarrow list 
\]

\[
\sin^{-1}(Value1) \text{ returns the angle whose sine is } Value1. \\
\sin^{-1}(List1) \text{ returns a list of the inverse sines of each element of List1.}
\]

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

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

\[
\sin^{-1}(squareMatrix1) \Rightarrow squareMatrix 
\]

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

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

**sinh()**

\[
\sinh(Numver1) \Rightarrow value \\
\sinh(List1) \Rightarrow list 
\]

\[
\sinh(Value1) \text{ returns the hyperbolic sine of the argument.} \\
\sinh(List1) \text{ returns a list of the hyperbolic sines of each element of List1.}
\]
\( \text{sinh}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)

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 \( \text{cos}() \).

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

\( \text{sinh}^{-1}() \)

\( \text{sinh}^{-1}(\text{Value1}) \Rightarrow \text{value} \)

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

\( \text{sinh}^{-1}(\text{Value1}) \) returns the inverse hyperbolic sine of the argument.

\( \text{sinh}^{-1}(\text{List1}) \) returns a list of the inverse hyperbolic sines of each element of \( \text{List1} \).

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

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

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 \( \text{cos}() \).

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

\( \text{SinReg} \)

\( \text{SinReg} X, Y[, \text{Iterations},[\text{Period}][, \text{Category}, \text{Include}]] \)

Computes the sinusoidal regression on lists \( X \) and \( Y \). A summary of results is stored in the \text{stat.results} variable. (See page 146.)

All the lists must have equal dimension except for \( \text{Include} \).
$X$ and $Y$ are lists of independent and dependent variables.

*Iterations* is a value that specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

*Period* specifies an estimated period. If omitted, the difference between values in $X$ should be equal and in sequential order. If you specify *Period*, the differences between $x$ values can be unequal.

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

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

The output of **SinReg** is always in radians, regardless of the angle mode setting.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $a \cdot \sin(bx+c)+d$</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.XReg</td>
<td>List of data points in the modified $X$ List actually used in the regression based on restrictions of <em>Freq</em>, <em>Category List</em>, and <em>Include Categories</em></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 <em>Freq</em>, <em>Category List</em>, and <em>Include Categories</em></td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to <em>stat.XReg</em> and <em>stat.YReg</em></td>
</tr>
</tbody>
</table>
SortA  List1[, List2] [, List3]...
SortA  Vector1[, Vector2] [, Vector3]...

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 212.

SortD  List1[, List2][, List3]...
SortD  Vector1[, Vector2][, Vector3]...

Identical to SortA, except SortD sorts the elements in descending order.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 212.

► 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 [ρ ∠ θ ∠ φ].

Vector must be of dimension 3 and can be either a row or a column vector.
**Sphere**

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

![Sphere diagram]

---

### sqrt()

<table>
<thead>
<tr>
<th>sqrt(Value) ⇒ value</th>
<th>sqrt(List) ⇒ list</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sqrt{4} ) = 2</td>
<td>( \sqrt{{9,2,4}} ) = {3.1414212}</td>
</tr>
</tbody>
</table>

Returns the square root of the argument.

For a list, returns the square roots of all the elements in **List**.

**Note:** See also **Square root template**, page 1.

---

### stat.results

**stat.results**

Displays results from a statistics calculation.

The results are displayed as a set of name-value pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

**Note:** Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

---

**stat.values**

<table>
<thead>
<tr>
<th>&quot;Linear Regression (mx+b)&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;m*<code>x</code>+b&quot;</td>
</tr>
<tr>
<td>3.2</td>
</tr>
<tr>
<td>1.2</td>
</tr>
<tr>
<td>0.996109</td>
</tr>
<tr>
<td>0.998053</td>
</tr>
<tr>
<td>&quot;{0.4,0.4,0.2,0.0}&quot;</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Vector ( {1,2,3,4,5} )</th>
<th>Vector ( {4,8,11,14,17} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinRegMx xlist,ylist,1: stat.results</td>
<td></td>
</tr>
<tr>
<td>[&quot;Title&quot;, &quot;Linear Regression (mx+b)&quot; ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;RegEqn&quot;, &quot;m*x+b&quot; ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;m&quot;, 3.2 ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;b&quot;, 1.2 ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;r^2&quot;, 0.996109 ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;t&quot;, 0.998053 ]</td>
<td></td>
</tr>
<tr>
<td>[&quot;Resid&quot;, &quot;{}&quot; ]</td>
<td></td>
</tr>
</tbody>
</table>

---

146 **Alphabetical Listing**
stat.a  stat.dfDenom  stat.MedianY  stat.Q3X  stat.SSBlock  
stat.AdjR²  stat.dfBlock  stat.MEDpred  stat.Q3Y  stat.SSCol  
stat.b  stat.dfCol  stat.MinX  stat.r  stat.SS  
stat.b0  stat.dfError  stat.MinY  stat.r²  stat.SSE  
stat.b1  stat.dfInteract  stat.MS  stat.RegEqn  stat.SSResid  
stat.b2  stat.dfReg  stat.MSBlock  stat.Resid  stat.SSInteract  
stat.b3  stat.dfNumer  stat.MSCol  stat.ResidTrans  stat.SSReg  
stat.b4  stat.dfRow  stat.MSError  stat.σx  stat.SSRow  
stat.b5  stat.DW  stat.MSInteract  stat.σy  stat.tList  
stat.b6  stat.e  stat.MSReg  stat.σx1  stat.UpperPred  
stat.b8  stat.F  stat.n  stat.Σx  stat.χ  
stat.b9  stat.FBlock  Stat.ₚ  stat.Σx²  stat.χ₁  
stat.b10  stat.Fcol  stat.ₚ₁  stat.Σxy  stat.χ₂  
stat.bList  stat.FInteract  stat.ₚ₂  stat.Σy  stat.χDiff  
stat.χ²  stat.FreqReg  stat.ₚDiff  stat.Σy²  stat.χList  
stat.c  stat.Frow  stat.PList  stat.s  stat.XReg  
stat.CLower  stat.Leverage  stat.PVal  stat.σE  stat.XVal  
stat.CLowerList  stat.LowerPred  stat.PValBlock  stat.σEList  stat.XValList  
stat.CompMatrix  stat.m  stat.PValInteract  stat.sResid  stat.ȳList  
stat.CookDist  stat.MaxX  stat.PValRow  stat.σslope  stat.YReg  
stat.CUpper  stat.MaxY  stat.Q1X  stat.sp  stat.YList  
stat.CUpperList  stat.ME  stat.Q1Y  stat.SS  
stat.d  stat.MedianX  

Note: Each time the Lists & Spreadsheet application calculates statistical results, it copies the “stat.” group variables to a “stat#.,” group, where # is a number that is incremented automatically. This lets you maintain previous results while performing multiple calculations.

stat.values

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike stat.results, stat.values omits the names associated with the values.

You can copy a value and paste it into other locations.
**stDevPop()**

**stDevPop(List [,freqList]) ⇒ expression**

Returns the population standard deviation of the elements in List.

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

**Note:** List must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 212.

**stDevPop(Matrix1 [,freqMatrix]) ⇒ matrix**

Returns a row vector of the population standard deviations of the columns in Matrix1.

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

**Note:** Matrix1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 212.

---

**stDevSamp()**

**stDevSamp(List [,freqList]) ⇒ expression**

Returns the sample standard deviation of the elements in List.

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

**Note:** List must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 212.
**stDevSamp()**

**stDevSamp**(Matrix1[, freqMatrix]) ⇒ matrix

Returns a row vector of the sample standard deviations of the columns in Matrix1.

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

**Note:** Matrix1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 212.

---

**Stop**

**Stop**

Programming command: Terminates the program.

Stop is not allowed in functions.

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

---

**Store**

See →(store), page 194.

---

**string()**

**string**(Expr) ⇒ string

Simplifies Expr and returns the result as a character string.
**subMat()**

*subMat*(Matrix1[, startRow][, startCol][, endRow][, endCol]) ⇒ matrix

Returns the specified submatrix of Matrix1.

Defaults: startRow=1, startCol=1, endRow=last row, endCol=last column.

<table>
<thead>
<tr>
<th>startRow</th>
<th>startCol</th>
<th>endRow</th>
<th>endCol</th>
<th>Matrix1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td>1 2 3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td>4 5 6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td>7 8 9</td>
</tr>
</tbody>
</table>

subMat(*m1*, 1, 1, 2, 2)

subMat(*m1*, 1, 2, 3, 3)

**Sum (Sigma)**

See Σ(), page 187.

**sum()**

*sum*(List[, Start[, End]]) ⇒ expression

Returns the sum of all elements in List.

Start and End are optional. They specify a range of elements.

Any void argument produces a void result. Empty (void) elements in List are ignored. For more information on empty elements, see page 212.

<table>
<thead>
<tr>
<th>List</th>
<th>Start</th>
<th>End</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, 3, 4, 5}</td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>{a, 2<em>a, 3</em>a}</td>
<td></td>
<td></td>
<td>&quot;Error: Variable is not defined&quot;</td>
</tr>
<tr>
<td>seq(n, n, 1, 10)</td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>{1, 3, 5, 7, 9}, 3</td>
<td></td>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix1</th>
<th>Start</th>
<th>End</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td></td>
<td>5 7 9</td>
</tr>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td></td>
<td>12 15 18</td>
</tr>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td>2,3</td>
<td>11 13 15</td>
</tr>
</tbody>
</table>

**sumIf()**

*sumIf*(List, Criteria[, SumList]) ⇒ value

Returns the accumulated sum of all elements in List that meet the specified Criteria. Optionally, you can specify an alternate list, sumList, to supply the elements to accumulate.

<table>
<thead>
<tr>
<th>List</th>
<th>Criteria</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2, e, 3, π, 4, 5, 6}</td>
<td>2.5&lt;?&lt;4.5</td>
<td>12.859874482</td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
<td>2&lt;?&lt;5, {10, 20, 30, 40}</td>
<td>70</td>
</tr>
</tbody>
</table>
**sumIf()**

`List` can be an expression, list, or matrix. `SumList`, if specified, must have the same dimension(s) as `List`.

**Criteria** can be:

- A value, expression, or string. For example, 34 accumulates only those elements in `List` that simplify to the value 34.
- A Boolean expression containing the symbol `?` as a placeholder for each element. For example, `?<10` accumulates only those elements in `List` that are less than 10.

When a `List` element meets the `Criteria`, the element is added to the accumulating sum. If you include `sumList`, the corresponding element from `sumList` is added to the sum instead.

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

Empty (void) elements are ignored. For more information on empty elements, see page 212.

**Note:** See also `countIf()`, page 29.

**sumSeq()**

See $\Sigma()$, page 187.

**system()**

`system(Value1[, Value2[, Value3[, ...]]])`

Returns a system of equations, formatted as a list. You can also create a system by using a template.
\[ T \]

### T (transpose)

Matrix1\( T \) ⇒ matrix

Returns the complex conjugate transpose of Matrix1.

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

### tan()

\( \tan(Value1) \) ⇒ value
\( \tan(List1) \) ⇒ list

**tan(Value1)** returns the tangent of the argument.

**tan(List1)** returns a list of the tangents of all elements in List1.

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

\[ \tan(squareMatrix1) \] ⇒ squareMatrix

Returns the matrix tangent of squareMatrix1. This is not the same as calculating the tangent of each element. For information about the calculation method, refer to cos().
\textbf{\textnormal{tan}()} \gridcell{trig\;key}

\textit{squareMatrix1} must be diagonalizable. The result always contains floating-point numbers.

\textbf{\textnormal{tan}⁻¹()} \gridcell{trig\;key}

\textbf{\textnormal{tan}⁻¹(}Value\textbf{\textnormal{)}} \Rightarrow \textbf{value} \begin{align*}
\text{In Degree angle mode:} \\
\text{tan}^{-1}(1) & \Rightarrow 45 \\
\text{In Gradian angle mode:} \\
\text{tan}^{-1}(1) & \Rightarrow 50 \\
\text{In Radian angle mode:} \\
\text{tan}^{-1}\left\{0.2, 0.5\right\} & \Rightarrow \{0.0, 0.197396, 0.463648\}
\end{align*}

\textbf{\textnormal{tan}⁻¹(}List\textbf{\textnormal{)}} \Rightarrow \textbf{list}

\textbf{\textnormal{tan}⁻¹(}Value\textbf{\textnormal{)}} returns the angle whose tangent is \textit{Value1}.

\textbf{\textnormal{tan}⁻¹(}List\textbf{\textnormal{)}} returns a list of the inverse tangents of each element of \textit{List1}.

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

\textbf{Note:} You can insert this function from the keyboard by typing \texttt{arctan(...)\;}. \begin{align*}
\textbf{\textnormal{tan}⁻¹(}\textbf{squareMatrix1}\textbf{\textnormal{)}} \Rightarrow \textbf{squareMatrix} \\
\text{In Radian angle mode:} \\
\text{\begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & 2 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} -0.083658 & 1.26629 & 0.62263 \\ 0.748539 & 0.630015 & -0.070012 \\ 1.68608 & -1.18244 & 0.455126 \end{pmatrix}}
\end{align*}

\textbf{\textnormal{tanh}()} \gridcell{Catalog \;> \; 2\;3}

\textbf{\textnormal{tanh}(}Value\textbf{\textnormal{)}} \Rightarrow \textbf{value}

\textbf{\textnormal{tanh}(}List\textbf{\textnormal{)}} \Rightarrow \textbf{list}

\textbf{\textnormal{tanh}(}Value\textbf{\textnormal{)}} returns the hyperbolic tangent of the argument.

\textbf{\textnormal{tanh}(}List\textbf{\textnormal{)}} returns a list of the hyperbolic tangents of each element of \textit{List1}.

\textbf{\textnormal{tanh}(}squareMatrix\textbf{\textnormal{)}} \Rightarrow \textbf{squareMatrix} \begin{align*}
\text{In Radian angle mode:} \\
\text{tanh}(1.2) & \Rightarrow 0.833655 \\
\text{tanh}(\{0,1\}) & \Rightarrow \{0., 0.761594\}
\end{align*}
**tanh()**  

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

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

| tanh | Catalog >  
|------|------------
| \[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\] | \[
\begin{bmatrix}
-0.097966 & 0.933436 & 0.425972 \\
0.488147 & 0.538881 & 0.129382 \\
1.28295 & -1.03425 & 0.428817
\end{bmatrix}
\] |

**tanh⁻¹()**

- `tanh⁻¹(Value1) ⇒ value`
- `tanh⁻¹(List1) ⇒ list`

`tanh⁻¹(Value1)` returns the inverse hyperbolic tangent of the argument.

`tanh⁻¹(List1)` returns a list of the inverse hyperbolic tangents of each element of `List1`.

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

- `tanh⁻¹(squareMatrix1) ⇒ squareMatrix`

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

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

In Rectangular complex format:

```
tanh⁻¹(0)
```

```
0.
```

```
tanh⁻¹\{1,2,1,3\}
```

```
\{undef,0.518046−1.5708·i,0.346574−1.5708·i,...\}
```

To see the entire result, press `▲` and then use `◄` and `►` to move the cursor.

In Radian angle mode and Rectangular complex format:

```
tanh⁻¹\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & 2 & 1
\end{bmatrix}
\]
```

```
\[
\begin{bmatrix}
-0.099353+0.164058·i & 0.267834−1.4908·i \\
-0.087596−0.725533·i & 0.479679−0.94730·i \\
0.511463−2.08316·i & -0.878563+1.7901·i
\end{bmatrix}
\]
```

To see the entire result, press `▲` and then use `◄` and `►` to move the cursor.

**tCdf()**

- `tCdf(lowBound,upBound,df) ⇒ number` if `lowBound` and `upBound` are numbers, `list` if `lowBound` and `upBound` are lists

Computes the Student-\( t \) distribution probability between `lowBound` and `upBound` for the specified degrees of freedom `df`.

---

154  Alphabetical Listing
For $P(X \leq upBound)$, set $lowBound = '9E999$.

**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 126, or **RequestStr**, page 127.

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

Define a program that pauses to display each of five random numbers in a dialog box.

Within the Prgm...EndPrgm template, complete each line by pressing `@` instead of `·`. On the computer keyboard, hold down `Alt` and press `Enter`.

```plaintext
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:

![Dialog Box Example](image)

**tInterval**

**tInterval** `List[, Freq[, CLevel]]`

(Data list input)

**tInterval** $\bar{x}, sx, n[, CLevel]$

(Summary stats input)

---

**Then**

See If, page 68.

---

**tCdf()**  
**Catalog >**
**tInterval**

Computes a $t$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 146.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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**

`tInterval_2Samp List1, List2 [,Freq1 [,Freq2 [,CLevel [,Pooled]]]]`

(Data list input)

`tInterval_2Samp x1, sx1, n1, x2, sx2, n2 [,CLevel [,Pooled]]`

(Summary stats input)

Computes a two-sample $t$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 146.)

*Pooled=1* pools variances; *Pooled=0* does not pool variances.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.(\bar{x}_1), stat.(\bar{x}_2)</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.(\sigma_1), stat.(\sigma_2)</td>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>stat.sp</td>
<td>The pooled standard deviation. Calculated when Pooled = YES</td>
</tr>
</tbody>
</table>

**tPdf()**

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

Computes the probability density function (pdf) for the Student-\(t\) distribution at a specified \(x\) value with specified degrees of freedom \(df\).

**trace()**

\[ \text{trace}(\text{squareMatrix}) \Rightarrow \text{value} \]

Returns the trace (sum of all the elements on the main diagonal) of \(\text{squareMatrix}\).
Try

\begin{align*}
\text{Try} & \quad \text{block1} \\
\text{Else} & \quad \text{block2} \\
\text{EndTry}
\end{align*}

Executes block1 unless an error occurs. Program execution transfers to block2 if an error occurs in block1. System variable errCode contains the error code to allow the program to perform error recovery. For a list of error codes, see “Error codes and messages,” page 222.

block1 and block2 can be either a single statement or a series of statements separated with the “;” character.

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

To see the commands Try, ClrErr, and PassErr in operation, enter the eigenvals() program shown at the right. Run the program by executing each of the following expressions.

\[
\begin{bmatrix}
  -3 \\ -41 \\ 5
\end{bmatrix}
\]

Note: See also ClrErr, page 22, and PassErr, page 110.
**tTest**

\[ \text{tTest} \mu_0,\text{List}[,.\text{Freq}[.,\text{Hypothe}]] \]

(Data list input)

\[ \text{tTest} \mu_0,\overline{x},sx,n,[\text{Hypothe}] \]

(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 \( \text{stat.results} \) variable. (See page 146.)

Test \( H_0 : \mu = \mu_0 \), against one of the following:

For \( H_a : \mu < \mu_0 \), set \( \text{Hypothe}<0 \)
For \( H^a_0 : \mu \neq \mu_0 \) (default), set \( \text{Hypothe}=0 \)
For \( H^a_1 : \mu > \mu_0 \), set \( \text{Hypothe}>0 \)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.t} )</td>
<td>( (\overline{x} - \mu_0) / (\text{stdev} / \sqrt{n}) )</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</td>
</tr>
<tr>
<td>( \text{stat.}\overline{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</td>
</tr>
<tr>
<td>( \text{stat.n} )</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

**tTest_2Samp**

\[ \text{tTest}_2\text{Samp} \text{List1},\text{List2}[,.\text{Freq1}[,.\text{Freq2} [,.\text{Hypothe}[,.\text{Pooled}]]]) \]

(Data list input)

\[ \text{tTest}_2\text{Samp} \overline{x}_1,sx_1,n_1,\overline{x}_2,sx_2,n_2,[,\text{Hypothe} [,\text{Pooled}]]] \]

(Summary stats input)
**tTest_2Samp**

Computes a two-sample t test. A summary of results is stored in the `stat.results` variable. (See page 146.)

Test \( H_0 \): \( \mu_1 = \mu_2 \), against one of the following:

For \( H_a \): \( \mu_1 < \mu_2 \), set \( Hypoth < 0 \)

For \( H_a \): \( \mu_1 \neq \mu_2 \) (default), set \( Hypoth = 0 \)

For \( H_a \): \( \mu_1 > \mu_2 \), set \( Hypoth > 0 \)

\( Pooled = 1 \) pools variances

\( Pooled = 0 \) does not pool variances

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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>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.( s_x )1, stat.( s_x )2</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(N,\( PV\),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 162. See also `amortTbl()`, page 7.

```
tvmFV(120,5,0,-500,12,12) \Rightarrow 77641.1
```

**tvml()**

```
tvml(N,\( PV\),Pmt,\( FV\),[PpY],[CpY],[PmtAt]) \Rightarrow value
```

```
tvml(240,100000,-1000,0,12,12) \Rightarrow 10.5241
```
tvmI()  
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 162. See also amortTbl(), page 7.

tvmN()  
tvmN(I,PV,Pmt,FV,[PpY],[CpY],[PmtAt]) ⇒ 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 162. See also amortTbl(), page 7.

tvmPmt()  
tvmPmt(N,I,PV,FV,[PpY],[CpY],[PmtAt]) ⇒ 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 162. See also amortTbl(), page 7.

tvmPV()  
tvmPV(N,I,Pmt,FV,[PpY],[CpY],[PmtAt]) ⇒ value  
Financial function that calculates the present value.

Note: Arguments used in the TVM functions are described in the table of TVM arguments, page 162. See also amortTbl(), page 7.
<table>
<thead>
<tr>
<th>TVM argument*</th>
<th>Description</th>
<th>Data type</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>
<tr>
<td>PV</td>
<td>Present value</td>
<td>real number</td>
</tr>
<tr>
<td>Pmt</td>
<td>Payment amount</td>
<td>real number</td>
</tr>
<tr>
<td>FV</td>
<td>Future value</td>
<td>real number</td>
</tr>
<tr>
<td>PpY</td>
<td>Payments per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>CpY</td>
<td>Compounding periods per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>PmtAt</td>
<td>Payment due at the end or beginning of each period, default=end</td>
<td>integer (0=end, 1=beginning)</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**

**TwoVar** *X*, *Y*, [*Freq*], [*Category*], [*Include*])

Calculates the TwoVar statistics. A summary of results is stored in the *stat.results* variable. (See page 146.)

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 212.

<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.$\sum x$</td>
<td>Sum of $x$ values</td>
</tr>
<tr>
<td>stat.$\sum x^2$</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.$\sigma x$</td>
<td>Population standard deviation of $x$</td>
</tr>
<tr>
<td>stat.$n$</td>
<td>Number of data points</td>
</tr>
<tr>
<td>stat.$\bar{y}$</td>
<td>Mean of $y$ values</td>
</tr>
<tr>
<td>stat.$\sum y$</td>
<td>Sum of $y$ values</td>
</tr>
<tr>
<td>stat.$\sum y^2$</td>
<td>Sum of $y^2$ values</td>
</tr>
<tr>
<td>stat.$sy$</td>
<td>Sample standard deviation of $y$</td>
</tr>
<tr>
<td>stat.$\sigma y$</td>
<td>Population standard deviation of $y$</td>
</tr>
<tr>
<td>stat.$\sum xy$</td>
<td>Sum of $x \times y$ values</td>
</tr>
<tr>
<td>stat.$r$</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.$\text{Min}X$</td>
<td>Minimum of $x$ values</td>
</tr>
<tr>
<td>stat.$Q_1X$</td>
<td>1st Quartile of $x$</td>
</tr>
<tr>
<td>stat.$\text{Median}X$</td>
<td>Median of $x$</td>
</tr>
<tr>
<td>stat.$Q_3X$</td>
<td>3rd Quartile of $x$</td>
</tr>
<tr>
<td>stat.$\text{Max}X$</td>
<td>Maximum of $x$ values</td>
</tr>
<tr>
<td>stat.$\text{Min}Y$</td>
<td>Minimum of $y$ values</td>
</tr>
<tr>
<td>stat.$Q_1Y$</td>
<td>1st Quartile of $y$</td>
</tr>
<tr>
<td>stat.$\text{Med}Y$</td>
<td>Median of $y$</td>
</tr>
<tr>
<td>stat.$Q_3Y$</td>
<td>3rd Quartile of $y$</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.MaxY</td>
<td>Maximum of y values</td>
</tr>
<tr>
<td>stat.Σ(x-\bar{x})^2</td>
<td>Sum of squares of deviations from the mean of x</td>
</tr>
<tr>
<td>stat.Σ(y-\bar{y})^2</td>
<td>Sum of squares of deviations from the mean of y</td>
</tr>
</tbody>
</table>

U

**unitV()**

\[ \text{unitV}(\text{Vector1}) \Rightarrow \text{vector} \]

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.

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

\[
\begin{bmatrix}
0.408248 & 0.816497 & 0.408248
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
0.267261 \\
0.534522 \\
0.801784
\end{bmatrix}
\]

unLock

\text{unLock} \ Var1, Var2 \ [, Var3] ... unLock \ Var.

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

See Lock, page 86, and getLockInfo(), page 64.

V

**varPop()**

\[ \text{varPop}(\text{List[, freqList]}) \Rightarrow \text{expression} \]

Returns the population variance of List.

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

**Note:** List must contain at least two elements.

\[
\text{varPop}\{5,10,15,20,25,30\} \Rightarrow 72.9167\]
**varPop()**

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 212.

**varSamp()**

\[ \text{varSamp(List[, freqList])} \Rightarrow \text{expression} \]

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 212.

\[ \text{varSamp(Matrix1[, freqMatrix])} \Rightarrow \text{matrix} \]

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 212.

**Note:** \( \text{Matrix1} \) must contain at least two rows.

**W**

**Wait**

\[ \text{Wait timeInSeconds} \]

To wait 4 seconds:

```
Wait 4
```
Wait

Suspends execution for a period of \textit{timeInSeconds} seconds.

\textbf{Wait} is particularly useful in a program that needs a brief delay to allow requested data to become available.

The argument \textit{timeInSeconds} must be an expression that simplifies to a decimal value in the range 0 through 100. The command rounds this value up to the nearest 0.1 seconds.

To cancel a \textbf{Wait} that is in progress,

- \textbf{Handheld}: Hold down the [ \texttt{On} ] key and press [ \texttt{enter} ] repeatedly.
- \textbf{Windows®}: Hold down the \texttt{F12} key and press \texttt{Enter} repeatedly.
- \textbf{Macintosh®}: Hold down the \texttt{F5} key and press \texttt{Enter} repeatedly.
- \textbf{iPad®}: The app displays a prompt. You can continue waiting or cancel.

\textbf{Note}: You can use the \textbf{Wait} command within a user-defined program but not within a function.

\textbf{warnCodes ( )}

\textbf{warnCodes}(Expr1, StatusVar) \Rightarrow expression

Evaluates expression \textit{Expr1}, returns the result, and stores the codes of any generated warnings in the \textit{StatusVar} list variable. If no warnings are generated, this function assigns \textit{StatusVar} an empty list.

\textit{Expr1} can be any valid TI-Nspire™ or TI-Nspire™ CAS math expression. You cannot use a command or assignment as \textit{Expr1}.

\textit{StatusVar} must be a valid variable name.

For a list of warning codes and associated messages, see page 230.

To wait 1/2 second:
Wait 0.5

To wait 1.3 seconds using the variable \textit{seccount}:
seccount:=1.3
Wait seccount

This example switches a green LED on for 0.5 seconds and then switches it off.
Send "SET GREEN 1 ON"
Wait 0.5
Send "SET GREEN 1 OFF"
### when()

**When**\( \text{Condition, trueResult [, falseResult] [, unknownResult]} \) \( \Rightarrow \) \text{expression}

Returns \text{trueResult}, \text{falseResult}, or \text{unknownResult}, depending on whether \text{Condition} is true, false, or unknown. Returns the input if there are too few arguments to specify the appropriate result.

Omit both \text{falseResult} and \text{unknownResult} to make an expression defined only in the region where \text{Condition} is true.

Use an \text{undef} \text{falseResult} to define an expression that graphs only on an interval.

\text{when()} is helpful for defining recursive functions.

| when\(\{x<0,x+3\}|x=5\) | undefined |
|--------------------------|------------|

### While

\textbf{While} \text{Condition}  
\textbf{Block}  
\textbf{EndWhile}

Executes the statements in \text{Block} as long as \text{Condition} is true.

\text{Block} can be either a single statement or a sequence of statements separated with the \text{";"} character.

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

### xor

\text{BooleanExpr1 xor BooleanExpr2} \text{returns Boolean expression}  
\text{Boolean expressionBooleanList1}  
\text{xor BooleanList2} \text{returns Boolean list}  
\text{BooleanMatrix1}  
\text{xor BooleanMatrix2} \text{returns Boolean}
Alphabetical Listing

**xor**

**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 108.

**Integer1 xor Integer2 ⇒ integer**

Comparer two real integers bit-by-bit using an xor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

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

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ►Base2, page 16.

**Note:** See or, page 108.

**Z**

**zInterval**

```plaintext
zInterval σ,List[,Freq[,CLevel]]
```

(Data list input)

```plaintext
zInterval σ,\bar{x},n [,CLevel]
```

(Summary stats input)
**zInterval**

Computes a $z$ confidence interval. A summary of results is stored in the `stat.results` variable. (See page 146.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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.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.σ</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 146.)

$x$ is a non-negative integer.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<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.\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>
<tr>
<td><code>stat.n</code></td>
<td>Number of samples in data sequence</td>
</tr>
</tbody>
</table>

**zInterval_2Prop**

`zInterval_2Prop x1,n1,x2,n2 [,CLevel]`
zInterval_2Prop

Computes a two-proportion $z$ confidence interval. A summary of results is stored in the stat.results variable. (See page 146.)

$x_1$ and $x_2$ are non-negative integers.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>stat.( \hat{p} ) Diff</td>
<td>The calculated difference between proportions</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.( \hat{p} )_1</td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td>stat.( \hat{p} )_2</td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td>stat.n1</td>
<td>Sample size in data sequence one</td>
</tr>
<tr>
<td>stat.n2</td>
<td>Sample size in data sequence two</td>
</tr>
</tbody>
</table>

zInterval_2Samp

\[ zInterval_2Samp \sigma_1, \sigma_2, List1, List2[, Freq1, Freq2[, CLevel]] ]

(Data list input)

\[ zInterval_2Samp \sigma_1, \sigma_2, \bar{x}_1, n_1, \bar{x}_2, n_2[, CLevel] ]

(Summary stats input)

Computes a two-sample $z$ confidence interval. A summary of results is stored in the stat.results variable. (See page 146.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.(\bar{x}_1 - \bar{x}_2)</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.(\bar{x}_1), stat.(\bar{x}_2)</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.(\sigma_1), stat.(\sigma_2)</td>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>stat.r1, stat.r2</td>
<td>Known population standard deviations for data sequence List 1 and List 2</td>
</tr>
</tbody>
</table>

**zTest**

zTest \(\mu_0, \sigma, \text{List},[\text{Freq},[,\text{Hypoth}]\]  
(Data list input)

zTest \(\mu_0, \sigma, \bar{x}, n[,\text{Hypoth}]\]  
(Summary stats input)

Performs a z test with frequency freqlist. A summary of results is stored in the stat.results variable. (See page 146.)

Test \(H_0: \mu = \mu_0\), against one of the following:

For \(H^<\) : \(\mu < \mu_0\), set Hypoth<0  
For \(H^\neq\) : \(\mu \neq \mu_0\) (default), set Hypoth=0  
For \(H^>\) : \(\mu > \mu_0\), set Hypoth>0  

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>(\frac{(\bar{x} - \mu_0)}{\sigma / \sqrt{n}})</td>
</tr>
<tr>
<td>stat.P Value</td>
<td>Least probability at which the null hypothesis can be rejected</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. Only returned for Data input.</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>
**zTest_1Prop**

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.p0</td>
<td>Hypothesized population proportion</td>
</tr>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the proportion</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.( \hat{p} )</td>
<td>Estimated sample proportion</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

**zTest_2Prop**

\texttt{zTest_2Prop \( x1,n1,x2,n2[,Hypoth] \)}

Computes a two-proportion \( z \) test. A summary of results is stored in the \texttt{stat.results} variable. (See page 146.)

\( x1 \) and \( x2 \) are non-negative integers.

Test \( H_0 : p1 = p2 \), against one of the following:

For \( H_a : p1 > p2 \), set \texttt{Hypoth}>0
For \( H_a : p1 \neq p2 \) (default), set \texttt{Hypoth}=0
For \( H_a : p < p0 \), set \texttt{Hypoth}<0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the difference of proportions</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.( \hat{p} ) 1</td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td>stat.( \hat{p} ) 2</td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td>stat.( \hat{p} )</td>
<td>Pooled sample proportion estimate</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples taken in trials 1 and 2</td>
</tr>
</tbody>
</table>

**zTest_2Samp**

\texttt{zTest_2Samp \( \sigma_1,\sigma_2,List1,List2[,Freq]\)}
zTest_2Samp

[,Freq2[,Hypoth]]

(Data list input)

zTest_2Samp σ₁,σ₂,\bar{x}_1,n₁,\bar{x}_2,n₂[,Hypoth]

(Summary stats input)

Computes a two-sample z test. A summary of results is stored in the stat.results variable. (See page 146.)

Test H₀: μ₁ = μ₂, against one of the following:

For H₀ : μ₁ < μ₂, set Hypoth<0
For H₀ : μ₁ ≠ μ₂ (default), set Hypoth=0
For H₀ : μ₁ > μ₂, Hypoth>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 212.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the difference of means</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.\bar{x}_1, stat.\bar{x}_2</td>
<td>Sample means of the data sequences in List1 and List2</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations of the data sequences in List1 and List2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>
Symbols

+ (add)

Value1 + Value2 ⇒ value

Returns the sum of the two arguments.

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.

Value + List1 ⇒ list

List1 + Value ⇒ list

Returns a list containing the sums of Value and each element in List1.

Value + Matrix1 ⇒ matrix

Matrix1 + Value ⇒ matrix

Returns a matrix with Value added to each element on the diagonal of Matrix1. Matrix1 must be square.

Note: Use .+ (dot plus) to add an expression to each element.

− (subtract)

Value1−Value2 ⇒ value

Returns Value1 minus Value2.

List1−List2 ⇒ list

Matrix1−Matrix2 ⇒ matrix

174 Symbols
\( - \) \text{(subtract)}

Subtracts each element in \( \text{List2} \) (or \( \text{Matrix2} \)) from the corresponding element in \( \text{List1} \) (or \( \text{Matrix1} \)), and returns the results.

Dimensions of the arguments must be equal.

\[ \text{Value} - \text{List1} \Rightarrow \text{list} \]

\[ \text{List1} - \text{Value} \Rightarrow \text{list} \]

Subtracts each \( \text{List1} \) element from \( \text{Value} \) or subtracts \( \text{Value} \) from each \( \text{List1} \) element, and returns a list of the results.

\[ \text{Value} - \text{Matrix1} \Rightarrow \text{matrix} \]

\[ \text{Matrix1} - \text{Value} \Rightarrow \text{matrix} \]

\( \text{Value} - \text{Matrix1} \) returns a matrix of \( \text{Value} \) times the identity matrix minus \( \text{Matrix1} \). \( \text{Matrix1} \) must be square.

\( \text{Matrix1} - \text{Value} \) returns a matrix of \( \text{Value} \) times the identity matrix subtracted from \( \text{Matrix1} \). \( \text{Matrix1} \) must be square.

\textbf{Note:} Use \( - \) (dot minus) to subtract an expression from each element.

\( \times \) \text{(multiply)}

\[ \text{Value1} \times \text{Value2} \Rightarrow \text{value} \]

Returns the product of the two arguments.

\[ \text{List1} \times \text{List2} \Rightarrow \text{list} \]

Returns a list containing the products of the corresponding elements in \( \text{List1} \) and \( \text{List2} \).

Dimensions of the lists must be equal.

\[ \text{Matrix1} \times \text{Matrix2} \Rightarrow \text{matrix} \]

Returns the matrix product of \( \text{Matrix1} \) and \( \text{Matrix2} \).

The number of columns in \( \text{Matrix1} \) must equal the number of rows in \( \text{Matrix2} \).
• (multiply)

Value • List1 ⇒ list

List1 • Value ⇒ list

Returns a list containing the products of Value and each element in List1.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix} \times \begin{bmatrix}
0.01 \\
0.03 \\
\end{bmatrix} = \begin{bmatrix}
0.01 & 0.02 \\
0.03 & 0.04 \\
\end{bmatrix}
\]

\[
6 \cdot \text{id}(3) = \begin{bmatrix}
6 & 0 & 0 \\
0 & 6 & 0 \\
0 & 0 & 6 \\
\end{bmatrix}
\]

Note: Use \( \cdot \) (dot multiply) to multiply an expression by each element.

/(divide)

Value1 / Value2 ⇒ value

Returns the quotient of Value1 divided by Value2.

\[
\frac{2}{3.45} = 0.57971
\]

Note: See also Fraction template, page 1.

List1 / List2 ⇒ list

Returns a list containing the quotients of List1 divided by List2.

\[
\frac{\{1,2,3\}}{\{4,5,6\}} = \left\{ \frac{1.25}{6}, \frac{2}{5}, \frac{1}{2} \right\}
\]

Dimensions of the lists must be equal.

Value / List1 ⇒ list

List1 / Value ⇒ list

Returns a list containing the quotients of Value divided by List1 or List1 divided by Value.

\[
\frac{6}{\{3,6,9\}} = \left\{ \frac{2}{3}, \frac{1.2}{6}, \frac{1}{18} \right\}
\]

\[
\frac{\{7,9,2\}}{7.9.2} = \left\{ \frac{1}{18}, \frac{1.1}{14}, \frac{1}{63} \right\}
\]

Value / Matrix1 ⇒ matrix

Matrix1 / Value ⇒ matrix

Returns a matrix containing the quotients of Matrix1 / Value.

Note: Use ./ (dot divide) to divide an expression by each element.
(power)

Value1 ^ Value2 ⇒ value

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.

Value ^ List1 ⇒ list

Returns Value raised to the power of the elements in List1.

List1 ^ Value ⇒ list

Returns the elements in List1 raised to the power of Value.

squareMatrix1 ^ 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.
### $x^2$ (square)

$Value^2 \Rightarrow value$

Returns the square of the argument.

$List^2 \Rightarrow list$

Returns a list containing the squares of the elements in $List$.

$squareMatrix^2 \Rightarrow matrix$

Returns the matrix square of $squareMatrix$. This is not the same as calculating the square of each element. Use `.^2` to calculate the square of each element.

<table>
<thead>
<tr>
<th>$x^2$ key</th>
<th>$x^2$</th>
<th>$x^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4^2$</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>${2,4,6}^2$</td>
<td>${4,16,36}$</td>
<td></td>
</tr>
</tbody>
</table>

### .+ (dot add)

$Matrix1 .+ Matrix2 \Rightarrow matrix$

$Value .+ Matrix1 \Rightarrow matrix$

$Matrix1 .+ Matrix2$ returns a matrix that is the sum of each pair of corresponding elements in $Matrix1$ and $Matrix2$.

$Value .+ Matrix1$ returns a matrix that is the sum of $Value$ and each element in $Matrix1$.

### .⁻ (dot subt.)

$Matrix1 .⁻ Matrix2 \Rightarrow matrix$

$Value .⁻ Matrix1 \Rightarrow matrix$

$Matrix1 .⁻ Matrix2$ returns a matrix that is the difference between each pair of corresponding elements in $Matrix1$ and $Matrix2$.

$Value .⁻ Matrix1$ returns a matrix that is the difference of $Value$ and each element in $Matrix1$. 
Matrix1 .\* Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40
\end{bmatrix}
= 
\begin{bmatrix}
10 & 40 \\
90 & 160
\end{bmatrix}
\]

Value .\* Matrix1 ⇒ matrix

\[
\begin{bmatrix}
5
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40
\end{bmatrix}
= 
\begin{bmatrix}
50 & 100 \\
150 & 200
\end{bmatrix}
\]

Matrix1 .\* Matrix2 returns a matrix that is the product of each pair of corresponding elements in Matrix1 and Matrix2.  

Value .\* Matrix1 returns a matrix containing the products of Value and each element in Matrix1.

Matrix1 ./ Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 \\
1 & 1
\end{bmatrix}
\]

Value ./ Matrix1 ⇒ matrix

\[
\begin{bmatrix}
5
\end{bmatrix}
\begin{bmatrix}
10 & 20 \\
30 & 40
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 \\
2 & 4
\end{bmatrix}
\]

Matrix1 ./ Matrix2 returns a matrix that is the quotient of each pair of corresponding elements in Matrix1 and Matrix2.  

Value ./ Matrix1 returns a matrix that is the quotient of Value and each element in Matrix1.

Matrix1 .^ Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
0 & 2 \\
3 & -1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 4 \\
27 & \frac{1}{4}
\end{bmatrix}
\]

Value .^ Matrix1 ⇒ matrix

\[
\begin{bmatrix}
5
\end{bmatrix}
\begin{bmatrix}
0 & 2 \\
3 & -1
\end{bmatrix}
= 
\begin{bmatrix}
1 & 25 \\
125 & \frac{1}{5}
\end{bmatrix}
\]

Matrix1 .^ Matrix2 returns a matrix where each element in Matrix2 is the exponent for the corresponding element in Matrix1.  

Value .^ Matrix1 returns a matrix where each element in Matrix1 is the exponent for Value.

\(-\) (negate)

\(-Value1\) ⇒ value

\[-2.43\]

\(-List1\) ⇒ list

\[\{-1,0.4,1.2\times 19\}\]

\(-Matrix1\) ⇒ matrix

\[\{1,-0.4,-1.2\times 19\}\]
- (negate)

Returns the negation of the argument.

For a list or matrix, returns all the elements negated.

If the argument is a binary or hexadecimal integer, the negation gives the two’s complement.

In Bin base mode:

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

```
-0b100101
0b11111111111111111111111111111111
```

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

% (percent)

```
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1% ⇒ value</td>
<td>Returns 13%</td>
</tr>
<tr>
<td>List1% ⇒ list</td>
<td>Returns {1.0, 1.0, 1.0}</td>
</tr>
<tr>
<td>Matrix1% ⇒ matrix</td>
<td>Returns 0.01, 0.1, 1.0</td>
</tr>
</tbody>
</table>
```

% key

Returns 100

For a list or matrix, returns a list or matrix with each element divided by 100.

= (equal)

```
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expr1=Expr2 ⇒ Boolean expression</td>
<td>Returns true if Expr1 is determined to be equal to Expr2.</td>
</tr>
<tr>
<td>List1=List2 ⇒ Boolean list</td>
<td>Returns false if Expr1 is determined to not be equal to Expr2.</td>
</tr>
<tr>
<td>Matrix1=Matrix2 ⇒ Boolean matrix</td>
<td>Anything else returns a simplified form of the equation.</td>
</tr>
</tbody>
</table>
```

= key

For lists and matrices, returns comparisons element by element.

Example function that uses math test symbols: =, ≠, <, ≤, >, ≥

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

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

≠ (not equal)  

Expr1≠Expr2 ⇒ Boolean expression  
List1≠List2 ⇒ Boolean list  
Matrix1≠Matrix2 ⇒ Boolean matrix  

Returns true if Expr1 is determined to be not equal to Expr2.

Returns false if Expr1 is determined to be equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing /=

< (less than)  

Expr1<Expr2 ⇒ Boolean expression  
List1<List2 ⇒ Boolean list  
Matrix1<Matrix2 ⇒ Boolean matrix  

Returns true if Expr1 is determined to be less than Expr2.

Returns false if Expr1 is determined to be greater than or equal to Expr2.
< (less than) 

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≤ (less or equal) 

$Expr1 \leq Expr2 \Rightarrow Boolean\ expression$  

See “=” (equal) example. 

List1 ≤ List2 ⇒ Boolean list

Matrix1 ≤ Matrix2 ⇒ Boolean matrix

Returns true if $Expr1$ is determined to be less than or equal to $Expr2$.

Returns false if $Expr1$ is determined to be greater than $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing <=

> (greater than) 

$Expr1 > Expr2 \Rightarrow Boolean\ expression$  

See “=” (equal) example. 

List1 > List2 ⇒ Boolean list

Matrix1 > Matrix2 ⇒ Boolean matrix

Returns true if $Expr1$ is determined to be greater than $Expr2$.

Returns false if $Expr1$ is determined to be less than or equal to $Expr2$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.
\( \geq \) (greater or equal)

Expressions:
- \( Expr1 \geq Expr2 \Rightarrow \) Boolean expression
- \( List1 \geq List2 \Rightarrow \) Boolean list
- \( Matrix1 \geq Matrix2 \Rightarrow \) Boolean matrix

Returns true if \( Expr1 \) is determined to be greater than or equal to \( Expr2 \).

Returns false if \( Expr1 \) is determined to be less than \( Expr2 \).

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing \( \geq \)

\( \Rightarrow \) (logical implication)

- \( \text{BooleanExpr1} \Rightarrow \text{BooleanExpr2} \) returns Boolean expression
- \( \text{BooleanList1} \Rightarrow \text{BooleanList2} \) returns Boolean list
- \( \text{BooleanMatrix1} \Rightarrow \text{BooleanMatrix2} \) returns Boolean matrix
- \( \text{Integer1} \Rightarrow \text{Integer2} \) returns Integer

Evaluates the expression not \(<\text{argument1}>\) 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 \( \Leftrightarrow \)

\( \\) (factorial)

Value1! \( \Rightarrow \) value

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.

\( \& \) (append)

String1 \& String2 \( \Rightarrow \) string

Returns a text string that is String2 appended to String1.
**d() (derivative)**

\[
d(Expr1, Var[, Order]) | \text{Var=Value } \Rightarrow \text{value}
\]

\[
d(Expr1, Var[, Order]) \Rightarrow \text{value}
\]

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

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

Except when using the first syntax, you must store a numeric value in variable \(\text{Var}\) before evaluating \(d()\). Refer to the examples.

\(d()\) can be used for calculating first and second order derivative at a point numerically, using auto differentiation methods.

\(\text{Order},\) if included, must be \(1\) or \(2\). The default is \(1\).

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

**Note:** See also First derivative, page 5 or Second derivative, page 5.

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

Consider the example on the right. The first derivative of \(x(x^2+x)^{1/3}\) at \(x=0\) is equal to 0. However, because the first derivative of the subexpression \((x^2+x)^{1/3}\) is undefined at \(x=0\), and this value is used to calculate the derivative of the total expression, \(d()\) reports the result as undefined and displays a warning message.

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

| \(x:=0\) | \(\frac{d}{dx}(|x|)\) at \(x=0\) | undefined |
|---|---|---|
| \(x:=3\) | \(\frac{d}{dx}\left\{\left(\frac{1}{x^2}, x^3, x^4\right)\right\}\) | \(\{6, 27, 108\}\) |

| \(\frac{d}{dx}\left(\frac{1}{x(x^2+x)^{1/3}}\right)\) at \(x=0\) | undefined |

| \(\text{centralDiff}\left(\frac{1}{x(x^2+x)^{1/3}}, x\right)\) at \(x=0\) | 0.000033 |
\[ \int (\text{integral}) \]
\[ \int(\text{Expr1, Var, Lower, Upper}) \Rightarrow \text{value} \]

Returns the integral of \( \text{Expr1} \) with respect to the variable \( \text{Var} \) from \( \text{Lower} \) to \( \text{Upper} \). Can be used to calculate the definite integral numerically, using the same method as \( \text{nInt}() \).

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

**Note:** See also \text{nInt}(), page 102, and Definite integral template, page 6.

\[ \sqrt() (\text{square root}) \]
\[ \sqrt(\text{Value1}) \Rightarrow \text{value} \]
\[ \sqrt(\text{List1}) \Rightarrow \text{list} \]

Returns the square root of the argument.

For a list, returns the square roots of all the elements in \( \text{List1} \).

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

**Note:** See also Square root template, page 1.

\[ \Pi() (\text{prodSeq}) \]
\[ \Pi(\text{Expr1, Var, Low, High}) \Rightarrow \text{expression} \]

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

Evaluates \( \text{Expr1} \) for each value of \( \text{Var} \) from \( \text{Low} \) to \( \text{High} \), and returns the product of the results.

**Note:** See also Product template (\( \Pi \)), page 5.

\( \Pi(\text{Expr1, Var, Low, Low−1}) \Rightarrow 1 \)

\( \Pi(\text{Expr1, Var, Low, High}) \Rightarrow 1/\Pi(\text{Expr1, Var, High+1, Low−1}) \) if \( \text{High} < \text{Low−1} \)
The product formulas used are derived from the following reference:


\[ \prod_{k=4}^{6} \left( \frac{1}{k} \right) \]

\[ \sum_{k=4}^{1} \frac{1}{k} \cdot \sum_{k=2}^{4} \frac{1}{k} \]

Σ() (sumSeq)

\[ \sum(Expr1, Var, Low, High) \Rightarrow expression \]

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

Evaluates Expr1 for each value of Var from Low to High, and returns the sum of the results.

Note: See also Sum template, page 5.

\[ \sum(Expr1, Var, Low, Low-1) \Rightarrow 0 \]

\[ \sum(Expr1, Var, Low, High) \Rightarrow \mu \]

\[ \sum(Expr1, Var, High+1, Low-1) \text{ if } High < Low-1 \]

The summation formulas used are derived from the following reference:


\[ \sum_{n-1}^{5} \frac{1}{n} \]

\[ \sum_{k=4}^{3} \frac{1}{k} \]

\[ \sum_{k=4}^{1} \frac{1}{k} \cdot \sum_{k=2}^{4} \frac{1}{k} \]

ΣInt()

\[ \sum\text{Int}(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) \Rightarrow value \]

\[ \sum\text{Int}(NPmt1, NPmt2, amortTable) \Rightarrow value \]

\[ \sum\text{Int}(1,3,12,4,75,20000,,12,12) \Rightarrow -213.48 \]
Amortization function that calculates the sum of the interest during a specified range of payments.

\( NPmt1 \) and \( NPmt2 \) define the start and end boundaries of the payment range.

\( N, I, PV, Pmt, FV, PpY, CpY, \) and \( PmtAt \) are described in the table of TVM arguments, page 162.

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

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

\( \Sigma Int(NPmt1,NPmt2,amortTable) \) calculates the sum of the interest based on amortization table \( amortTable \). The \( amortTable \) argument must be a matrix in the form described under \( amortTbl() \), page 7.

**Note:** See also \( \Sigma Prn() \), below, and \( Bal() \), page 15.

\( \Sigma Prn() \) Amortization function that calculates the sum of the principal during a specified range of payments.

\( NPmt1 \) and \( NPmt2 \) define the start and end boundaries of the payment range.

\( N, I, PV, Pmt, FV, PpY, CpY, \) and \( PmtAt \) are described in the table of TVM arguments, page 162.
• If you omit \( Pmt \), it defaults to 
\[ Pmt = \text{tvmPmt}(N,I,PV,FV,PpY,CpY,PmtAt) \].
• If you omit \( FV \), it defaults to \( FV = 0 \).
• The defaults for \( PpY, CpY, \) and \( PmtAt \) are the same as for the TVM functions.

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

\[ \Sigma \text{Prn}(NPmt_1,NPmt_2,\text{amortTable}) \]
calculates the sum of the principal paid based on amortization table \( \text{amortTable} \). The \( \text{amortTable} \) argument must be a matrix in the form described under \( \text{amortTbl}() \), page 7.

**Note:** See also \( \Sigma \text{Int}() \), above, and \( \text{Bal}() \), page 15.

### # (indirection)

#### \# varNameString

Refers to the variable whose name is \( \text{varNameString} \). This lets you use strings to create variable names from within a function.

| \# \( \text{xyc} := 12 \) | 12 |
| \#(\"x" \& \"y" \& \"z") | 12 |

Creates or refers to the variable \( \text{xyc} \).

| \( 10 \rightarrow r \) | 10 |
| \( \text{"r"} \rightarrow \text{s1} \) | \( \text{"r"} \) |
| \#\( \text{s1} \) | 10 |

Returns the value of the variable \( r \) whose name is stored in variable \( s1 \).

### E (scientific notation)

#### mantissa\( \times 10^{\text{exponent}} \)

Enters a number in scientific notation. The number is interpreted as \( \text{mantissa} \times 10^{\text{exponent}} \).

Hint: If you want to enter a power of 10 without causing a decimal value result, use \( 10^{\text{integer}} \).
### Symbols

#### E (scientific notation)

**Note:** You can insert this operator from the computer keyboard by typing @E. For example, type 2 . 3@E4 to enter 2.3E4.

#### g (gradian)

**Expr1g** ⇒ expression  
**List1g** ⇒ list  
**Matrix1g** ⇒ matrix

In Degree, Gradian or Radian mode:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cos(50g) )</td>
<td>0.707107</td>
</tr>
<tr>
<td>( \cos({0,100g,200g}) )</td>
<td>{1,0,-1}</td>
</tr>
</tbody>
</table>

This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies **Expr1** by \( \pi/200 \).

In Degree angle mode, multiplies **Expr1** by \( g/100 \).

In Gradian mode, returns **Expr1** unchanged.

**Note:** You can insert this symbol from the computer keyboard by typing @g.

#### r (radian)

**Value1r** ⇒ value  
**List1r** ⇒ list  
**Matrix1r** ⇒ matrix

In Degree, Gradian or Radian angle mode:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cos(\frac{\pi}{4r}) )</td>
<td>0.707107</td>
</tr>
<tr>
<td>( \cos(\left{0,\frac{\pi}{12},-(\pi)\right}) )</td>
<td>{1,0.965926,-1}</td>
</tr>
</tbody>
</table>

This function gives you a way to specify a radian angle while in Degree or Gradian mode.

In Degree angle mode, multiplies the argument by 180/\( \pi \).

In Radian angle mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by 200/\( \pi \).
Hint: Use \( r \) if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

**Note:** You can insert this symbol from the computer keyboard by typing @r.

\[ r \] (radian)

\( \pi \) key

\( \circ \) (degree)

\( \pi \) key

**Value**\( \circ \) ⇒ \textit{value}

**List**\( \circ \) ⇒ \textit{list}

**Matrix**\( \circ \) ⇒ \textit{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.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cos(45^\circ) )</td>
<td>0.707107</td>
</tr>
<tr>
<td>( \cos\left{0, \frac{\pi}{4}, 90^\circ, 30.12^\circ\right} )</td>
<td>{1, 0.707107, 0, 0.864976}</td>
</tr>
</tbody>
</table>

\( \circ, ' , " \) (degree/minute/second)

\( \text{ctrl} \) \( \text{<right shift>} \) \( \text{keys} \)

\( dd' mm' ss'' \Rightarrow \textit{expression} \)

\( dd \) 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 (").

Symbols 191
\( \angle \) (angle)

\[ \text{[Radius}, \angle \theta_{-} \text{Angle}] \Rightarrow \text{vector} \]
(polar input)

\[ \text{[Radius}, \angle \theta_{-} \text{Angle}, Z_{-} \text{Coordinate}] \Rightarrow \text{vector} \]
(cylindrical input)

\[ \text{[Radius}, \angle \theta_{-} \text{Angle}, \angle \theta_{-} \text{Angle}] \Rightarrow \text{vector} \]
(spherical input)

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

Note: You can insert this symbol from the computer keyboard by typing @<.

\((\text{Magnitude} \angle \text{Angle}) \Rightarrow \text{complexValue}\)
(polar input)

Enters a complex value in \((r \angle \theta)\) polar form. The Angle is interpreted according to the current Angle mode setting.

\_ (underscore as an empty element)

See “Empty (Void) Elements,” page 212.

10^()

10^ (Value1) \Rightarrow \text{value}

10^ (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.
$10^()$

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.

$^\text{−1} \text{(reciprocal)}$

Returns the reciprocal of the argument. For a list, returns the reciprocals of the elements in `List1`.

`squareMatrix1^\text{−1} ⇒ squareMatrix`

Returns the inverse of `squareMatrix1`.

`squareMatrix1` must be a non-singular square matrix.

$\mid \text{(constraint operator)}$

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 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.

If the variable Var does not exist, creates it and initializes it to Value, List, or Matrix.

If the variable Var already exists and is not locked or protected, replaces its contents with Value, List, or Matrix.
Note: You can insert this operator from the keyboard by typing := as a shortcut. For example, type pi/4 := myvar.

:= (assign)  

Var := Value

Var := List

Var := Matrix

Function(Param1,...) := Expr

Function(Param1,...) := List

Function(Param1,...) := Matrix

If variable Var does not exist, creates Var and initializes it to Value, List, or Matrix.

If Var already exists and is not locked or protected, replaces its contents with Value, List, or Matrix.

© (comment)  

© [text]

© processes text as a comment line, allowing you to annotate functions and programs that you create.

© can be at the beginning or anywhere in the line. Everything to the right of ©, to the end of the line, is the comment.

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

Ob, Oh  

Ob binaryNumber

Oh hexadecimalNumber

In Dec base mode:
196  Symbols

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

<table>
<thead>
<tr>
<th>0b10+0hF+10</th>
<th>27</th>
</tr>
</thead>
</table>

In Bin base mode:

| 0b10+0hF+10 | 0b11011 |

In Hex base mode:

| 0b10+0hF+10 | 0h1B   |
TI-Nspire™ CX II - Draw Commands

This is a supplemental document for the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide. All TI-Nspire™ CX II commands will be incorporated and published in version 5.1 of the TI-Nspire™ Reference Guide and the TI-Nspire™ CAS Reference Guide.

Graphics Programming

New commands have been added on TI-Nspire™ CX II Handhelds and TI-Nspire™ desktop applications for graphics programming.

The TI-Nspire™ CX II Handhelds will switch into this graphics mode while executing graphics commands and switch back to the context in which the program was executed after completion of the program.

The screen will display “Running...” in the top bar while the program is being executed. It will show “Finished” when the program completes. Any key-press will transition the system out of the graphics mode.

- The transition to graphics mode is triggered automatically when one of the Draw (graphics) commands is encountered during execution of the TI Basic program.
- This transition will only happen when executing a program from calculator; in a document or calculator in scratchpad.
- The transition out of graphics mode happens upon termination of the program.
- The graphics mode is only available on the TI-Nspire™ CX II Handhelds and the desktop TI-Nspire™ CX II Handhelds view. This means it is not available in the computer document view or PublishView (.tnsp) on the desktop nor on iOS.
  - If a graphics command is encountered while executing a TI Basic program from the incorrect context, an error message is displayed and the TI Basic program is terminated.

Graphics Screen

The graphics screen will contain a header at the top of the screen that cannot be written to by graphics commands.

The graphics screen drawing area will be cleared (color = 255,255,255) when the graphics screen is initialized.

<table>
<thead>
<tr>
<th>Graphics Screen</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>212</td>
</tr>
<tr>
<td>Width</td>
<td>318</td>
</tr>
<tr>
<td>Color</td>
<td>white: 255,255,255</td>
</tr>
</tbody>
</table>
**Default View and Settings**

- The status icons in the top bar (battery status, press-to-test status, network indicator etc.) will not be visible while a graphics program is running.
- Default drawing color: Black (0,0,0)
- Default pen style - normal, smooth
  - Thickness: 1 (thin), 2 (normal), 3 (thickest)
  - Style: 1 (smooth), 2 (dotted), 3 (dashed)
- All drawing commands will use the current color and pen settings; either default values or those which were set via TI-Basic commands.
- Text font is fixed and cannot be changed.
- Any output to the graphics screen will be drawn within a clipping window which is the size of the graphics screen drawing area. Any drawn output that extends outside of this clipped graphics screen drawing area will not be drawn. No error message will be displayed.
- All x,y coordinates specified for drawing commands are defined such that 0,0 is at the top left corner of the graphics screen drawing area.
  - **Exceptions:**
    - **DrawText** uses the coordinates as the bottom left corner of the bounding box for the text.
    - **SetWindow** uses the bottom left corner of the screen
- All parameters for the commands can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.
**Graphics Screen Errors Messages**

If the validation fails, an error message will display.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Description</th>
<th>View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Syntax</td>
<td>If the syntax checker finds any syntax errors, it displays an error message and tries to position the cursor near the first error so you can correct it.</td>
<td><img src="image" alt="Error Syntax" /></td>
</tr>
<tr>
<td>Error Too few arguments</td>
<td>The function or command is missing one or more arguments</td>
<td><img src="image" alt="Error Too few arguments" /></td>
</tr>
<tr>
<td>Error Too many arguments</td>
<td>The function or command contains and excessive number of arguments and cannot be evaluated.</td>
<td><img src="image" alt="Error Too many arguments" /></td>
</tr>
<tr>
<td>Error Invalid data type</td>
<td>An argument is of the wrong data type.</td>
<td><img src="image" alt="Error Invalid data type" /></td>
</tr>
</tbody>
</table>

**Invalid Commands While in Graphics Mode**

Some commands are not allowed once the program switches to graphics mode. If these commands are encountered while in graphics mode and error will be displayed and the program will be terminated.

<table>
<thead>
<tr>
<th>Disallowed Command</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Request cannot be executed in graphics mode</td>
</tr>
<tr>
<td>RequestStr</td>
<td>RequestStr cannot be executed in graphics mode</td>
</tr>
<tr>
<td>Text</td>
<td>Text cannot be executed in graphics mode</td>
</tr>
</tbody>
</table>

The commands that print text to the calculator - **disp** and **dispAt** - will be supported commands in the graphics context. The text from these commands will be sent to the Calculator screen (not on Graphics) and will be visible after the program exits and the system switches back to the Calculator app.
**Clear**

**Clear** \( x, y, \) width, height

Clears entire screen if no parameters are specified.

If \( x, y, \) width and height are specified, the rectangle defined by the parameters will be cleared.

Example:

Clear 10,10,100,50

Clears a rectangle area with top left corner on (10, 10) and with width 100, height 50
**DrawArc**

**DrawArc** \(x, y, \text{width}, \text{height}, \text{startAngle}, \text{arcAngle}\)

Draw an arc within the defined bounding rectangle with the provided start and arc angles.

\(x, y\): upper left coordinate of bounding rectangle

\(\text{width}, \text{height}\): dimensions of bounding rectangle

The "arc angle" defines the sweep of the arc.

These parameters can be provided as expressions that evaluate to a number which is then rounded to the nearest integer.

**See Also:** [FillArc](#)

---

**DrawCircle**

**DrawCircle** \(x, y, \text{radius}\)

\(x, y\): coordinate of center

\(\text{radius}\): radius of the circle

**See Also:** [FillCircle](#)
**DrawLine**

**DrawLine** \( x_1, y_1, x_2, y_2 \)

Draw a line from \( x_1, y_1, x_2, y_2 \).

Expressions that evaluate to a number which is then rounded to the nearest integer.

**Screen bounds:** If the specified coordinates causes any part of the line to be drawn outside of the graphics screen, that part of the line will be clipped and no error message will be displayed.

**DrawPoly**

The commands have two variants:

**DrawPoly** \( xlist, ylist \)

or

**DrawPoly** \( x_1, y_1, x_2, y_2, x_3, y_3...x_n, y_n \)

**Note:** **DrawPoly** \( xlist, ylist \)
Shape will connect \( x_1, y_1 \) to \( x_2, y_2, x_2, y_2 \) to \( x_3, y_3 \) and so on.

**Note:** **DrawPoly** \( x_1, y_1, x_2, y_2, x_3, y_3...x_n, y_n \)
\( x_n, y_n \) will **NOT** be automatically connected to \( x_1, y_1 \).

Expressions that evaluate to a list of real floats
\( xlist, ylist \)

Expressions that evaluate to a single real float
\( x_1, y_1...x_n, y_n = \) coordinates for vertices of polygon

---

202  TI-Nspire™ CX II - Draw Commands
**DrawPoly**

**Note:** **DrawPoly**: Input size dimensions (width/height) relative to drawn lines. The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn polygon will be larger than the width and height.

See Also: **FillPoly**

**DrawRect**

**DrawRect** *x, y, width, height*

*x, y*: upper left coordinate of rectangle  
*width, height*: width and height of rectangle (rectangle drawn down and right from starting coordinate).

**Note:** The lines are drawn in a bounding box around the specified coordinate and dimensions such that the actual size of the drawn rectangle will be larger than the width and height indicate.

See Also: **FillRect**

**DrawText**

**DrawText** *x, y, exprOrString1 [,exprOrString2]...*

*x, y*: coordinate of text output  

Draws the text in *exprOrString* at the specified *x, y* coordinate location.

The rules for *exprOrString* are the same as for **Disp** – **DrawText** can take multiple arguments.
**FillArc**

FillArc $x, y, width, height, startAngle, arcAngle$

$x, y$: upper left coordinate of bounding rectangle

Draw and fill an arc within the defined bounding rectangle with the provided start and arc angles.

Default fill color is black. The fill color can be set by the `SetColor` command.

The "arc angle" defines the sweep of the arc.

**FillCircle**

FillCircle $x, y, radius$

$x, y$: coordinate of center

Draw and fill a circle at the specified center with the specified radius.

Default fill color is black. The fill color can be set by the `SetColor` command.

**FillPoly**

FillPoly $xlist, ylist$

or

FillPoly $x1, y1, x2, y2, x3, y3...xn, yn$

Note: The line and color are specified by `SetColor` and `SetPen`
**FillPoly**

```
FillPoly 0,10,200,20,150,150,0,10
```

**FillRect**

```
FillRect 25,25,100,50
```

**FillPoly**

**Catalog > CXII**

**FillRect**

**Catalog > CXII**

**FillRect** \( x, y, \text{width}, \text{height} \)

\( x, y \): upper left coordinate of rectangle
\( \text{width}, \text{height} \): width and height of rectangle

Draw and fill a rectangle with the top left corner at the coordinate specified by \((x,y)\)

Default fill color is black. The fill color can be set by the **SetColor** command

**Note:** The line and color are specified by **SetColor** and **SetPen**
getPlatform()

getPlatform()

Returns:
“dt” on desktop software applications
“hh” on TI-Nspire™ CX handhelds
“ios” on TI-Nspire™ CX iPad® app
PaintBuffer

Paint graphics buffer to screen

This command is used in conjunction with UseBuffer to increase the speed of display on the screen when the program generates multiple graphical objects.

See Also: UseBuffer

UseBuffer
For n,1,10
    x:=randInt(0,300)
    y:=randInt(0,200)
    radius:=randInt(10,50)
    Wait 0.5
    DrawCircle x,y,radius
EndFor
PaintBuffer

This program will display all the 10 circles at once.

If the “UseBuffer” command is removed, each circle will be displayed as it is drawn.
**PlotXY**

plotxy \( x, \ y, \ shape \)

\( x, \ y \): coordinate to plot shape

\( shape \): a number between 1 and 13 specifying the shape

1 - Filled circle
2 - Empty circle
3 - Filled square
4 - Empty square
5 - Cross
6 - Plus
7 - Thin
8 - medium point, solid
9 - medium point, empty
10 - larger point, solid
11 - larger point, empty
12 - largest point, solid
13 - largest point, empty

Example:

\[
\text{PlotXY} \ 100,100,1 \\
\text{PlotXY} \ 100,100,2 \\
\text{PlotXY} \ 100,100,3 \\
\text{PlotXY} \ 100,100,4 \\
\text{PlotXY} \ 100,100,5 \\
\text{PlotXY} \ 100,100,6 \\
\text{PlotXY} \ 100,100,7 \\
\text{PlotXY} \ 100,100,8 \\
\text{PlotXY} \ 100,100,9 \\
\text{PlotXY} \ 100,100,10 \\
\text{PlotXY} \ 100,100,11 \\
\text{PlotXY} \ 100,100,12 \\
\text{PlotXY} \ 100,100,13 \\
\]

For \( n, 1, 13 \)

\[
\text{DrawText} \ \text{1+22*n,40,n} \\
\text{PlotXY} \ \text{5+22*n,50,n} \\
\text{EndFor}
\]
**SetColor**

**SetColor**

Red-value, Green-value, Blue-value

Valid values for red, green and blue are between 0 and 255

Sets the color for subsequent Draw commands

```
SetColor 255,0,0
DrawCircle 150,150,100
```

**SetPen**

**SetPen**

thickness, style

thickness: $1 \leq \text{thickness} \leq 3$ | 1 is thinnest, 3 is thickest

style: 1 = Smooth, 2 = Dotted, 3 = Dashed

Sets the pen style for subsequent Draw commands

```
SetPen 3,3
DrawCircle 150,150,50
```

**SetWindow**

**SetWindow**

xMin, xMax, yMin, yMax

Establishes a logical window that maps to the graphics drawing area. All parameters are required.

If the part of drawn object is outside the window, the output will be clipped (not shown) and no error message is displayed.

```
SetWindow 0,160,0,120
DrawLine 0,0,100,100
SetWindow 0,160,0,120
SetPen 3,3
DrawLine 0,0,100,100
```
If xmin is greater than or equal to xmax or ymin is greater than or equal to ymax, an error message is shown.

Any objects drawn before a SetWindow command will not be re-drawn in the new configuration.

To reset the window parameters to the default, use:

SetWindow 0,0,0,0
**UseBuffer**

Draw to an off screen graphics buffer instead of screen (to increase performance)

This command is used in conjunction with PaintBuffer to increase the speed of display on the screen when the program generates multiple graphical objects.

With UseBuffer, all the graphics are displayed only after the next PaintBuffer command is executed.

UseBuffer only needs to be called once in the program i.e. every use of PaintBuffer does not need a corresponding UseBuffer

**See Also:** PaintBuffer
Empty (Void) Elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ Software allows empty, or void, data elements so you can proceed with the nearly complete data rather than having to start over or discard the incomplete cases.

You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “Graphing spreadsheet data.”

The delVoid() function lets you remove empty elements from a list. The isVoid() function lets you test for an empty element. For details, see delVoid(), page 39, and isVoid(), page 75.

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 \[\text{ctrl} \ _\].

Calculations involving void elements

The majority of calculations involving a void input will produce a void result. See special cases below.

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[_]</td>
<td>_</td>
</tr>
<tr>
<td>gcd(100,_)</td>
<td>_</td>
</tr>
<tr>
<td>[3+_]</td>
<td>_</td>
</tr>
<tr>
<td>[{5,_}..10} - {3,6,9}</td>
<td>{2,_}</td>
</tr>
</tbody>
</table>

List arguments containing void elements

The following functions and commands ignore (skip) void elements found in list arguments.

- count, countIf, cumulativeSum, freqTable►list, frequency, max, mean, median, product, stDevPop, stDevSamp, sum, sumIf, varPop, and varSamp, as well as regression calculations, OneVar, TwoVar, and FiveNumSummary statistics, confidence intervals, and stat tests.

<table>
<thead>
<tr>
<th>Function</th>
<th>Argument</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>{2,_3,5,6,6}</td>
<td>16.6</td>
</tr>
<tr>
<td>median</td>
<td>{1,2,_}..3}</td>
<td>2</td>
</tr>
<tr>
<td>cumulativeSum</td>
<td>{1,2,_,4,5}</td>
<td>{1,3,_,7,12}</td>
</tr>
<tr>
<td>cumulativeSum</td>
<td>[1 \ 2 \ _]</td>
<td>[1 \ 2 \ _]</td>
</tr>
<tr>
<td></td>
<td>[3 \ _]</td>
<td>[4 \ _]</td>
</tr>
<tr>
<td></td>
<td>[5 \ 6]</td>
<td>[9 \ 8]</td>
</tr>
</tbody>
</table>

SortA and SortD move all void elements within the first argument to the bottom.

<table>
<thead>
<tr>
<th>List1</th>
<th>List2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{5,4,3,_,1}</td>
<td>{5,4,3,_,1}</td>
</tr>
<tr>
<td>{5,4,3,2,1}</td>
<td>{5,4,3,2,1}</td>
</tr>
<tr>
<td>SortA list1,list2</td>
<td>Done</td>
</tr>
<tr>
<td>list1</td>
<td>{1,3,4,5,_}</td>
</tr>
<tr>
<td>list2</td>
<td>{1,3,4,5,2}</td>
</tr>
</tbody>
</table>
List arguments containing void elements

In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

An omitted category in regressions introduces a void for the corresponding element of the residual.

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.
Shortcuts for Entering Math Expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type `sqrt(6)` on the entry line. When you press [enter], the expression `sqrt(6)` is changed to $\sqrt{6}$. Some shortcuts are useful from both the handheld and the computer keyboard. Others are useful primarily from the computer keyboard.

From the Handheld or Computer Keyboard

<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>$</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{()}$</td>
<td>sqrt(...)</td>
</tr>
<tr>
<td>$\Sigma()$ (Sum template)</td>
<td>sumSeq(...)</td>
</tr>
<tr>
<td>$\Pi()$ (Product template)</td>
<td>prodSeq(...)</td>
</tr>
<tr>
<td>$\sin^{-1}()$, $\cos^{-1}()$, ...</td>
<td>arcsin(...), arccos(...), ...</td>
</tr>
<tr>
<td>$\Delta List()$</td>
<td>deltaList(...)</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>$i$ (imaginary constant)</td>
<td>@i</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>To enter this:</td>
<td>Type this shortcut:</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</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>(@\angle)</td>
</tr>
<tr>
<td>► (conversion)</td>
<td>(@)</td>
</tr>
<tr>
<td>►Decimal, ►approxFraction(), and so on.</td>
<td>►Decimal, ►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™ 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 (°,'&quot;,), factorial (!), percentage (%), radian (r), subscript ([ ]), transpose (↑)</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 not</td>
</tr>
<tr>
<td>12</td>
<td>Logical and</td>
</tr>
<tr>
<td>13</td>
<td>Logical or</td>
</tr>
<tr>
<td>14</td>
<td>xor, nor, nand</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 (“</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™ 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 \([-]\) followed by the number. Post operations and exponentiation are performed before negation. For example, the result of \(-x^2\) is a negative number, and \(-9^2 = -81\). Use parentheses to square a negative number such as \((-9)^2\) to produce 81.

**Constraint (“|”)**

The argument following the constraint (“|”) operator provides a set of constraints that affect the evaluation of the argument preceding the operator.
TI-Nspire CX II - TI-Basic Programming Features

Auto-indentation in Programming Editor

The TI-Nspire™ program editor now auto-indents statements inside a block command. Block commands are If/EndIf, For/EndFor, While/EndWhile, Loop/EndLoop, Try/EndTry

The editor will automatically prepend spaces to program commands inside a block command. The closing command of the block will be aligned with the opening command.

The example below shows auto-indentation in nested block commands.

Code fragments that are copied and pasted will retain the original indentation.

Opening a program created in an earlier version of the software will retain the original indentation.

Improved Error Messages for TI-Basic

Errors

<table>
<thead>
<tr>
<th>Error Condition</th>
<th>New message</th>
</tr>
</thead>
</table>
| Error in condition statement (If/While) | A conditional statement did not resolve to **TRUE** or **FALSE**  
**NOTE:** With the change to place the cursor on the line with the error, we no longer need to specify if the error is in an "If" statement or a "While" statement. |
| Missing EndIf                  | Expected **EndIf** but found a different end statement                       |
| Missing EndFor                  | Expected **EndFor** but found a different end statement                      |
| Missing EndWhile                | Expected **EndWhile** but found a different end statement                    |
| Missing EndLoop                 | Expected **EndLoop** but found a different end statement                     |
Error Condition | New message
--- | ---
Missing EndTry | Expected EndTry but found a different end statement
“Then” omitted after If <condition> | Missing If..Then
“Then” omitted after ElseIf <condition> | Then missing in block: ElseIf.
When “Then”, “Else” and “ElseIf” were encountered outside of control blocks | ElseIf invalid outside of block: If..Then..EndIf or Try..EndTry
“ElseIf” appears outside of “If..Then..EndIf” block | ElseIf invalid outside of block: If..Then..EndIf
“Then” appears outside of “If…EndIf” block | Then invalid outside of block: If..EndIf

**Syntax Errors**

In case commands that expect one or more arguments are called with an incomplete list of arguments, a “Too few argument error” will be issued instead of “syntax” error.
<table>
<thead>
<tr>
<th>Current behavior</th>
<th>New CX II behavior</th>
</tr>
</thead>
</table>

![Current behavior](image1)

![New CX II behavior](image2)

**Note:** When an incomplete list of arguments is not followed by a comma, the error message is: “too few arguments”. This is the same as previous releases.
The following table lists the constants and their values that are available when performing unit conversions. They can be typed in manually or selected from the Constants list in Utilities > Unit Conversions (Handheld: Press \( \text{Handheld: Press } 3 \)).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( _c )</td>
<td>Speed of light</td>
<td>299792458 ( \text{m/s} )</td>
</tr>
<tr>
<td>( _Cc )</td>
<td>Coulomb constant</td>
<td>8987551787.3682 ( \text{m/F} )</td>
</tr>
<tr>
<td>( _Fc )</td>
<td>Faraday constant</td>
<td>96485.33289 ( \text{coul/mol} )</td>
</tr>
<tr>
<td>( _g )</td>
<td>Acceleration of gravity</td>
<td>9.80665 ( \text{m/s}^2 )</td>
</tr>
<tr>
<td>( _Gc )</td>
<td>Gravitational constant</td>
<td>6.67408 ( 10^{-11} ) ( \text{m}^3/\text{kg/s}^2 )</td>
</tr>
<tr>
<td>( _h )</td>
<td>Planck's constant</td>
<td>6.626070040 ( 10^{-34} ) ( \text{j/s} )</td>
</tr>
<tr>
<td>( _k )</td>
<td>Boltzmann's constant</td>
<td>1.38064852 ( 10^{-23} ) ( \text{J/°K} )</td>
</tr>
<tr>
<td>( _\mu 0 )</td>
<td>Permeability of a vacuum</td>
<td>1.2566370614359 ( 10^{-6} ) ( \text{N/A} )</td>
</tr>
<tr>
<td>( _\mu b )</td>
<td>Bohr magneton</td>
<td>9.274009994 ( 10^{-24} ) ( \text{J/m}^2 )</td>
</tr>
<tr>
<td>( _Me )</td>
<td>Electron rest mass</td>
<td>9.10938356 ( 10^{-31} ) ( \text{kg} )</td>
</tr>
<tr>
<td>( _M\mu )</td>
<td>Muon mass</td>
<td>1.883531594 ( 10^{-28} ) ( \text{kg} )</td>
</tr>
<tr>
<td>( _Mn )</td>
<td>Neutron rest mass</td>
<td>1.674927471 ( 10^{-27} ) ( \text{kg} )</td>
</tr>
<tr>
<td>( _Mp )</td>
<td>Proton rest mass</td>
<td>1.67261898 ( 10^{-27} ) ( \text{kg} )</td>
</tr>
<tr>
<td>( _Na )</td>
<td>Avogadro's number</td>
<td>6.0221408 ( 10^{23} ) ( \text{mol} )</td>
</tr>
<tr>
<td>( _q )</td>
<td>Electron charge</td>
<td>1.6021766 ( 10^{-19} ) ( \text{coul} )</td>
</tr>
<tr>
<td>( _Rb )</td>
<td>Bohr radius</td>
<td>5.291772106 ( 10^{-11} ) ( \text{m} )</td>
</tr>
<tr>
<td>( _Rc )</td>
<td>Molar gas constant</td>
<td>8.3144598 ( \text{J/mol/°K} )</td>
</tr>
<tr>
<td>( _Rdb )</td>
<td>Rydberg constant</td>
<td>10973731.58508 ( \text{m} )</td>
</tr>
<tr>
<td>( _Re )</td>
<td>Electron radius</td>
<td>2.81794032 ( 10^{-15} ) ( \text{m} )</td>
</tr>
<tr>
<td>( _u )</td>
<td>Atomic mass</td>
<td>1.66053904 ( 10^{-27} ) ( \text{kg} )</td>
</tr>
<tr>
<td>( _Vm )</td>
<td>Molar volume</td>
<td>2.2413962 ( 10^{-2} ) ( \text{m}^3/\text{mol} )</td>
</tr>
<tr>
<td>( _\varepsilon 0 )</td>
<td>Permittivity of a vacuum</td>
<td>8.85418781 ( 10^{-12} ) ( \text{F/m} )</td>
</tr>
<tr>
<td>( _\sigma )</td>
<td>Stefan-Boltzmann constant</td>
<td>5.670367 ( 10^{-8} ) ( \text{W/m}^2/\text{°K}^4 )</td>
</tr>
<tr>
<td>( _\phi 0 )</td>
<td>Magnetic flux quantum</td>
<td>2.06783383 ( 10^{-15} ) ( \text{Wb} )</td>
</tr>
</tbody>
</table>
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 158.

**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>
</tbody>
</table>
| 140        | Argument must be a variable name. Make sure that the name:  
|            | • does not begin with a digit  
|            | • does not contain spaces or special characters  
|            | • does not use underscore or period in invalid manner  
|            | • does not exceed the length limitations  
|            | See the Calculator section in the documentation for more details. |
| 160        | Argument must be an expression |
| 165        | Batteries too low for sending or receiving  
<p>|            | Install new batteries before sending or receiving. |
| 170        | Bound The lower bound must be less than the upper bound to define the search interval. |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>Break</td>
</tr>
<tr>
<td>The esc or on key was pressed during a long calculation or during program execution.</td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>Circular definition</td>
</tr>
<tr>
<td>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>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Constraint expression invalid</td>
</tr>
<tr>
<td>For example, solve(3x^2-4=0,x)</td>
<td>x&lt;0 or x&gt;5 would produce this error message because the constraint is separated by “or” instead of “and.”</td>
</tr>
<tr>
<td>210</td>
<td>Invalid Data type</td>
</tr>
<tr>
<td>An argument is of the wrong data type.</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>Dependent limit</td>
</tr>
<tr>
<td>230</td>
<td>Dimension</td>
</tr>
<tr>
<td>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>
<td></td>
</tr>
<tr>
<td>235</td>
<td>Dimension Error. Not enough elements in the lists.</td>
</tr>
<tr>
<td>240</td>
<td>Dimension mismatch</td>
</tr>
<tr>
<td>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>
<td></td>
</tr>
<tr>
<td>250</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>260</td>
<td>Domain error</td>
</tr>
<tr>
<td>An argument must be in a specified domain. For example, rand(0) is not valid.</td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>Duplicate variable name</td>
</tr>
<tr>
<td>280</td>
<td>Else and Elself 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 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</td>
</tr>
<tr>
<td>For example, solve(3x^2-4,x) is invalid because the first argument is not an equation.</td>
<td></td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</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>
</tbody>
</table>
| 380        | Undefined Ans  
Either the previous calculation did not create Ans, or no previous calculation was entered. |
| 390        | Invalid assignment |
| 400        | Invalid assignment value |
| 410        | Invalid command |
| 430        | Invalid for the current mode settings |
| 435        | Invalid guess |
| 440        | Invalid implied multiply  
For example, x(x+1) is invalid; whereas, x*(x+1) is the correct syntax. This is to avoid confusion between implied multiplication and function calls. |
| 450        | Invalid in a function or current expression  
Only certain commands are valid in a user-defined function. |
| 490        | Invalid in Try..EndTry block |
| 510        | Invalid list or matrix |
| 550        | Invalid outside function or program  
A number of commands are not valid outside a function or program. For example, Local cannot be used unless it is in a function or program. |
| 560        | Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks  
For example, the Exit command is valid only inside these loop blocks. |
| 565        | Invalid outside program |
| 570        | Invalid pathname  
For example, \var is invalid. |
| 575        | Invalid polar complex |
| 580        | Invalid program reference  
Programs cannot be referenced within functions or expressions such as 1+p(x) where p is a program. |
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>Invalid table</td>
</tr>
<tr>
<td>605</td>
<td>Invalid use of units</td>
</tr>
<tr>
<td>610</td>
<td>Invalid variable name in a Local statement</td>
</tr>
<tr>
<td>620</td>
<td>Invalid variable or function name</td>
</tr>
<tr>
<td>630</td>
<td>Invalid variable reference</td>
</tr>
<tr>
<td>640</td>
<td>Invalid vector syntax</td>
</tr>
</tbody>
</table>
| 650        | Link transmission  
A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends. |
| 665        | Matrix not diagonalizable |
| 670        | Low Memory  
1. Delete some data in this document  
2. Save and close this document  
If 1 and 2 fail, pull out and re-insert batteries |
| 672        | Resource exhaustion |
| 673        | Resource exhaustion |
| 680        | Missing ( |
| 690        | Missing ) |
| 700        | Missing “ |
| 710        | Missing ] |
| 720        | Missing } |
| 730        | Missing start or end of block syntax |
| 740        | Missing Then in the If..EndIf block |
| 750        | Name is not a function or program |
| 765        | No functions selected |
| 780        | No solution found |
| 800        | Non-real result  
For example, if the software is in the Real setting, √(-1) is invalid. |
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>226</td>
<td>Error Codes and Messages</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>830</td>
<td>Overflow</td>
</tr>
<tr>
<td>850</td>
<td>Program not found</td>
</tr>
<tr>
<td></td>
<td>A program reference inside another program could not be found in the provided path during execution.</td>
</tr>
<tr>
<td>855</td>
<td>Rand type functions not allowed in graphing</td>
</tr>
<tr>
<td>860</td>
<td>Recursion too deep</td>
</tr>
<tr>
<td>870</td>
<td>Reserved name or system variable</td>
</tr>
<tr>
<td>900</td>
<td>Argument error</td>
</tr>
<tr>
<td></td>
<td>Median-median model could not be applied to data set.</td>
</tr>
<tr>
<td>910</td>
<td>Syntax error</td>
</tr>
<tr>
<td>920</td>
<td>Text not found</td>
</tr>
<tr>
<td>930</td>
<td>Too few arguments</td>
</tr>
<tr>
<td></td>
<td>The function or command is missing one or more arguments.</td>
</tr>
<tr>
<td>940</td>
<td>Too many arguments</td>
</tr>
<tr>
<td></td>
<td>The expression or equation contains an excessive number of arguments and cannot be evaluated.</td>
</tr>
<tr>
<td>950</td>
<td>Too many subscripts</td>
</tr>
<tr>
<td>955</td>
<td>Too many undefined variables</td>
</tr>
<tr>
<td>960</td>
<td>Variable is not defined</td>
</tr>
<tr>
<td></td>
<td>No value is assigned to variable. Use one of the following commands:</td>
</tr>
<tr>
<td></td>
<td>• sto →</td>
</tr>
<tr>
<td></td>
<td>• :=</td>
</tr>
<tr>
<td></td>
<td>• Define</td>
</tr>
<tr>
<td></td>
<td>to assign values to variables.</td>
</tr>
<tr>
<td>965</td>
<td>Unlicensed OS</td>
</tr>
<tr>
<td>970</td>
<td>Variable in use so references or changes are not allowed</td>
</tr>
<tr>
<td>980</td>
<td>Variable is protected</td>
</tr>
<tr>
<td>990</td>
<td>Invalid variable name</td>
</tr>
<tr>
<td></td>
<td>Make sure that the name does not exceed the length limitations</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1000</td>
<td>Window variables domain</td>
</tr>
<tr>
<td>1010</td>
<td>Zoom</td>
</tr>
<tr>
<td>1020</td>
<td>Internal error</td>
</tr>
<tr>
<td>1030</td>
<td>Protected memory violation</td>
</tr>
<tr>
<td>1040</td>
<td>Unsupported function. This function requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1045</td>
<td>Unsupported operator. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1050</td>
<td>Unsupported feature. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1060</td>
<td>Input argument must be numeric. Only inputs containing numeric values are allowed.</td>
</tr>
<tr>
<td>1070</td>
<td>Trig function argument too big for accurate reduction</td>
</tr>
<tr>
<td>1080</td>
<td>Unsupported use of Ans. This application does not support Ans.</td>
</tr>
<tr>
<td>1090</td>
<td>Function is not defined. Use one of the following commands:</td>
</tr>
<tr>
<td></td>
<td>• Define</td>
</tr>
<tr>
<td></td>
<td>• :=</td>
</tr>
<tr>
<td></td>
<td>• sto →</td>
</tr>
<tr>
<td></td>
<td>to define a function.</td>
</tr>
<tr>
<td>1100</td>
<td>Non-real calculation</td>
</tr>
<tr>
<td></td>
<td>For example, if the software is in the Real setting, $\sqrt{-1}$ is invalid.</td>
</tr>
<tr>
<td></td>
<td>To allow complex results, change the “Real or Complex” Mode Setting to RECTANGULAR or POLAR.</td>
</tr>
<tr>
<td>1110</td>
<td>Invalid bounds</td>
</tr>
<tr>
<td>1120</td>
<td>No sign change</td>
</tr>
<tr>
<td>1130</td>
<td>Argument cannot be a list or matrix</td>
</tr>
<tr>
<td>1140</td>
<td>Argument error</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>1150</td>
<td>Argument error</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>1160</td>
<td>Invalid library pathname</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| A pathname must be in the form `xxx/yyyy`, 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. |
<p>| 1230 | Domain error. |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1250       | Argument Error  
Use a system of linear equations.  
Example of a system of two linear equations with variables x and y:  
3x+7y=5  
2y-5x=-1 |
| 1260       | Argument Error:  
The first argument of `nfMin` or `nfMax` must be an expression in a single variable. It cannot contain a non-valued variable other than the variable of interest. |
| 1270       | Argument Error  
Order of the derivative must be equal to 1 or 2. |
| 1280       | Argument Error  
Use a polynomial in expanded form in one variable. |
| 1290       | Argument Error  
Use a polynomial in one variable. |
| 1300       | Argument Error  
The coefficients of the polynomial must evaluate to numeric values. |
| 1310       | Argument error:  
A function could not be evaluated for one or more of its arguments. |
| 1380       | Argument error:  
Nested calls to domain() function are not allowed. |

Trigonometric conversion operators are not supported in Degree or Gradian angle modes.
### 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 166.

<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>cSolve might specify more zeros.</td>
</tr>
<tr>
<td>10006</td>
<td>Solve 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)` |
<p>| 10008        | Domain of the result might be smaller than the domain of the input. |
| 10009        | Domain of the result might be larger than the domain of the input. |
| 10012        | Non-real calculation |
| 10013        | ∞^0 or undef^0 replaced by 1 |
| 10014        | undef^0 replaced by 1 |
| 10015        | 1^∞ or 1^undef replaced by 1 |
| 10016        | 1^undef replaced by 1 |
| 10017        | Overflow replaced by ∞ or −∞ |
| 10018        | Operation requires and returns 64 bit value. |
| 10019        | Resource exhaustion, simplification might be incomplete. |
| 10020        | Trig function argument too big for accurate reduction. |
| 10021        | Input contains an undefined parameter. Result might not be valid for all possible parameter values. |</p>
<table>
<thead>
<tr>
<th>Warning code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10022</td>
<td>Specifying appropriate lower and upper bounds might produce a solution.</td>
</tr>
<tr>
<td>10023</td>
<td>Scalar has been multiplied by the identity matrix.</td>
</tr>
<tr>
<td>10024</td>
<td>Result obtained using approximate arithmetic.</td>
</tr>
<tr>
<td>10025</td>
<td>Equivalence cannot be verified in EXACT mode.</td>
</tr>
<tr>
<td>10026</td>
<td>Constraint might be ignored. Specify constraint in the form &quot;&quot; 'Variable MathTestSymbol Constant' or a conjunct of these forms, for example 'x&lt;3 and x&gt;-12'</td>
</tr>
</tbody>
</table>
General Information

Online Help

education.ti.com/eguide
Select your country for more product information.

Contact TI Support

education.ti.com/ti-cares
Select your country for technical and other support resources.

Service and Warranty Information

education.ti.com/warranty
Select your country for information about the length and terms of the warranty or about product service.

Limited Warranty. This warranty does not affect your statutory rights.
Index

- subtract .......................... 174

!, factorial .......................... 184

" second notation .................. 191

#, indirection ........................ 189

#, indirection operator .............. 217

%, percent .......................... 180

&, append ............................ 184

*, multiply .......................... 175

., dot subtraction .................... 178

.*, dot multiplication ............... 179

./, dot division ....................... 179

.^, dot power ........................ 179

.+ , dot addition ..................... 178

/, divide ............................ 176

:=, assign ........................... 195

^, power ............................. 177

|, constraint operator ............... 193

\, minute notation .................. 191

+, add ............................... 174

=, equal ............................. 180

≠, not equal .......................... 181

≤, less than or equal ................. 182

≥, greater than or equal ............. 183

>, greater than ....................... 182

Π, product ........................... 186

Σ, sum ................................. 187

Σ(), sum ............................... 187

ΣInt(), sum ........................... 188

ΣPrn(), sum ........................... 188

√, square root ....................... 186

∠ (angle) ............................ 192

∫, integral ........................... 186

► approxFraction( ) ................. 12

► Base10, display as decimal integer 17

► Base16, display as hexadecimal ... 17

► Base2, display as binary ........... 16
Cylind, display as cylindrical vector 34
DD, display as decimal angle 35
Decimal, display result as decimal 36
DMS, display as degree/minute/second 42
Grad, convert to gradian angle 68
Polar, display as polar vector 111
Rad, convert to radian angle 120
Rect, display as rectangular vector 123
Sphere, display as spherical vector 145
⇒, logical implication 183, 214
⇒, store variable 194
⇔, logical double implication 184, 214
©, comment 195
°, degree notation 191
°, degrees/minutes/seconds 191
0
0b, binary indicator 195
0h, hexadecimal indicator 195
add, + 174
amortization table, amortTbl() 7, 15
amortTbl(), amortization table 7, 15
and, Boolean operator 8
angle(), angle 8
angle, angle() 8
ANOVA, one-way variance analysis 9
ANOVA2way, two-way variance analysis 10
Ans, last answer 12
answer (last), Ans 12
append, & 184
approx(), approximate 12
approximate, approx() 12
approxRational() 13
arccos(), cos−1() 13
arccosh(), cosh−1() 13
arccot(), cot−1() 13
arccoth(), coth−1() 13
arccsc(), csc−1() 13
arccsch(), csch−1() 13
arcsec(), sec−1() 14
arcsech(), sech−1() 14
arcsin(), sin−1() 14
arcsinh(), sinh−1() 14
arctan(), tan−1() 14
arctanh(), tanh−1() 14
arguments in TVM functions 162
augment(), augment/concatenate 14
augment/concatenate, augment() 14
average rate of change, avgRC() 15
avgRC(), average rate of change 15
B
binary display, ►Base2 16
indicator, 0b 195
binomCdf() 18, 73
binomPdf() 18
Boolean operators
⇒ 183, 214
⇔ 184
and 8
nand 99
nor .................................. 103
not .................................. 104
or .................................. 108
xor .................................. 167

C
cdf() .................................. 52
ceiling(), ceiling .................. 19
ceiling, ceiling() ............... 19, 30
centralDiff() ....................... 19
char(), character string .......... 20
character string, char() ........ 20
characters
numeric code, ord() ............. 109
string, char() ..................... 20
\chi^2\text{way} ...................... 20
clear
error, ClrErr ....................... 22
Clear .................................. 200
ClearAZ ............................... 22
ClrErr, clear error ............... 22
colAugment ......................... 23
colDim(), matrix column dimension
... 23
colNorm(), matrix column norm 23
combinations, nCr() ............. 100
comment, © ........................... 195
complex
conjugate, conj() ................. 23
conj(), complex conjugate ...... 23
constraint operator "|" .......... 193
constraint operator, order of
evaluation .......................... 216
construct matrix, constructMat() 24
constructMat(), construct matrix 24
convert

\text{\textbullet}Grad .................. 68
\text{\textbullet}Rad .................. 120
copy variable or function, CopyVar
... 24
correlation matrix, corrMat() ... 25
corrMat(), correlation matrix ... 25
cos^{-1}, \arccosine ............ 26
cos(), cosine ..................... 25
cosh^{-1}(), \text{hyperbolic } \arccosine 27
cosh(), \text{hyperbolic cosine} ... 27
cosine, cos() ..................... 25
cot^{-1}(), \text{arccotangent} ... 28
cot(), cotangent ................ 28
cotangent, cot() ............... 28
coth^{-1}(), \text{hyperbolic } \arccotangent 
... 29
coth(), \text{hyperbolic cotangent} ... 28
count days between dates, dbd() ... 35
count items in a list conditionally,
countif() ............................. 29
count items in a list, count() .... 29
count(), count items in a list ...... 29
countif(), conditionally count items in a list 29
cPolyRoots() ....................... 30
cross product, crossP() ..... 30
crossP(), cross product .. 30
csc^{-1}(), \text{inverse cosecant} ... 31
csc(), cosecant .................. 31
csch^{-1}(), \text{inverse hyperbolic cosecant} ... 32
csch(), \text{hyperbolic cosecant} ... 32
cubic regression, CubicReg ... 32
CubicReg, cubic regression ... 32
cumulative sum, cumulativeSum() 33
cumulativeSum(), cumulative sum 33
cycle, Cycle .......................... 34
Cycle, cycle ......................... 34
cylindrical vector display, \text{\textbullet}Cylind ... 34

d()
\text{\textbullet}Grad .................. 68
days between dates, dbd() ...... 35
dbd(), days between dates ...... 35
decimal
angle display, \text{\textbullet}DD ... 35
integer display, \text{\textbullet}Base10 ... 17
Define ............................. 36
Define LibPriv .................... 37
Define LibPub ..................... 37
define, Define ..................... 36
Define, define ..................... 36
defining
private function or program ... 37

Index 235
<table>
<thead>
<tr>
<th>Function/Program</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>public function or program</td>
<td>37</td>
</tr>
<tr>
<td>definite integral template for</td>
<td>6</td>
</tr>
<tr>
<td>degree notation, °</td>
<td>191</td>
</tr>
<tr>
<td>degree/minute/second display, ►DMS</td>
<td>42</td>
</tr>
<tr>
<td>degree/minute/second notation</td>
<td>191</td>
</tr>
<tr>
<td>delete void elements from list</td>
<td>39</td>
</tr>
<tr>
<td>deleting variable, DelVar</td>
<td>38</td>
</tr>
<tr>
<td>deltaList()</td>
<td>38</td>
</tr>
<tr>
<td>DelVar, delete variable</td>
<td>38</td>
</tr>
<tr>
<td>delVoid(), remove void elements</td>
<td>39</td>
</tr>
<tr>
<td>derivatives first derivative, d()</td>
<td>185</td>
</tr>
<tr>
<td>numeric derivative, nDeriv()</td>
<td>101-102</td>
</tr>
<tr>
<td>numeric derivative, nDerivative()</td>
<td>100</td>
</tr>
<tr>
<td>det(), matrix determinant</td>
<td>39</td>
</tr>
<tr>
<td>diag(), matrix diagonal</td>
<td>40</td>
</tr>
<tr>
<td>dim(), dimension</td>
<td>40</td>
</tr>
<tr>
<td>dimension, dim()</td>
<td>40</td>
</tr>
<tr>
<td>Disp, display data</td>
<td>40, 135</td>
</tr>
<tr>
<td>DispAt</td>
<td>41</td>
</tr>
<tr>
<td>display as binary, ►Base2</td>
<td>16</td>
</tr>
<tr>
<td>cylindrical vector, ►Cylind</td>
<td>34</td>
</tr>
<tr>
<td>decimal angle, ►DD</td>
<td>35</td>
</tr>
<tr>
<td>decimal integer, ►Base10</td>
<td>17</td>
</tr>
<tr>
<td>degree/minute/second, ►DMS</td>
<td>42</td>
</tr>
<tr>
<td>hexadecimal, ►Base16</td>
<td>17</td>
</tr>
<tr>
<td>polar vector, ►Polar</td>
<td>111</td>
</tr>
<tr>
<td>rectangular vector, ►Rect</td>
<td>123</td>
</tr>
<tr>
<td>spherical vector, ►Sphere</td>
<td>145</td>
</tr>
<tr>
<td>display data, Disp</td>
<td>40, 135</td>
</tr>
<tr>
<td>distribution functions binomCdf()</td>
<td>18, 73</td>
</tr>
<tr>
<td>binomPdf()</td>
<td>18</td>
</tr>
<tr>
<td>invNorm()</td>
<td>74</td>
</tr>
<tr>
<td>invt()</td>
<td>74</td>
</tr>
<tr>
<td>Invx²()</td>
<td>72</td>
</tr>
<tr>
<td>normCdf()</td>
<td>104</td>
</tr>
<tr>
<td>normPdf()</td>
<td>104</td>
</tr>
<tr>
<td>poissCdf()</td>
<td>111</td>
</tr>
<tr>
<td>poissPdf()</td>
<td>111</td>
</tr>
<tr>
<td>tCdf()</td>
<td>154</td>
</tr>
<tr>
<td>tPdf()</td>
<td>157</td>
</tr>
<tr>
<td>χ²2way()</td>
<td>20</td>
</tr>
<tr>
<td>χ²Cdf()</td>
<td>20</td>
</tr>
<tr>
<td>χ²GOF()</td>
<td>21</td>
</tr>
<tr>
<td>χ²Pdf()</td>
<td>21</td>
</tr>
<tr>
<td>divide, /</td>
<td>176</td>
</tr>
<tr>
<td>dot addition, .+</td>
<td>178</td>
</tr>
<tr>
<td>division, ./</td>
<td>179</td>
</tr>
<tr>
<td>multiplication, .*</td>
<td>179</td>
</tr>
<tr>
<td>power, .^</td>
<td>179</td>
</tr>
<tr>
<td>product, dotP()</td>
<td>43</td>
</tr>
<tr>
<td>subtraction, -.</td>
<td>178</td>
</tr>
<tr>
<td>dotP(), dot product</td>
<td>43</td>
</tr>
<tr>
<td>draw</td>
<td>201-203</td>
</tr>
<tr>
<td>E e exponent template for</td>
<td>2</td>
</tr>
<tr>
<td>e to a power, e^()</td>
<td>43, 49</td>
</tr>
<tr>
<td>E, exponent</td>
<td>189</td>
</tr>
<tr>
<td>e^(), e to a power</td>
<td>43</td>
</tr>
<tr>
<td>eff(), convert nominal to effective rate</td>
<td>44</td>
</tr>
<tr>
<td>effective rate, eff()</td>
<td>44</td>
</tr>
<tr>
<td>eigenvalue, eigVl()</td>
<td>45</td>
</tr>
<tr>
<td>eigenvector, eigVc()</td>
<td>44</td>
</tr>
<tr>
<td>eigVc(), eigenvector</td>
<td>44</td>
</tr>
<tr>
<td>eigVl(), eigenvalue</td>
<td>45</td>
</tr>
<tr>
<td>else if, ElseIf</td>
<td>45</td>
</tr>
<tr>
<td>else, Else</td>
<td>68</td>
</tr>
<tr>
<td>ElseIf, else if</td>
<td>45</td>
</tr>
<tr>
<td>empty (void) elements end for, EndFor</td>
<td>212</td>
</tr>
<tr>
<td>function, EndFunc</td>
<td>54</td>
</tr>
<tr>
<td>if, EndIf</td>
<td>57</td>
</tr>
<tr>
<td>loop, EndLoop</td>
<td>68</td>
</tr>
<tr>
<td>program, EndPrgm</td>
<td>90</td>
</tr>
<tr>
<td>try, EndTry</td>
<td>114</td>
</tr>
<tr>
<td>while, EndWhile</td>
<td>158</td>
</tr>
<tr>
<td>end function, EndFunc</td>
<td>167</td>
</tr>
</tbody>
</table>
getVarInfo(), get/return variables information ........................................ 67
go to, Goto ........................................................................ 68
Goto, go to ......................................................................... 68
gradian notation, g .......................................................... 190
greater than or equal, ≥ .................................................... 183
greater than, > .................................................................. 182
greatest common divisor, gcd() ........................................... 58

groups, locking and unlocking ............................................. 86, 164
groups, testing lock status ................................................... 64

H

hexadecimal
 display, •Base16 .............................................................. 17
 indicator, 0h .................................................................. 195

hyperbolic
 arccosine, cosh⁻¹( ) ......................................................... 27
 arcsine, sinh⁻¹( ) ............................................................... 143
 arctangent, tanh⁻¹( ) ......................................................... 154
 cosine, cosh( ) ................................................................ 27
 sine, sinh( ) ..................................................................... 142
 tangent, tanh( ) ............................................................... 153

I

identity matrix, identity( ) .................................................... 68
identity( ), identity matrix .................................................. 68
if, If ................................................................................. 68
If, if ................................................................................. 68
ifFn( ) .............................................................................. 70
imag( ), imaginary part ..................................................... 70
 imaginary part, imag( ) ....................................................... 70
indirection operator (#) ....................................................... 217
indirection, # ................................................................... 189
inString( ), within string .................................................... 71
int( ), integer .................................................................... 71
intDiv( ), integer divide ..................................................... 72
 integer divide, intDiv( ) .................................................... 72
 integer part, iPart( ) ........................................................ 74
 integer, int( ) .................................................................... 71
integral, ∫ ......................................................................... 186
interpolate(), interpolate ................................................... 72
 inverse cumulative normal distribution (invNorm( ) ........ 74

L

label, Lbl ........................................................................... 76
language
 get language information .................................................. 64
Lbl, label ........................................................................... 76
lcm, least common multiple .................................................. 76
 least common multiple, lcm .............................................. 76
 left( ), left( ) .................................................................... 76
 length of string .................................................................. 40
 less than or equal, ≤ .......................................................... 182
 LibPriv ............................................................................. 37
 LibPub ............................................................................... 37
 library
 create shortcuts to objects .................................................. 77
 libShortcut( ), create shortcuts to library objects .................. 77
 linear regression, LinRegAx .............................................. 78
 linear regression, LinRegBx .............................................. 78
 LinRegBx, linear regression .............................................. 77
 LinRegMx, linear regression .............................................. 78
 LinRegtIntervals, linear regression .................................... 79
 LinRegtTest ....................................................................... 81
 linSolve() ........................................................................ 82
 Δlist( ), list difference ......................................................... 83
 list to matrix, list•mat( ) ..................................................... 83
 list, conditionally count items in ....................................... 29
 list, count items in ............................................................. 29
 list•mat( ), list to matrix .................................................... 83
 lists
 augment/concatenate, augment( ) .................................... 14
<table>
<thead>
<tr>
<th>term</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix ((m \times n))</td>
<td>4</td>
</tr>
<tr>
<td>matrix to list, mat(\text{list})</td>
<td>91</td>
</tr>
<tr>
<td>max(), maximum</td>
<td>91</td>
</tr>
<tr>
<td>maximum, max()</td>
<td>91</td>
</tr>
<tr>
<td>mean(), mean</td>
<td>92</td>
</tr>
<tr>
<td>mean, mean()</td>
<td>92</td>
</tr>
<tr>
<td>median(), median</td>
<td>92</td>
</tr>
<tr>
<td>median, median()</td>
<td>92</td>
</tr>
<tr>
<td>medium-medium line regression, MedMed</td>
<td>93</td>
</tr>
<tr>
<td>MedMed, medium-medium line regression</td>
<td>93</td>
</tr>
<tr>
<td>mid-string, mid()</td>
<td>94</td>
</tr>
<tr>
<td>mid(), mid-string</td>
<td>94</td>
</tr>
<tr>
<td>min(), minimum</td>
<td>94</td>
</tr>
<tr>
<td>minimum, min()</td>
<td>94</td>
</tr>
<tr>
<td>minute notation, (\text{'})</td>
<td>191</td>
</tr>
<tr>
<td>mirr(), modified internal rate of return</td>
<td>95</td>
</tr>
<tr>
<td>mixed fractions, using propFrac()</td>
<td>116</td>
</tr>
<tr>
<td>mod(), modulo</td>
<td>95</td>
</tr>
<tr>
<td>mode settings, getMode()</td>
<td>65</td>
</tr>
<tr>
<td>modes</td>
<td>137</td>
</tr>
<tr>
<td>setting, setMode()</td>
<td>137</td>
</tr>
<tr>
<td>modified internal rate of return, mirr()</td>
<td>95</td>
</tr>
<tr>
<td>modulo, mod()</td>
<td>95</td>
</tr>
<tr>
<td>mRow(), matrix row operation</td>
<td>96</td>
</tr>
<tr>
<td>mRowAdd(), matrix row multiplication and addition</td>
<td>96</td>
</tr>
<tr>
<td>Multiple linear regression t test</td>
<td>98</td>
</tr>
<tr>
<td>multiply, (*)</td>
<td>175</td>
</tr>
<tr>
<td>MultReg</td>
<td>96</td>
</tr>
<tr>
<td>MultRegIntervals()</td>
<td>97</td>
</tr>
<tr>
<td>MultRegTests()</td>
<td>98</td>
</tr>
<tr>
<td>(n\text{Cr}())</td>
<td>83</td>
</tr>
<tr>
<td>combinations</td>
<td>100</td>
</tr>
<tr>
<td>nDerivative(), numeric derivative</td>
<td>100</td>
</tr>
<tr>
<td>negation, entering negative numbers</td>
<td>217</td>
</tr>
<tr>
<td>net present value, npv()</td>
<td>106</td>
</tr>
<tr>
<td>new list, newList()</td>
<td>101</td>
</tr>
<tr>
<td>matrix, newMat()</td>
<td>101</td>
</tr>
<tr>
<td>newList(), new list</td>
<td>101</td>
</tr>
<tr>
<td>newMat(), new matrix</td>
<td>101</td>
</tr>
<tr>
<td>nfMax(), numeric function maximum</td>
<td>101</td>
</tr>
<tr>
<td>nfMin(), numeric function minimum</td>
<td>102</td>
</tr>
<tr>
<td>nInt(), numeric integral</td>
<td>102</td>
</tr>
<tr>
<td>nom(), convert effective to nominal rate</td>
<td>103</td>
</tr>
<tr>
<td>nominal rate, nom()</td>
<td>103</td>
</tr>
<tr>
<td>nor, Boolean operator</td>
<td>103</td>
</tr>
<tr>
<td>norm(), Frobenius norm</td>
<td>104</td>
</tr>
<tr>
<td>normal distribution probability, normCdf()</td>
<td>104</td>
</tr>
<tr>
<td>normCdf()</td>
<td>104</td>
</tr>
<tr>
<td>normPdf()</td>
<td>104</td>
</tr>
<tr>
<td>not equal, (\neq)</td>
<td>181</td>
</tr>
<tr>
<td>not, Boolean operator</td>
<td>104</td>
</tr>
<tr>
<td>nPr(), permutations</td>
<td>105</td>
</tr>
<tr>
<td>npv(), net present value</td>
<td>106</td>
</tr>
<tr>
<td>nSolve(), numeric solution</td>
<td>106</td>
</tr>
<tr>
<td>nth root</td>
<td>106</td>
</tr>
<tr>
<td>template for</td>
<td>1</td>
</tr>
<tr>
<td>numeric derivative, nDeriv()</td>
<td>101-102</td>
</tr>
<tr>
<td>derivative, nDerivative()</td>
<td>100</td>
</tr>
<tr>
<td>integral, nInt()</td>
<td>102</td>
</tr>
<tr>
<td>solution, nSolve()</td>
<td>106</td>
</tr>
<tr>
<td>(O)</td>
<td></td>
</tr>
<tr>
<td>objects</td>
<td>77</td>
</tr>
<tr>
<td>create shortcuts to library</td>
<td>107</td>
</tr>
<tr>
<td>one-variable statistics, OneVar</td>
<td>107</td>
</tr>
<tr>
<td>OneVar, one-variable statistics</td>
<td>107</td>
</tr>
<tr>
<td>operators</td>
<td>216</td>
</tr>
<tr>
<td>order of evaluation</td>
<td>216</td>
</tr>
<tr>
<td>or (Boolean), or</td>
<td>108</td>
</tr>
<tr>
<td>or, Boolean operator</td>
<td>108</td>
</tr>
<tr>
<td>ord(), numeric character code</td>
<td>109</td>
</tr>
<tr>
<td>(P)</td>
<td>109</td>
</tr>
<tr>
<td>P(\text{Rx}()), rectangular x coordinate</td>
<td>109</td>
</tr>
</tbody>
</table>
P•Ry(), rectangular y coordinate .......................... 110
pass error, PassErr ........................................ 110
PassErr, pass error ........................................ 110
Pdf() .................................................................... 55
percent, % .......................................................... 180
permutations, nPr() ............................................. 105

piecewise function (2-piece) template for .................. 2
piecewise function (N-piece) template for ................. 2
piecewise() ......................................................... 111
poissCdf() .......................................................... 111
poissPdf() .......................................................... 111
polar coordinate, R•Pr() ....................................... 120
coordinate, R•Pθ() .............................................. 119
vector display, •Polar .......................................... 111
polyEval(), evaluate polynomial ............................ 112
polynomials evaluate, polyEval() ......................... 112
random, randPoly() ........................................... 122
PolyRoots() ....................................................... 113
power of ten, 10^() ............................................. 192
power regression, PowerReg .................................. 113, 126-127, 155
power, ^ ........................................................... 177
PowerReg, power regression ................................. 113
Prgm, define program ......................................... 114
prime number test, isPrime() ............................... 75
probability density, normPdf() ............................ 104
prodSeq() .......................................................... 115
product(), product ............................................. 115
product, Π() ........................................................ 186
template for ...................................................... 5
product, product() ............................................. 115
programming define program, Prgm .......................... 114
display data, Disp ............................................. 40, 135
pass error, PassErr ............................................. 110
programs defining private library ............................ 37
defining public library ........................................ 37
programs and programming clear error, ClrErr .......... 22
display I/O screen, Disp ...................................... 40, 135
end program, EndPrgm ..................................... 114
define program, Prgm ....................................... 114
try, EndTry ...................................................... 158
try, Try ............................................................ 158
proper fraction, propFrac ...................................... 116
propFrac, proper fraction ..................................... 116

Q
QR factorization, QR ............................................. 116
QR, QR factorization .......................................... 116
quadratic regression, QuadReg ............................ 117
QuadReg, quadratic regression .............................. 117
quartic regression, QuartReg ............................... 118
QuartReg, quartic regression ................................ 118

R
R, radian .......................................................... 190
R•Pr(), polar coordinate ...................................... 120
R•Pθ(), polar coordinate ..................................... 119
radian, R .......................................................... 190
random, rand() .................................................. 120
randBin(), random number ................................ 120
randInt(), random integer .................................. 121
randMat(), random matrix .................................. 121
randNorm(), random norm .................................. 122
random
matrix, randMat() ............................................ 121
norm, randNorm() ............................................. 122
number seed, RandSeed ..................................... 122
polynomial, randPoly() ....................................... 122
random sample .................................................. 122
randPoly(), random polynomial .......................... 122
randSamp() ..................................................... 122
RandSeed, random number seed ......................... 122
real(), real ..................................................... 122
real(), real ..................................................... 122
reciprocal, ^⁻¹ .................................................. 193
rectangular-vector display, •Rect .......................... 123
rectangular x coordinate, P•Rx() .......................... 109
rectangular y coordinate, P•Ry() .......................... 110
reduced row echelon form, rref() ......................... 133
rref(), row echelon form .................................... 124

Index 241
RefreshProbeVars .......................... 125
regressions
  cubic, CubicReg  ..................... 32
  exponential, ExpReg  ............... 50
  linear regression, LinRegAx  ...... 78
  linear regression, LinRegBx  ...... 77, 79
logistic, Logistic  ....................... 87
medium-medium line, MedMed  ........ 93
MultReg  .................................. 96
power regression,
  PowerReg  . 113, 126-127, 155
quadratic, QuadReg  ...................... 117
quartic, QuartReg  ...................... 118
sinusoidal, SinReg  ....................... 143
remain(), remainder  .................... 126
remainder, remain()  ..................... 126
remove
  void elements from list  .......... 39
Request  .................................. 126
RequestStr  ................................ 127
result values, statistics  ............. 147
results, statistics  ...................... 146
return, Return  ............................ 128
Return, return  ............................ 128
right(), right  ............................ 128
rk23(), Runge Kutta function  ....... 129
rotate(), rotate  .......................... 130
rotate, rotate()  .......................... 130
round(), round  ............................ 132
round, round()  ............................ 132
row echelon form, ref()  .............. 124
rowAdd(), matrix row addition  ...... 132
rowDim(), matrix row dimension  ...... 132
rowNorm(), matrix row norm  .......... 132
rowSwap(), matrix row swap  .......... 133
rref(), reduced row echelon form  ... 133

sech(), hyperbolic secant  .......... 134
second derivative
  template for  ........................... 5
second notation, "  ...................... 191
seq(), sequence  ........................ 135
seqGen()  .................................. 136
seqn()  .................................... 136
sequence, seq()  ......................... 135-136
set
  mode, setMode()  ....................... 137
setMode(), set mode  ..................... 137
settings, get current  ................. 65
shift(), shift  ............................ 138
shift, shift()  ............................ 138
sign(), sign  .............................. 140
sign, sign()  .............................. 140
simult(), simultaneous equations  ... 140
simultaneous equations, simult()  ... 140
sin⁻¹(), arcsine  .......................... 142
sin(), sine  ............................... 141
sine, sin()  ............................... 141
sinh⁻¹(), hyperbolic arcsine  .......... 143
sinh(), hyperbolic sine  .............. 142
SinReg, sinusoidal regression  ....... 143
sinusoidal regression, SinReg  ...... 143
SortA, sort ascending  .................. 145
SortD, sort descending  ............... 145
sorting
  ascending, SortA  ...................... 145
descending, SortD  ....................... 145
spherical vector display, Sphere  .... 145
sqrt(), square root  ...................... 146
square root
  template for  ........................... 1
standard deviation, stdDev()  ....... 148, 164
stat.results  ............................. 146
stat.values  .............................. 147
statistics
  combinations, nCr()  .......... 100
factorial, !  .............................. 184
mean, mean()  ............................ 92
median, median()  ....................... 92
one-variable statistics, OneVar  ... 107
<table>
<thead>
<tr>
<th>term</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>permutations, nPr()</td>
<td>105</td>
</tr>
<tr>
<td>random norm, randNorm()</td>
<td>122</td>
</tr>
<tr>
<td>random number seed, RandSeed</td>
<td>122</td>
</tr>
<tr>
<td>standard deviation, stdDev()</td>
<td>148</td>
</tr>
<tr>
<td>two-variable results, TwoVar</td>
<td>162</td>
</tr>
<tr>
<td>variance, variance()</td>
<td>165</td>
</tr>
<tr>
<td>stdDevPop(), population standard deviation</td>
<td>148</td>
</tr>
<tr>
<td>stdDevSamp(), sample standard deviation</td>
<td>148</td>
</tr>
<tr>
<td>Stop command</td>
<td>149</td>
</tr>
<tr>
<td>store variable -&gt;</td>
<td>194</td>
</tr>
<tr>
<td>storing symbol, &amp;</td>
<td>195</td>
</tr>
<tr>
<td>string dimension, dim()</td>
<td>40</td>
</tr>
<tr>
<td>length</td>
<td>40</td>
</tr>
<tr>
<td>string(), expression to string</td>
<td>149</td>
</tr>
<tr>
<td>strings append, &amp;</td>
<td>184</td>
</tr>
<tr>
<td>character code, ord()</td>
<td>109</td>
</tr>
<tr>
<td>character string, char()</td>
<td>20</td>
</tr>
<tr>
<td>expression to string, string()</td>
<td>149</td>
</tr>
<tr>
<td>format, format()</td>
<td>54</td>
</tr>
<tr>
<td>formatting</td>
<td>54</td>
</tr>
<tr>
<td>indirection, #</td>
<td>189</td>
</tr>
<tr>
<td>left, left()</td>
<td>76</td>
</tr>
<tr>
<td>mid-string, mid()</td>
<td>94</td>
</tr>
<tr>
<td>right, right()</td>
<td>46, 72, 128-129</td>
</tr>
<tr>
<td>rotate, rotate()</td>
<td>130</td>
</tr>
<tr>
<td>shift, shift()</td>
<td>138</td>
</tr>
<tr>
<td>string to expression, expr()</td>
<td>50</td>
</tr>
<tr>
<td>using to create variable names</td>
<td>217</td>
</tr>
<tr>
<td>within, InString</td>
<td>71</td>
</tr>
<tr>
<td>student-t distribution probability, tCdf()</td>
<td>154</td>
</tr>
<tr>
<td>student-t probability density, tPdf()</td>
<td>157</td>
</tr>
<tr>
<td>subMat(), submatrix</td>
<td>150-151</td>
</tr>
<tr>
<td>submatrix, subMat()</td>
<td>150-151</td>
</tr>
<tr>
<td>substitution with &quot;</td>
<td>&quot; operator</td>
</tr>
<tr>
<td>subtract, -</td>
<td>174</td>
</tr>
<tr>
<td>sum of interest payments</td>
<td>187</td>
</tr>
<tr>
<td>sum of principal payments</td>
<td>188</td>
</tr>
<tr>
<td>sum(), summation</td>
<td>150</td>
</tr>
<tr>
<td>sum, Σ()</td>
<td>187</td>
</tr>
<tr>
<td>template for system of equations (2-equation)</td>
<td>3</td>
</tr>
<tr>
<td>template for system of equations (N-equation)</td>
<td>3</td>
</tr>
<tr>
<td>t test, tTest</td>
<td>159</td>
</tr>
<tr>
<td>T, transpose</td>
<td>152</td>
</tr>
<tr>
<td>tan⁻¹(), arctangent</td>
<td>153</td>
</tr>
<tr>
<td>tan(), tangent</td>
<td>152</td>
</tr>
<tr>
<td>tangent, tan()</td>
<td>152</td>
</tr>
<tr>
<td>tanh⁻¹(), hyperbolic arctangent</td>
<td>154</td>
</tr>
<tr>
<td>tanh(), hyperbolic tangent</td>
<td>153</td>
</tr>
<tr>
<td>tCdf(), student distribution probability</td>
<td>154</td>
</tr>
<tr>
<td>templates absolute value</td>
<td>3-4</td>
</tr>
<tr>
<td>definite integral</td>
<td>6</td>
</tr>
<tr>
<td>e exponent</td>
<td>2</td>
</tr>
<tr>
<td>exponent</td>
<td>1</td>
</tr>
<tr>
<td>first derivative</td>
<td>5</td>
</tr>
<tr>
<td>fraction</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>2</td>
</tr>
<tr>
<td>matrix (1 × 2)</td>
<td>4</td>
</tr>
<tr>
<td>matrix (2 × 1)</td>
<td>4</td>
</tr>
<tr>
<td>matrix (2 × 2)</td>
<td>4</td>
</tr>
<tr>
<td>matrix (m × n)</td>
<td>4</td>
</tr>
<tr>
<td>nth root</td>
<td>1</td>
</tr>
<tr>
<td>piecewise function (2-piece)</td>
<td>2</td>
</tr>
<tr>
<td>piecewise function (N-piece)</td>
<td>2</td>
</tr>
<tr>
<td>product, Π()</td>
<td>5</td>
</tr>
<tr>
<td>second derivative</td>
<td>5</td>
</tr>
<tr>
<td>square root</td>
<td>1</td>
</tr>
<tr>
<td>sum, Σ()</td>
<td>5</td>
</tr>
<tr>
<td>system of equations (2-equation)</td>
<td>3</td>
</tr>
<tr>
<td>system of equations (N-equation)</td>
<td>3</td>
</tr>
</tbody>
</table>

Index 243
test for void, isVoid( ) .................. 75
Test_2S, 2-sample F test .................. 57
Text command .......................... 155
time value of money, Future Value .... 160
time value of money, Interest .......... 160
time value of money, number of payments .......... 161
time value of money, payment amount .......... 161
time value of money, present value .......... 161
tInterval, t confidence interval ........... 155
tInterval_2Samp, two-sample t confidence interval .......... 156
tPdf( ), student probability density ...... 157
transpose, T .......................... 152
Try, error handling command .............. 158
tTest, t test ................................ 159
tTest_2Samp, two-sample t test .......... 159
TVM arguments .......................... 162
tmFV( ) .................................. 160
tmI( ) .................................. 160
tmN( ) .................................. 161
tmPmt( ) .................................. 161
tmPV( ) .................................. 161
two-variable results, TwoVar .......... 162
TwoVar, two-variable results .......... 162

U
unit vector, unitV( ) ..................... 164
unitV( ), unit vector ..................... 164
unLock, unlock variable or variable group ..................... 164
unlocking variables and variable group ..................... 164
user-defined functions .................... 36
user-defined functions and programs ..................... 37

V
variable creating name from a character string ..................... 217
variable and functions copying ..................... 24
variables clear all single-letter ..................... 22
delete, DelVar ..................... 38
local, Local ..................... 86
variables, locking and unlocking .......... 64, 86, 164
variance, variance( ) .................. 165
varPop( ) ..................... 164
varSamp( ), sample variance .......... 165
vectors cross product, crossP( ) ......... 30
cylindrical vector display, ► Cylind .......... 34
dot product, dotP( ) .................. 43
unit, unitV( ) ..................... 164
void elements ..................... 212
void elements, remove .................. 39
void, test for ..................... 75

W
Wait command ..................... 165
warnCodes( ), Warning codes .......... 166
warning codes and messages .......... 230
when( ), when ..................... 167
when, when( ) ..................... 167
while, While ..................... 167
While, while ..................... 167
with, | ..................... 193
within string, inString( ) .......... 71

X
x², square ..................... 178
XNOR ..................... 184
xor, Boolean exclusive or ..................... 167

Z
zInterval, z confidence interval .......... 168
zInterval_1Prop, one-proportion z confidence interval .......... 169
zInterval_2Prop, two-proportion z confidence interval .......... 169
zInterval_2Samp, two-sample z confidence interval .......... 170
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>zTest</td>
<td>171</td>
</tr>
<tr>
<td>zTest_1Prop, one-proportion z test</td>
<td>172</td>
</tr>
<tr>
<td>zTest_2Prop, two-proportion z test</td>
<td>172</td>
</tr>
<tr>
<td>zTest_2Samp, two-sample z test</td>
<td>172</td>
</tr>
<tr>
<td>( \chi^2 \text{Cdf()} )</td>
<td>20</td>
</tr>
<tr>
<td>( \chi^2 \text{GOF} )</td>
<td>21</td>
</tr>
<tr>
<td>( \chi^2 \text{Pdf()} )</td>
<td>21</td>
</tr>
</tbody>
</table>