A Guide to the HP38G

Math Menu

and

Programming

Acknowledgments:

Thanks to Charles Patton of the design team of the HP38G for his help, both by sharing an advance copy of the MATH menu specifications, and by providing answers to many of my questions regarding syntax and usage. Also, many thanks to Mark Howell, Sam Gough, and Jill Lovorn for sharing their programs and expertise.

June 1995
A Guide to the HP38G

Math Menu

and

Programming

Acknowledgments:

Thanks to Charles Patton of the design team of the HP38G for his help, both by sharing an advance copy of the MATH menu specifications, and by providing answers to many of my questions regarding syntax and usage. Also, many thanks to Mark Howell, Sam Gough, and Jill Lovorn for sharing their programs and expertise.

June 1995
Contents

1. MATH FUNCTIONS

A catalog of all the built-in functions that don't appear on the keyboard of the HP38G, along with specific syntax and explanations of the arguments and the result returned by each function.

2. PROGRAM COMMANDS

A catalog of all the built-in commands that can be used on the HP38G (either on the HOME screen or from within a program) along with specific syntax and explanations of the arguments and action of each command.

3. PROGRAM CONSTANTS

A catalog of all the built-in constants which the HP38G uses to set variables.

4. PROGRAM HINTS AND EXAMPLES

A discussion of programming basics, along with a collection of examples of programs written for the HP38G.
Introduction

This is reference material in addition to that found in the Quick Guide for the HP38G.

It includes a listing of all the different functions, commands, and constants available on the HP38G, along with some hints on programming.

The MATH key provides the access to the catalogs for all the functions and commands available that don't appear directly on the keyboard. When you press MATH, you'll see the following menu labels along the bottom of the screen:

MTH      CMD  CONS      CANCL      OK

The first three of these labels each activate a different "double" menu.

The currently active menu is indicated by a little white square lit up on one of these three labels and also by a title bar you see at the top of the menu.

MTH activates the menu titled MATH FUNCTIONS
CMDS activates the menu titled PROGRAM COMMANDS
CONS activates the menu titled PROGRAM CONSTANTS
Notes on Terminology and Conventions

What's the difference between a Function and a Command?

A Function always takes zero or more arguments and then returns exactly one result. For example, RAND takes no arguments and returns a random number between 0 and 1. COMB takes two arguments m and n and returns the number of combinations of n items that can be chosen from m objects: COMB(m,n) = m!/(n!((m-n)!)). Functions can be used in any expression.

A Command also takes zero or more arguments, but it does not return a result. Instead, a command may affect the value of a variable (STO> is a good example) or it may perform some specific action (BEEP is an example). Commands are used in programs, but you can also execute them from the HOME screen.

For each function and command, the syntax is given showing the required punctuation (parentheses, comma's, semicolon's, etc.) and the order of any needed arguments. Some specific types of arguments are indicated as follows:

- `<name>` a variable name
- `<expr>` any expression
- `<N#>` a positive integer or expression that has a positive integer value
- `<R#>` a real number or expression that has a real numeric value
- `<C#>` a complex number or expression that has a complex value
- `<#>` a real or complex number or expression
- `<{}>` a list or an expression that has a list value
- `<M#>` a matrix or an expression that has a matrix value
- `<V#>` a vector or an expression that has a vector value

Other more descriptive abbreviations are sometimes used.

In this reference material you will find the following information provided for each math function or programming command:

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax of usage (showing type and order of arguments)</th>
<th>Description of Arguments</th>
<th>Description of Result or Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCN or CMD</td>
<td>FCN(&lt;arg1&gt;,&lt;arg2&gt;...) or CMD &lt;arg1&gt;;&lt;arg2&gt;;...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additional notes of explanation are also provided as needed.

HELPWITH

If you type HELPWITH FCN or HELPWITH CMD on the HOME screen (where FCN or CMD is a specific function name or programming command) a box comes up which shows the syntax needed.
# 1. Math Functions

## Overview of the MATH FUNCTIONS Menu

<table>
<thead>
<tr>
<th>Calculus</th>
<th>Complex</th>
<th>Constant</th>
<th>Hyperb.</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>ARG</td>
<td>$e$</td>
<td>ACOSH</td>
<td>CONCAT</td>
</tr>
<tr>
<td>$\int$</td>
<td>CONJ</td>
<td>$i$</td>
<td>ASINH</td>
<td>$\Delta$LIST</td>
</tr>
<tr>
<td>TAYLOR</td>
<td>IM</td>
<td>MAXREAL</td>
<td>ATANH</td>
<td>MAKELIST</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>MINREAL</td>
<td>COSH</td>
<td>$\Pi$LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi$</td>
<td>SINH</td>
<td>POS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TANH</td>
<td>REVERSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALOG</td>
<td>SIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXP</td>
<td>$\Sigma$LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EXPM1</td>
<td>SORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LNP1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop</th>
<th>Matrix</th>
<th>Polynom.</th>
<th>Prob.</th>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITERATE</td>
<td>COLNORM</td>
<td>POLYCOEF</td>
<td>COMB</td>
<td>CEILING</td>
</tr>
<tr>
<td>RECURSE</td>
<td>COND</td>
<td>POLYEVAL</td>
<td>!</td>
<td>DEG→RAD</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>CROSS</td>
<td>POLYFORM</td>
<td>PERM</td>
<td>FLOOR</td>
</tr>
<tr>
<td></td>
<td>DET</td>
<td>POLYROOT</td>
<td>RANDOM</td>
<td>FNROOT</td>
</tr>
<tr>
<td></td>
<td>DOT</td>
<td></td>
<td>UTPC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EIGENVAL</td>
<td></td>
<td>UTPF</td>
<td>HMS→</td>
</tr>
<tr>
<td></td>
<td>EIGENVV</td>
<td></td>
<td>UTPN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDENMAT</td>
<td></td>
<td>UTPT</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td>INVERSE</td>
<td></td>
<td></td>
<td>MANT</td>
</tr>
<tr>
<td></td>
<td>LQ</td>
<td></td>
<td></td>
<td>MAX</td>
</tr>
<tr>
<td></td>
<td>LSQ</td>
<td></td>
<td></td>
<td>MIN</td>
</tr>
<tr>
<td></td>
<td>LU</td>
<td></td>
<td></td>
<td>MOD</td>
</tr>
<tr>
<td></td>
<td>MAKEMAT</td>
<td></td>
<td></td>
<td>%</td>
</tr>
<tr>
<td></td>
<td>QR</td>
<td></td>
<td></td>
<td>%CHANGE</td>
</tr>
<tr>
<td></td>
<td>RANK</td>
<td></td>
<td></td>
<td>%TOTAL</td>
</tr>
<tr>
<td></td>
<td>ROWNORM</td>
<td></td>
<td></td>
<td>RAD→DEG</td>
</tr>
<tr>
<td></td>
<td>RREF</td>
<td></td>
<td></td>
<td>ROUND</td>
</tr>
<tr>
<td></td>
<td>SCHUR</td>
<td></td>
<td></td>
<td>SIGN</td>
</tr>
<tr>
<td></td>
<td>SIZE</td>
<td></td>
<td></td>
<td>TRUNCATE</td>
</tr>
<tr>
<td></td>
<td>SPECNORM</td>
<td></td>
<td></td>
<td>XPON</td>
</tr>
<tr>
<td></td>
<td>SPECRAD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SVD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SVL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRACE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TRN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stat-Two</th>
<th>Symbolic</th>
<th>Test</th>
<th>Trig</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREDX</td>
<td>$=$</td>
<td>$&lt;$</td>
<td>ACOT</td>
</tr>
<tr>
<td>PREDY</td>
<td>ISOLATE</td>
<td>$\leq$</td>
<td>ACSC</td>
</tr>
<tr>
<td></td>
<td>LINEAR?</td>
<td>$=$</td>
<td>ASEC</td>
</tr>
<tr>
<td></td>
<td>QUAD</td>
<td>$\neq$</td>
<td>COT</td>
</tr>
<tr>
<td></td>
<td>QUOTE</td>
<td>$&gt;$</td>
<td>CSC</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>$</td>
<td>$\geq$</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XOR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Syntax Guide to MATH FUNCTIONS

Calculus

\( \delta \) \delta<name>(<expr>)
\( x, expression \) \longrightarrow derivative of the expression with respect to variable x

\( \int \) \int(<R#>,<R#>,<expr>,<name>)
\( a,b,expression,x \) \longrightarrow integral from a to b of expression with respect to x

TAYLOR TAYLOR(<expr>,<name>,<N#>)
\( expression,x,n \) \longrightarrow Taylor polynomial about x = 0 of order n

Complex

ARG ARG(<C#>)
\( complex number \) \longrightarrow argument of complex number [i.e., if the complex number z has polar form (r,t), then ARG(z) = t]

CONJ CONJ(<C#>)
\( complex number \) \longrightarrow conjugate of complex number

IM IM(<C#>)
\( complex number \) \longrightarrow imaginary part

RE RE(<C#>)
\( complex number \) \longrightarrow real part

Constant

(These take no arguments; they are special numbers with special names.)

e the base of the natural log (rounded to 2.71828182846)
i (0,1) the standard square root of -1
MAXREAL the constant 9.9999999E499 (largest positive real number that can be represented on the HP38G)
MINREAL the constant 1.0000000E-499 (smallest positive real number that can be represented on the HP38G)
\( \pi \) the constant \( \pi \) (rounded to 3.14159265359)
### Hyperbolic Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOSH((x))</td>
<td>inverse hyperbolic cosine of (x)</td>
</tr>
<tr>
<td>ASINH((x))</td>
<td>inverse hyperbolic sine of (x)</td>
</tr>
<tr>
<td>ATANH((x))</td>
<td>inverse hyperbolic tangent of (x)</td>
</tr>
<tr>
<td>COSH((x))</td>
<td>hyperbolic cosine of (x)</td>
</tr>
<tr>
<td>SINH((x))</td>
<td>hyperbolic sine of (x)</td>
</tr>
<tr>
<td>TANH((x))</td>
<td>hyperbolic tangent of (x)</td>
</tr>
<tr>
<td>ALOG((x))</td>
<td>(10^x)</td>
</tr>
<tr>
<td>EXP((x))</td>
<td>(e^x)</td>
</tr>
<tr>
<td>EXPM1((x))</td>
<td>exponential of (x) minus one</td>
</tr>
<tr>
<td>LNP1((x))</td>
<td>logarithm of ((x+1))</td>
</tr>
</tbody>
</table>

**NOTE:** More accurate than the usual operation.
List

**CONCAT**  
CONCAT(<{}>,<{}>) Concatenate two lists.  
\{X1,...,xn\}, \{y1,...,ym\}  
\{x1,...,xn,y1,...,ym\}  
NOTE: Joins two lists end to end to make a new list.

**ΔLIST**  
ΔLIST(<{}>) List of "first differences"  
\{x1,x2,...,xn\}  
\{x2-x1,x3-x2,...xn-x(n-1)\}

**ΠLIST**  
ΠLIST(<{}>) Product of elements  
\{x1,...,xn\}  
\x1*x2*...*xn

**ΣLIST**  
ΣLIST(<{}>) Sum of elements  
\{x1,...,xn\}  
\x1+...+xn

**MAKELIST**  
MAKELIST(<expr>,<name>,<#,<#,<#>))  
evaluation of expression for each value of the variable name obtained by stepping  
--------->  
\{expr|(name =start),expr|(name =start+step), ..., expr|(name =end) \}

NOTE: Creates a list by evaluating the expression for each value of  
the variable name obtained by stepping from the starting  
value to the end value by the step size.

EXAMPLE:  
MAKELIST(X^2,X,1,5,2) creates \{1,9,25\}

**POS**  
POS(<{}>,<N#>) Position of value in list  
\{x1,...,xn\}, y  
\index value \(i\) such that \(xi = y\),  
or 0 if no such \(i\) exists

**REVERSE**  
REVERSE(<{}>)  
\list \in reverse order

**SIZE**  
SIZE(<{}>)  
\size of the list (# of elements)

**SORT**  
SORT(<{}>)  
\list sorted in increasing order
Loop

ITERATE

ITERATE(<expr>,<name>,<R#>,<N#>)
expression,name,x0,n--------> iterates the function
expression in variable name n
times starting with input x0

EXAMPLE: ITERATE(X^2,X,5,3) results in 390625 because we
have 5^2 = 25, 25^2 = 625, and 625^2 = 390625.

RECURSE

RECURSE(<name>,<expr>,<N#>,<N#>)
index variable, Nth term, first term, second term
----------> allows you to store a sequence definition from within a program or from
the HOME screen.
EXAMPLE: RECURSE(N, N^2, 1, 4) STO > U1(N)

Σ

Σ(<name> = <N#>,<N#>,<expr>)
index n, starting index n1, ending index n2, expression for nth term
----------> summation of expression in index variable n from n = n1 to n = n2

Matrix

COLNORM

COLNORM(<M#>)
matrix [[M]]----------> column-norm of M

COND

COND(<M#>)
matrix [[M]]----------> condition number of M

CROSS

CROSS(<V#>,<V#>)
vectors [v1], [v2]----------> cross-product of v1 and v2

DET

DET(<M#>)
matrix [[M]]----------> determinant of M

DOT

DOT(<V#>,<V#>)
vectors [v1], [v2]----------> dot-product of v1 and v2

EIGENVAL

EIGENVAL(<M#>)
matrix [[M]]----------> [eigenvalues] (vector of
eigenvalues of M)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIGENVV</td>
<td>Computes the eigenvectors and eigenvalues of a matrix.</td>
</tr>
<tr>
<td>IDENMAT</td>
<td>Creates an nxn identity matrix.</td>
</tr>
<tr>
<td>INVERSE</td>
<td>Computes the multiplicative inverse of a matrix.</td>
</tr>
<tr>
<td>LQ</td>
<td>Decomposes a matrix into L and Q.</td>
</tr>
<tr>
<td>LSQ</td>
<td>Solves the least squares problem.</td>
</tr>
<tr>
<td>LU</td>
<td>Decomposes a matrix into L, U, and P.</td>
</tr>
<tr>
<td>MAKEMAT</td>
<td>Creates a matrix from an expression.</td>
</tr>
<tr>
<td>QR</td>
<td>Decomposes a matrix into Q, R, and P.</td>
</tr>
<tr>
<td>RANK</td>
<td>Computes the rank of a matrix.</td>
</tr>
<tr>
<td>ROWNORM</td>
<td>Computes the row-norm of a matrix.</td>
</tr>
<tr>
<td>RREF</td>
<td>Computes the reduced row-echelon form of a matrix.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
Matrix Continued

EIGENVV(EIGENVV(<M#>))

matrix [[M]]

NOTE: list of matrix of eigenvectors and a vector of eigenvalues

IDENMAT(IDENMAT(<N#>))

integer n

NOTE: Creates an nxn identity matrix

INVERSE(INVERSE(<M#>))

matrix [[M]]

NOTE: Multiplicative inverse M^-1

LQ(LQ(<M#>))

matrix [[M]]

NOTE: Decomposes into L and Q

LSQ(LSQ(<M#>,<M#>))

matrices [[B]], [[A]]

NOTE: X is the least squares solution of A*X=B

LU(LU(<M#>))

matrix [[M]]

NOTE: Result list of three matrices L, U, and P

MAKEMAT(MAKEMAT(<expr>,<n1#>,<n2#>))

expression in I and J, n1 (number of rows), n2 (number of columns)

NOTE: Matrix M(I,J) where the (I,J) entry is the value of the expression

QR(QR(<M#>))

matrix [[M]]

NOTE: Result list of three matrices Q, R, and P

RANK(RANK(<M#>))

matrix [[M]]

NOTE: Computed rank of M

ROWNORM(ROWNORM(<M#>))

matrix [[M]]

NOTE: Row-norm of M

RREF(RREF(<M#>))

matrix [[M]]

NOTE: Reduced row-echelon form of M

1. Math Functions
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| SCHUR    | \( \text{SCHUR}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \{ [[Q]] [[U]] \} \)  
  \( \text{NOTE: result is list of two matrices where } Q^*U^*Q^H = M \) |
| SIZE     | \( \text{SIZE}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \{n1, n2\} \)  
  \( \text{(size of the matrix } M) \) |
| SPECNORM | \( \text{SPECNORM}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \text{spectral norm of } M \) |
| SPECRAD  | \( \text{SPECRAD}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \text{spectral radius of } M \) |
| SVD      | \( \text{SVD}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \{ [[U]] [[V]] [S] \} \)  
  \( \text{NOTE: result is a list of two matrices and a vector where } U^H * M * V^H = \text{diag}(S) \text{ and } U * \text{diag}(S) * V = M \) |
| SVL      | \( \text{SVL}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \text{vector } [S] \)  
  \( \text{NOTE: Result is the vector } [S] \text{ described in SVD as above} \) |
| TRACE    | \( \text{TRACE}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \text{trace of } M \) |
| TRN      | \( \text{TRN}(\langle M\rangle) \)  
  \( \text{matrix } [[M]] \quad \text{--------} \quad \text{matrix transpose of } M \) |
### Polynom

**POLYCOEF**

POLYCOEF(<V#>)

\[ [r_N,...,r_1] \quad \rightarrow \quad [a_N,...,a_1,a_0] \]

*vector of roots* \hspace{1cm} *vector of coefficients*

**NOTE:** The r's are the roots of the polynomial \( a_N x^N + a_{N-1} x^{N-1} + \ldots + a_1 x + a_0 \)

**POLYEVAL**

POLYEVAL(<V#>,<#>)

\[ [a_N,...,a_1,a_0], \, r \quad \rightarrow \quad a_N r^N + \ldots + a_1 r + a_0 \]

*vector of coefficients, real number*

**NOTE:** The r's are the roots of the polynomial \( a_N x^N + a_{N-1} x^{N-1} + \ldots + a_1 x + a_0 \)

**POLYFORM**

POLYFORM(<expr>,<name>,<name>...)

\[ expression, \, x, \, y, \ldots \quad \rightarrow \quad expression \]

**NOTE:** Result is a polynomial in x whose coefficients are polynomials in y whose coefficients are...

**EXAMPLE:** POLYFORM(X^2*Y^2+X^2*Y,X)

results in the polynomial \( (Y^2 + Y) \cdot X^2 \)

**POLYROOT**

POLYROOT(<V#>)

\[ [a_N,...,a_0] \quad \rightarrow \quad [r_N,...,r_1] \]

*vector of coefficients* \hspace{1cm} *vector of roots*

**NOTE:** The r's are the roots of the polynomial \( a_N x^N + a_{N-1} x^{N-1} + \ldots + a_1 x + a_0 \)

### Prob

**COMB**

COMB(<N#>,<N#>)

\[ integers \, n, \, k \quad \rightarrow \quad \binom{n}{k} \]

\[ \frac{n!}{k!(n-k)!} \]

\[ \frac{(n-1)\cdots[k+1]}{1\cdot2\cdots k} \]

**PERM**

PERM(<N#>,<N#>)

\[ integers \, n, \, k \quad \rightarrow \quad n! \]

\[ \frac{n!}{k!(n-k)!} \]

**RAND**

RAND

\[ no \, arguments \quad \rightarrow \quad random \, \# \, between \, 0 \, and \, 1 \]
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTPC</td>
<td>upper-tail chi-square distribution. Of order ( N ) evaluated at ( X )</td>
</tr>
<tr>
<td>UTPF</td>
<td>upper-tail F-distribution of real ( X ) order ( N1, N2 ), evaluated at ( X )</td>
</tr>
<tr>
<td>UTPN</td>
<td>upper-tail normal distribution. Of mean ( m ), variance ( v ) evaluated at ( X )</td>
</tr>
<tr>
<td>UTPT</td>
<td>upper-tail Student's T-distribution. ( N ) of order ( N ) evaluated at ( X )</td>
</tr>
<tr>
<td>CEILING</td>
<td>least integer ( \geq x )</td>
</tr>
<tr>
<td>DEG→RAD</td>
<td>convert ( x ) degrees to corresponding radians</td>
</tr>
<tr>
<td>FLOOR</td>
<td>greatest integer ( \leq x )</td>
</tr>
<tr>
<td>FNROOT</td>
<td>( r ) such that ( \text{expr},(x=r) = 0 ). Search is started near ( x0 ).</td>
</tr>
<tr>
<td>FRAC</td>
<td>fractional part of ( x )</td>
</tr>
<tr>
<td>HMS→</td>
<td>convert hours/minutes/seconds format to decimal hours</td>
</tr>
<tr>
<td>→HMS</td>
<td>converts decimal hours to hours/minutes/seconds format</td>
</tr>
</tbody>
</table>
INT

\text{INT}(\langle R\# \rangle)
\text{real number } x \quad \longrightarrow \quad \text{integer part of } x

\text{EXAMPLE: } \text{INT}(23.438) \text{ returns 23 as result.}

MANT

\text{MANT}(\langle R\# \rangle)
\text{real number } x \quad \longrightarrow \quad \text{mantissa of } x

MAX

\text{MAX}(\langle R\# \rangle,\langle R\# \rangle)
\text{real numbers } x,y \quad \longrightarrow \quad \text{maximum of } x,y

MIN

\text{MIN}(\langle R\# \rangle,\langle R\# \rangle)
\text{real number } x,y \quad \longrightarrow \quad \text{minimum of } x,y

MOD

\langle N\# \rangle \text{ MOD } \langle N\# \rangle
\text{integers } m, n \quad \longrightarrow \quad \text{integer remainder when } m \text{ is divided by } n

\text{EXAMPLE: } 27 \text{ MOD } 4 \text{ returns a result of 3}

\%

\%\langle R\# \rangle,\langle R\# \rangle
\text{real numbers } x,p \quad \longrightarrow \quad x*p*0.01

\%\text{CHANGE}

\%\text{CHANGE}(\langle R\# \rangle,\langle R\# \rangle)
\text{real numbers } x,y \quad \longrightarrow \quad 100*(y-x)/x

\%\text{TOTAL}

\%\text{TOTAL}(\langle R\# \rangle,\langle R\# \rangle)
\text{real numbers } x,y \quad \longrightarrow \quad 100*y/x

\text{RAD->DEG}

\text{RAD->DEG}(\langle R\# \rangle)
\text{real number } x \quad \longrightarrow \quad \text{convert } x \text{ radians to degrees}

\text{ROUND}

\text{ROUND}(\langle R\# \rangle,\langle N\# \rangle)
\text{real } x,\text{integer } n \quad \longrightarrow \quad \text{round } x \text{ to } n \text{ digits}

\text{SIGN}

\text{SIGN}(\langle \# \rangle)
\text{real or complex } x \quad \longrightarrow \quad x/\text{ABS}(x) \text{ if } x \neq 0 \text{ if } x = 0

\text{TRUNCATE}

\text{TRUNCATE}(\langle R\# \rangle,\langle N\# \rangle)
\text{real } x,\text{integer } n \quad \longrightarrow \quad \text{truncate } x \text{ to } n \text{ digits}

\text{XPON}

\text{XPON}(\langle R\# \rangle)
\text{real number } x \quad \longrightarrow \quad \text{exponent of } x \text{ when written in scientific notation}

1. Math Functions
Stat-Two

PREDX

\[ \text{PREDX}(\text{name}, y) \]

\(--->\)

\textit{predicted value for the "independent" variable of the indicated statistical dataset, given the "dependent" value}

PREDY

\[ \text{PREDY}(\text{name}, x) \]

\(--->\)

\textit{predicted value for the "dependent" variable of the indicated statistical dataset, given the "independent" value}

Symbolic

\[ = \]

This is an equational operator used to define equations like \( A + C = C/2 \), but it is not a predicate (like \( = \) in Test).

ISOLATE

\[ \text{ISOLATE}(\text{expr}, \text{name}) \]

\textit{expression or equation, name of a specific variable}

\(--->\)

\textit{symbolic expression for specific variable in terms of the others}

\textbf{NOTE:} An expression is interpreted as an equation with the expression on one side and 0 on the other side of the \( = \).

LINEAR?

\[ \text{LINEAR?}(\text{name}) \]

\textit{expression name} \(--->\) \textit{flag indicating whether the expression is linear}

QUAD

\[ \text{QUAD}(\text{expr}, \text{name}) \]

\textit{quadratic expression or equation, name of a specific variable}

\(--->\)

\textit{symbolic expression for the two complex roots}

\textbf{NOTE:} An expression is interpreted as an equation with the expression on one side and 0 on the other side of the \( = \).

QUOTE

\[ \text{QUOTE}(\text{expr}) \]

\textit{expression} \(--->\) \textit{suppresses expression eval.}

\textbf{NOTE:} Single quote marks also work as in '\text{expr}'

(WHERE)

\[ (\text{expr} | (\text{name} = \#, \text{name} = \#, \ldots) "substitution" \text{expr} , x, p1, y, p2,... \)--->\]

\textit{expression evaluated where x=p1, y=p2, etc.}
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Expression</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>a, b</code></td>
<td>1 if <code>a &lt; b</code>, 0 otherwise</td>
</tr>
<tr>
<td><code>≤</code></td>
<td>less than or equal to</td>
<td><code>a, b</code></td>
<td>1 if <code>a ≤ b</code>, 0 otherwise</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal</td>
<td><code>a, b</code></td>
<td>1 if <code>a = b</code>, 0 otherwise</td>
</tr>
<tr>
<td><code>≠</code></td>
<td>not equal to each other</td>
<td><code>a, b</code></td>
<td>1 if <code>a ≠ b</code>, 0 otherwise</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>a, b</code></td>
<td>1 if <code>a &gt; b</code>, 0 otherwise</td>
</tr>
<tr>
<td><code>≥</code></td>
<td>greater than or equal to</td>
<td><code>a, b</code></td>
<td>1 if <code>a ≥ b</code>, 0 otherwise</td>
</tr>
<tr>
<td>AND</td>
<td>logical AND</td>
<td><code>a, b</code></td>
<td>1 if both <code>a</code> and <code>b</code> ≠ 0, 0 otherwise</td>
</tr>
<tr>
<td>IFTE</td>
<td>IFTE(&lt;predicate&gt;, &lt;true-clause&gt;, &lt;false-clause&gt;)</td>
<td><code>a, expression1, expression2</code></td>
<td>expression1 if <code>a ≠ 0</code>, expression2 otherwise</td>
</tr>
<tr>
<td>NOT</td>
<td>logical NOT</td>
<td><code>a</code></td>
<td>1 if <code>a = 0</code>, 0 otherwise</td>
</tr>
<tr>
<td>OR</td>
<td>logical OR</td>
<td><code>a, b</code></td>
<td>1 if <code>a</code> or <code>b</code> are non-zero, 0 otherwise</td>
</tr>
<tr>
<td>XOR</td>
<td>exclusive OR</td>
<td><code>a, b</code></td>
<td>1 if <code>a ≠ 0</code> or <code>b ≠ 0</code>, but not both, 0 otherwise</td>
</tr>
</tbody>
</table>

1. Math Functions
<table>
<thead>
<tr>
<th>Function</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOT</td>
<td>ACOT( &lt;#&gt; )</td>
<td>inverse cotangent of x</td>
</tr>
<tr>
<td>ACSC</td>
<td>ACSC( &lt;#&gt; )</td>
<td>inverse cosecant of x</td>
</tr>
<tr>
<td>ASEC</td>
<td>ASEC( &lt;#&gt; )</td>
<td>inverse secant of x</td>
</tr>
<tr>
<td>COT</td>
<td>COT( &lt;#&gt; )</td>
<td>cotangent of x</td>
</tr>
<tr>
<td>CSC</td>
<td>CSC( &lt;#&gt; )</td>
<td>cosecant of x</td>
</tr>
<tr>
<td>SEC</td>
<td>SEC( &lt;#&gt; )</td>
<td>secant of x</td>
</tr>
</tbody>
</table>

**Functions which are not in the MATH menu and not on the keyboard:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG</td>
<td>NEG( &lt;# or V# or M# or {} or expr or grob&gt; )</td>
<td>NOTE: same functionality as -x key.</td>
</tr>
</tbody>
</table>
2. PROGRAM COMMANDS

Overview of the PROGRAM COMMANDS Menu

<table>
<thead>
<tr>
<th>Aplet</th>
<th>Branch</th>
<th>Drawing</th>
<th>Graphic</th>
<th>Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>IF</td>
<td>ARC</td>
<td>DISPLAY</td>
<td>FOR</td>
</tr>
<tr>
<td>SELECT</td>
<td>THEN</td>
<td>BOX</td>
<td>DISPLAY</td>
<td>=</td>
</tr>
<tr>
<td>SETVIEWS</td>
<td>ELSE</td>
<td>ERASE</td>
<td>GROB</td>
<td>TO</td>
</tr>
<tr>
<td>UNCHECK</td>
<td>END</td>
<td>FREEZE</td>
<td>GROBNOT</td>
<td>STEP</td>
</tr>
<tr>
<td>CASE</td>
<td></td>
<td>LINE</td>
<td>GROBOR</td>
<td>END</td>
</tr>
<tr>
<td>IFERR</td>
<td>PIXOFF</td>
<td>GROBXOR</td>
<td>DO</td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td>PIXON</td>
<td>MAKEGROB</td>
<td>UNTIL</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>TLINE</td>
<td>PLOT</td>
<td>END</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WHILE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REPLACE</td>
<td>REPEAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUB</td>
<td>END</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ZEROGROB</td>
<td>BREAK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Print</th>
<th>Prompt</th>
<th>Stat-One</th>
<th>Stat-Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDCOL</td>
<td>PRDISPLAY</td>
<td>BEEP</td>
<td>DO1VSTATS</td>
<td>DO2VSTATS</td>
</tr>
<tr>
<td>ADDROW</td>
<td>PRHISTORY</td>
<td>CHOOSE</td>
<td>RANDSEED</td>
<td>SETDEPEND</td>
</tr>
<tr>
<td>DELCOL</td>
<td>PRVAR</td>
<td>DISP</td>
<td>SETSAMPLE</td>
<td>SETINDEP</td>
</tr>
<tr>
<td>DELROW</td>
<td>DISPTIME</td>
<td>SETSAMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDITMAT</td>
<td></td>
<td>EDITMAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RANDMAT</td>
<td>FREEZE</td>
<td>GETKEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REDIM</td>
<td>INPUT</td>
<td>MSGBOX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPLACE</td>
<td>SUB</td>
<td>WAIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCALE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCALEADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPCOL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPROW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not In Any Menu

DEMO
ON-1
ON-PLOT
LIBEVAL (not documented in manual)
PINIT (not documented in manual)
SYSEVAL (not documented in manual)
VERSION (not documented in manual)
WSLOG (not documented in manual)
Syntax Guide to PROGRAM COMMANDS

Aplet These commands control Aplets.

CHECK CHECK <N#>
integer n ------- > checks the corresponding function in current aplet

EXAMPLE: CHECK 3 means check F3 if the current aplet is Function.

SELECT SELECT <ApLetname>

UNCHOOSE UNCHECK <N#>
integer n ------- > unchecks the corresponding function in current Aplet

EXAMPLE: UNCHECK 3 means uncheck F3 if current ApLet is Function.

SETVIEWS SETVIEWS <prompt_1>; <program_name_1>; <view# 1>; <prompt_2>; <program_name_2>; <view# 2>; ...
<prompt_n>; <program_name_n>; <view# n>

Usage of the SETVIEWS command:

Each triple prompt/program/view defines one line of the menu as follows:

<prompt> is a string to display in the menu.
<program_name> is name of a program to run if this line is selected.
<View#> is number of view to start after the program finishes running.

The prompt can be specified in two ways besides an explicit prompt string <prompt>. First, if <prompt> is an empty string (e.g., nothing appears between two semicolons), the program name is used as the prompt. Second, if both <prompt> and <program> are empty strings, and <view> is one of the default `special views', the normal prompt for that view (e.g., `Auto Scale') is used.

The program can prompt for information and display information, and it can modify all sorts of variables, and it can select another aplet, but it can't directly start a view. That happens only indirectly when the program is...
finished. If no program is desired, specify an empty string. The views are numbered as follows:

1. PLOT  2. SYMB  3. NUM
4. PLOT setup  5. SYMB setup  6. NUM setup
7. VIEWS  8. NOTE  9. SKETCH

In addition, five catalogs are available as views numbered 10 - 14, and if the ApLet has N default `special views', they are numbered views 15 through N +14.)

10. LIB  11. LIST  12. MATRIX
13. NOTE PAD  14. PROGRAM  15-? SPECIAL

If the prompt specified for a view is exactly the string `Reset', or 'Start', the associated program is considered the ApLet's special reset or start mechanism. Pressing the RESET or START softkey in the LIB catalog will run the special reset or start program.

It can be useful to associate a program with an ApLet even though the program isn't used directly in the menu of custom views. For example, the program may be a subroutine called by other programs. To accommodate this case, if <prompt> is exactly one space (" "), this triple is not presented in the menu.

Branch These commands work in various combinations to make one or more tests and to execute one clause of several. (See PART 4 for examples of branching structures)

IF
THEN
ELSE
END

CASE

IFERR

RUN       RUN <progname>
program name       ---------> runs the specified program

STOP       STOP
no arguments       ---------> stops the current program
### Drawing
These commands act on the display.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC</td>
<td>Draws an arc with center ((x, y)), radius (r), start (s), and end (e) on the display.</td>
</tr>
<tr>
<td>BOX</td>
<td>Draws a box with opposite corners ((x_1, y_1)) and ((x_2, y_2)) in the display.</td>
</tr>
<tr>
<td>ERASE</td>
<td>Causes the display to be erased.</td>
</tr>
<tr>
<td>FREEZE</td>
<td>Causes display to not be updated.</td>
</tr>
<tr>
<td>NOTE: This can be used to &quot;freeze&quot; the display the way it was when the program stopped running.</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td>Draws a line from ((x_1, y_1)) to ((x_2, y_2)) in the display.</td>
</tr>
<tr>
<td>PIXOFF</td>
<td>Turns off pixel with indicated (x, y) coordinates in the display.</td>
</tr>
<tr>
<td>PIXON</td>
<td>Turns on pixel with indicated (x, y) coordinates in the display.</td>
</tr>
<tr>
<td>TLINE</td>
<td>Toggles a line from ((x_1, y_1)) to ((x_2, y_2)) in the display.</td>
</tr>
</tbody>
</table>
These program commands use grob-variables (G0,...,G9) as arguments.

**DISPLAY** → DISPLAY → <name>

name

stores display as a grob in name

**DISPLAY** → DISPLAY <grob>

grob

puts grob in the display

**GROB** → GROB <name>;<font#> <object>

name,f, obj

creates a display grob using font number f and stores the result in name.

NOTE: The text that is converted into a grob is specified by object, which can be either a grob-variable-name or a grob-expression. (A grob-expression is an expression which returns a grob.)

**GROBNOT** → GROBNOT <name>

name

replaces the grob in name with the "inverted" grob from name

**GROBOR** → GROBOR <name1>;<pos>;<name2>

name1,position, name2

replaces the grob in name1 with the bitwise OR with the grob in name2 starting at pos. (name2 can also be a grob-expression)

**GROBXOR** → GROBXOR <name1>;<pos>;<name2>

name1, position, name2

replaces the grob in name1 with the bitwise XOR with the grob in name2 starting at pos (name2 can also be a grob-expression)

**MAKEGROB** → MAKEGROB <name>;<width> <height> <hex-data>

name, width, height, hex-data

creates a GROB with the given width height and hex-data and stores it in name

**PLOT** → PLOT → <name>

name

stores the Plot view display as a grob in name

2. Program Commands
Graphic

\[ \rightarrow \text{PLOT} \]

\[ \rightarrow \text{PLOT} \ < \text{name} > \]

\[ \text{name} \quad \rightarrow \quad \text{puts grob from name into the} \]
\[ \text{Plot view display.} \]

**NOTE:** Plot view is erased first, and grob is placed in upper left corner of Plot view display.

\[ \text{ZEROGROB} \]

\[ \text{ZEROGROB} \ < \text{name} > ; < \text{width} > ; < \text{height} > \]

\[ \text{name of grob variable, width, height} \quad \rightarrow \quad \text{creates a blank GROB with given width and height and stores it in name} \]

**The following two commands also apply to matrices and lists.**

\[ \text{SUB} \]

\[ \text{SUB} \ < \text{name} > ; < \text{object} > ; < \text{start} > < \text{end} > \]

\[ \text{name, obj, start, end} \quad \rightarrow \quad \text{stores the indicated subobject in name} \]

**NOTE:** SUB takes matrix, list, or grob arguments. The object can be a matrix-variable-name, a list-variable-name, a grob-variable-name, a matrix-expression, a list-expression, or a grob-expression.

\[ \text{REPLACE} \]

\[ \text{REPLACE} \ < \text{name} > ; < \text{start} > ; < \text{object} > \]

\[ \text{name, start, object} \quad \rightarrow \quad \text{replaces the object in} \]
\[ < \text{name} > \text{with} < \text{object} > \text{starting at} < \text{start} > \]

**NOTE:** REPLACE takes matrix, list, or grob arguments. The object can be a matrix-variable-name, a list-variable-name, a grob-variable-name, a matrix-expression, a list-expression, or a grob-expression.

**Loop**

These commands are used in combinations to conditionally repeat a clause several times. (See **PART 4** for examples of looping structures.)

\[ \text{FOR} \]

\[ \text{FOR} \ ... \ = \ ... \ \text{TO} \ ... \ \text{STEP} \ ... \ \text{END} \]

\[ \text{DO} \]

\[ \text{DO} \ ... \ \text{UNTIL} \ ... \ \text{END} \]

\[ \text{WHILE} \]

\[ \text{WHILE} \ ... \ \text{REPEAT} \ ... \ \text{END} \]

\[ \text{BREAK} \]
Matrix

These commands store their results in specified matrix variables.

ADDCOL          ADDCOL <name>;<V#>;,<N#>
matrix name, vector of column data, column position n
-------->  inserts the vector of column data as column n of the matrix stored in name

ADDR0W          ADDROW <name>;(<V#>;,<N#>
matrix name, vector of row data, column position n
-------->  inserts the vector of row data as column n of the matrix stored in name

DELCOL          DELCOL <name>;,<N#>
matrix name, position n
-------->  deletes column n of the matrix stored in name

DELROW          DELROW <name>;,<N#>
name, position n  -------->  deletes row n of the matrix stored in name

EDITMAT         EDITMAT <name>
matrix name  -------->  fires up the matrix editor on the specified matrix

NOTE: Returns to program execution after you press OK

RANDMAT         RANDMAT <name>;,<row#>;,<col#>
matrix name, number of rows n, number of columns m
-------->  creates a random matrix with n rows, m columns, and stores the result in <name>.

REDIM           REDIM <name>;,<size>
redimensions the object in <name> to <size>

REPLACE         REPLACE <name>;,<start>;,<object>
NOTE: See under Graphic for explanation.

SCALE           SCALE <name>;,<factor>;,<row#>
matrix name, scale factor s, row number n
-------->  multiplies row n of the matrix stored in name by factor s
Matrix  Continued

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCALEADD</td>
<td>SCALEADD &lt;name&gt;;&lt;factor&gt;;&lt;row#1&gt;;&lt;row#2&gt;</td>
</tr>
<tr>
<td></td>
<td>matrix name, scale factor s, row numbers n1,n2</td>
</tr>
<tr>
<td></td>
<td>--------&gt;</td>
</tr>
<tr>
<td></td>
<td>adds the product of s and row n1 of the matrix stored in name to row n2</td>
</tr>
<tr>
<td>SUB</td>
<td>SUB &lt;name&gt;;&lt;object&gt;;&lt;start&gt; &lt;end&gt;</td>
</tr>
<tr>
<td></td>
<td>NOTE: see under Graphic for explanation.</td>
</tr>
<tr>
<td>SWAPCOL</td>
<td>SWAPCOL &lt;name&gt;;&lt;pos1&gt;;&lt;pos2&gt;</td>
</tr>
<tr>
<td></td>
<td>matrix name, column position n1, column position n2</td>
</tr>
<tr>
<td></td>
<td>--------&gt;</td>
</tr>
<tr>
<td></td>
<td>swaps columns n1 and n2 of the matrix stored in name</td>
</tr>
<tr>
<td>SWAPROW</td>
<td>SWAPROW &lt;name&gt;;&lt;pos1&gt;;&lt;pos2&gt;</td>
</tr>
<tr>
<td></td>
<td>matrix name, row position n1, row position n2</td>
</tr>
<tr>
<td></td>
<td>--------&gt;</td>
</tr>
<tr>
<td></td>
<td>swaps rows n1 and n2 of the matrix stored in name</td>
</tr>
</tbody>
</table>

Print

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRDISPLAY</td>
<td>PRDISPLAY</td>
</tr>
<tr>
<td></td>
<td>NOTE: Prints the current display on the IR printer</td>
</tr>
<tr>
<td>PRHISTORY</td>
<td>PRHISTORY</td>
</tr>
<tr>
<td></td>
<td>NOTE: Prints the history &quot;stack&quot; of results from the HOME screen on the IR printer</td>
</tr>
<tr>
<td>PRVAR</td>
<td>PRVAR &lt;name&gt;</td>
</tr>
<tr>
<td></td>
<td>NOTE: Prints a variable's name and contents on the IR printer</td>
</tr>
<tr>
<td>PRVAR</td>
<td>PRVAR &lt;name&gt;;PROG</td>
</tr>
<tr>
<td></td>
<td>NOTE: Prints program's name and contents on the IR printer</td>
</tr>
<tr>
<td>PRVAR</td>
<td>PRVAR &lt;name&gt;;NOTE</td>
</tr>
<tr>
<td></td>
<td>NOTE: Prints a note's name and contents on the IR printer</td>
</tr>
</tbody>
</table>

26  2. Program Commands
<table>
<thead>
<tr>
<th>Prompt</th>
<th>These commands ask for information or provide information to the user.</th>
</tr>
</thead>
</table>

**BEEP**

BEEP `<freq#>;<time#>`  

*frequency f (cycles per second), duration time s (seconds)*  

--------->  

emits a tone at frequency f for s seconds

**CHOOSE**

CHOOSE `<var-name>;<prompt>;<item-1>;...;<item-n>`  

*<var-name>* is the name of the variable from which the number of the initially-highlighted item will be gotten and into which the number of the chosen item will be stored;  

*<prompt>* is the top-of-box prompt or null, meaning no prompt;  

*<item-1>* through *<item-n>* are the items to be displayed.  

--------->  

CHOOSE displays a choose box and sets the specified variable to real number 0 through n corresponding to whether the choose box is cancelled (0) or an item is chosen (1 through n).

**DISP**

DISP `<N#>;<item>`  

*line n, text-item*  

--------->  

display on line n of the display text that was created from the text item `<item>`

**DISPTIME**

DISPTIME  

*no arguments*  

--------->  

Displays current date and time.

**EDITMAT**

EDITMAT `<name>`  

*matrix name*  

--------->  

fires up the matrix editor on the specified matrix; then returns to program execution with OK
FREEZE
no arguments

Causes display to not be updated.

NOTE: Freezes the display the way it was when the program stopped running.

GETKEY
GETKEY <name>

NOTE: Waits for a keystroke, then stores the corresponding keycode in the given name

INPUT
INPUT <name>;<title>;<label>;<help>;<default>

NOTE: 1) prompts the user with title, label, and help,
2) initializes a command line with default, and
3) saves the resulting input in name.

NOTE: <title>, <label>, and <help> are text-tems, and <default> is an expression and is evaluated. (See Text Item Conventions below.)

MSGBOX
MSGBOX <text-item>

NOTE: Displays a text-item in a message box.

WAIT
WAIT <time#>

time in seconds

waits s seconds

Text Item Conventions

A text item is either a quoted string of text characters (for example, "ABC") or an expression which will be evaluated and turned into a string of characters (for example, SIN(0) is equivalent to "0") or a sequence of these. The results of all of these will be concatenated together to form a single string of characters.

An explicit character string can be delimited by either double quotes ("ABC") or single quotes ("new" improved!). Sequences of explicit character strings will be concatenated ("Alice's " "new" " ApLet is here" ). (Note that sequences of expressions may be interpreted using implicit multiplication.)

All this allows you to display labels and values. For example, with the text item "A:" SIN(0), the displayed string would be "A: 0". Null arguments are allowed.
Stat-One  These commands are used for one-variable statistics.

DO1VSTATS  DO1VSTATS <datasetname>
<datasetname> may be H1, H2, ..., or H5
--------->
 calculates STATS using <datasetname> and stores results in the
following variables: NΣ, TOTΣ, MEANΣ, PVARΣ, SVARΣ, PSDEV,
SSDEV, MINΣ, Q1, MEDIAN, Q3, and MAXΣ.

RANDSEED  RANDSEED <R#>
seed value s
--------->
 sets random number seed to s

SETFREQ  SETFREQ <datasetname>; <definition>
<datasetname> may be H1, H2, ..., or H5
--------->
 defines <datasetname> frequency according to <definition>
expression.

SETSAMPLE  SETSAMPLE <datasetname>; <definition>
<datasetname> may be H1, H2, ..., or H5
--------->
 Defines <datasetname> sample according to <definition>
expression.

Stat-Two  These commands are used for two-variable statistics.

DO2VSTATS  DO2VSTATS <datasetname>
<datasetname> may be S1, S2, ..., or S5
--------->
 calculates STATS using <datasetname> and stores results in
 corresponding variables: MEANX, SX, ΣX^2, MEANY, SY, ΣY^2,
ΣX*Y, CORR, COV, and FIT.

SETDEPEND  SETDEPEND <datasetname>; <definition>
<datasetname> may be S1, S2, ..., or S5
--------->
 Defines <datasetname> dependent according to <definition>
expression.

SETINDEP  SETINDEP <datasetname>; <definition>
<datasetname> may be S1, S2, ..., or S5
--------->
 defines <datasetname> dependent according to <definition>
expression.
Program Commands Not Found In Any Menu

Documented In Manual

DEMO

NOTE: This displays the animated demonstration of some of the HP38G’s features.

ON-1

NOTE: Pressing ON and 1 keys at the same time causes the current display information to be sent out through the cable connection.

ON-PLOT

NOTE: Pressing ON and PLOT keys at the same time causes the current display to be stored as a grob in GO.

Not Documented In Manual

LIBEVAL LIBEVAL <library_number>; <routine_number>

NOTE: This executes the specified rompointer.

PINIT

NOTE: This is for port initialization.

SYSEVAL SYSEVAL <address>

NOTE: This executes system object at specified address.

VERSION

NOTE: This displays the version and HP copyright message in a message box.

WSLOG

NOTE: This displays the warmstart log.
3. PROGRAM CONSTANTS

Used for setting variables. All constants listed below (with the exception of StatMode constants) also appear in input forms.

The name in the MATH menu is always the same as the name that appears in the corresponding input form field. In some cases, the CHOOS-list that corresponds to the input form field will show longer names.

The value of each constant is the same as its position in the MATH Menu (which is also the same as its position in the CHOOS-list).

EXAMPLE: Degrees = 1, Radians = 2, and Grads = 3.

Overview of the PROGRAM CONSTANTS Menu

<table>
<thead>
<tr>
<th>Angle</th>
<th>Format</th>
<th>SeqPlot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees</td>
<td>Standard</td>
<td>Stairstep</td>
</tr>
<tr>
<td>Radians</td>
<td>Fixed</td>
<td>Cobweb</td>
</tr>
<tr>
<td>Grads</td>
<td>Sci</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eng</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fraction</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1...5fit*</th>
<th>StatMode</th>
<th>StatPlot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>Stat1Var</td>
<td>Hist</td>
</tr>
<tr>
<td>LogFit</td>
<td>Stat2Var</td>
<td>BoxW</td>
</tr>
<tr>
<td>ExpFit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuadFit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cubic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* constants in S1...5fit category are used to set S1fit,...,S5fit

CHOOS-list Versions of the Constant Names

<table>
<thead>
<tr>
<th>Angle</th>
<th>Format</th>
<th>SeqPlot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degrees</td>
<td>Standard</td>
<td>Stairstep</td>
</tr>
<tr>
<td>Radians</td>
<td>Fixed</td>
<td>Cobweb</td>
</tr>
<tr>
<td>Grads</td>
<td>Scientific</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Engineering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fraction</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1...5fit</th>
<th>StatPlot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>Histogram</td>
</tr>
<tr>
<td>Logarithmic</td>
<td>BoxWhisker</td>
</tr>
<tr>
<td>Exponential</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>Quadratic</td>
<td></td>
</tr>
<tr>
<td>Cubic</td>
<td></td>
</tr>
<tr>
<td>Logistic</td>
<td></td>
</tr>
<tr>
<td>User Defined</td>
<td></td>
</tr>
</tbody>
</table>
4. PROGRAM HINTS AND EXAMPLES

What is a program?

At its very simplest, a program is just a sequence of commands, each of which performs an action. On the HP38G, a colon (:) is used to separate commands from one another, while a semicolon (;) is used to separate the arguments to a single command.

It is possible to write a program directly on the HOME screen in the Editline.

For example,

```
1 STO> A: 2 STO> B: 3 STO> C
```

followed by ENTER will return a result of 3 in Ans (the last command was 3 STO> C), but all three STO> commands were executed.

The PROGRAM CATALOG

To save a program and give it a specific name, press PROGRAM, and you will see a title bar for the PROGRAM CATALOG. This contains a directory of all your programs (even including the Editline). The menu keys here are:

```
EDIT     NEW     SEND     RECV     RUN
```

To create a new program, press NEW. You are prompted for a name for the program. After you type in the name of your choice (it can be virtually anything you please and of any length), press ENTER or OK.

The PROGRAM EDITOR

Now you are in the editor for your specific program, and the title bar should indicate the name of the program. For example, if you named your program MINE, then the title bar should read MINE PROGRAM at the top of the screen. The menu keys now are:

```
STO>     SPACE     A...Z     BKSP
```

The A...Z key is an "alpha-lock" which is can be toggled on or off (when it is on, you will see a white square lit up on the menu label and the α symbol lit at the top of the screen). If you press the colored shift key before toggling on A...Z, it changes the key to a...z, a lower-case "alpha-lock".
Typing in your program

The flashing cursor arrow is an insertion marker. While typing a program you can use the directional arrows for moving from line to line or from character to character.

**SPACE** is on the menu for typing convenience

**BKSP** (backspace) is on the menu for typing convenience

**STO>** is the one command provided on the menu (performs the same action as the **STO>** key in HOME).

**DEL** deletes the character under the insertion arrow (or the last character if you are at the very end)

**ENTER** is a line feed

Numbers, letters, and other characters can be typed from the keyboard as usual. You will find yourself making frequent use of the **CHARS** menu (for retrieving relational symbols like <, >, ≤, ≥ and the ' or " symbols). To use **CHARS**, use the directional arrows to highlight the character you want, and then press **OK**. If there are several characters in a row that you want to retrieve from the **CHARS** menu, press **ECHO** first and then each of the characters you want (press **OK** when you are done). Functions and variable names can either be typed in character-by-character or retrieved from the **VAR** or the **MATH** menus.

All other programming commands (other than **STO>**) are found in the **MATH** menu when **CMDS** is toggled on. An overview and the syntax for all these commands are found in PART 2 of this reference.

When you are finished...

Once you are finished typing in a program, there is no special action necessary to "EXIT". Simply go somewhere else! For example, you could press **HOME** to return to the **HOME** screen, or you could press **PROGRAM** to create a new program or **RUN** the program you have just written.

If you highlight the name of a program in the PROGRAM CATALOG, you can **EDIT** it (this returns you to the program editor where you can make changes). **SEND** and **RECV** are used to send and receive programs through infra-red communications with another HP38G, or by serial wire connection to a disk drive.

Cutting and pasting the contents of one program into another

Sometimes you may find that you have several lines of program code that you want to use in another program (perhaps with some slight editing). Rather than type the code in again "from scratch", there is a neat way to cut and paste one program into another:
1. Position the flashing cursor at the location in the current program where you want to paste in the contents of another program
2. Press VAR
3. Highlight the Program category and press OK
4. Highlight the name of the program whose contents you want to paste
5. Toggle on the VALUE menu key (instead of NAME)
6. Press OK

Program structures

Programs that are simply sequences of commands can be very useful, but the real power of programs becomes evident when they allow for conditional execution of commands (branching) or repeated execution of several commands over and over again (looping). You will see that the HP38G’s programming language does not use line numbers or labels and GO TO statements to accomplish branching and looping.

HP38G Branching structures

You can either type these commands in character-by-character or retrieve them from the Branch category of the CMDS menu in MATH.

IF ... THEN ... ELSE ... END

The syntax for this structure is as follows:

IF test clause THEN
    do some thing:
    do some other thing: These statements are executed if the test clause is true and so on:
ELSE
    do some thing:
    do some other thing: These statements are executed if the test clause is false and so on:
END: Marks the end of the IF THEN ELSE structure

NOTES:
1. In the statements that follow either THEN or ELSE, it’s possible to have loops or additional branch structures, including other IF THEN ELSE END structures.
2. The test clause can actually be any expression, and could have a value other than 1 or 0. The THEN part is executed if the test clause has a nonzero value, while the ELSE part is executed only if the test clause has the value zero.
Example:

If \( A < 0 \)
Then
\(-A \) STO> A:
(If \( A \) is negative, then the sign of \( A \) is reversed. Note that this piece of code has no ELSE part.)
End:

Example: This program records the sign of \( A \) in \( S \)

If \( A < 0 \)
Then
\(-1 \) STO> S: \( S = -1 \) if \( A \) is negative
Else
If \( A = 0 \)
Then
0 STO> S: \( S = 0 \) if \( A = 0 \)
Else
1 STO> S: \( S = 1 \) if \( A \) is positive
End:
End:

CASE

This structure is handy if you have several clauses to test in order, and you want to branch when you encounter the first true clause. The syntax for CASE is:

CASE
If first test clause
Then
do some thing:
do some other thing:
and so on:
End
If second test clause
Then
do some thing:
do some other thing:
and so on:
End
.
.
.
End:

4. Program Hints and Examples
Here are two examples to illustrate the distinction between the use of CASE and simply a sequence of several IF THEN statements.

**Example 1:**
```
CASE
IF I > 5
THEN I-5 STO> S:
END
IF I ≥ 5
THEN 0 STO> S:
END
IF I < 5
THEN 5 - I STO> S:
END
END:
```

**Example 2:**
```
CASE
IF I > 5
THEN I-5 STO> S:
END
IF I ≥ 5
THEN 0 STO> S:
END
IF I < 5
THEN 5 - I STO> S:
END
END:
```

Suppose we store 7 in I (7 STO> I on the HOME screen), and then run the first program involving CASE. The result will be that 2 is stored in S, because the clause in the second IF THEN statement won't even be tested.

In contrast, if the second program is run, the final result will be that 0 is stored in S, for all three of the clauses in the IF THEN statements will be tested.

*NOTE the punctuation differences between these two examples, particularly the use of colons.*

**IFERR ** **THEN ** **ELSE**

This structure is one designed for "error trapping." Some program statements may result in an error (thus causing the program to stop) depending on the values of certain variables at the time of execution. IFERR will check any number of statements for such errors and give you the chance to gracefully branch.
The syntax for IFERR is:

```
IFERR
first command:
second command:
et cetera:
THEN
 do some thing:
do some other thing:
and so on:
ELSE
 do some thing else:
do some other thing
else:
and so on:
```

If an error resulted from any of these commands
  then these statements are executed

Optional—if no error resulted
  then these statements are executed instead

Example:

```
IFERR
ABS(COT(X)) STO> C:
THEN
  1 STO> A:
  MAXREAL STO> C:
ELSE
  0 STO> A:
END:
```

If this statement results in an error
  (for example, when X = 0)
  then store 1 in A
  and store the largest real number in C
If no error resulted,
  then store 0 in A instead

For example, if X had the value 0 at execution, the attempt to store ABS(COT(X))
as the value of C would result in an error.

Rather than stopping, this program branches to store the largest real number that
can be represented on the HP38G as the value of C, and 1 is stored in A (as a
marker that an error occurred, perhaps).

If the X has a value that does not result in an error, then ABS(COT(X)) is stored in
C and 0 is stored in A.
RUN

You can also branch to other programs using the RUN command. The syntax is:

RUN name: where name is the name of the program.

**HP38G Looping Structures**

You can either type these commands in character-by-character or retrieve them from the Loop category of the CMDS menu in MATH.

**FOR = TO ... STEP ... END**

The syntax for this basic loop structure is:

FOR <index variable> = <start value> TO <end value> STEP <increment value>;
do some thing:
do some other thing:
and so on:
END:

*Example*:

0 STO> S:
1 STO> P:
FOR I = 1 TO 10 STEP 1;
S+I STO> S:
P*I STO> P:
END:

The result of this program is that the sum of the integers 1 through 10 is stored in S, and their product is stored in P.
DO ... UNTIL ... END

This loop structure will repeatedly execute a body of statements until some test clause is true. The syntax is:

DO
some thing:
some other thing:
and so on:
UNTIL <test clause>
END:

Example:

0 STO> S:
1 STO> P:
1 STO> I:
DO
S+I STO> S:
P*I STO> P:
I+1 STO> I:
UNTIL I > 10
END:

The result of this program is the same as the previous one: the sum of the integers 1 through 10 is stored in S, and their product is stored in P.

WHILE ... REPEAT ... END

This structure repeats the execution of some body of statements while a test clause remains true. The syntax is:

WHILE <test clause>
REPEAT
do some thing:
do some other thing:
and so on:
END:
Example:

0 STO> S:
1 STO> P:
1 STO> I:
WHILE I ≤ 10
REPEAT
S+I STO> S:
P*I STO> P:
I+1 STO> I:
END:

The result of this program is the same as the previous two: the sum of the integers 1 through 10 is stored in S, and their product is stored in P.

Example:

This example assumes that a positive integer has been stored in M. Then it sets N=M and calculates 3N+1 if N is odd or N/2 if N is even. This becomes the new value of N and the process is repeated until N=1. The value of S at the end of the program's execution is the number of different values of N. (S=0 if M is not a positive integer). NOTE: It is an open question whether S is finite for all positive integers M.

IF (M ≤ 0) OR (M ≠ INT(M))
THEN 0 STO> S:
ELSE
1 STO> S:
M STO> N:
WHILE N ≠ 1
REPEAT
IF N MOD 2
THEN
3*N+1 STO> N:
S+1 STO> S:
ELSE
N/2 STO> N:
S+1 STO> S:
END:
END:
EXAMPLE PROGRAMS

Probably the best way to learn how to program the HP38G is to study some interesting working programs that actually use many of the basic programming constructions. We provide a few examples here along with line-by-line documentation.

Simply entering these programs will give you some practice in finding commands and familiarize you with the most important elements of program syntax. Some suggestions are made for ways that you could adapt some of the programs to perform other tasks.

HP38G EXAMPLE 1.

This program looks for perfect numbers (positive integers \( N \) with the property that \( N = S \), the sum of all positive integer factors < \( N \)). It tests all the positive integers from 2 to 1000, displaying each along with the sum of its divisors. When \( N = S \), the program displays that value with the label "PERFECT". Hence, you will see the last perfect number found by the program as it runs.

The program illustrates the use of WHILE REPEAT loops as well as IF THEN statements. It also illustrates the use of the DISP command for displaying text and values on specified lines of the display screen.

Program PERFECTN

```
ERASE:
2 STO> N:
WHILE N ≤ 1000
REPEAT
2 STO> F:
1 STO> S:
WHILE F ≤ \( \sqrt{N} \)
REPEAT
IF N MOD F = 0 THEN S+F+(N/F) STO> S:
END:
F+1 STO> F:
END:
IF N=S THEN
DISP 3; "PERFECT: "N:
END:
DISP 1;"N= ";N:
DISP 2;"S= ";S:
N+1 STO> N:
```

NOTE: Here are some possible adaptations you could make to this program:

a) Save the list of the perfect numbers found in LO.

b) Allow the user to input the value of \( N \) (the largest integer checked).

c) Search for prime numbers (or other integers with special properties).
HP38G EXAMPLE 2.

This program illustrates the use of PIXON to light specific pixels on the display screen, as well as the use of nested WHILE REPEAT loops. It assumes that your viewing window is the default window [-6.5, 6.5] by [-3.1, 3.2]. The program checks the coordinates of each pixel in the third quadrant and uses the integer part of the value of \(X^2 + Y^2 \mod 2\) (i.e., the remainder after division by 2) to decide whether to light up the pixels or not (1 for yes, 0 for no). It also lights up the other three symmetrically located pixels in the other three quadrants. It takes a while to run, but the result is a striking picture exhibiting some interesting diffraction patterns. (The picture is stored in the SKETCH view of the current ApLet.)

NOTE: Try replacing the expression \(X^2 + Y^2\) with another expression symmetric in \(X\) and \(Y\) to produce a more exotic picture.

Program Ripples

ERASE:

\[
\begin{align*}
& \text{Clear the display screen} \\
&(X_{\text{max}} - X_{\text{min}})/130 \text{ STO> } H: \\
&(Y_{\text{max}} - Y_{\text{min}})/63 \text{ STO> } K: \\
&X_{\text{min}} \text{ STO> } X: \\
&\text{OuterWHILE loop begins} \\
&\text{While } X \leq 0 \\
&\text{REPEAT} \\
&Y_{\text{min}} \text{ STO> } Y: \\
&\text{InnerWHILE loop begins} \\
&\text{While } Y \leq 0 \\
&\text{REPEAT} \\
&\text{If the integer part of } \frac{X^2 + Y^2}{2} \text{ is 1, then} \\
&P\text{IXON } X;Y: \\
&P\text{IXON } X;Y_{\text{max}}-(Y-Y_{\text{min}}): \\
&P\text{IXON } X_{\text{max}}-(X-X_{\text{min}});Y: \\
&P\text{IXON } X_{\text{max}}-(X-X_{\text{min}});Y_{\text{max}}-(Y-Y_{\text{min}}): \\
&\text{InnerWHILE loop ends} \\
&Y+K \text{ STO> } Y: \\
&\text{InnerWHILE loop ends} \\
&X+H \text{ STO> } X: \\
&\text{OuterWHILE loop ends} \\
&\text{Save finished picture in SKETCH} \\
\end{align*}
\]
This program "animates" a numerical limit by displaying a simultaneous readout of X and F(X) values. (In this case, the particular function used is SIN(X)/X.) Pay particular attention to the INPUT statements, for they provide a very simple way to give a professional look to your programs, complete with a title bar, a highlighted input form (with a default value supplied if you wish), and a help message to the user to prompt for the necessary input.

NOTE: This program can be adapted to investigate the limit of an arbitrary function by adding an input statement that allows the user to select a function in the SYMB menu. This program can also be used to investigate limits "at infinity" by using MAXREAL or -MAXREAL as the TARGET value A, and setting a suitably large increment H.

**Program LIMITS**

```plaintext
INPUT X;
"LIMIT ANIMATION";
"INITIAL X";
"ENTER STARTING X";
1:
INPUT A;
"LIMIT ANIMATION";
"TARGET";
"ENTER DESTINATION";
0:
INPUT H;
"LIMIT ANIMATION";
"DELTA X";
"ENTER INCREMENT";
.1:
ERASE:
IF (X-A)H>0
THEN -H STO> H:
END:
WHILE 1
REPEAT
DISP 1;"X 
IFERR SIN(X)/X STO> Z
THEN 0 STO> Z:
END:
DISP 3;
"SIN(X)/X "Z:
```

4. Program Hints and Examples
X STO> 0:
X+H STO> X:
IF (A·X)(A·0) ≤ 0
THEN
ERASE:
0 STO> X:
H/10 STO> H:
END:
END:

Store current value of X in 0
Increment X by H
If the new value of X is equal to A or on the opposite side as the previous value of X, then clear the display, restore the old value of X, and reduce increment by a factor of 10
HP38G EXAMPLE 4.

This is actually a "suite" of four programs (three are used as subroutines of the main program, called ARCHIMEDES). They provide an exploration of the Archimedean method for approximating the area of a circle. This program shows how the SETVIEWS command operates. Look at PLOT to see the graph of a regular polygon that can be inscribed in the unit circle. Look at VIEWS after running the program ARCHIMEDES for special interactive options to change the number of sides and to see the computed area of the regular polygon.

Program ARCHIMEDES

SELECT Polar:          Activate Polar ApLet
1 STO> Angle:          Set ApLet mode to degrees
1 STO> HAngle:         Set Home mode to degrees
1 STO> R1(0):          Set unit circle polar equation in R1
0 STO> Omin:           Set starting angle to 0
361 STO> Omax:         Set ending angle to 361
CHECK 1:               Check R1 in SYMB
3 STO> N:              Initialize number of sides to N=3
120 STO> Ostep:        Initialize angle step to 360/3 = 120
SETVIEWS               Set up special views
"DOUBLE # SIDES";       First special view allows user
to double number of sides
ARCHI.1:1;              Second special view allows user
to set number of views
"INPUT # SIDES";         Third special view will display
ARCHI.2:1;              area of the inscribed N-gon
"AREA A=?";            ARCHI.3:1

Program ARCHI.1

2*N STO> N:            Double size of N, the number of sides
360/N STO> Ostep:      Set the corresponding new angle step

Program ARCHI.2

INPUT N;               INPUT for N, number of sides
"ARCHIMEDES";          Title bar
"# SIDES";             Label for input blank
"ENTER NUMBER OF SIDES";N:  Help message to prompt user
360/N STO> Ostep:      Set the corresponding new angle step

Program ARCHI.3

ERASE:                 Erase the display
DISP 1;"A= " .5*N*SIN(360/N):  Compute the area of the polygon
FREEZE:                Freeze the display screen
4. Program Hints and Examples