HP 49G Advanced Users Guide

Volume 1

Part A:

Computer Algebra Commands

Introduction

This volume details the computer algebra operations that are available on the HP 49G.

For each operation, the following details are provided:

Type: Function or command. Functions can be used as a part of an algebraic objects and commands cannot. When working with functions or commands within Equation Writer:

- When you apply a function to an expression, the function appears as part of the expression. You need to ensure that the expression is selected, then press ↱ EVAL to apply the function to the selection.
- When you apply a command to an expression in Equation Writer, it is evaluated immediately.

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys.

Input: The input argument or arguments that the operation needs. If the operation uses more than one input argument, details of the arguments and the order in which you supply them are provided: argument order for algebraic mode and stack order for RPN mode.

Output: The output that the operation produces.

- In RPN mode, the outputs are placed on the stack.
- In algebraic mode, the outputs are written to a list.

As with the input arguments, the outputs for both algebraic and RPN mode are described.

Flags: Details of how flag settings affect the operation of the function or command.

Example: An example of the function or command.

See also: Related functions or commands.

Computer algebra command categories

Algebra commands

Arithmetic commands

Arithmetic Integer commands

Arithmetic Polynomial commands

Arithmetic Modulo commands

Calculus commands

Derivation and integration commands

Limits and series commands

Differential equations commands

Exp and Lin commands

Matrix-related commands

Create

Operations

Quadratic form

Linear systems

Eigenvector

Symbolic solve commands

Trigonometry commands

Alphabetical command list

The following pages contain the commands in alphabetical order. See "Computer algebra command categories" on page 3 to view the commands in the order that they appear on the menus.

ABCUV

Type: Command

Description: Returns a solution in polynomials u and v of $au+bv=c$ where a and b are polynomials, and c is a value.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 3/Argument 1: The polynomial corresponding to a.
Level 2/Argument 2: The polynomial corresponding to b.
Level 1/Argument 3: The value corresponding to c.

Output: Level 2/Item 1: The solution corresponding to u.
Level 1/Item 2: The solution corresponding to v.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find a solution in polynomials u and v for the following equation:

$(x^2 + x + 1)u + (x^2 + 4)v = 13$

Command:
```
ABCUV(X^2+X+1,X^2+4,13)
```

Result:
```
{-(X+3),X+4}
```

See also: IABCUV
EGCD

ACOS2S

Type: Command

Description: Transforms an expression by replacing acos(x) in subexpressions with $\pi/2$–asin(x).

Access: Trigonometry, ↱ TRIG

Input: The expression to transform.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Simplify the following expression:

$$arc\cos\left(\frac{2}{3}\right)+arc\cos(x)$$

Command: `ACOS2S(ACOS(2/3)+ACOS(X))`

Result: `π/2-ASIN(2/3)+π/2-ASIN(X)`

See also: ASIN2C
ASIN2T
ATAN2S

ADDTMOD

Type: Function

Description: Adds two expressions or values, modulo the current modulus.

Access: Arithmetic, ↰ (ARITH) MODULO

Input: Level 2/Argument 1: The first expression.
Level 1/Argument 2: The second expression.

Output: The sum of the two expressions, modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Express the result of the following addition in modulo 7.
$(x^2+3x+6)+(9x+3)$
Note: Use the CAS modes input form to set the modulo to 7

Command: `ADDTMOD(X^2+3*X+6,9*X+3)`

Result: `X^2-2*X+2`

ASIN2C

Type: Command

Description: Transforms an expression by replacing asin(*x*) subexpressions with π/2–acos(*x*) subexpressions.

Access: Trigonometry, ↱ TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

ASIN2T

Type: Command

Description: Transforms an expression by replacing asin(*x*) subexpressions with the following:

$$\text{atan}\left(\frac{x}{\sqrt{1-x^2}}\right)$$

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

ATAN2S

Type: Command

Description: Transforms an expression by replacing atan(*x*) subexpressions with the following:

$$\text{asin}\left(\frac{x}{\sqrt{x^2+1}}\right)$$

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –flag –03 clear).

AXL

Type: Command

Description: Converts a list to an array, or an array to a list.

Access: Convert, ↰ CONVERT

Input: A list or an array.

Output: If the input is a list, returns the corresponding array. If the input is an array, returns the corresponding list.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Convert the following matrix to a list:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Command: `AXL([[0,1][1,0]])`

Result: `{{1,0},{0,1}}`

See also: AXM

AXM

Type: Command

Description: Converts a numeric array to a symbolic matrix.

Access: Matrices, ↰ MATRICES OPERATIONS

Input: An array.

Output: The corresponding matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: AXL
AXQ

AXQ

Type: Command

Description: Converts a square matrix into the associated quadratic form.

Access: Convert, ↰ CONVERT

Input: Level 2/Argument 1: An $n \times n$ matrix.
Level 1/Argument 2: A vector containing n variables.

Output: Level 2/Item 1: The corresponding quadratic form.
Level 1/Item 2: The vector containing the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the quadratic form, expressed in terms of x, y, and z, associated with the following matrix:

$$\begin{bmatrix} 3 & 6 & 0 \\ 2 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Command:
```
AXQ([[3,6,0][2,4,1][1,1,1]],[X,Y,Z])
```

Result:
```
{3*X^2+(8*Y+Z)*X+(4*Y^2+2*Z*Y+Z^2),[X,Y,Z]}
```

See also: GAUSS
QXA

CASCFG

Type: Command

Description: Restores the default CAS mode settings. This command is equivalent to pressing RESET when the CAS Modes input form is displayed.

Access: CAT CASCFG

CHINREM

Type: Command

Description: Solves a system of simultaneous polynomial congruences in the ring Z[*x*].

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: A vector of the first congruence (expression and modulus).
Level 1/Argument 2: A vector of the second congruence (expression and modulus).

Output: A vector of the solution congruence (expression and modulus).

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Solve the following simultaneous congruences for the polynomial *u*:

$u \equiv x^2 + 1 \pmod{x+2}$

$u \equiv x - 1 \pmod{x+3}$

Command: `CHINREM([X^2+1,X+2],[X-1,X+3])`

Result: `[X^3+2*X^2+5,-(X^2+5*X+6)]`

See also: ICHINREM

CURL

Type: Function

Description: Returns the curl of a three-dimensional vector function.

Access: Calculus, ↰ CALC DERIV AND INTEG

Input: Level 2/Argument 1: A three-dimensional vector function of three variables.
Level 1/Argument 2: An array comprising the three variables.

Output: The curl of the vector function with respect to the specified variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the curl of the following vector function:

$' = x^2y\underset{\sim}{i} + x^2y\underset{\sim}{j} + y^2z\hat{k}$

Command: `CURL([X^2*Y, X^2*Y, Y^2*Z],[X,Y,Z])`

Result: `[2*X*Y,X^2,Y^2]`

See Also: DIV
HESS

DERIV

Type: Function

Description: Returns the partial derivatives of a function, with respect to the specified variables.

Access: Calculus, ↰ CALC DERIV. & INTEG

Input: Level 2/Argument 1: A function or a list of functions.
Level 1/Argument 2: A variable, or a vector of variables.

Output: The derivative, or a vector of the derivatives, of the function or functions.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the gradient of the following function of the spatial variables x, y, and z:

$2x^2y + 3y^2z + zx$

Command:
```
DERIV(2*X^2*Y+3*Y^2*Z+Z*X, [X,Y,Z])
EXPAND(ANS(1))
```

Result: `[4*Y*X+Z,2*X^2+6*Z*Y,X+3*Y^2]`

See also: DERVX

DERVX

Type: Function

Description: Returns the derivative of a function with respect to the current variable.

Access: Calculus, ↰ CALC DERIV. & INTEG.

Input: The function or list of functions to be differentiated.

Output: The derivative, or a vector of the derivatives, of the function or functions.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: DERIV

DESOLVE

Type: Command

Description: Solves certain first-order ordinary differential equations with respect to the current variable.

Access: Symbolic solve, ↰ S.SLV

Input: Level 2/Argument 1: A first-order differential equation.
Level 1/Argument 2: The function to solve for.

Output: The solution to the equation, either y as a function of x or x as a function of y, or x and y as functions of a parameter.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Solve the following differential equation:

$$y'(x) + 2y(x) = e^{3x}$$

Command:
```
DESOLVE(d1Y(X)+2*Y(X)=EXP(3*X),Y(X))
```

Result:
```
{Y(X)=(1/5*EXP(5*X)+C0*)(1/EXP(X)^2)}
```

See also: LDEC

DIV

Type: Command

Description: Returns the divergence of a vector function.

Access: Calculus, ↰ CALC DERIV. & INTEG.

Input: Level 2/Argument 1: An array representing a vector function.
Level 1/Argument 2: An array containing the variables.

Output: The divergence of the vector function with respect to the specified variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the divergence of the following vector function:

$$v = x^2y\underset{\sim}{i} + x^2y\underset{\sim}{j} + y^2z\underset{\sim}{k}$$

Command: `DIV([X^2*Y, X^2*Y, Y^2*Z],[X,Y,Z])`

Result: `Y*2*X+X^2+Y^2`

See also: CURL, HESS

DIV2

Type: Command

Description: Performs euclidean division on two expressions. Step-by-step mode is available with this command.

Access: Arithmetic, ↰ (ARITH) POLYNOMIAL

Input: Level 2/Argument 1: The dividend.
Level 1/Argument 2: The divisor.

Output: Level 2/Item 1: The quotient.
Level 1/Item 2: The remainder.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Perform the following division:

$$\frac{x^2 + x + 1}{2x + 4}$$

Command: `DIV2(X^2+X+1,2*X+4)`

Result: `{1/2(X-1),3}`

DIV2MOD

Type: Command

Description: Performs euclidean division on two expressions modulo the current modulus.

Access: Arithmetic, ↰ (ARITH) MODULO

Input: Level 2/Argument 1: The dividend.
Level 1/Argument 2: The divisor.

Output: Level 2/Item 1: The quotient.
Level 1/Item 2: The remainder.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the result of $\frac{x^3+4}{x^2-1}$, modulo the default modulus, 3.

Command: `DIV2MOD(X^3+4,X^2-1)`

Result: `{X X+1}`

DIVIS

Type: Command

Description: Returns a list of divisors of a polynomial or an integer.

Access: Arithmetic, ↰ ARITH

Input: A polynomial or an integer.

Output: A list containing the expressions or integers that exactly divide into the input.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the divisors of the following polynomial:

$x^2 + 3x + 2$

Command: `DIVIS(X^2+3*X+2)`

Result: `{1,X+1,X+2,X^2+3*X+2}`

See also: `DIV2`

DIVMOD

Type: Function

Description: Divides two expressions modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO

Input: Level 2/Argument 1: The dividend.
Level 1/Argument 2: The divisor.

Output: The quotient of the terms modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Modulo the default modulus, 3, divide $5x^2+4x+2$ by x^2+1.

Command: `DIVMOD(5*X^2+4*X+2,X^2+1)`

Result: `-((X^2-X+1)/X^2+1))`

DIVPC

Type: Command

Description: Returns a Taylor polynomial for the quotient of two expressions.

Access: Calculus, ↰ CALC LIMITS & SERIES

Input: Level 3/Argument 1: The numerator expression.
Level 2/Argument 2: The denominator expression.
Level 1/Argument 3: The degree of the Taylor polynomial.

Output: The Taylor polynomial at $x = 0$ of the quotient of the two expressions, to the specified degree.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the fourth degree Taylor polynomial for the following:

$$\frac{x^3 + 4x + 12}{11x^{11} + 1}$$

Command: `DIVPC(X^3+4*X+12,11*X^11+1,4)`

Result: `12+4*X+X^3`

EGCD

Type: Command

Description: Given two polynomials, a and b, returns polynomials u, v, and c where:
$au+bv=c$
In the equation, c is the greatest common divisor of a and b.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: The expression corresponding to a in the equation.
Level 1/Argument 2: The expression corresponding to b in the equation.

Output: Level 3/Item 1: The result corresponding to c in the equation.
Level 2/Item 2: The result corresponding to u in the equation.
Level 1/Item 3: The result corresponding to v in the equation.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the polynomials for u, v, and c, where c is the greatest common divisor of a and b such that:

$$u(x^2 + 1) + v(x - 1) = c$$

Command: `EGCD(X^2+1,X-1)`

Result: `{2,1,-(X+1)}`

See also: IEGCD
ABCUV

EPSX0

Type: Function

Description: Replaces all coefficients in a polynomial that have an absolute value less than that held in the variable EPS, with 0. The value in EPS must be less than 1.

Access: Catalog, CAT

Input: A polynomial.

Output: The polynomial with conforming coefficients replaced with 0.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

EULER

Type: Function

Description: For a given integer, returns the number of integers less than the integer that are co-prime with the integer. (Euler's Φ function.)

Access: ↰ (ARITH) INTEGER

Input: A non-negative integer.

Output: The number of positive integers, less than, and co-prime with, the integer.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

EXLR

Type: Command

Description: Returns the left- and right-hand sides of an equation as discrete expressions.

Access: Catalog, (CAT)

Input: An equation.

Output: Level 2/Argument 1: The expression to the left of the "=" sign in the original equation.
Level 1/Argument 2: The expression to the right of the "=" sign in the original equation

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Split the following equation into its two component expressions:
$\sin(x)=5x+y$

Command: `EXLR(SIN(X)=5*X+Y)`

Result: `{SIN(X) 5*X+Y}`

EXPAN

Type: Command

Description: Expands and simplifies an algebraic expression. This command is identical to the EXPAND command. It is included to ensure backward-compatibility with the HP 48-series calculators.

Access: Catalog, CAT

Input: An expression

Output: The expanded and simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: EXPAND

EXPAND

Type: Command

Description: Expands and simplifies an algebraic expression.

Access: Algebra, ↱ ALG

Input: An expression, or an array of expressions.

Output: The expanded and simplified expression or array of expressions.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Simplify the following expression:

$$\frac{(x^2 + 2x + 1)}{x + 1}$$

Command: `EXPAN((X^2+2*X+1)/(X+1))`

Result: `X+1`

See also: EXPAN

EXPANDMOD

Type: Function

Description: Expands and simplifies an algebraic expression, modulo the current modulus.

Access: ↰ (ARITH) MODULO

Input: An expression.

Output: The expanded and simplified expression modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Expand the following expression and give the result modulo 3 (the default modulo setting):
$(x + 3)(x + 4)$

Command:
```
EXPANDMOD((X+3)*(X+4))
```

Result:
```
X^2+X
```

EXPLN

Type: Command

Description: Transforms the trigonometric terms in an expression to exponential and logarithmic terms.

Access: Convert, ↰ (CONVERT)

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).
Complex mode must be set (flag –103 set).

Example: Transform the following expression and simplify the result using the EXPAND command:

$2\cos(x^2)$

Command:
```
EXPLN(2*COS(X^2))
EXPAND(ANS(1))
```

Result: $\frac{\mathrm{EXP}(\dot{i}X^2)^2 + 1}{\mathrm{EXP}(\ddot{i}X^2)}$

See also: SINCOS

FACTOR

Type: Command

Description: Factorizes a polynomial or an integer:

- The function expresses a polynomial as the product of irreducible polynomials.
- The function expresses an integer as the product of prime numbers.

Access: Algebra, ↱ ALG

Input: An expression or an integer.

Output: The factorized expression, or the integer expressed as the product of prime numbers.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Factorize the following:

$x^2 + 5x + 6$

Command: `FACTOR(X^2+5*X+6)`

Result: `(X+2)(X+3)`

See also: EXPAND

FACTORMOD

Type: Function

Description: Factorizes a polynomial modulo the current modulus. The modulus must be less than 100, and a prime number.

Access: Arithmetic, ↰ ARITH MODULO

Input: The expression to be factorized.

Output: The factorized expression modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Factorize the following expression modulo the default modulus, 3.

x^2+2

Command: `FACTORMOD(X^2+2)`

Result: `(X+1)*(X-1)`

FACTORS

Type: Command

Description: For a value or expression, returns a list of prime factors and their multiplicities.

Access: Arithmetic, ↰ ARITH

Input: A value or expression.

Output: A list of prime factors of the value or expression, with each factor followed by its multiplicity.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example 1: Find the prime factors of 100.

Command: `FACTORS(100)`

Result: `{5 2. 2 2.}`

Example 2: Find the irreducible factors of: $x^2 + 4x + 4$

Command: `FACTORS(X^2+4*X+4)`

Result: `{X+2,2.}`

FCOEF

Type: Command

Description: From an array of roots and multiplicities/poles, returns a rational polynomial with a leading coefficient of 1, with the specified set of roots or poles, and with the specified multiplicities.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: An array of the form [Root 1, multiplicity/pole 1, Root 2, Multiplicity/pole 2, . .] The multiplicity/pole must be an integer. A positive number signifies a multiplicity. A negative number signifies a pole.

Output: The rational polynomial with the specified roots and multiplicities/poles.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the rational polynomial corresponding to the following set of roots and poles:
1, 2, 3, –1

Command: `FCOEF([1,2,3,-1])`

Result: `(X-1)^2/(X-3)`

See also: FROOTS

FOURIER

Type: Function

Description: Returns the n^{th} coefficient of a complex Fourier series expansion. The `PERIOD` variable must be in the current path, and set to hold *L*, the period of the input function.

Access: Calculus ↰ CALC DERIV. & INTEG

Input: Level 1/Argument 2: An expression
Level 2/Argument 1: The number, *n*, of the coefficient to return.

Output: The *n*th Fourier coefficient of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).
Complex mode must be set, that is, flag –103 must be set.

FROOTS

Type: Command

Description: For a rational polynomial, returns an array of its roots and poles, with their corresponding multiplicities.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: A rational polynomial.

Output: An array of the form [Root 1, Multiplicity 1, Root 2, Multiplicity 2 . . .]
A negative multiplicity indicates a pole.

Flags: Exact mode must be set (flag –105 clear). Numeric mode must not be set (flag –03 clear). If complex mode is set (flag –103 set), FROOTS looks for complex solutions as well as real solutions.

If approximate mode is set (flag –105 set) FROOTS searches for numeric roots.

See also: FCOEF

FXND

Type: Command

Description: Splits an object into a numerator and a denominator.

Access: Catalog, CAT

Input: A fraction, or an object that evaluates to a fraction.

Output: The object split into numerator and denominator.
Level 2/Item 1: The numerator.
Level 1/Item 2: The denominator.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Return the numerator and the denominator of the following expression:

$$\frac{(x-3)^2}{z+4}$$

Command: `FXND((X-3)^2/(Z+4))`

Result: Level 2/Item1: `(X-3)^2`
Level 1/Item 2: `Z+4`

GAUSS

Type: Command

Description: Returns the diagonal representation of a quadratic form.

Access: Matrices, ↰ MATRICES QUADRATIC FORM

Input: Level 2/Argument 1: The quadratic form.
Level 1/Argument 2: A vector containing the independent variables.

Output: Level 4/Argument 1: An array of the coefficients of the diagonal.
Level 3/Argument 2: A matrix, P, such that the quadratic form is represented as P^TDP, where the diagonal matrix D contains the coefficients of the diagonal representation.
Level 2/Argument 3: The diagonal representation of the quadratic form.
Level 1/Argument 4: A list of the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the Gaussian symbolic quadratic form of the following:

$x^2 + 2axy$

Command: `GAUSS(X^2+2*A*X*Y,[X,Y])`

Result: `{[1,-A^2] [[1,A][0,1]],-(A^2*Y^2)+(A*Y+X)^2,[X,Y])`

See also: AXQ
QXA

GCD

Type: Function

Description: Returns the greatest common divisor of two objects.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: An expression, or an object that evaluates to a number.
Level 1/Argument 2: An expression, or an object that evaluates to a number.

Output: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Flags: For a symbolic result, clear the CAS modes Numeric option (flag –03 clear).

Example: Find the greatest common divisor of 2805 and 99.

Command: `GCD(2805,99)`

Result: `33`

See also: GCDMOD
EGCD
IEGCD
LCM

GCDMOD

Type: Function

Description: Finds the greatest common divisor of two polynomials modulo the current modulus.

Access: Arithmetic, ↰ (ARITH) MODULO

Input: Level 2/Argument 1: A polynomial expression.
Level 1/Argument 2: A polynomial expression.

Output: The greatest common divisor of the two expressions modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: GCD

HADAMARD

Type: Command

Description: Performs an element by element multiplication of two matrices (Hadamard product).

Access: Matrices, ↰ (MATRICES) OPERATIONS

Input: Level 2/Argument 1: Matrix 1.
Level 1/Argument 2: Matrix 2.
The matrices must have the same order.

Output: The matrix representing the result of the multiplication.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the Hadamard product of the following two matrices:

$$\begin{bmatrix} 3 & -1 & 2 \\ 0 & 1 & 4 \end{bmatrix} \text{ and } \begin{bmatrix} 2 & 3 & 0 \\ 1 & 5 & 2 \end{bmatrix}$$

Command: `HADAMARD([[3,-1,2][0,1,4]],[2,3,0][1,5,2]])`

Result: `[[6,-3,-0][0,5,8]]`

HALFTAN

Type: Command

Description: Transforms an expression by replacing sin(x), cos(x) and tan(x) subexpressions with tan($x/2$) terms.

Access: Trigonometry, ↱ TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

HERMITE

Type: Function

Description: Returns the nth Hermite polynomial.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: A non-negative integer.

Output: The corresponding polynomial expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the Hermite polynomial with degree 4.

Command: `HERMITE(4)`

Result: `16*X^4-48*X^2+12`

HESS

Type: Command

Description: Returns the Hessian matrix and the gradient of an expression with respect to the specified variables.

Access: Calculus ↰ CALC DERIV & INTEG

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: A vector of the variables.

Output: Level 3/Item 1: The Hessian matrix with respect to the specified variables.
Level 2/Item 2: The gradient with respect to the variables.
Level 1/Item 3: The vector of the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: CURL
DIV

HILBERT

Type: Command

Description: Returns a square Hilbert matrix of the specified order.

Access: Matrices, ↰ MATRICES CREATE

Input: A positive integer, representing the order.

Output: The Hilbert matrix of the specified order.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the order 3 Hilbert matrix.

Command: `HILBERT(3)`

Result:

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

HORNER

Type: Command

Description: Executes a Horner scheme on a polynomial. That is, for a given polynomial *P*, and a number *r*, HORNER returns $P/(x-r)$, *r* AND $P(r)$

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: A polynomial, *P*.
Level 1/Argument 2: A number, *r*.

Output: Level 3/Item 1: $P/(x-r)$
Level 2/Item 2: r
Level 1/item 3: $P(r)$, the remainder of the division process.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: For r = 3, find the result of executing a Horner scheme on the following polynomial:

$x^2 + x + 1$

Command: `HORNER(X^2+X+1,3)`

Results: `(X+4,3,13)`

I→R

Type: Command

Description: Converts an integer into a real number.

Access: Catalog, CAT

Input: Level 1/Argument 1: An integer.

Output: Level 1/Item 1: The integer converted to a real number.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: R→I

IABCUV

Type: Command

Description: Returns a solution in integers u and v of $au + bv = c$, where a, b, and c are integers.

Access: Arithmetic, ↰ ARITH INTEGER

Input: Level 3/Argument 1: the value of a.
Level 2/Argument 2: the value of b.
Level 1/Argument 3: the value of c.

Output: Level 2/Item 1: The value for u.
Level 1/Item 2: The value for v.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find a solution in integers of the equation:
$6a + 11b = 3$

Command: `IABCUV(6,11,3)`

Result: `{6,-3}`

See also: ABCUV
IEGCD

IBERNOULLI

Type: Function

Description: Returns the nth Bernoulli number for a given integer.

Access: Catalog, CAT

Input: Level 1/Argument 1: an integer.

Output: Level 1/Item 1: The corresponding nth Bernoulli number for the integer.

Flags: Numeric mode must not be set (flag –03 clear).

IBP

Type: Command

Description: Performs integration by parts on a function. The function must be able to be represented as a product of two functions, where the antiderivative of one of the functions is known:
`f(x) = u(x).v'(x)`

Note that the command is designed for use in RPN mode only.

Access: Calculus, ↰ CALC DERIV & INTEG

Input: Level 2: The integrand expressed as a product of two functions, `u(x).v'(x)`
Level 1: The antiderivative of one of the component functions, v(x).

Output: Level 2: u(*x*)v(*x*)
Level 1: -u'(*x*)v(*x*)

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Use integration by parts to calculate the following:
$\int x\cos(x)dx$

Command 1: Apply the IBP command:
Level 2: `X*COS(X)`
Level 1: `SIN(X)`

Result: Level 2: `SIN(X).X`
Level 1: `-SIN(X)`

Command 2: Apply the INTVX command to level 1, `-SIN(X)`

Result: Level 2: `SIN(X).X`
Level 1: `COS(X)`

Command 3: Press + to add the result to the value at level 2 to obtain the final result.

Result: `SIN(X).(X)+COS(X)`

See also: INTVX, INT, PREVAL, RISCH

ICHINREM

Type: Command

Description: Solves a system of two congruences in integers using the Chinese Remainder theorem.

Access: Arithmetic, ↰ (ARITH) INTEGER

Input: Level 2/Argument 1: A vector of the first value and the modulus.
Level 1/Argument 2: A vector of the second value and the modulus.

Output: A vector of the solution.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Solve the following system of congruences:
$x \equiv 2$ Modulo 3
$x \equiv 1$ Modulo 5

Command: `ICHINREM([2,3],[1,5])`

Results: `[-4, 15]`

See also: CHINREM

IDIV2

Type: Command

Description: For two integers, *a* and *b*, returns the integer part of *a*/*b*, and the remainder, *r*.

Access: Arithmetic, ↰ (ARITH) INTEGER

Input: Level 2/Argument 1: *a*.
Level 1/Argument 2: *b*.

Output: Level 2/Item 1: The integer part of *a*/*b*.
Level 1/Item 2: The remainder.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Return the integer part and the remainder of 11632/864.

Command: `IDIV2(11632,864)`

Result: `{13,400}`

See also: DIV2

IEGCD

Type: Command

Description: Given two integers x and y, returns three integers, a, b, and c, such that:
$ax+by=c$

Access: Arithmetic, ↰ ARITH INTEGER

Input: Level 2/Argument 1: x.
Level 1/Argument 2: y.

Output: Level 3/Item 1: c.
Level 2/Item 2: a.
Level 1/Item 3: b.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: EGCD

ILAP

Type: Function

Description: Returns the inverse Laplace transform of an expression. The expression must evaluate to a rational fraction.

Access: Calculus, ↰ CALC DIFFERENTIAL EQNS

Input: A rational expression.

Output: The inverse Laplace transformation of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the inverse Laplace transform of:

$$\frac{1}{(x-5)^2}$$

Command: `ILAP(1/(X-5)^2)`

Result: `X*EXP(5*X)`

INT

Type: Function

Description: Calculates the antiderivative of a function for a given variable, at a given point.

Access: Catalog, CAT

Input: Level 3/Item 1: A function.
Level 2/Item 2: The variable to obtain the derivative with respect to.
Level 1/Item 3: The point at which to calculate the antiderivative. This point can be a variable or an expression.

Output: The antiderivative of the function for the given variable, at the point you specified.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: INTVX
RISCH

INTVX

Type: Function

Description: Finds the antiderivative of a function symbolically, with respect to the current default variable.

Access: Calculus, ↰ CALC DERIV. & INTEG

Input: An expression.

Output: The antiderivative of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the antiderivative of the following:

$x^2 \ln x$

Command: `INTVX(X^2*LN(X))`

Result: `1/3*X^3*LN(X)+(-1/9)X^3`

See also: IBP, RISCH, PREVAL

INVMOD

Type: Function

Description: Performs modular inversion on an object modulo the current modulus.

Access: Arithmetic, ↰ (ARITH) MODULO

Input: An object.

Output: The modular inverse of the object.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Solve the following for *x*, modulo the default modulus, 3.
$(2x \equiv 1)$

Command: `INVMOD(2)`

Result: `-1`

IQUOT

Type: Function

Description: Returns the integer quotient of two integers. That is, given two integers, *a* and *b*, returns the integer *q*, such that:
$a = qb + r$, and $0 \le r < b$

Access: Arithmetic, ↰ (ARITH) INTEGER

Input: Level 2/Item 1: The dividend.
Level 1/Item 2: The divisor.

Output: The integer quotient.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: QUOT
IDIV2

IREMAINDER

Type: Function

Description: Returns the remainder of an integer division.

Access: Catalog, CAT

Input: Level 2/Argument 1: The numerator.
Level 1/Argument 2: The denominator.

Output: The remainder.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: IDIV2

ISPRIME?

Type: Function

Description: Tests if a number is prime.

Access: Arithmetic, ↰ ARITH INTEGER

Input: An object that evaluates to a number.

Output: 1 (True) if the number is prime, 0 (False) if it is not.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: NEXTPRIME
PREVPRIME

JORDAN

Type: Command

Description: Computes the eigenvalues, eigenvectors, minimum polynomial, and characteristic polynomial of a matrix.

Access: Matrices, ↰ MATRICES EIGENVECTORS

Input: An $n \times n$ matrix.

Output: Level 4/Item 1: The minimum polynomial.
Level 3/Item 2: The characteristic polynomial.
Level 2/Item 3: A list of characteristic spaces tagged by the corresponding eigenvalue (either a vector or a list of Jordan chains, each of them ending with an "Eigen:"-tagged eigenvector).
Level 1/Item 4: An array of the eigenvalues, with multiplicities

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

LAGRANGE

Type: Command

Description: Returns the interpolating polynomial of minimum degree for a pair of values.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: A two × *n* matrix of the *n* pairs of values.

Output: The polynomial that results from the Lagrange interpolation of the data.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find an interpolating polynomial for the data (1,6), (3,7), (4,8), (2,9)

Command: LAGRANGE($\begin{bmatrix} 1 & 3 & 4 & 2 \\ 6 & 7 & 8 & 9 \end{bmatrix}$)

Result: $\frac{8x^3 - 63x^2 + 151x - 60}{6}$

LAP

Type: Function

Description: Performs a Laplace transform on an expression with respect to the current default variable.

Access: Calculus, ↰ CALC DIFFERENTIAL EQNS

Input: An expression.

Output: The Laplace transform of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the Laplace transform of e^x.

Command: `LAP(EXP(X))`

Result: `1/(X-1)`

LAPL

Type: Command

Description: Returns the Laplacian of a function with respect to a list of variables.

Access: ↰ CALC DERIV & INTEG

Input: Level 2/Argument 1:An expression.
Level 1/Argument 2: A vector of variables.

Output: The Laplacian of the expression with respect to the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find, and simplify, the Laplacian of the following expression:

$e^x \cos(zy)$

Command:
```
LAPL(EXP(X)*COS(Z*Y),[X,Y,Z])
EXPAND(ANS(1))
```

Result: `-((Y^2+Z^2-1)*EXP(X)*COS(Z*Y))`

LCM

Type: Function

Description: Returns the least common multiple of two objects.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: An expression, a number, or object that evaluates to a number.
Level 1/Argument 2: An expression, a number, or object that evaluates to a number.

Output: The least common multiple of the objects.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the least common multiple of the following two expressions:

$x^2 - 1$

$x-1$

Command: `LCM(X^2-1,X-1)`

Results: `X^2-1`

See also: GCD

LCXM

Type: Command

Description: From a program with two arguments, builds a matrix with the specified number of rows and columns, with $a_{ij} = f(i,j)$.

Access: Catalog, CAT

Input: Level 3/Argument 1: The number of rows you want in the resulting matrix.
Level 2/Argument 2: The number of columns you want in the resulting matrix.
Level 1/Argument 3: A program that uses two arguments.

Output: The resulting matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Build a 2 × 3 matrix with a_{ij}=i+2j.

Command: `LCXM(2,3,<<→I J 'I+2*J'>>)`

Result:

$$\begin{bmatrix} 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}$$

LDEC

Type: Command

Description: Solves a linear differential equation with constant coefficients, or a system of first order linear differential equations with constant coefficients.

Access: Symbolic solve, ↰ S.SLV

Input: Level 2/Argument 1: For a single equation, the function forming the right hand side of the equation. For a system of equations, an array comprising the terms not containing the dependent variables.
Level 1/Argument 2: For one equation, the auxiliary polynomial. For a system of equations, the matrix of coefficients of the dependent variables.

Output: The solution.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

LEGENDRE

Type: Function

Description: Returns the *n*th degree Legendre polynomial.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: An integer, *n*.

Output: The *n*th Legendre polynomial.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the Legendre polynomial with degree 4.

Command: `LEGENDRE(4)`

Result: `(35*X^4-30*X^2+3)/8`

LGCD

Type: Function

Description: Returns the greatest common divisor of a list of expressions or values.

Access: Arithmetic, ↰ ARITH

Input: A list of expressions or values.

Output: Level 2/Item 1: The list of elements.
Level 1/Item 2: The greatest common divisor of the elements.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

LIMIT

Type: Function

Description: Returns the limit of a function as it approaches a specified value.

Access: Calculus, ↰ CALC LIMITS&SERIES

Input: Level 2/Argument 1: An expression.

Level 1/Argument 2: An expression of the form $x = y$, where x is the variable and y is the value at which the limit is to be evaluated.

Output: The limit of the expression at the limit point.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the following limit:

$$\lim_{x \to y} \frac{x^n - y^n}{x - y}$$

Command: `LIMIT((X^N-Y^N)/(X-Y), X=Y)`

Result: `N*EXP(N*LN(Y))/Y`

See also: SERIES

LIN

Type: Command

Description: Linearizes expressions involving exponential terms.

Access: Exponential and logarithm, ↰ EXP&LN

Input: An expression.

Output: The linearized expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Linearize the following expression:

$x(e^x e^y)^4$

Command: `LIN(X*(EXP(X)*EXP(Y))^4)`

Result: `X*EXP(4X+4Y)`

LINSOLVE

Type: Command

Description: Solves a system of linear equations.

Access: Symbolic solve, ↰ S.SLV

Input: Level 2/Argument 1: An array of equations.
Level1/Argument 2: A vector of the variables to solve for.

Output: Level 3/Item 1: The system of equations.
Level 2/Item 2: A list of the pivot points.
Level 1/Item 3: The solution.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

LNAME

Type: Command

Description: Returns the variable names contained in a symbolic expression.

Access: Catalog, CAT

Input: A symbolic expression.

Output: A vector containing the variable names.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: LVAR

LNCOLLECT

Type: Command

Description: Simplifies an expression by collecting logarithmic terms.

Access: Algebra, ↱ ALG

Input: An expression.

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Simplify the following expression:
$2(\ln(x)+\ln(y))$

Command: `LNCOLLECT(2(LN(X)+LN(Y))`

Result: `LN(X^2+Y)`

LVAR

Type: Command

Description: Returns a list of variables in an algebraic object.

Access: Catalog, CAT

Input: An algebraic object.

Output: Level 2/Item 1: The algebraic object.
Level 1/Item 2: A vector of the variables that the object contains.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

MAD

Type: Command

Description: Returns details of a square matrix.

Access: Matrices, ↰ MATRICES OPERATIONS

Input: A square matrix

Output: Level 4/Item 1: The determinant.
Level 3/Item 2: The formal inverse.
Level 2/Item 3: The matrix coefficients of the polynomial, p, defined by $(xi–a)p(x)=m(x)i$, where a is the matrix, and m is the characteristic polynomial of a.
Level 1/Item 4: The characteristic polynomial.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

MENUXY

Type: Command

Description: Displays a function key menu of the computer algebra commands in the specified range.

Access: Catalog, CAT

Input: Level 2/Argument 1: The number of the first command in the range that you want to display.
Level 1/Argument 2: The number of the last command in the range that you want to display.

Output: On the function key menu, a list of the computer algebra commands in the range that you specified.

MODSTO

Type: Command

Description: Changes the modulo setting to the specified number. The number that you set is reflected in the CAS Modes input form.

Access: Arithmetic, ↰ ARITH MODULO

Input: The modulo value that you want to set.

Output: The modulo setting is changed to the specified number.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

MULTMOD

Type: Function

Description: Performs modular multiplication of two objects, modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO

Input: Level 2/Argument 1: A number or an expression.
Level 1/Argument 2: A number or an expression.

Output: The result of modular multiplication of the two objects, modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the product of $2x$ and $38x^2$, modulo the default modulus, 3.

Command: `MULTMOD(2*X,38*X^2)`

Result: `X^3`

NEXTPRIME

Type: Function

Description: Given an integer, returns the next prime number larger than the integer.

Access: Arithmetic, ↰ ARITH INTEGER

Input: An integer.

Output: The next prime number larger than the integer.

Example: Find the closest, larger prime number to 145.

Command: `NEXTPRIME(145)`

Result: `149`

See also: ISPRIME?
PREVPRIME

PA2B2

Type: Command

Description: Takes a prime number, p, such that $p=2$ or $p \equiv 1 \mod 4$, and returns a Gaussian integer $a + ib$ such that $p = a^2 + b^2$. This function is useful for factorizing Gaussian integers.

Access: Arithmetic, ↰ ARITH INTEGER

Input: A prime number, p, such that $p=2$ or $p \equiv 1 \mod 4$

Output: A Gaussian integer a+ib such that $p=a^2+b^2$

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: GAUSS

PARTFRAC

Type: Command

Description: Performs partial fraction decomposition on a partial fraction.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: An algebraic expression.

Output: The partial fraction decomposition of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Perform a partial fraction decomposition of the following expression:

$$\frac{1}{x^2-1}$$

Command: `PARTFRAC(1/(X^2-1))`

Result: `(-1+(1/X+^2))`

PCAR

Type: Command

Description: Returns the characteristic polynomial of an $n \times n$ matrix.

Access: Matrices, ↰ MATRICES EIGENVECTORS

Input: A square matrix.

Output: The characteristic polynomial of the matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the characteristic polynomial of the following matrix:

$$\begin{bmatrix} 5 & 8 & 16 \\ 4 & 1 & 8 \\ -4 & -4 & -11 \end{bmatrix}$$

Command: `PCAR([[5,8,16][4,1,8][-4,-4,-11]])`

Result: `X^3+5*X^2+3*X-9`

POWMOD

Type: Function

Description: Raises an object (number or expression) to the specified power, and expresses the result modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO

Input: Level 2/Argument 1: The object.
Level 1/Argument 2: The exponent.

Output: The result of the object raised to the exponent, modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

PREVAL

Type: Function

Description: With respect to the current default variable, returns the difference between the values of a function at two specified values of the variable.

PREVAL can be used in conjunction with INTVX to evaluate definite integrals. See the example below.

Access: Calculus, ↰ CALC DERIV. & INTEG.

Input: Level 3/Argument 1: A function.
Level 2/Argument 2: The lower bound.
Level 3/Argument 1: The upper bound.
The bounds can be expressions.

Output: The result of the evaluation.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Evaluate the following:

$$\int_0^3 (x^3 + 3x)dx$$

Command: `PREVAL(INTVX(X^3+3*X),0,3)`

Result: `135/4`

PREVPRIME

Type: Function

Description: Given an integer, finds the closest prime number smaller than the integer.

Access: Arithmetic, ↰ ARITH INTEGER

Input: An integer.

Output: The closest prime number smaller than the integer.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the closest, smaller prime number to 145.

Command: `PREVPRIME(145)`

Result: `139`

See also: ISPRIME?
NEXTPRIME

PROPFRAC

Type: Command

Description: Splits an improper fraction into an integer part and a fraction part.

Access: Arithmetic, ↰ ARITH

Input: An improper fraction, or an object that evaluates to an improper fraction.

Output: A proper fraction.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Express the following as a proper fraction:

$$\frac{x^3+4}{x^2}$$

Command: `PROPFRAC((X^3+4)/X^2))`

Result: `X+(4/X^2)`

PSI

Type: Function

Description: Calculates the polygamma function in one point.

Access: Catalog, CAT

Input: A complex expression.

Output: The polygamma function.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Psi

Type: Function

Description: Calculates the digamma function in one point. The digamma function is the derivative of the natural logarithm (ln) of the gamma function. The function can be represented as follows:

$$\Psi(z) = \frac{d}{dz}(\ln\Gamma(z)) = \frac{\Gamma'(z)}{\Gamma(z)}$$

Access: Catalog, CAT

Input: Level 2/Argument 1: A complex expression
Level 1/Argument 2: A non-negative integer.

Output: The digamma function at the specified point.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

PTAYL

Type: Function

Description: Returns the Taylor polynomial at $x = a$ for a specified polynomial.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: A polynomial, P.
Level 1/Argument 2: A number, *a*.

Output: A polynomial, Q such that $Q(x-a)=P(x)$.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

QUOT

Type: Function

Description: Returns the quotient part of the Euclidean division of two polynomials.

Access: Arithmetic, ↰ (ARITH) POLYNOMIAL

Input: Level 2/Argument 1: The numerator polynomial.
Level 1/Argument 2: The denominator polynomial.

Output: The quotient of the Euclidean division.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the quotient of the division of $x^3 + 6x^2 + 11x + 6$ by $x^2 + 5x + 6$.

Command: `QUOT(X^3+6*X^2+11*X+6, X^2+5*X+6)`

Result: `X+1`

See also: REMAINDER

QXA

Type: Command

Description: Expresses a quadratic form in matrix form.

Access: Catalog, (CAT)

Input: Level 2/Argument 1: A quadratic form.
Level 1/Argument 2: A vector containing the variables.

Output: Level 2/Item 1: The quadratic form expressed in matrix form.
Level 1/Item 2: A vector containing the variables.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Express the following quadratic form in matrix form:
$x^2 + xy + y^2$

Command: `QXA(X^2+X*Y+Y^2, [X,Y])`

Result: `{[[1,1/2][1/2,1]],[X,Y]}`

See also: AXQ
GAUSS

R→I

Type: Command

Description: Converts a real number to an integer.

Access: Catalog, CAT

Input: Level 1/Argument 1: A real number.

Output: Level 1/Item 1: The real number converted to an integer.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: I→R

REF

Type: Command

Description: Reduces a matrix to echelon form.

Access: Matrices, ↰ MATRICES LINEAR SYSTEMS

Input: A matrix.

Output: The equivalent matrix in echelon form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: RREF

REMAINDER

Type: Function

Description: Returns the remainder of the Euclidean division of two polynomials.

Access: Arithmetic, ↰ ARITH POLYNOMIAL

Input: Level 2/Argument 1: The numerator polynomial.
Level 1/Argument 2: The denominator polynomial.

Output: The remainder resulting from the Euclidean division.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: QUOT

REORDER

Type: Function

Description: Given a polynomial expression and a variable, reorders the variables in the expression in the order of powers set on the CAS Modes screen, that is, either in increasing or decreasing order.

Access: Catalog, CAT

Input: Level 2/Argument 1: The polynomial expression.
Level 1/Argument 2: The variable with respect to which the reordering is performed.

Output: The reordered expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

RESULTANT

Type: Function

Description: Returns the resultant of two polynomials of the current variable. That is, it returns the determinant of the Sylvester matrices of the two polynomials.

Access: Catalog, CAT

Input: Level 2/Argument 1: The first polynomial.
Level 1/Argument 2: The second polynomial.

Output: The determinent of the two matrices that correspond to the polynomials.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

RISCH

Type: Function

Description: Performs symbolic integration on a function using the Risch algorithm. RISCH is similar to the INTVX command, except that it allows you to specify the variable of integration.

Access: Calculus ↰ CALC DERIV. & INTEG.

Input: Level 2/Argument 1: The function to integrate.
Level 1/Argument 2: The variable of integration.

Output: The antiderivative of the function with respect to the variable.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Find the antiderivative of the following function, with respect to *y*:

$y^2 + 3y + 2$

Command: `RISCH(Y^3-3*Y+2,Y)`

Result: `1/3*Y^3+3/2*Y^2+2*Y`

See also: IBP
INT
INTVX

RREF

Type: Command

Description: Reduces a martrix to row-reduced echelon form.

Access: Matrices, ↰ MATRICES LINEAR SYSTEMS

Input: A matrix.

Output: An equivalent matrix in row reduced echelon form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

---

rref

Type: Command

Description: Reduces a martrix to row-reduced echelon form.

Access: Matrices, ↰ (MATRICES) LINEAR SYSTEMS

Input: A matrix.

Output: Level 2/Argument 1: The pivot points.

Level 1/Argument 2: An equivalent matrix in row reduced echelon form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

RREFMOD

Type: Command

Description: Performs modular row-reduction to echelon form on a matrix, modulo the current modulus.

Access: Catalog, (CAT)

Input: A matrix.

Output: The modular row-reduced matrix. The modulo value is set using the Modes CAS input form.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

SERIES

Type: Command

Description: For a given function, computes Taylor series, asymptotic development and limit at finite and infinite points.

Access: Calculus, ↰ (CALC) LIMITS & SERIES

Input: Level 3/Argument 1: The function $f(x)$
Level 2/Argument 2: The variable if the limit point is 0, or an equation $x = a$ if the limit point is a.
Level 1/Argument 3: The order for the series expansion. Note the following points:

- The minimum value is 2, and the maximum value is 20.
- If the order is a positive or negative real number, the series is unidirectional.
- For bidirectional series expansions, you need to give the order as a binary integer, for example, #5d.

Output: Level 2/Item 1: A list containing the bidirectional limit, an expression approximating the function near the limit point, and the order of the remainder. These are expressed in terms of a small parameter h.
Level 1/Item 2: An expression for h in terms of the original variable.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

SEVAL

Type: Function

Description: In the given expression, evaluates any existing variables that the expression contains and substitutes these back into the expression.

Access: Catalog, CAT

Input: Level 1/Item 1: An algebraic expression.

Output: The expression with existing variables evaluated.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

SIGMA

Type: Function

Description: Calculates the discrete antiderivative of a function with respect to the variable that you define.

Access: Catalog, CAT

Input: Level 2/Item 1: A function
Level 1/Item 2: The variable to calculate the antiderivative with respect to.

Output: The antiderivative of the function.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: SIGMAVX, RISCH

SIGMAVX

Type: Function

Description: Calculates the discrete antiderivative of a function with respect to the current variable.

Access: Catalog, CAT

Input: Level 2/Item 1: A function.

Output: The antiderivative of the function.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: SIGMA, RISCH

SIGNTAB

Type: Command

Description: Tabulates the sign of a rational function of one variable.

Access: Catalog, CAT

Input: An algebraic expression.

Output: A list containing, the points where the expression changes sign, and for each point, the sign of the expression between the points.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

SIMP2

Type: Command

Description: Simplifies two objects by dividing them by their greatest common divisor.

Access: Arithmetic, ↰ ARITH

Input: Level 2/Argument 1: The first object.
Level 1/Argument 2: The second object.

Output: Level 2/Item 1: The first object divided by the greatest common divisor.
Level 1/Item 2: The second object divided by the greatest common divisor.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Divide the following expressions by their greatest common divisor:

$x^3 + 6x^2 + 11x + 6$

$x^3 - 7x - 6$

Command: `SIMP2(X^3+6*X^2+11*X+6, X^3-7*X-6)`

Result: `{X+3,X-3}`

SINCOS

Type: Command

Description: Converts complex logarithmic and exponential expressions to expressions with trigonometric terms.

Access: Trigonometry, ↱ TRIG

Input: An expression with complex linear and exponential terms.

Output: The expression with logarithmic and exponential subexpressions converted to trigonometric and inverse trigonometric expressions.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).
Must be in complex mode (flag –103 set).

Example: Express e^{ix} in trigonometric terms.

Command: `SINCOS(EXP(i*X))`

Result: `COS(X)+iSIN(X)`

SOLVE

Type: Command

Description: Finds zeros of an expression equated to 0, or solves an equation.

Access: Symbolic solve, ↰ S.SLV

Input: Level 2/Argument 1: The expression or equation.
Level 1/Argument 2: The variable to solve for.

Output: A list of zeros or solutions.

Flags: If exact mode is set (flag –105 clear) and there are no exact solutions, the command returns a null list even when there are approximate solutions.

Example: Find the zeros of the following expression:

$x^3 - x - 9$

Command: `SOLVE(X^3-X-9,X)`

Result: `{X=2.24004098747}`

See also: SOLVEVX

SOLVEVX

Type: Command

Description: Finds zeros of an expression with respect to the current variable, or solves an equation with respect to the current variable. (You use the CAS modes input form to set the current variable.)

Access: Symbolic solve, ↰ S.SLV

Input: A function or equation in the current variable.

Output: A list of zeros or solutions.

Flags: For a symbolic result, clear the CAS modes Numeric option (flag –03 clear).
If Exact mode is set (flag –105 clear) and there are no exact solutions, the command returns a null list even when there are approximate solutions.

Example: Solve the following expression for 0, where X is the default variable on the calculator:

$x^3 - x - 9$

Command: `SOLVEVX(X^3-X-9)`

Result: `{X=2.2400}`
Note that if exact mode is set, this example returns a null list as there are no exact solutions to the equation.

See also: SOLVE

SUBST

Type: Function

Description: Substitutes a value for a variable in an expression. The value can be numeric or an expression.

Access: Algebra, ↱ ALG

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: The value or expression to be substituted.

Output: The expression with the substitution made.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Substitute $x = z+1$ for x in the following expression, and apply the EXPAND command to simplify the result:

$x^2 + 3x + 7$

Command:
```
SUBST(X^2+3*X+7,X=Z+1)
EXPAND(ANS(1))
```

Result: `Z^2+5*Z+11`

SUBTMOD

Type: Function

Description: Performs a subtraction, modulo the current modulus.

Access: Arithmetic, ↰ ARITH MODULO

Input: Level 2/Argument 1: The object or number to be subtracted from.
Level 1/Argument 2: The object or number to subtract.

Output: The result of the subtraction, modulo the current modulus.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

SYLVESTER

Type: Command

Description: For a symmetric matrix A, returns D and P where D is a diagonal matrix and $A = P^T DP$

Access: Catalog, CAT

Input: A symmetric matrix.

Output: Level 2/Item 1: the diagonal martrix, D.
Level 1 1/Item 2: The matrix P.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TABVAL

Type: Command

Description: For an expression and a list of values, returns the results of substituting the values for the default variable in the expression.

Access: Catalog, CAT

Input: Level 2/Argument 1: An algebraic expression in terms of the current variable.
Level 1/Argument 2: A list of values for which the expression is to be evaluated.

Output: Level 2/Item 1: The algebraic expression.
Level 1/Item 2: A list containing two lists: a list of the values and a list of the corresponding results.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Substitute 1, 2, and 3 into $x^2 + 1$.

Command: `TABVAL(X^2+1,{1 2 3})`

Result: `{{1 2 3},{2 5 10}}`

TABVAR

Type: Command

Description: For a rational function of the current variable, computes the variation table, that is the turning points of the function and where the function is increasing or decreasing.

Access: Catalog, CAT

Input: A rational function of the current variable.

Output: Level 3/Item 1: The original rational function.
Level 2/Item 2: A list of two lists. The first list indicates the variation of the function (where it is increasing or decreasing) in terms of the independent variable. The second list indicates the variation in terms of the dependent variable.
Level 1/Item 3: A graphic object that shows how the variation table was computed.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TAN2SC

Type: Command

Description: Replaces tan(*x*) sub-expressions with sin(*x*)/(1-cos(2*x*) or (1-cos(2*x*))/sin(2*x*).

Access: Trigonometry, ↱ TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TAN2SC2

Type: Command

Description: Replaces tan(*x*) terms in an expression with sin(2*x*)/1+cos(2*x*) terms.

Access: Trigonometry, ⮡ TRIG

Input: An expression

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).
Flag –116 – Prefer sin/cos affects the result:

- If flag –116 is set (Prefer sin()) then Tan(x) terms are replaced with:
  1 – Cos(2*x*)/Sin(2*x*)
- If flag –116 is clear (prefer Cos()) then Tan(*x*) terms are replaced with:
  Sin(2*x*)/1 + Cos(2*x*)

TAYLOR0

Type: Function

Description: Performs a fourth-order Taylor expansion of an expression at $x = 0$.

Access: Calculus, ⮢ CALC LIMITS & SERIES

Input: An expression

Output: The Taylor expansion of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TCHEBYCHEFF

Type: Function

Description: Returns the nth Tchebycheff polynomial.

Access: Catalog, CAT

Input: A non-negative integer, *n*.

Output: The *n*th Tchebycheff polynomial.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TCOLLECT

Type: Command

Description: Linearizes products in a trigonometric expression by collecting sine and cosine terms, and by combining sine and cosine terms of the same argument.

Access: Trigonometry, → TRIG

Input: An expression with trigonometric terms.

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TEXPAND

Type: Command

Description: Expands transcendental functions.

Access: Trigonometry, ← TRIG

Input: An expression.

Output: The transformation of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Simplify the following expression:
ln(sin(*x*+*y*))

Command: `TEXPAND(LN(SIN(X+Y)))`

Result: `LN(COS(Y)*SIN(X)+SIN(Y)* COS(X))`

TLIN

Type: Command

Description: Linearizes and simplifies trigonometric expressions. Note that this function does not collect sin and cos terms of the same angle.

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformation of the expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Linearize and simplify the following:
$(\cos(x))^4$

Command: `TLIN(COS(X)^4)`

Result: `(1/8)*COS(4X)+(1/2)*COS(2X)+(3/8)`

TRAN

Type: Command

Description: Returns the transpose of a matrix.

Access: Matrices, ↰ MATRICES OPERATIONS

Input: A matrix.

Output: The transposed matrix.

Flags: Exact mode must be set (flag –105 clear). Numeric mode must not be set (flag –03 clear).

Example: Transpose the following matrix: $\begin{bmatrix} 1 & 7 \\ 2 & -3 \end{bmatrix}$

Command: `TRAN([[1,7][2,-3]])`

Result: `[[1,2][7,-3]]`

TRIG

Type: Command

Description: Converts complex logarithmic and exponential subexpressions into their equivalent trigonometric expressions.

Access: Trigonometry, ↱ TRIG

Input: A complex expression with logarithmic and/or exponential terms.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).
Must be in Complex mode (flag –103 set).

Example: Express the following in trigonometric terms:
$\ln(x + i)$

Command: `TRIG(LN(X+i))`

Result: $\dfrac{\mathrm{LN}(X^2 + 1) + 2 \bullet i \bullet \mathrm{ATAN}\left(\frac{1}{x}\right)}{2}$

TRIGCOS

Type: Command

Description: Simplifies a trigonometric expression by applying the identity:
$(\sin x)^2 + (\cos x)^2 = 1$
Returns only cosine terms if possible.

Access: Trigonometry, ↱ TRIG

Input: An expression with trigonometric terms.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: TRIGSIN, TRIGTAN

TRIGSIN

Type: Command

Description: Simplifies a trigonometric expression by applying the identity:
$(\sin x)^2 + (\cos x)^2 = 1$
Returns only sine terms if possible.

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: TRIGCOS, TRIGTAN

TRIGTAN

Type: Command

Description: Replaces sin and cos terms in a trigonometric expression with tan terms.

Access: Trigonometry, ↱ TRIG

Input: An expression.

Output: The transformed expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: TRIGCOS, TRIGSIN

TRUNC

Type: Command

Description: Truncates a series expansion.

Access: Catalog, CAT

Input: Level 2/Argument 1: The expression that you want to truncate.
Level 1/Argument 2: The expression to truncate with respect to.

Output: The expression from Level 2/Argument 1, with terms of order greater than or equal to the order of th expression in Level 1/Argument 2 removed.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

TSIMP

Type: Command

Description: Performs simplifications on expressions involving exponentials and logarithms.

Access: Exponential and logarithms, ↰ EXP&LN

Input: An expression

Output: The simplified expression.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

See also: TEXPAND
TLIN

VANDERMONDE

Type: Command

Description: Builds a Vandermonde matrix from a list of objects. That is, for a list of n objects, the command creates an $n \times n$ matrix. The i^{th} row in the matrix consists of the list items raised to the power of $(i–1)$.

Access: Matrices, ↰ MATRICES CREATE

Input: A list of objects.

Output: The corresponding Vandermonde matrix.

Flags: Exact mode must be set (flag –105 clear).
Numeric mode must not be set (flag –03 clear).

Example: Build a Vandermonde matrix from the following list of objects:
$\{x, y, z\}$

Command: `VANDERMONDE({x,y,z})`

Result:

$$\begin{bmatrix} 1 & 1 & 1 \\ x & y & z \\ x^2 & y^2 & z^2 \end{bmatrix}$$

VER

Type: Command

Description: Returns the Computer Algebra System version number, and date of release.

Access: Catalog, CAT

Input: No input required.

Output: The version and release date of the Computer Algebra System software.

XNUM

Type: Command

Description: Converts an object or a list of objects to approximate numeric format.

Access: Catalog, CAT

Input: An object or list of objects.

Output: The objects in numeric format.

Example: Find the approximate value of $\pi/2$, $3e$, and $4\cos(2)$.

Command: `XNUM({π/2,3*e,4*COS(2})`

Results: `{1.5707963268 8.15484548538 -1.66458734619}`

XQ

Type: Command

Description: Converts a number, or a list of numbers in decimal format, to rational format.

Access: Catalog, CAT

Input: A number, or a list of numbers.

Output: The number or list of numbers in rational format.

Example: Express .3658 in rational format:

Command: `XQ(.3658)`

Results: `1829/5000`

ZEROS

Type: Command

Description: Returns the zeros of a function of one variable, without multiplicity.

Access: Symbolic solve, ↰ S.SLV

Input: Level 2/Argument 1: An expression.
Level 1/Argument 2: The variable to solve for.

Output: The solution or solutions for the expression equated to 0.

Flags: For a symbolic result, clear the CAS modes Numeric option (flag –03 clear).
The following flag –settings affect the result:

- If Exact mode is set (flag –105 is clear), the function attempts to find exact solutions only. This may return a null list, even if approximate solutions exist.
- If Approximate mode is set (flag –105 set), the function finds numeric roots.
- If Complex mode is set flag –103 set, the function searches for real and complex roots.

HP 49G Advanced Users Guide

Volume 1

Part B

Other Commands: A to F

Go to Index

Introduction

This volume details the HP 49G commands and functions that are not computer algebra-specific. See Volume 1, Computer algebra commands and functions, for information on computer algebra commands.

For each operation, the following details are provided:

Type: Function or command. Functions can be used as a part of an algebraic objects and commands cannot.

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys.

Input/Output: The input argument or arguments that the operation needs, and the outputs it produces.

See also: Related functions or commands

A to B

ABS

Type: Function

Description: Absolute Value Function: Returns the absolute value of its argument.

ABS has a derivative (SIGN) but not an inverse.

In the case of an array, ABS returns the Frobenius (Euclidean) norm of the array, defined as the square root of the sum of the squares of the absolute values of all *n* elements. That is:

$$\sqrt{\sum_{i=1}^{n} |z_i|^2}$$

Access: ↰ ABS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	$\lvert x \rvert$
(x,y)	→	$\sqrt{x^2 + y^2}$
x_unit	→	$\lvert x \rvert$*_unit*
[array]	→	$\lVert$ *array* $\rVert$
'symb'	→	*'ABS(symb)'*

See also: NEG, SIGN

ACK

Type: Command

Description: Acknowledge Alarm Command: Acknowledges the oldest past-due alarm.

ACK clears the alert annunciator if there are both no other past-due alarms and no other active alert sources (such as a low battery condition).

ACK has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Access: ↱ TIME TOOLS ALRM ACK

See also: ACKALL

ACKALL

Type: Command

Description: Acknowledge All Alarms Command: Acknowledges all past-due alarms.

ACKALL clears the alert annunciator if there are no other active alert sources (such as a low battery condition).

ACKALL has no effect on control alarms. Control alarms that come due are automatically acknowledged *and* saved in the system alarm list.

Access: ↱ TIME TOOLS ALRM ACK

See also: ACK

ACOS

Type: Analytic Function

Description: Arc Cosine Analytic Function: Returns the value of the angle having the given cosine.

For a real argument x in the domain $-1 \le x \le 1$, the result ranges from 0 to 180 degrees (0 to π radians; 0 to 200 grads).

A real argument outside of this domain is converted to a complex argument, $z = x + 0i$, and the result is complex.

The inverse of COS is a *relation*, not a function, since COS sends more than one argument to the same result. The inverse relation for COS is expressed by ISOL as the *general solution*

s1*ACOS(Z)+2*π*n1

The function ACOS is the inverse of a *part* of COS, a part defined by restricting the domain of COS such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of COS are called the *principal values* of the inverse relation. ACOS in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOS to the boundary of the restricted domain of COS form the *branch cuts* of ACOS.

The principal branch used by the HP 49 for ACOS was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ACOS. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation s1*ACOS(Z)+2*π*n1 for the case *s1*=1 and *n1* = 0. For other values of *s1* and *n1*, the vertical band in the lower graph is translated to the right or to the left. Taken together, the bands cover the whole complex plane, which is the domain of COS.

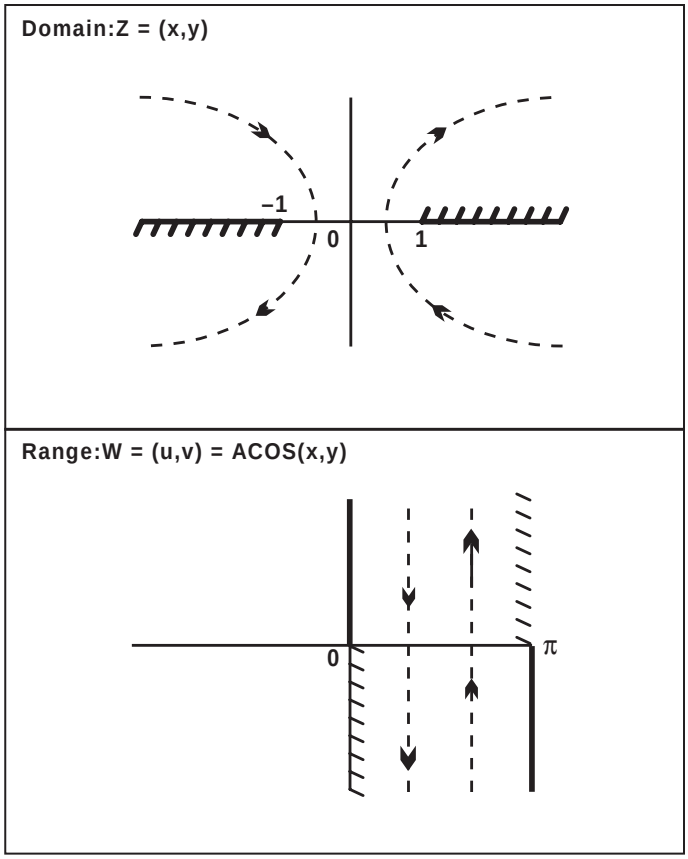
View these graphs with domain and range reversed to see how the domain of COS is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). COS sends this domain onto the whole complex plane in the range W = (*u*, *v*) = COS(*x*, *y*) in the upper graph.

Access: ↰ ACOS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*acos z*
'symb'	→	*'ACOS(symb)'*

See also: ASIN, ATAN, COS, ISOL.



ACOSH

Type: Analytic Function

Description: Inverse Hyperbolic Cosine Analytic Function: Returns the inverse hyperbolic cosine of the argument.

For real arguments $x < 1$, ACOSH returns the complex result obtained for the argument $(x, 0)$.

The inverse of ACOSH is a *relation*, not a function, since COSH sends more than one argument to the same result. The inverse relation for COSH is expressed by ISOL as the *general solution*:

s1*ACOSH(Z)+2*π*i*n1

The function ACOSH is the inverse of a *part* of COSH, a part defined by restricting the domain of COSH such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of COSH are called the *principal values* of the inverse relation. ACOSH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOSH to the boundary of the restricted domain of COSH form the *branch cuts* of ACOSH.

The principal branch used by the HP 49 for ACOSH was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued hyperbolic arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation s1*ACOSH(Z)+2*π*i*n1 for the case *s1* = 1 and *n1* = 0. For other values of *s1* and *n1*, the horizontal half-band in the lower graph is rotated to the left and translated up and down. Taken together, the bands cover the whole complex plane, which is the domain of COSH.

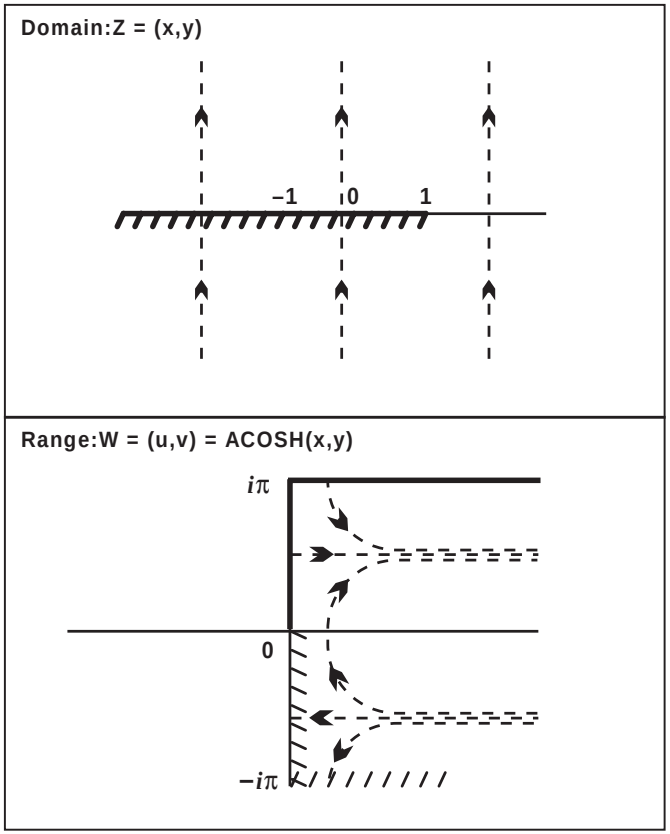
View these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse *function* possible. Consider the horizontal half-band in the lower graph as the restricted domain Z = (*x*, *y*). COSH sends this domain onto the whole complex plane in the range W = (*u*, *v*) = COSH(*x*, *y*) in the upper graph.

Access: ↱ TRIG HYPERBOLIC ACOSH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*acosh z*
'symb'	→	*'ACOSH(symb)'*

See also: ASINH, ATANH, COSH, ISOL



ADD

Type: Command

Description: Add List Command: Adds corresponding elements of two lists or adds a number to each of the elements of a list.

ADD executes the + command once for each of the elements in the list. If two lists are the arguments, they must have the same number of elements as ADD will execute the + command once for each corresponding pair of elements. If one argument is a non-list object, ADD will attempt to execute the + command using the non-list object and each element of the list argument, returning the result to the corresponding position in the result. (See the + command entry to see the object combinations that are defined.) If an undefined addition is encountered, a Bad Argument Type error results.

Access: CAT ADD

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ $list_1$ }	{ $list_2$ }	→	{ $list_{result}$ }
{ $list$ }	$obj_{non\text{-}list}$	→	{ $list_{result}$ }
$obj_{non\text{-}list}$	{ $list$ }	→	{ $list_{result}$ }

See also: ΔLIST, ΠLIST, ΣLIST

ADDTOREAL

Type: Command

Description: Add to Real List List Command: Adds the specified global name to the reserved variable REALASSUME. REALASSUME is a list of the global variables that will be considered by a CAS operation to represent *real* numbers.

Access: CAT ADDTOREAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'global'	→	

ALOG

Type: Analytic Function

Description: Common Antilogarithm Analytic Function: Returns the common antilogarithm; that is, 10 raised to the given power.

For complex arguments: $10^{(x,y)} = e^{cx} \cos cy + i\, e^{cx} \sin cy$ where $c = \ln 10$.

Access: ↰ 10^x

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	10^z
'symb'	→	*'ALOG(symb)'*

See also: EXP, LN, LOG

AMORT

Type: Command

Description: Amortize Command: Amortizes a loan or investment based upon the current amortization settings.

Values must be stored in the TVM variables (*1%YR*, *PV*, *PMT*, and *PYR*). The number of payments *n* is taken from the input together with flag –14.

Access: ↰ FINANCE AMOR

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
n	→	*principal*	*interest*	*balance*

See also: TVM, TVMBEG, TVMEND, TVMROOT

AND

Type: Function

Description: And Function: Returns the logical AND of two arguments.

When the arguments are binary integers or strings, AND does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table.

bit_1	bit_2	bit_1 **AND** bit_2
0	0	0
0	1	0
1	0	0
1	1	1

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must have the same number of characters.

When the arguments are real numbers or symbolics, AND simply does a true/false test. The result is 1 (true) if both arguments are non-zero; it is 0 (false) if either or both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic expressions, then the result is an algebraic of the form $symb_1$ AND $symb_2$. Execute →NUM (or set flag –3 before executing AND) to produce a numeric result from the algebraic result.

Access: ↱ BASE LOGIC AND

test and

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
#n_1	#n_2	→	#n_3
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	*0/1*
T/F	'*symb*'	→	'*T/F AND symb*'
'*symb*'	*T/F*	→	'*symb AND T/F*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ *AND* $symb_2$'

See also: NOT, OR , XOR

ANIMATE

Type: Command

Description: Animate Command: Displays graphic objects in sequence.

ANIMATE displays a series of graphics objects (or variables containing them) one after the other. You can use a list to specify the area of the screen you want to animate (pixel coordinates #*X* and #*Y*), the number of seconds before the next grob is displayed (*delay*), and the number of times the sequence is run (*rep*). If *rep* is set to 0, the sequence is played one million times, or until you press CANCEL.

The animation displays PICT while displaying the grobs. The grobs and the animate parameters are left on the stack.

Access: ↰ PRG GROB ANIMATE

Input/Output:

$L_{n+1}.../A_1$	L_1/A_{n+1}		L_1/I_1
$grob_n...grob_1$	n_{grobs}	→	*same stack*
$grob_n...grob_1$	{ *n* {#*X*#*Y* } *delay* *rep* }	→	*same stack*

L = level, A = argument, I = item

ANS

Type: Command

Description: Recalls the *n*th answer from history.

Access: ↰ ANS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	obj_n

See also: LASTARG

APPLY

Type: Function

Description: Apply to Arguments Function: Creates an expression from the specified function name and arguments.

A user-defined function *f* that checks its arguments for special cases often can't determine whether a symbolic argument *x* represents one of the special cases. The function *f* can use APPLY to create a new expression *f(x)*. If the user now evaluates *f(x)*, *x* is evaluated before *f*, so the argument to *f* will be the result obtained by evaluating *x*.

When evaluated in an algebraic expression, APPLY evaluates the arguments (to resolve local names in user-defined functions) before creating the new object.

Access: CAT APPLY

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ $symb_1$ ... $symb_n$ }	'*name*'	→	'*name*($symb_1$ ... $symb_n$)'

See also: QUOTE, |

ARC

Type: Command

Description: Draw Arc Command: Draws an arc in *PICT* counterclockwise from $x_{\theta 1}$ to $x_{\theta 2}$, with its center at the coordinate specified in argument 1 or level 4 and its radius specified in argument 2 or level 3.

ARC always draws an arc of constant radius in pixels, even when the radius and center are specified in user-units, regardless of the relative scales in user-units of the x- and y-axes. With user-unit arguments, the arc starts at the pixel specified by $(x, y) + (a, b)$, where (a, b) is the rectangular conversion of the polar coordinate $(x_{radius}, x_{\theta 1})$. The resultant distance in pixels from the starting point to the centre pixel is used as the actual radius, r'. The arc stops at the pixel specified by $(r', x_{\theta 2})$.

If $x_{\theta 1} = x_{\theta 2}$, ARC plots one point. If $|x_{\theta 1} - x_{\theta 2}| > 360$ degrees, 2π radians, or 400 grads, ARC draws a complete circle.

Access: ↰ PRG PICT ARC

Input/Output:

Level 4/Argument 1	Level 3/Argument 2	Level 2/Argument 3	Level 1/Argument 4		Level 1/Item 1
(x, y)	x_{radius}	$x_{\theta 1}$	$x_{\theta 2}$	→	
{ #*n*,#*m* }	$\#n_{radius}$	$x_{\theta 1}$	$x_{\theta 2}$	→	

See also: BOX, LINE, TLINE

ARCHIVE

Type: Command

Description: Archive HOME Command: Creates a backup copy of the *HOME* directory (that is, all variables), the user-key assignments, and the alarm catalog in the specified backup object (:n_{port}:*name*) in RAM or flash ROM.

The specified port number can be 0 through 2. An error will result if there is not enough memory in the specified port to copy the HOME directory.

If the backup object is :IO:*name*, then the copied directory is transmitted in binary via Kermit protocol through the current I/O port to the specified filename.

To save flag settings, execute RCLF and store the resulting list in a variable.

Access: ↰ PRG MEMORY ARCHIVE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
:n_{port} :*name*	→	
:*IO* :*name*	→	

See also: RESTORE

ARG

Type: Function

Description: Argument Function: Returns the (real) polar angle θ of a complex number (x, y).

The polar angle θ is equal to:

- atan y/x for $x \geq 0$
- atan $y/x + \pi$ sign y for $x < 0$, Radians mode
- atan $y/x + 180$ sign y for $x < 0$, Degrees mode
- atan $y/x + 200$ sign y for $x < 0$, Grads mode

A real argument x is treated as the complex argument $(x,0)$.

Access: ↱ ARG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	θ
'symb'	→	*'ARG(symb)'*

ARIT

Type: Command

Description: Displays a menu of arithmetic commands.

Access: CAT ARIT

Input: None

Output: None

See also: BASE, CMPLX, DIFF, EXP&LN, SOLVER, TRIGO

ARRY→

Type: Command

Description: Array to Stack Command: Takes an array and returns its elements as separate real or complex numbers. Also returns a list of the dimensions of the array.

If the argument is an n-element vector, the first element is returned to level $n + 1$ (not level $nm + 1$), and the nth element to level 2.

Access: CAT ARRY→

Input/Output:

Level 1/Argument 1		$L_{nm+1}/A_1 \ldots L_2/A_{nm}$	$Level_1/Item_{nm+1}$
[vector]	→	$z_1 \cdots z_n$	{ $n_{element}$ }
[[matrix]]	→	$z_{11} \cdots z_{nm}$	{ n_{row} m_{col} }

L = level; I = item

Flags: None

See also: →ARRY, DTAG, EQ→, LIST→, OBJ→, STR→

→ARRY

Type: Command

Description: Stack to Array Command: Returns a vector of *n* real or complex elements or a matrix of *n* × *m* real or complex elements.

The elements of the result array should be entered in row order. If one or more of the elements is a complex number, the result array will be complex.

Access: CAT →ARRY

Input/Output:

$Level_{nm+1}/Argument_1 \ldots Level_2/Argument_{nm}$	$Level_1/Argument_{nm+1}$		$Level_1/Item_1$
$z_1 \cdots z_n$	$n_{element}$	→	[*vector*]
$z_{11} \cdots z_{nm}$	{ n_{row}, m_{col} }	→	[[*matrix*]]

See also: ARRY→, LIST→, →LIST, OBJ→, STR→, →TAG, →UNIT

ASIN

Type: Analytic Function

Description: Arc Sine Analytic Function: Returns the value of the angle having the given sine.

For a real argument x in the domain $-1 \le x \le 1$, the result ranges from –90 to +90 degrees ($-\pi/2$ to $+\pi/2$ radians; –100 to +100 grads).

A real argument outside of this domain is converted to a complex argument $z = x + 0i$, and the result is complex.

The inverse of SIN is a *relation*, not a function, since SIN sends more than one argument to the same result. The inverse relation for SIN is expressed by ISOL as the *general solution*:

```
ASIN(Z)*(-1)^n1+π*n1
```

The function ASIN is the inverse of a *part* of SIN, a part defined by restricting the domain of SIN such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

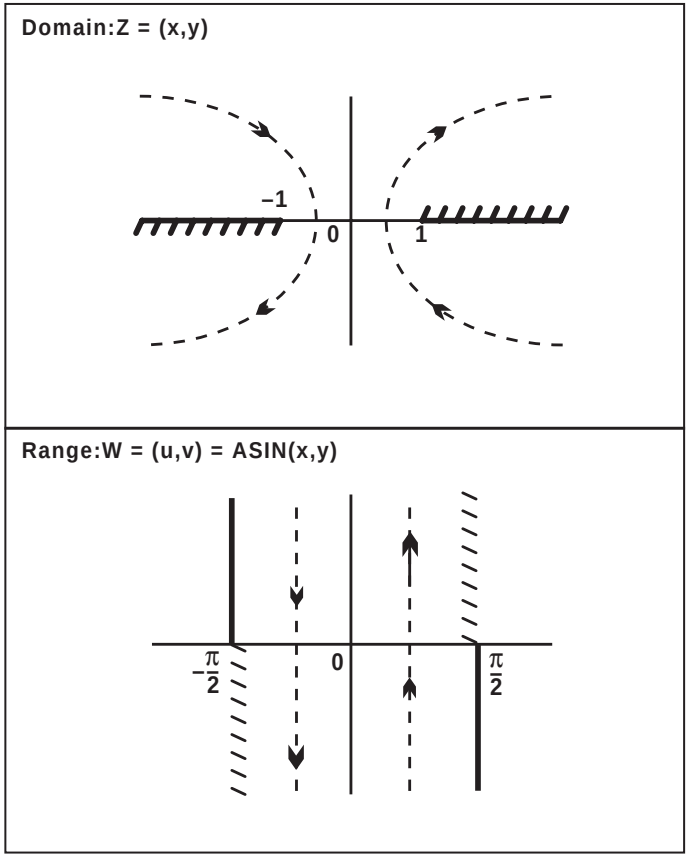
The points in this restricted domain of SIN are called the *principal values* of the inverse relation. ASIN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASIN to the boundary of the restricted domain of SIN form the *branch cuts* of ASIN.

The principal branch used by the HP 49 for ASIN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc sine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ASIN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation ASIN(Z)*(–1)^n1+π*n1 for the case *n1*=0. For other values of *n1*, the vertical band in the lower graph is translated to the right (for *n1* positive) or to the left (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of SIN.

View these graphs with domain and range reversed to see how the domain of SIN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). SIN sends this domain onto the whole complex plane in the range
W = (*u*, *v*) = SIN(*x*, *y*) in the upper graph.



Access: ↰ ASIN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*asin z*
'symb'	→	*'ASIN(symb)'*

See also: ACOS, ATAN, ISOL, SIN

ASINH

Type: Analytic Function

Description: Arc Hyperbolic Sine Analytic Function: Returns the inverse hyperbolic sine of the argument.

The inverse of SINH is a *relation*, not a function, since SINH sends more than one argument to the same result. The inverse relation for SINH is expressed by ISOL as the *general solution*:

```
ASINH(Z)*(-1)^n1+π*i*n1
```

The function ASINH is the inverse of a *part* of SINH, a part defined by restricting the domain of SINH such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of SINH are called the *principal values* of the inverse relation. ASINH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASINH to the boundary of the restricted domain of SINH form the *branch cuts* of ASINH.

The principal branch used by the HP 49 for ASINH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ASINH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for ASINH can be found from the graph for ASIN (see ASIN) and the relationship asinh $z = -i$ asin iz.

Access: ↱ TRIG HYPERBOLIC ASINH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*asinh* z
'*symb*'	→	'*ASINH(symb)*'

See also: ACOSH, ATANH, ISOL, SINH

ASN

Type: Command

Description: Assign Command: Defines a single key on the user keyboard by assigning the given object to the key x_{key}, which is specified as *rc.p*.

The argument x_{key} is a real number *r.p* specifying the key by its row number *r*, column number *c*, and plane (shift) *p*. The legal values for *p* are as follows:

Plane, *p*	Shift
0 or 1	unshifted
2	◁ (left-shifted)
3	▷ (right-shifted)
4	ALPHA (alpha-shifted)
5	ALPHA ◁ (alpha left-shifted)
6	ALPHA ▷ (alpha right-shifted)

Once ASN has been executed, pressing a given key in User or 1-User mode executes the user-assigned object. The user key assignment remains in effect until the assignment is altered by ASN, STOKEYS, or DELKEYS. Keys without user assignments maintain their standard definitions.

If the argument *obj* is the name SKEY, then the specified key is restored to its *standard key* assignment on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command S DELKEYS (see DELKEYS).

To make multiple key assignments simultaneously, use STOKEYS. To delete key assignments, use DELKEYS.

Be careful not to reassign or suppress the keys necessary to cancel User mode. If this happens, exit User mode by doing a system halt ("warm start"): press and hold ON and F3 simultaneously, releasing F3 first. This cancels User mode.

Access: CAT ASN

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	x_{key}	→	
'SKEY'	x_{key}	→	

Output: None

See also: DELKEYS, RCLKEYS, STOKEYS

ASR

Type: Command

Description: Arithmetic Shift Right Command: Shifts a binary integer one bit to the right, except for the most significant bit, which is maintained.

The most significant bit is preserved while the remaining (*wordsize* –1) bits are shifted right one bit. The second-most significant bit is replaced with a zero. The least significant bit is shifted out and lost.

An arithmetic shift is useful for preserving the sign bit of a binary integer that will be shifted. Although the HP 49 makes no special provision for signed binary integers, you can still *interpret* a number as a signed quantity.

Access: ↱ BASE BIT ASR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: SL, SLB, SR, SRB

ATAN

Type: Analytic Function

Description: Arc Tangent Analytic Function: Returns the value of the angle having the given tangent.

For a real argument, the result ranges from –90 to +90 degrees (–π/2 to +π/2 radians; –100 to +100 grads).

The inverse of TAN is a *relation*, not a function, since TAN sends more than one argument to the same result. The inverse relation for TAN is expressed by ISOL as the *general solution*:

ATAN(Z)+π*n1

The function ATAN is the inverse of a *part* of TAN, a part defined by restricting the domain of TAN such that:

- each argument is sent to a distinct result, and

- each possible result is achieved.

The points in this restricted domain of TAN are called the *principal values* of the inverse relation. ATAN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATAN to the boundary of the restricted domain of TAN form the *branch cuts* of ATAN.

The principal branch used by the HP 49 for ATAN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cuts for the complex-valued arc tangent function occur where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of ATAN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

These graphs show the inverse relation ATAN(Z)+π*n1 for the case *n1* = 0. For other values of *n1*, the vertical band in the lower graph is translated to the right (for *n1* positive) or to the left (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of TAN.

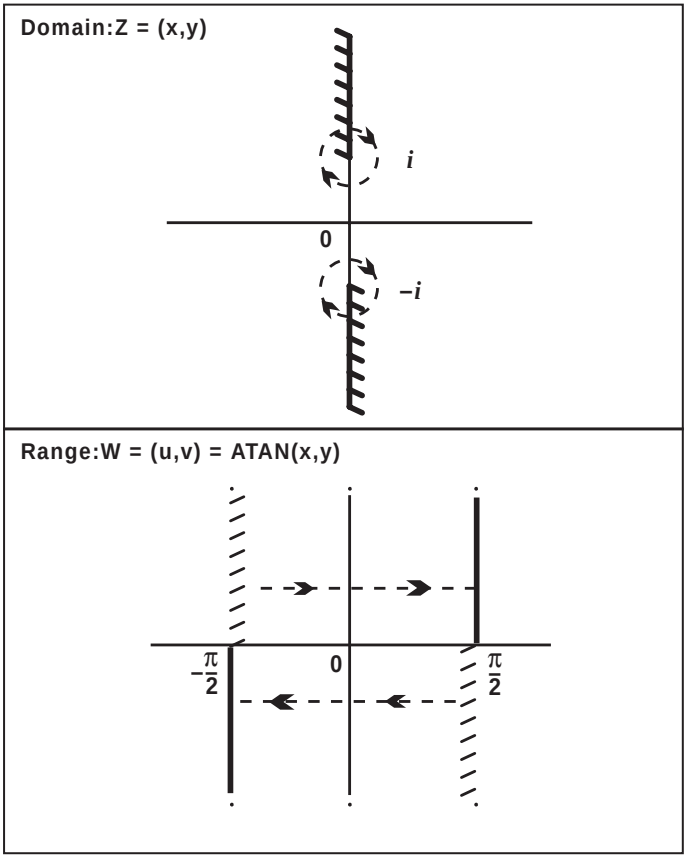
View these graphs with domain and range reversed to see how the domain of TAN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (x, y). TAN sends this domain onto the whole complex plane in the range W = (u, v) = TAN(x, y) in the upper graph.

Access: ↰ ATAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*atan z*
'*symb*'	→	'*ATAN(symb)*'

See also: ACOS, ASIN, ISOL, TAN



ATANH

Type: Analytic Function

Description: Arc Hyperbolic Tangent Analytic Function: Returns the inverse hyperbolic tangent of the argument.

For real arguments $|x| > 1$, ATANH returns the complex result obtained for the argument $(x, 0)$. For a real argument $x=\pm 1$, an Infinite Result exception occurs. If flag –22 is set (no error), the sign of the result (MAXR) matches that of the argument.

The inverse of TANH is a *relation*, not a function, since TANH sends more than one argument to the same result. The inverse relation for TANH is expressed by ISOL as the *general solution*;

ATANH(Z)+π*i*n1

The function ATANH is the inverse of a *part* of TANH, a part defined by restricting the domain of TANH such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

The points in this restricted domain of TANH are called the *principal values* of the inverse relation. ATANH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATANH to the boundary of the restricted domain of TANH form the *branch cuts* of ATANH.

The principal branch used by the HP 49 for ATANH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ATANH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graph for ATANH can be found from the graph for ATAN (see ATAN) and the relationship atanh $z = -i$ atan iz.

Access: ↱ TRIG HYPERBOLIC ATAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*atanh z*
'*symb*'	→	'*ATANH(symb)*'

See also: ACOSH, ASINH, ISOL, TANH

ATICK

Type: Command

Description: Axes Tick-Mark Command: Sets the axes tick-mark annotation in the reserved variable *PPAR*.

Given *x*, ATICK sets the tick-mark annotation to *x* units on both the *x*- and the *y*-axis. For example, 2 would place tick-marks every 2 units on both axes.

Given #*n*, ATICK sets the tick-mark annotation to #*n* pixels on both the *x*- and the *y*-axis. For example, #5 would place tick-marks every 5 pixels on both axes.

Given { *x*,*y*}, ATICK sets the tick-mark unit annotation for each axis individually. For example, { 10, 3 } would mark the *x*-axis at every multiple of 10 units, and the *y*-axis at every multiple of 3 units.

Given { #*n* #*m* } ATICK sets the tick-mark pixel annotation for each axis individually. For example, {#6, #2 } would mark the *x*-axis every 6 pixels, and the *y*-axis every 2 pixels.

Access: CAT ATICK

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	
#*n*	→	
{ *x*,*y* }	→	
{ #*n*, #*m* }	→	

See also: AXES, DRAX

ATTACH

Type: Command

Description: Attach Library Command: Attaches the library with the specified number to the current directory. Each library has a unique number. If a port number is specified, it is ignored.

To use a library object, it must be in a port and it must be attached. A library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

Some libraries require you to ATTACH them.

You can ascertain whether a library is attached to the current directory by executing LIBS.

A library that has been copied into RAM and then stored (with STO) into a port can be attached *only after the calculator has been turned off and then on again* following the STO command. This action (off/on) creates a *system halt*, which makes the library object "attachable." Note that it also clears the stack, local variables, and the LAST stack, and it displays the TOOL menu. (To save the stack first, execute DEPTH →LIST '*name*' STO.)

The number of libraries that can be attached to the HOME directory is limited only by the available memory. However, only one library at a time can be attached to any other directory.

If you attempt to attach a second library to a non-*HOME* directory, the new library will overwrite the old one.

Access: CAT ATTACH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{library}	→	
$:n_{\text{port}}\ :n_{\text{library}}$	→	

See also: DETACH, LIBS

AUTO

Type: Command

Description: Autoscale Command: Calculates a *y*-axis display range, or an *x*- and *y*-axis display range.

The action of AUTO depends on the plot type as follows:

Plot Type	Scaling Action
FUNCTION	Samples the equation in *EQ* at 40 values of the independent variable, equally spaced through the *x*-axis plotting range, discards points that return ±∞, then sets the *y*-axis display range to include the maximum, minimum, and origin.
CONIC	Sets the *y*-axis scale equal to the *x*-axis scale.
POLAR	Samples the equation in *EQ* at 40 values of the independent variable, equally spaced through the plotting range, discards points that return ±∞, then sets both the *x*- and *y*-axis display ranges in the same manner as for plot type FUNCTION.
PARAMETRIC	Same as POLAR.

Plot Type	Scaling Action (Continued)
TRUTH	No action.
BAR	Sets the *x*-axis display range from 0 to the number of elements in Σ*DAT*, plus 1. Sets the *y*-range to the minimum and maximum of the elements. The *x*-axis is always included.
HISTOGRAM	Sets the *x*-axis display range to the minimum and maximum of the elements in Σ*DAT*. Sets the *y*-axis display range from 0 to the number of rows in Σ*DAT*.
SCATTER	Sets the *x*-axis display range to the minimum and maximum of the independent variable column (XCOL) in Σ*DAT*. Sets the *y*-axis display range to the minimum and maximum of the dependent variable column (YCOL).

AUTO does not affect 3D plots.

AUTO actually calculates a *y*-axis display range and then expands that range so that the menu labels do not obscure the resultant plot.

AUTO does not draw a plot – execute DRAW to do so.

Access: CAT AUTO

Input: None

Output: None

See also: DRAW, SCALEH, SCALE, SCLΣ, SCALEW, XRNG, YRNG

AXES

Type: Command

Description: Axes Command: Specifies the intersection coordinates of the *x*- and *y*-axes, tick-mark annotation, and the labels for the *x*- and *y*-axes. This information is stored in the reserved variable *PPAR*.

The argument for AXES (a complex number or list) is stored as the fifth parameter in the reserved variable *PPAR*. How the argument is used depends on the type of object it is:

- If the argument is a complex number, it replaces the current entry in *PPAR*.
- If the argument is a list containing any or all of the above variables, only variables that are specified are affected.

atick has the same format as the argument for the ATICK command. This is the variable that is affected by the ATICK command.

The default value for AXES is (0,0).

Axes labels are not displayed in *PICT* until subsequent execution of LABEL.

Access: CAT AXES.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	
{ (x, y) *atick* ,"*x-axis label*", "*y-axis label*" }	→	

See also: ATICK, DRAW, DRAX, LABEL

BAR

Type: Command

Description: Bar Plot Type Command: Sets the plot type to BAR.

When the plot type is BAR, the DRAW command plots a bar chart using data from one column of the current statistics matrix (reserved variable Σ*DAT*). The column to be plotted is specified by the XCOL command, and is stored in the first parameter of the reserved variable Σ*PAR*. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type BAR, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers, with the smaller of the numbers specifying the horizontal location of the first bar. The default value of *indep* is *X*.
- *res* is a real number specifying the bar width in user-unit coordinates, or a binary integer specifying the bar width in pixels. The default value is 0, which specifies a bar width of 1 in user-unit coordinates.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command BAR places the command name BAR in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

A bar is drawn for each element of the column in Σ*DAT*. Its width is specified by *res* and its height is the value of the element. The location of the first bar can be specified by *indep*; otherwise, the value in (x_{min},y_{min}) is used.

Access: CAT BAR

Input: None

Output: None

See also: CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

BARPLOT

Type: Command

Description: Draw Bar Plot Command: Plots a bar chart of the specified column of the current statistics matrix (reserved variable Σ*DAT*).

The data column to be plotted is specified by XCOL and is stored as the first parameter in reserved variable Σ*PAR*. The default column is 1. Data can be positive or negative, resulting in bars above or below the axis. The *y*-axis is autoscaled, and the plot type is set to BAR.

When BARPLOT is executed from a program, the resulting plot does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Access: CAT BARPLOT

Input: None

Output: A bar chart based on ΣDAT.

See also: FREEZE, HISTPLOT, PICTURE, PVIEW, SCATRPLOT, XCOL

BASE

Type: Command

Description: Displays a menu of basic algebra commands.

Access: CAT BASE

Input: None

Output: None

See also: ARIT, CMPLX, DIFF, EXP&LN, SOLVER, TRIGO

BAUD

Type: Command

Description: Baud Rate Command: Specifies bit-transfer rate.

Legal baud rates are 1200, 2400, 4800, and 9600 (default).

Access: CAT BAUD

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{baudrate}	→	

See also: CKSM, PARITY, TRANSIO

BEEP

Type: Command

Description: Beep Command: Sounds a tone at *n* hertz for *x* seconds.

The frequency of the tone is subject to the resolution of the built-in tone generator. The maximum frequency is approximately 4400 Hz; the maximum duration is 1048.575 seconds. Arguments greater than these maximum values default to the maxima.

Access: ↰ PRG OUT BEEP

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$n_{\text{frequency}}$	x_{duration}	→	

See also: HALT, INPUT, PROMPT, WAIT

BESTFIT

Type: Command

Description: Best-Fitting Model Command: Executes LR with each of the four curve fitting models, and selects the model yielding the largest correlation coefficient.

The selected model is stored as the fifth parameter in the reserved variable Σ*PAR*, and the associated regression coefficients, intercept and slope, are stored as the third and fourth parameters, respectively.

Access: CAT BESTFIT

Input: None

Output: None

See also: EXPFIT, LINFIT, LOGFIT, LR, PWRFIT

BIN

Type: Command

Description: Binary Mode Command: Selects binary base for binary integer operations. (The default base is decimal.)

Binary integers require the prefix #. Binary integers entered and returned in binary base automatically show the suffix b. If the current base is not binary, binary numbers can still be entered by using the suffix b (the numbers are displayed in the current base, however).

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: (CAT) BIN

Input: None

Output: None

See also: DEC, HEX, OCT, STWS, RCWS

BINS

Type: Command

Description: Sort into Frequency Bins Command: Sorts the elements of the independent column (XCOL) of the current statistics matrix (the reserved variable Σ*DAT*) into (n_{bins} + 2) bins, where the left edge of bin 1 starts at value x_{min} and each bin has width x_{width}.

BINS returns a matrix containing the frequency of occurrences in each bin, and a 2-element array containing the frequency of occurrences falling below or above the defined range of *x*-values. The array can be stored into the reserved variable Σ*DAT* and used to plot a bar histogram of the bin data (for example, by executing BARPLOT).

For each element *x* in ΣDAT, the *n*th bin count $n_{freq\ bin\ n}$ is incremented, where:

$$n_{\ freqbinn} = IP\left[\frac{x - x_{min}}{x_{width}}\right]$$

for $x_{min} \leq x \leq x_{max}$, where $x_{max} = x_{min} + (n_{bins})(x_{width})$.

Access: (CAT) BINS

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
x_{min}	x_{width}	n_{bins}	→	$[[\ n_{bin\ 1}\ \ldots\ n_{bin\ n}\ ]]$	$[\ n_{bin\ L}\ n_{bin\ R}\ ]$

L = level; A = argument; I = item

See also: BARPLOT, XCOL

BLANK

Type: Command

Description: Blank Graphics Object Command: Creates a blank graphics object of the specified width and height.

Access: ↰ PRG GROB BLANK

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_{width}$	$\#m_{height}$	→	$grob_{blank}$

See also: →GROB, LCD→

BOX

Type: Command Operation

Description: Box Command: Draws in *PICT* a box whose opposite corners are defined by the specified pixel or user-unit coordinates.

Access: ↰ PRG PICT BOX

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\{\ \#n_1\ \#m_1\ \}$	$\{\ \#n_2\ \#m_2\ \}$	→	
(x_1, y_1)	(x_2, y_2)	→	

See also: ARC, LINE, TLINE

BUFLEN

Type: Command

Description: Buffer Length Command: Returns the number of characters in the HP 49's serial input buffer and a single digit indicating whether an error occurred during data reception.

The digit returned is 1 if no framing, UART overrun, or input-buffer overflow errors occurred during reception, or 0 if one of these errors did occur. (The input buffer holds up to 255 bytes.) When a framing or overrun error occurs, data reception ceases until the error is cleared (which BUFLEN does); therefore, *n* represents the data received *before* the error.

Use ERRM to see which error has occurred when BUFLEN returns 0 to level 1.

Access: (CAT) BUFLEN.

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	n_{chars}	*0/1*

See also: CLOSEIO, OPENIO, SBRK, SRECV, STIME, XMIT

BYTES

Type: Command

Description: Byte Size Command: Returns the number of bytes and the checksum for the given object.

If the argument is a built-in object, then the size is 2.5 bytes and the checksum is #0.

If the argument is a global name, then the size represents the name *and* its contents, while the checksum represents the contents only. The size of the name alone is (3.5 + *n*), where *n* is the number of characters in the name.

Access: (↰) (PRG) MEMORY BYTES

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
obj	→	$\#n_{checksum}$	x_{size}

See also: MEM

B→R

Type: Command

Description: Binary to Real Command: Converts a binary integer to its floating-point equivalent.

If # *n* ≥ # 1000000000000 (base 10), only the 12 most significant decimal digits are preserved in the resulting mantissa.

Access: ↱ BASE B→R

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#*n*	→	*n*

See also: R→B

C to D

CASE

Type: Command

Description: CASE Conditional Structure Command: Starts CASE … END conditional structure.

The CASE … END structure executes a series of *cases* (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE … END structure. A default clause can also be included: this clause executes if all tests evaluate to false.

The CASE command is available in RPN programming only. You cannot use it in algebraic programming.

The CASE … END structure has this syntax:

CASE

test-clause$_1$ THEN *true-clause*$_1$ END

test-clause$_2$ THEN *true-clause*$_2$ END

.
.
.

test-clause$_n$ THEN *true-clause*$_n$ END

default-clause (*optional*)

END

When CASE executes, *test-clause*$_1$ is evaluated. If the test is true, *true-clause*$_1$ executes, then execution skips to END. If *test-clause*$_1$ is false, *test-clause*$_2$ executes. Execution within the CASE structure continues until a true clause is executed, or until all the test clauses evaluate to false. If the default clause is included, it executes if all test clauses evaluate to false.

Access: ↰ PRG BRCH CASE

Input/Output:

Level 1/Argument 1			Level 1/Item 1
CASE		→	
THEN	*T/F*	→	
END		→	
END		→	

See also: END, IF, IFERR, THEN

CEIL

Type: Function

Description: Ceiling Function: Returns the smallest integer greater than or equal to the argument.

Access: ↰ MTH REAL CEIL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'*CEIL(symb)*'

See also: FLOOR, IP, RND, TRNC

CENTR

Type: Command

Description: Center Command: Adjusts the first two parameters in the reserved variable *PPAR*, (x_{min}, y_{min}) and (x_{max},y_{max}), so that the point represented by the argument (x, y) is the plot center. The center pixel is in row 32, column 65 when *PICT* is its default size (131 × 64).

If the argument is a real number x, CENTR makes the point (x,0) the plot center.

Access: CAT CENTR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	
x	→	

See also: SCALE

CF

Type: Command

Description: Clear Flag Command: Clears the specified user or system flag.

User flags are numbered 1 through 128. System flags are numbered –1 through –128. See the *Pocket Guide* for a listing of HP 49 system flags and their flag numbers.

Access: ↰ PRG TEST CF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flagnumber}}$	→	

See also: FC?, FC?C, FS?, FS?C, SF

%CH

Type: Function

Description: Percent Change Function: Returns the percent change from x to y as a percentage of x.

If both arguments are unit objects, the units must be consistent with each other. The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

For more information on using temperature units with arithmetic functions, refer to the keyword entry of +.

Access: ↰ MTH REAL %CH

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$100(y-x)/x$
x	'*symb*'	→	'%CH(x,*symb*)'
'*symb*'	x	→	'%CH(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%CH($symb_1$, $symb_2$)'
x_unit	y_unit	→	$100(y\_unit - x\_unit)/x\_unit$
x_unit	'*symb*'	→	'%CH(x_unit,*symb*)'
'*symb*'	x_unit	→	'%CH(*symb*,x_unit)'

See also: %, %T

CHOOSE

Type: Command

Description: Create User-Defined Choose Box Command: Creates a user-defined choose box.

CHOOSE creates a standard single-choice choose box based on the following specifications:

Variable	Function
"*prompt*"	A message that appears at the top of choose box. If "*prompt*" is an empty string (""), no message is displayed.
$\{c_1 \ldots c_n\}$	Definitions that appear within the choose box. A choice definition (c_x) can have two formats. • *obj*, any object • { $obj_{display}$ obj_{result} }, the object to be displayed followed by the result returned to the stack if that object is selected.
n_{pos}	The position number of an item definition. This item is highlighted when the choose box appears. If $n_{pos} = 0$, no item is highlighted, and the choose box can be used to view items only.

If you choose an item from the choose box and press OK, CHOOSE returns the *result* (or the object itself if no result is specified) to level 2 and 1 to level 1. If you press CANCEL, CHOOSE returns 0. Also, if $n_{pos} = 0$, CHOOSE returns 0.

Access: ← PRG IN CHOOS

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→	*obj or result*	"*1*"
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→		"*0*"

L = level; A = argument; I = item

See also: INFORM, NOVAL

CHR

Type: Command

Description: Character Command: Returns a string representing the character corresponding to the character code *n*.

The character codes are an extension of ISO 8859/1. Codes 128 through 160 are unique to the HP 49.

The default character ■ is supplied for all character codes that are *not* part of the normal HP 49 display character set.

Character code 0 is used for the special purpose of marking the end of the command line. Attempting to edit a string containing this character causes the error Can't Edit CHR(0).

You can use the CHARS application to find the character code for any character used by the HP 49. See "Entering Special Characters" in chapter 2 of the *HP 49 User's Guide*.

Access: ↰ PRG TYPE CHR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*"string"*

See also: NUM, POS, REPL, SIZE, SUB

CKSM

Type: Command

Description: Checksum Command: Specifies the error-detection scheme.

Legal values for $n_{checksum}$ are as follows:

- 1-digit arithmetic checksum.
- 2-digit arithmetic checksum.
- 3-digit cyclic redundancy check (default).

The CKSM specified is the error-detection scheme that will be requested by KGET, PKT, or SEND. If the sender and receiver disagree about the request, however, then a 1-digit arithmetic checksum will be used.

Access: (CAT) CKSM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{checksum}$	→	

See also: BAUD, PARITY, TRANSIO

CLEAR

Type: Command

Description: Clear Command: Removes all objects from the stack or history.

To recover a cleared stack or history, press (↱)(UNDO) before executing any other operation. There is no programmable command to recover the stack or history.

Access: (↱)(CLEAR)

Input/Output:

$Level_n$/Argument 1 ... Level 1/$Argument_n$		$Level_n$/Item 1 ... Level 1/$Item_n$
$obj_n ... obj_1$	→	

See also: CLVAR, PURGE

CLKADJ

Type: Command

Description: Adjust System Clock Command: Adjusts the system time by x clock ticks, where 8192 clock ticks equal 1 second.

If x is positive, x clock ticks are added to the system time. If x is negative, x clock ticks are subtracted from the system time.

Access: (↱)(TIME) TOOLS CLKADJ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	

See also: →TIME

CLLCD

Type: Command

Description: Clear LCD Command: Clears (blanks) the stack display.

The menu labels continue to be displayed after execution of CLLCD.

When executed from a program, the blank display persists only until the keyboard is ready for input. To cause the blank display to persist until a key is pressed, execute FREEZE after executing CLLCD. (When executed from the keyboard, CLLCD *automatically* freezes the display.)

Access: ↰ PRG OUT CLLCD

Input: None

Output: None

See also: DISP, FREEZE

CLOSEIO

Type: Command

Description: Close I/O Port Command: Closes the serial port, and clears the input buffer and any error messages for KERRM.

When the HP 49 turns off, it automatically closes the serial, but does not clear KERRM. Therefore, CLOSEIO is not needed to close the port, but can conserve power without turning off the calculator.

Executing HP 49 Kermit protocol commands automatically clears the input buffer; however, executing non-Kermit commands (such as SRECV and XMIT) does not.

CLOSEIO also clears error messages from KERRM. This can be useful when debugging.

Access: CAT CLOSEIO

Input: None

Output: None

See also: BUFLEN, OPENIO

CLΣ

Type: Command

Description: Purges the current statistics matrix (reserved variable ΣDAT).

Access: CAT CLΣ

Input: None

Output: None

See also: RCLΣ, STOΣ, Σ+, Σ–

CLVAR

Type: Command

Description: Clear Variables Command: Purges all variables and empty subdirectories in the current directory.

Access: CAT CLVAR

Input: None

Output: None

See also: PGDIR, PURGE

CMPLX

Type: Command

Description: Displays a menu of commands pertaining to complex numbers.

Access: CAT CMPLX

Input: None

Output: None

See also: ARIT, BASE, DIFF, EXP&LN, SOLVER, TRIGO

CNRM

Type: Command

Description: Column Norm Command: Returns the column norm (one-norm) of the array argument.

The column norm of a matrix is the maximum (over all columns) of the sum of the absolute values of all elements in each column. For a vector, the column norm is the sum of the absolute values of the vector elements. For complex arrays, the absolute value of a given element (x, y) is $\sqrt{x^2 + y^2}$.

Access: ↰ MATRICES OPERATIONS CNRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[array]	→	$x_{\text{columnnorm}}$

See also: CROSS, DET, DOT, RNRM

→COL

Type: Command

Description: Transforms a matrix into a series of column vectors and returns the vectors and a column count, or transforms a vector into its elements and returns the elements and an element count.

→COL introduces no rounding error.

Access: ↰ MTH MATRIX COL →COL

↰ MATRICES CREATE COLUMN →COL

Input/Output:

Level 1/Argument 1		Level$_{n+1}$/Item 1 ...	Level 2/Item 2	Level 1/Item 3
[[*matrix*]]	→	[*vector*]$_{\text{col1}}$	*[vector]*$_{\text{coln}}$	n_{colcount}
[*vector*]	→	$element_1$	$element_n$	$n_{\text{elementcount}}$

See also: COL→, →ROW, ROW→

COL→

Type: Command

Description: Transforms a series of column vectors and a column count into a matrix containing those columns, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

All vectors must have the same length. The column or element count is rounded to the nearest integer.

Access: ↰ MTH MATRIX COL COL→

↰ MATRICES CREATE COLUMN COL→

Input/Output:

L_{n+1}/A_1 ...	L_2/A_2	L_1/A_{n+1}		Level 1/Item 1
$[\ vector\ ]_{col1}$	$[\ vector\ ]_{coln}$	$n_{colcount}$	→	$[[\ matrix\ ]]$
$element_1$	$element_n$	$n_{elementcount}$	→	$[\ vector\ ]$

L = level; A = argument; I = item

See also: →COL, →ROW, ROW→

COL–

Type: Command

Description: Delete Column Command: Deletes column *n* of a matrix (or element *n* of a vector), and returns the modified matrix (or vector) and the deleted column (or element).

n is rounded to the nearest integer.

Access: ↰ MTH MATRIX COL COL–

↰ MATRICES CREATE COLUMN COL–

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
$[[\ matrix\ ]]_1$	n_{column}	→	$[[\ matrix\ ]]_2$	$[\ vector\ ]_{column}$
$[\ vector\ ]_1$	$n_{element}$	→	$[\ vector\ ]_2$	$element_n$

See also: COL+, CSWP, ROW+, ROW–

COL+

Type: Command

Description: Insert Column Command: Inserts an array (vector or matrix) into a matrix (or one or more elements into a vector) at the position indicated by n_{index}, and returns the modified array.

The inserted array must have the same number of rows as the target array.

n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and to the right of the insertion point are shifted to the right.

Access: ↰ MTH MATRIX COL COL+

↰ MATRICES CREATE COLUMN COL+

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$[[\ matrix\ ]]_2$	n_{index}	→	$[[\ matrix\ ]]_3$
$[[\ matrix\ ]]_1$	$[\ vector\ ]_{column}$	n_{index}	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{element}$	n_{index}	→	$[\ vector\ ]_2$

See also: COL–, CSWP, ROW+, ROW–

COLCT

Type: Command

Description: Factorizes a polynomial or an integer. Identical to FACTOR (see volume 1, CAS Commands).

Access: CAT COLCT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$'symb_1'$	→	$'symb_2'$
x	→	x
(x, y)	→	(x, y)

See also: EXPAN, FACTOR, ISOL, QUAD, SHOW

COLΣ

Type: Command

Description: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable ΣDAT).

COLΣ combines the functionality of XCOL and YCOL. The independent-variable column number x_{xcol} is stored as the first parameter in the reserved variable ΣPAR (the default is 1). The dependent-variable column number x_{ycol} is stored as the second parameter in the reserved variable ΣPAR (the default is 2).

COLΣ accepts and stores noninteger values, but subsequent commands that use these two parameters in ΣPAR will cause errors.

Access: CAT COLΣ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{col}	y_{col}	→	

See also: BARPLOT, BESTFIT, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL, YCOL

COMB

Type: Function

Description: Combinations Function: Returns the number of possible combinations of *n* items taken *m* at a time.

The following formula is used:

$$C_{n,m} = \frac{n!}{m! \cdot (n-m)!}$$

The arguments *n* and *m* must each be less than 10^{12}. If $n<m$, zero is returned.

Access: ↰ MTH PROBABILITY COMB

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*m*	→	$C_{n;m}$
'*symb*$_n$'	*m*	→	'*COMB(symb*$_n$*,m)*'
n	'*symb*$_m$'	→	'*COMB(n, symb*$_m$*)*'

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_n$'	'$symb_m$'	→	'$COMB(symb_n,symb_m)$'

See also: PERM, !

CON

Type: Command

Description: Constant Array Command: Returns a constant array, defined as an array whose elements all have the same value.

The constant value is a real or complex number taken from argument 2/level 1. The resulting array is either a new array, or an existing array with its elements replaced by the constant, depending on the object in argument 1/level 2.

- Creating a new array: If argument 1/level 2 contains a list of one or two integers, CON returns a new array. If the list contains a single integer $n_{columns}$, CON returns a constant vector with n elements. If the list contains two integers n_{rows} and $m_{columns}$, CON returns a constant matrix with n rows and m columns.
- Replacing the elements of an existing array: If argument 1/level 2 contains an array, CON returns an array of the same dimensions, with each element equal to the constant. If the constant is a complex number, the original array must also be complex.
  If argument 1/level 2 contains a name, the name must identify a variable that contains an array. In this case, the elements of the array are replaced by the constant. If the constant is a complex number, the original array must also be complex.

Access: ↰ MTH MATRIX MAKE CON

↰ MATRICES CREATE ON

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ $n_{columns}$ }	$z_{constant}$	→	[$vector_{constant}$]
{ n_{rows} $m_{columns}$ }	$z_{constant}$	→	[[$matrix_{constant}$]]
[R-*array*]	$x_{constant}$	→	[R-$array_{constant}$]
[C-*array*]	$z_{constant}$	→	[C-$array_{constant}$]
'*name*'	$z_{constant}$	→	

See also: IDN

COND

Type: Command

Description: Condition Number Command: Returns the 1-norm (column norm) condition number of a square matrix.

The condition number of a matrix is the product of the norm of the matrix and the norm of the inverse of the matrix. COND uses the 1-norm and computes the condition number of the matrix without computing the inverse of the matrix.

The condition number expresses the sensitivity of the problem of solving a system of linear equations having coefficients represented by the elements of the matrix (this includes inverting the matrix). That is, it indicates how much an error in the inputs may be magnified in the outputs of calculations using the matrix.

In many linear algebra computations, the base 10 logarithm of the condition number of the matrix is an estimate of the number of digits of precision that might be lost in computations using that matrix. A reasonable rule of thumb is that the number of digits of accuracy in the result is approximately MIN(12,15–$\log_{10}$(COND)).

Access: ↰ MTH MATRIX NORMALIZE COND

↰ MATRICES OPERATIONS COND

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[\ matrix\ ]]_{m\times n}$	→	$\varkappa_{conditionnumber}$

See also: SNRM, SRAD, TRACE

CONIC

Type: Command

Description: Conic Plot Type Command: Sets the plot type to CONIC.

When the plot type is CONIC, the DRAW command plots the current equation as a second-order polynomial of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

$$\{\ (x_{min}, y_{min})\ (x_{max}, y_{max})\ \textit{indep res axes ptype depend}\ \}$$

For plot type CONIC, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5, –3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5, 3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes, or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

The current equation is used to define a pair of functions of the independent variable. These functions are derived from the second-order Taylor's approximation to the current equation. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max}) (the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

Access: (CAT) CONIC

Input: None

Output: None

See also: BAR, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

CONJ

Type: Function

Description: Conjugate Analytic Function: Conjugates a complex number or a complex array.

Conjugation is the negation (sign reversal) of the imaginary part of a complex number. For real numbers and real arrays, the conjugate is identical to the original argument.

Access: ↱ CMPLX CONJ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*x*
(x, y)	→	*(x, –y)*
[R-array]	→	*[R-array]*
[C-array]$_1$	→	*[C-array]*$_2$
'symb'	→	*'CONJ(symb)'*

See also: ABS, IM, RE, SCONJ, SIGN

CONLIB

Type: Command

Description: Open Constants Library Command: Opens the Constants Library catalog.

Access: APPS CONSTANTS LIBRARY

Input: None

Output: None

See also: CONST

CONST

Type: Function

Description: Constant Value Command: Returns the value of a constant.

CONST returns the value of the specified constant. It chooses the unit type depending on flag –60: SI if clear, English if set, and uses the units depending on flag –61: units if clear, no units if set.

Access: CAT CONST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	*x*

See also: CONLIB

CONT

Type: Command

Description: Continue Program Execution Command: Resumes execution of a halted program.

Since CONT is a command, it can be assigned to a key or to a custom menu.

Access: ↩ CONT

Input: None

Output: None

See also: HALT, KILL, PROMPT

CONVERT

Type: Command

Description: Convert Units Command: Converts a source unit object to the dimensions of a target unit.

The source and target units must be compatible. The number part x_2 of the target unit object is ignored.

Access: ↩ CONVERT UNITS TOOLS CONVERT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_1\_units_{source}$	$x_2\_units_{target}$	→	$x_3\_units_{target}$

See also: UBASE, UFACT, →UNIT, UVAL

CORR

Type: Command

Description: Correlation Command: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable Σ*DAT*).

The columns are specified by the first two elements in the reserved variable Σ*PAR*, set by XCOL and YCOL, respectively. If Σ*PAR* does not exist, CORR creates it and sets the elements to their default values (1 and 2).

The correlation is computed with the following formula:

$$\frac{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})(x_{in_2}-\bar{x}_{n_2})}{\sqrt{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})^2\sum_{i=1}^{n}(x_{in_2}-\bar{x}_{n_2})^2}}$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\bar{x}_{n_1}$ is the mean of the data in column n_1, $\bar{x}_{n_2}$ is the mean of the data in column n_2, and n is the number of data points.

Access: CAT CORR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{\text{correlation}}$

Flags: None

See also: COLΣ, COV, PREDX, PREDY, XCOL, YCOL

COS

Type: Analytic Function

Description: Cosine Analytic Function: Returns the cosine of the argument.

For real arguments, the current angle mode determines the number's interpretation as an angle, unless the angular units are specified.

For complex arguments, $\cos(x + iy) = \cos x \cosh y - i \sin x \sinh y$.

If the argument for COS is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing COS with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Access: COS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	cos z
'*symb*'	→	'*COS(symb)*'
x_unit_{angular}	→	$cos\ (x\_unit_{\text{angular}})$

See also: ACOS, SIN, TAN

COSH

Type: Analytic Function

Description: Hyperbolic Cosine Analytic Function: Returns the hyperbolic cosine of the argument.

For complex arguments, $\cosh(x + iy) = \cosh x \cos y + i \sinh x \sin y$.

Access: ↱ TRIG HYPERBOLIC COSH

↰ MTH HYPERBOLIC COSH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	cosh z
'*symb*'	→	'*COSH(symb)*'

See also: ACOSH, SINH, TANH

COV

Type: Command

Description: Covariance Command: Returns the sample covariance of the independent and dependent data columns in the current statistics matrix (reserved variable Σ*DAT*).

The columns are specified by the first two elements in reserved variable Σ*PAR*, set by XCOL and YCOL respectively. If Σ*PAR* does not exist, COV creates it and sets the elements to their default values (1 and 2).

The covariance is calculated with the following formula:

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_{in_1}-\overline{x_{n_1}})(x_{in_2}-\overline{x_{n_2}})$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

Access: CAT COV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{\text{covariance}}$

See also: COLΣ, CORR, PCOV, PREDX, PREDY, XCOL, YCOL

CR

Type: Command

Description: Carriage Right Command: Prints the contents, if any, of the printer buffer.

CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable *PRTPAR*.

Access: CAT CR

Input: None

Output: None

See also: DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

CRDIR

Type: Command

Description: Create Directory Command: Creates an empty subdirectory with the specified name in the current directory.

CRDIR does not change the current directory; evaluate the name of the new subdirectory to make it the current directory.

Access: ↰ PRG MEMORY DIRECTORY CRDIR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'global'	→	

See also: HOME, PATH, PGDIR, UPDIR

CROSS

Type: Command

Description: Cross Product Command: CROSS returns the cross product C = A × B of vectors A and B.

The arguments must be vectors having two or three elements, and need not have the same number of elements. (The HP 49 automatically converts a two-element argument [d_1, d_2] to a three-element argument [d_1, d_2, 0].)

Access: ↰ MTH VECTOR CROSS

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\lceil$ *vector* $\rceil_A$	$\lceil$ *vector* $\rceil_B$	→	$\lceil$ *vector* $\rceil_{A \times B}$

See also: CNRM, DET, DOT, RNRM

CSWP

Type: Command

Description: Column Swap Command: Swaps columns *i* and *j* of the argument matrix and returns the modified matrix, or swaps elements *i* and *j* of the argument vector and returns the modified vector.

Column numbers are rounded to the nearest integer. Vector arguments are treated as row vectors.

Access: ↰ MATRICES CREATE COLUMN CSWP

↰ MTH MATRIX COL CSWP

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$n_{columni}$	$n_{columnj}$	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{elementi}$	$n_{elementj}$	→	$[\ vector\ ]_2$

See also: COL+, COL–, RSWP

CYLIN

Type: Command

Description: Cylindrical Mode Command: Sets Cylindrical coordinate mode.

CYLIN clears flag –15 and sets flag –16.

In Cylindrical mode, vectors are displayed as polar components.

Access: CAT CYLIN

Input: None

Output: None

See also: RECT, SPHERE

C→PX

Type: Command

Description: Complex to Pixel Command: Converts the specified user-unit coordinates to pixel coordinates.

The user-unit coordinates are derived from the (x_{min},y_{min}) and (x_{max},y_{max}) parameters in the reserved variable *PPAR*.

Access: ↰ PRG PICT C→PX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	{ #*n*, #*m* }

See also: PX→C

C→R

Type: Command

Description: Complex to Real Command: Separates the real and imaginary parts of a complex number or complex array.

The result in item 1/level 2 represents the real part of the complex argument. The result in item 2/ level 1 represents the imaginary part of the complex argument.

Access: ↰ PRG TYPE C→R

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
(x, y)	→	*x*	*y*
[C-array]	→	*[R-array]*$_1$	*[R-array]*$_2$

See also: R→C, RE, IM

DARCY

Type: Function

Description: Darcy Friction Factor Function: Calculates the Darcy friction factor of certain fluid flows.

DARCY calculates the Fanning friction factor and multiplies it by 4. $x_{e/D}$ is the relative roughness – the ratio of the conduit roughness to its diameter. y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{e/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Access: CAT DARCY

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_e / D	y_{Re}	→	x_{Darcy}

See also: FANNING

DATE

Type: Command

Description: Date Command: Returns the system date.

Access: ↱ TIME TOOLS DATE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*date*

See also: DATE+, DDAYS, TIME, TSTR

→DATE

Type: Command

Description: Set Date Command: Sets the system date to *date*.

date has the form *MM.DDYYYY* or *DD.MMYYYY*, depending on the state of flag –42. *MM* is month, *DD* is day, and *YYYY* is year. If *YYYY* is not supplied, the current specification for the year is used. The range of allowable dates is January 1, 1991 to December 31, 2090.

Access: ↱ TIME TOOLS →DATE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
date	→	

See also: →TIME

DATE+

Type: Command

Description: Date Addition Command: Returns a past or future date, given a date in argument 1/level 2 and a number of days in argument 2/level 1.

If x_{days} is negative, DATE+ calculates a past date. The range of allowable dates is October 15, 1582, to December 31, 9999.

Access: ↱ TIME TOOLS DATE+

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$date_1$	x_{days}	→	$date_{new}$

See also: DATE, DDAYS

DBUG

Type: Operation

Description: Debug Operation: Starts program execution, then suspends it as if HALT were the first program command.

DBUG is not programmable.

Access: CAT DEBUG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
« *program* » or '*program name*'	→	

See also: HALT, NEXT

DDAYS

Type: Command

Description: Delta Days Command: Returns the number of days between two dates.

If the argument 1/level 2 date is chronologically later than the argument 2/ level 1 date, the result is negative. The range of allowable dates is October 15, 1582, to December 31, 9999.

Access: ↱ TIME TOOLS DDAYS

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$date_1$	$date_2$	→	x_{days}

See also: DATE, DATE+

DEC

Type: Command

Description: Decimal Mode Command: Selects decimal base for binary integer operations. (The default base is decimal.)

Binary integers require the prefix #. Binary integers entered and returned in decimal base automatically show the suffix *d*. If the current base is not decimal, then you can enter a decimal number by ending it with d. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: CAT DEC

Input: None

Output: None

See also: BIN, HEX, OCT, RCWS, STWS

DECR

Type: Command

Description: Decrement Command: Takes a variable, subtracts 1, stores the new value back into the original variable, and returns the new value.

The contents of *name* must be a real number or an integer.

Access: ← PRG MEMORY ARITHMETIC DECR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	x_{new}

See also: INCR, STO+, STO–

DEFINE

Type: Command

Description: Define Variable or Function Command: Stores the expression on the right side of the = in the variable specified on the left side, or creates a user-defined function.

If the left side of the equation is *name* only, DEFINE stores *exp* in the variable *name*.

If the left side of the equation is *name* followed by parenthetical arguments $name_1$ … $name_n$, DEFINE creates a user-defined function and stores it in the variable *name*.

Access: ↰ DEF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name=exp*'	→	
'$name(name_1 \ldots name_n)=exp(name_1 \ldots name_n)$'	→	

See also: STO

DEG

Type: Command

Description: Degrees Command: Sets Degrees angle mode.

DEG clears flags –17 and –18, and clears the RAD and GRAD annunciators.

In Degrees angle mode, real-number arguments that represent angles are interpreted as degrees, and real-number results that represent angles are expressed in degrees.

Access: CAT DEG

Input: None

Output: None

See also: GRAD, RAD

DELALARM

Type: Command

Description: Delete Alarm Command: Deletes the specified alarm.

If n_{index} is 0, all alarms in the system alarm list are deleted.

Access: ↱ TIME TOOLS ALRM DELALARM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{index}	→	

Flags: None

See also: FINDALARM, RCLALARM, STOALARM

DELAY

Type: Command

Description: Delay Command: Specifies how many seconds the HP 49 waits between sending lines of information to the printer.

x_{delay} specifies the delay time in seconds. The default delay is 1.8 seconds. The maximum delay is 6.9 seconds. (The sign of x_{delay} is ignored, so –4 DELAY is equivalent to 4 DELAY.)

The delay setting is the first parameter in the reserved variable *PRTPAR*.

A shorter delay setting can be useful when the HP 49 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay *shorter* than the time to print one line, you may lose information.

Access: CAT DELAY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{delay}	→	

See also: CR, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

DELKEYS

Type: Command

Description: Delete Key Assignments Command: Clears user-defined key assignments.

The argument x_{key} is a real number *rc.p* specifying the key by its *r*ow number, its *c*olumn number, and its *p*lane (shift). For a definition of plane, see ASN.

Specifying 0 for x_{key} clears *all* user key assignments and restores the standard key assignments.

Specifying S as the argument for DELKEYS suppresses all standard key assignments on the user keyboard. This makes keys without user key assignments inactive on the user keyboard. (You can make exceptions using ASN, or restore them all using STOKEYS.) If you are stuck in User mode – probably with a "locked" keyboard – because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt ("warm start"): press and hold ON and F3 simultaneously, releasing F3 first. This cancels User mode.

Deleted user key assignments still take up from 2.5 to 15 bytes of memory each. You can free this memory by packing your user key assignments by executing RCLKEYS 0 DELKEYS STOKEYS.

Access: (CAT) DELKEYS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\mathcal{X}_{key}$	→	
$\{ \mathcal{X}_{key1}, \ldots, \mathcal{X}_{key\ n} \}$	→	
0	→	
'*S*'	→	

See also: ASN, RCLKEYS, STOKEYS

DEPND

Type: Command

Description: Dependent Variable Command: Specifies the dependent variable (and its plotting range for TRUTH plots).

The specification for the dependent variable name and its plotting range is stored in the reserved variable *PPAR* as follows:

- If the argument is a global variable name, that name replaces the dependent variable entry in *PPAR*.
- If the argument is a list containing a global name, that name replaces the dependent variable name but leaves unchanged any existing plotting range.
- If the argument is a list containing a global name and two real numbers, or a list containing a name, array, and real number, that list replaces the dependent variable entry.
- If the argument is a list containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the dependent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *Y*.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Access: (CAT) DEPND

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
	'global'	→	
	{ *global* }	→	
	{ *global*, y_{start}, y_{end} }	→	
	{y_{start}, y_{end} }	→	
y_{start}	y_{end}	→	

See also: INDEP

DEPTH

Type: RPN Command

Description: Depth Command: Returns a real number representing the number of objects present on the stack (before DEPTH was executed).

Access: ⮌ (PRG) STACK DEPTH

Input/Output:

$Level_n$...Level 1		Level 1
	→	*n*

DET

Type: Command

Description: Determinant Function: Returns the determinant of a square matrix.

The argument matrix must be square. DET computes the determinant of 1 × 1 and 2 × 2 matrices directly from the defining expression for the determinant. DET computes the determinant of a larger matrix by computing the Crout LU decomposition of the matrix and accumulating the product of the decomposition's diagonal elements.

Since floating-point division is used to do this, the computed determinant of an integer matrix is often not an integer, even though the actual determinant of an integer matrix must be an integer. DET corrects this by rounding the computed determinant to an integer value. This technique is also used for noninteger matrices with determinants having fewer than 15 nonzero digits: the computed determinant is rounded at the appropriate digit position to restore some or all of the accuracy lost to floating-point arithmetic.

This refining technique can cause the computed determinant to exhibit discontinuity. To avoid this, you can disable the refinement by setting flag –54.

Access: ↰ MATRICES OPERATIONS DET

↰ MTH NORMALIZE DET

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[matrix]]	→	$x_{\text{determinant}}$

See also: CNRM, CROSS, DOT, RNRM

DETACH

Type: Command

Description: Detach Library Command: Detaches the library with the specified number from the current directory. Each library has a unique number. If a port number is specified, it is ignored.

A RAM-based library object attached to the HOME directory must be detached before it can be purged, whereas a library attached to any other directory does not. Also, a library object attached to a non-*HOME* directory is *automatically* detached (without using DETACH) whenever a new library object is attached there.

Access: CAT DETACH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{library}	→	
$:n_{\text{port}}:n_{\text{library}}$	→	

See also: ATTACH, LIBS, PURGE

DIAG→

Type: Command

Description: Vector to Matrix Diagonal Command: Takes an array and a specified dimension and returns a matrix whose major diagonal elements are the elements of the array.

Real number dimensions are rounded to integers. If a single dimension is given, a square matrix is returned. If two dimensions are given, the proper order is { *number of rows, number of columns* }. No more than two dimensions can be specified.

If the main diagonal of the resulting matrix has more elements than the array, additional diagonal elements are set to zero. If the main diagonal of the resulting matrix has fewer elements than the array, extra array elements are dropped.

Access: ↰ MATRICES CREATE DIAG→

↰ MTH MATRIX MAKE DIAG→

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array]$_{\text{diagonals}}$	{ *dim* }	→	*[[matrix]]*

→DIAG

Type: Command

Description: Matrix Diagonal to Array Command: Returns a vector that contains the major diagonal elements of a matrix.

The input matrix does not have to be square.

Access: ↰ MATRICES CREATE →DIAG

↰ MTH MATRIX MAKE →DIAG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[*vector*]$_{\text{diagonals}}$

See also: DIAG→

DIFF

Type: Command

Description: Displays a menu of calculus commands.

Access: CAT DIFF

Input: None

Output: None

See also: ARIT, BASE, CMPLX, EXP&LN, SOLVER, TRIGO

DIFFEQ

Type: Command

Description: Differential Equation Plot Type Command: Sets the plot type to DIFFEQ.

When the plot type is DIFFEQ and the reserved variable *EQ* does not contain a list, the initial value problem is solved and plotted over an interval using the Runge–Kutta–Fehlberg (4,5) method. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$$\{ (x_{min}, y_{min})\ (x_{max}, y_{max})\ \textit{indep res axes ptype depend} \}$$

For plot type DIFFEQ, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a list, $\{ X\ x_0\ x_f \}$, containing a name that specifies the independent variable, and two numbers that specify the initial and final values for the independent variable. The default values for these elements are $\{ X\ 0\ x_{max} \}$.
- *res* is a real number specifying the maximum interval, in user-unit coordinates, between values of the independent variable. The default value is 0. If *res* does not equal zero, then the maximum interval is *res*. If *res* equals zero, the maximum interval is unlimited.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. If

the solution is real-valued, these strings can specify the dependent or the independent variable; if the solution is vector valued, the strings can specify a solution component:

- 0 specifies the dependent variable (X)
- 1 specifies the dependent variable (Y)
- *n* specifies a solution component Y_n

If *axes* contains any strings other than 0, 1 or *n*, the DIFFEQ plotter uses the default strings 0 and 1, and plots the independent variable on the horizontal axis and the dependent variable on the vertical.

- *ptype* is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in *PPAR*.
- *depend* is a list, { Y y_0 $\varkappa_{ErrTol}$ }, containing a name that specifies the dependent variable (the solution), and two numbers that specify the initial value of Y and the global absolute error tolerance in the solution Y. The default values for these elements are { Y 0 .0001 }

EQ must define the right-hand side of the initial value problem Y'(XF(X,Y). Y can return a real value or real vector when evaluated.

The DIFFEQ-plotter attempts to make the interval between values of the independent variable as large as possible, while keeping the computed solution within the specified error tolerance $\varkappa_{ErrTol}$. This tolerance may hold only at the computed points. Straight lines are drawn between computed step endpoints, and these lines may not accurately represent the actual shape of the solution. *res* limits the maximum interval size to provide higher plot resolution.

On exit from DIFFEQ plot, the first elements of *indep* and *depnd* (identifiers) contain the final values of X and Y, respectively.

If *EQ* contains a list, the initial value problem is solved and plotted using a combination of Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods. In this case DIFFEQ uses RRKSTEP to calculate y_f, and *EQ* must contain two additional elements:

- The second element of *EQ* must evaluate to the partial derivative of Y' with respect to X, and can return a real value or real vector when evaluated.
- The third element of *EQ* must evaluate to the partial derivative of Y' with respect to Y, and can return a real value or a real matrix when evaluated.

Access: CAT DIFFEQ

Input: None

Output: None

See also: AXES, CONIC, FUNCTION, PARAMETRIC, POLAR, RKFSTEP, RRKSTEP, TRUTH

DISP

Type: Command

Description: Display Command: Displays *obj* in the *n*th display line.

$n \leq 1$ indicates the top line of the display; $n \geq 7$ indicates the bottom line.

To facilitate the display of messages, strings are displayed without the surrounding " " delimiters. All other objects are displayed in the same form as would be used if the object were in level 1 in the multiline display format. If the object display requires more than one display line, the display starts in line *n*, and continues down the display either to the end of the object or the bottom of the display.

The object displayed by DISP persists in the display only until the keyboard is ready for input. The FREEZE command can be used to cause the object to persist in the display until a key is pressed.

Access: ↰ PRG OUT DISP

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	*n*	→	

See also: FREEZE, HALT, INPUT, PROMPT

DO

Type: Command Operation

Description: DO Indefinite Loop Structure Command: Starts DO … UNTIL … END indefinite loop structure.

DO … UNTIL … END executes a loop repeatedly until a test returns a true (nonzero) result. Since the test clause is executed after the loop clause, the loop is always executed at least once. The syntax is:

DO *loop-clause* UNTIL *test-clause* END

DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from

the stack. If its value is zero, the loop clause is executed again; otherwise, execution resumes following END.

Access: ⮌ PRG BRANCH DO

Input/Output:

Level 1/Argument 1			Level 1/Item 1
DO		→	
UNTIL		→	
END	*T/F*	→	

See also: END, UNTIL, WHILE

DOERR

Type: Command

Description: Do Error Command: Executes a "user-specified" error, causing a program to behave exactly as if a normal error had occurred during program execution.

DOERR causes a program to behave exactly as if a normal error has occurred during program execution. The error message depends on the argument provided to DOERR:

- n_{error} or $\#n_{error}$ display the corresponding built-in error message.
- "*error*" displays the contents of the string. (A subsequent execution of ERRM returns "*error*". ERRN returns # 70000h.)
- 0 abandons program execution without displaying a message. 0 DOERR is equivalent to pressing CANCEL.

Access: ⮌ PRG ERROR DOERR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{error}	→	
$\#n_{error}$	→	
"error"	→	
0	→	

See also: ERRM, ERRN, ERR0

DOLIST

Type: Command

Description: Do to List Command: Applies commands, programs, or user-defined functions to lists.

The number of lists, *n*, can be omitted when the first or level 1 argument is any of the following:

- A command.
- A program containing exactly one command (e.g. «DUP»)
- A program conforming to the structure of a user-defined function.

The final argument 1 (or level 1 object) can be a local or global name that refers to a program or command.

All lists must be the same length *l*. The program is executed *l* times: on the *i*th iteration, *n* objects each taken from the *i*th position in each list are entered on the stack in the same order as in their original lists, and the program is executed. The results from each execution are left on the stack. After the final iteration, any new results are combined into a single list.

Access: ↰ PRG LIST PROCEDURES DOLIST

Input/Output:

L_{n+2}/A_1 ... L_3/A_{n-2}	L_2/A_{n+1}	L_1/A_{n+2}		Level 1/Item 1
{ *list* }$_1$ ... { *list* }$_n$	*n*	*« program »*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*command*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*name*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*« program »*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*command*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*name*	→	{ *results* }

L = level; A = argument

See also: DOSUBS, ENDSUB, NSUB, STREAM

DOSUBS

Type: Command

Description: Do to Sublist Command: Applies a program or command to groups of elements in a list.

The real number *n* can be omitted when the first argument is one of the following:

- A command.
- A user program containing a single command.
- A program with a user-defined function structure.
- A global or local name that refers to one of the above.

The first iteration uses elements 1 through *n* from the list; the second iteration uses elements 2 through *n* + 1; and so on. In general, the m^{th} iteration uses the elements from the list corresponding to positions *m* through *m* + *n* – 1.

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the command ENDSUB. Both of these commands return an Undefined Local Name error if executed when DOSUBS is not active.

DOSUBS returns the Invalid User Function error if the object at level 1/argument 3 is a user program that does not contain only one command and does not have a user-defined function structure. DOSUBS also returns the Wrong Argument Count error if the object at level 1/argument 3 is a command that does not accept 1 to 5 arguments of specific types (DUP, ROT, or →LIST, for example).

Access: ⬅ PRG LIST PROCEDURES DOSUBS

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
{ *list* }$_1$	*n*	« *program* »	→	{ *list* }$_2$
{ *list* }$_1$	*n*	*command*	→	{ *list* }$_2$
{ *list* }$_1$	*n*	*name*	→	{ *list* }$_2$
	{ *list* }$_1$	« *program* »	→	{ *list* }$_2$
	{ *list* }$_1$	*command*	→	{ *list* }$_2$
	{ *list* }$_1$	*name*	→	{ *list* }$_2$

See also: DOLIST, ENDSUB, NSUB, STREAM

DOT

Type: Command

Description: Dot Product Command: Returns the dot product A·B of two arrays A and B, calculated as the sum of the products of the corresponding elements of the two arrays.

Both arrays must have the same dimensions.

Some authorities define the dot product of two complex arrays as the sum of the products of the conjugated elements of one array with their corresponding elements from the other array. The HP 49 uses the ordinary products without conjugation. If you prefer the alternative definition, apply CONJ to one array before using DOT.

Access: ↰ MATRICES VECTOR DOT

↰ MTH VECTOR DOT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array A]	*[array B]*	→	*x*

See also: CNRM, CROSS, DET, RNRM

DRAW

Type: Command Operation

Description: Draw Plot Command: Plots the mathematical data in the reserved variable *EQ* or the statistical data in the reserved variable Σ*DAT*, using the specified *x*- and *y*-axis display ranges.

The plot type determines if the data in the reserved variable *EQ* or the data in the reserved variable Σ*DAT* is plotted.

DRAW does not erase *PICT* before plotting; execute ERASE to do so. DRAW does not draw axes; execute DRAX to do so.

When DRAW is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Access: CAT DRAW

Input: None

Output: None

See also: AUTO, AXES, DRAX, ERASE, FREEZE, PICTURE, LABEL, PVIEW

DRAW3DMATRIX

Type: Command

Description: Draws a 3D plot from the values in a specified matrix.

The number of rows indicates the number of units along the x axis, the number of columns indicates the number of units along the y axis, and the values in the matrix give the magnitudes of the plotted points along the z axis. In other words, the coordinates of a plotted point are (r, c, v) where r is the row number, c the column number and v the value in the corresponding cell of the matrix.

You can limit the points that are plotted by specifing a minimum value (v_{min}) and a maximum value (v_{max}). Values in the matrix outside this range are not plotted. If all values are included, the total number of points plotted is $r \times c$.

Once the plot has been drawn, you can rotate it in various ways by pressing the following keys:

- ▶ and ◀ rotate the plot around the x axis (in different directions)
- ▲ and ▼ rotate the plot around the y axis (in different directions)
- TOOL and NXT rotate the plot around the z axis (in different directions)

Access: CAT FAST3DMATRIX

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[*matrix*]]	v_{min}	v_{max}	→	

See also: FAST3D

DRAX

Type: Command

Description: Draw Axes Command: Draws axes in *PICT*.

The coordinates of the axes intersection are specified by AXES. Axes tick-marks are specified in PPAR with the ATICK, or AXES command. DRAX does not draw axes labels; execute LABEL to do so.

Access: CAT DRAX

Input: None

Output: None

See also: AXES, DRAW, LABEL

DROP

Type: RPN Command

Description: Drop Object Command: Removes the level 1 object from the stack.

Access: ↰ PRG STACK DROP

Input/Output:

Level 1		Level 1
obj	→	

See also: CLEAR, DROPN, DROP2

DROP2

Type: RPN Command

Description: Drop 2 Objects Command: Removes the first two objects from the stack.

Access: ↰ PRG STACK DROP2

Input/Output:

Level 2	Level 1		Level 1
obj_1	obj_2	→	

See also: CLEAR, DROP, DROPN

DROPN

Type: RPN Command

Description: Drop n Objects Command: Removes the first $n + 1$ objects from the stack (the first n objects excluding the integer n itself).

Access: ↰ PRG STACK DROPN

Input/Output:

$Level_{n+1}$ ... Level 2	Level 1		Level 1
obj1 ... obj_n	*n*	→	

See also: CLEAR, DROP, DROP2

DTAG

Type: Command

Description: Delete Tag Command: DTAG removes all tags (labels) from an object.

The leading colon is not shown for readability when the tagged object is on the stack.

DTAG has no effect on an untagged object.

Access: ⮌ PRG TYPE DTAG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
tag:obj	→	*obj*

See also: LIST→, →TAG

DUP

Type: RPN Command

Description: Duplicate Object Command: DUP returns a copy of the argument (or the object on level 1).

Access: ⮌ PRG STACK DUP

Input/Output:

Level 1		Level 2	Level 1
obj	→	*obj*	*obj*

See also: DUPN, DUP2, PICK

DUP2

Type: RPN Command

Description: Duplicate 2 Objects Command: DUP2 returns copies of the two objects on levels 1 and 2 of the stack.

Access: ⮌ PRG STACK DUP2

Input/Output:

L_2	L_1		L_4	L_3	L_2	L_1
obj_2	obj_1	→	obj_2	obj_1	obj_2	obj_1

L = level

See also: DUP, DUPN, PICK

DUPDUP

Type: RPL Command

Description: Duplicates an object twice.

Access: PRG STACK DUPDUP

Input/Output:

Level 1		Level 3	Level 2	Level 1
obj	→	*obj*	*obj*	*obj*

See also: DUP, NDUPN, DUPN, DUP2

DUPN

Type: RPN Command

Description: Duplicate n Objects Command: Takes an integer *n* from level 1 of the stack, and returns copies of the objects on stack levels 2 through *n* + 1.

Access: ↰ PRG STACK DUPN

Input/Output:

L_{i+1}	L_i ...L_3	L_2	L_1		L_{i+n}	L_{i+n-1} ... L_2	L_1
obj_1	obj_2 ... obj_{i-1}	obj_i	n	→	obj_1	obj_2 ... obj_{i-1}	obj_i

L = level

See also: DUP, DUP2, PICK

D→R

Type: Function

Description: Degrees to Radians Function: Converts a real number representing an angle in degrees to its equivalent in radians.

This function operates independently of the angle mode.

Access: ↰ MTH REAL D→R

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*(π/180)x*
'symb'	→	*'D→R(symb)'*

See also: R→D

E to F

e

Type: Function

Description: e Function: Returns the symbolic constant *e* or its numerical representation, 2.71828182846.

The number returned for *e* is the closest approximation to 12-digit accuracy. For exponentiation, use the expression EXP(*x*) rather than *e*^*x*, since the function EXP uses a special algorithm to compute the exponential to greater accuracy.

Access: ALPHA ◁ E

◁ MTH CONSTANTS *e*

◁ MTH CONSTANTS 2.718281828…

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*'e'*
	→	*2.71828182846*

See also: EXP, EXPM, *i*, LN, LNP1, MAXR, MINR, π

EDIT

Type: Command

Description: Edit Command: Moves specified object to the command line where it can be edited.

Access: CAT EDIT

See also: VISIT

EDITB

Type: Command

Description: Edit Command: Opens the specified object in the most suitable editor. For example, if you use a matrix as the specified object, the command opens it in Matrix Writer.

Access: CAT EDIT

See also: VISIT

EGV

Type: Command

Description: Eigenvalues and Eigenvectors Command: Computes the eigenvalues and right eigenvectors for a square matrix.

The resulting vector EVal contains the computed eigenvalues. The columns of matrix EVec contain the right eigenvectors corresponding to the elements of vector EVal.

The computed results should minimize (within computational precision):

$$\frac{|A \cdot EVec - EVec \cdot diag(EVal)|}{n \cdot |A|}$$

where diag (*EVal*) denotes the $n \times n$ diagonal matrix containing the eigenvalues *EVal*.

Access: ↰ MATRICES EIGENVECTOR EGV

↰ MTH MATRIX EGV

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
$[[matrix]]_{A}$	→	$[[matrix]]_{EVec}$	$[vector]_{EVal}$

See also: EGVL

EGVL

Type: Command

Description: Eigenvalues Command: Computes the eigenvalues of a square matrix.

The resulting vector L contains the computed eigenvalues.

Access: ↰ MATRICES EIGENVECTOR EGVL

↰ MTH MATRIX EGVL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[matrix]]_{A}$	→	$[vector]_{EVal}$

See also: EGV

ELSE

Type: Command

Description: ELSE Command: Starts false clause in conditional or error-trapping structure.

See the IF and IFERR keyword entries for more information.

Access: ↰ PRG BRANCH ELSE

Input: None

Output: None

See also: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE

END

Type: Command

Description: END Command: Ends conditional, error-trapping, and indefinite loop structures.

See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

Access: ↰ PRG BRANCH END

Input: None

Output: None

See also: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE

ENDSUB

Type: Command

Description: Ending Sublist Command: Provides a way to access the total number of sublists contained in the list used by DOSUBS.

Returns an Undefined Local Name error if executed when DOSUBS is not active.

Access: ↰ PRG LIST PROCEDURES ENDSUB

Input: None

Output: None

See also: DOSUBS, NSUB

ENG

Type: Command

Description: Engineering Mode Command: Sets the number display format to engineering mode, which displays one to three digits to the left of the fraction mark (decimal point) and an exponent that is a multiple of three. The total number of significant digits displayed is $n + 1$.

Engineering mode uses $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for n outside this range are rounded up or down.) A number is displayed or printed as follows:

(*sign*) *mantissa* E (*sign*) *exponent*

where the mantissa is of the form $(nn)n.(n\ldots)$ (with up to 12 digits total) and the exponent has one to three digits.

A number with an exponent of –499 is displayed automatically in scientific mode.

Access: CAT ENG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	

See also: FIX, SCI, STD

EQW

Type: Command

Description: Opens Equation Writer, where you can edit an expression.

If the object you specify—or the object on level 1—is not the type of object that can be edited in Equation Writer, EQW places the object on the command line, where you edit it.

Access: CAT EQW

Input/Output:

Level 1/Argument 1		Level 1/Item 1
exp_1	→	exp_2

See also: EDIT, EDITB, VISIT, VISITB

EQ→

Type: Command

Description: Equation to Stack Command: Separates an equation into its left and right sides.
If the argument is an expression, it is treated as an equation whose right side equals zero.

Access: ↰ PRG TYPE EQ→

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
'$symb_1$=$symb_2$'	→	'$symb_1$'	'$symb_2$'
z	→	z	*0*
'*name*'	→	'*name*'	*0*
'*x_unit*'	→	'*x_unit*'	*0*
'*symb*'	→	'*symb*'	*0*

See also: ARRY→, DTAG, LIST→, OBJ→, STR→

ERASE

Type: Command

Description: Erase PICT Command: Erases *PICT*, leaving a blank *PICT* of the same dimensions.

Access: CAT ERASE

Input: None

Output: None

See also: DRAW

ERR0

Type: Command

Description: Clear Last Error Number Command: Clears the last error number so that a subsequent execution of ERRN returns # 0h, and clears the last error message.

Access: ⮌ PRG ERROR ERR0

Input: None

Output: None

See also: DOERR, ERRM, ERRN

ERRM

Type: Command

Description: Error Message Command: Returns a string containing the error message of the most recent calculator error.

ERRM returns the string for an error generated by DOERR. If the argument to DOERR was 0, the string returned by ERRM is empty.

Access: ⮌ PRG ERROR ERRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*"error message"*

See also: DOERR, ERRN, ERR0

ERRN

Type: Command

Description: Error Number Command: Returns the error number of the most recent calculator error.

If the most recent error was generated by DOERR with a string argument, ERRN returns #70000h. If the most recent error was generated by DOERR with a binary integer argument, ERRN returns that binary integer. (If the most recent error was generated by DOERR with a real number argument, ERRN returns the binary integer conversion of the real number.)

Access: ⮌ PRG ERROR ERRN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	#*nerror*

See also: DOERR, ERRM, ERR0

EVAL

Type: Command

Description: Evaluate Object Command: Evaluates the object.

The following table describes the effect of the evaluation on different object types.

Obj. Type	Effect of Evaluation
Local Name	Recalls the contents of the variable.
Global Name	*Calls* the contents of the variable: • A name is evaluated. • A program is evaluated. • A directory becomes the current directory. • Other objects are put on the stack. If no variable exists for a given name, evaluating the name returns the name to the stack.
Program	*Enters* each object in the program: • Names are evaluated (unless quoted). • Commands are evaluated. • Other objects are put on the stack.

Obj. Type	Effect of Evaluation (Continued)
List	*Enters* each object in the list: • Names are evaluated. • Commands are evaluated • Programs are evaluated. • Other objects are put on the stack.
Tagged	If the tag specifies a port, recalls and evaluates the specified object. Otherwise, puts the untagged object on the stack.
Algebraic	*Enters* each object in the algebraic expression: • Names are evaluated. • Commands are evaluated. • Other objects are put on the stack.
Command, Function, XLIB Name	Evaluates the specified object.
Other Objects	Puts the object on the stack.

To evaluate a symbolic argument to a numerical result, evaluate the argument in Numerical Result mode (flag –3 set) or execute →NUM on that argument.

Access: ↱ EVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*(see above)*

See also: →NUM, SYSEVAL

EXP

Type: Analytic Function

Description: Exponential Analytic Function: Returns the exponential, or natural antilogarithm, of the argument; that is, *e* raised to the given power.

EXP uses extended precision constants and a special algorithm to compute its result to full 12-digit precision for all arguments that do not trigger an underflow or overflow error.

EXP provides a more accurate result for the exponential than can be obtained by using *e* y^x. The difference in accuracy increases as *z* increases. For example:

z	EXP(*z*)	e^z
3	20.0855369232	20.0855369232
10	22026.4657948	22026.4657949
100	2.68811714182E43	2.68811714191E43
500	1.40359221785E217	1.40359221809E217
1000	1.9707111402E434	1.9707111469E434

For complex arguments:

$$e^{(x,y)} = e^x \cos y + ie^x \sin y$$

Access: ↰ e^x

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	e^z
'*symb*'	→	'*EXP(symb)*'

See also: ALOG, EXPM, LN, LOG

EXPAN

Type: Command

Description: Expand Products Command: Rewrites an algebraic expression or equation by expanding products and powers. Same as the EXPAND command (see volume 1, CAS Commands).

Access: CAT EXPAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	x
'$symb_1$'	→	'$symb_2$'
(x, y)	→	(x, y)

See also: COLCT, EXPLAND, ISOL, QUAD, SHOW

EXPFIT

Type: Command

Description: Exponential Curve Fit Command: Stores EXPFIT as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the exponential curve fitting model.

LINFIT is the default specification in Σ*PAR*.

Access: CAT EXPFIT

Input: None

Output: None

See also: BESTFIT, LR, LINFIT, LOGFIT, PWRFIT

EXPM

Type: Analytic Function

Description: Exponential Minus 1 Analytic Function: Returns $e^x - 1$.

For values of x close to zero, EXPM(x) returns a more accurate result than does EXP(x)–1. (Using EXPM allows both the argument and the result to be near zero, and avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.)

Access: ↰ MTH HYPERBOLIC EXPM

↰ EXP&LN EXPM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	$e^x - 1$
'*symb*'	→	'*EXPM(symb)*'

See also: EXP, LNP1

EYEPT

Type: Command

Description: Eye Point Command: Specifies the coordinates of the eye point in a perspective plot.

x_{point}, y_{point}, and z_{point} are real numbers that set the x-, y-, and z-coordinates as the eye-point from which to view a 3D plot's view volume. The y-coordinate must always be 1 unit less than the view volume's nearest point (y_{near} of YVOL). These coordinates are stored in the reserved variable *VPAR*.

Access: CAT EYEPT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
x_{point}	y_{point}	z_{point}	→	

See also: NUMX, NUMY, XVOL, XXRNG, YVOL, YYRNG, ZVOL

F0λ

Type: Function

Description: Black Body Emissive Power Function: Returns the fraction of total black-body emissive power at temperature x_T between wavelengths 0 and y_{lambda}. If units are not specified, y_{lambda} has implied units of meters and x_T has implied units of K.

F0λ returns a dimensionless fraction.

Access: CAT F0λ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{lambda}	x_T	→	x_{power}
y_{lambda}	'*symb*'	→	'*F0λ(*y_{lambda}*,symb)*'
'*symb*'	x_T	→	'*F0λ(symb,*x_T*)*'
'$symb_1$'	'$symb_2$'	→	'*F0λ(*$symb_1$*,*$symb_2$*)*'

FACT

Type: Command

Description: Factorial (Gamma) Function: FACT is the same as !. See !.

Access: None. Must be typed in.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*n!*
x	→	$\Gamma(x + 1)$
'*symb*'	→	'*(symb)!*'

FANNING

Type: Function

Description: Fanning Friction Factor Function: Calculates the Fanning friction factor of certain fluid flows.

FANNING calculates the Fanning friction factor, a correction factor for the frictional effects of fluid flows having constant temperature, cross-section, velocity, and viscosity (a typical pipe flow, for example). $x_{x/D}$ is the relative roughness (the ratio of the conduit roughness to its diameter). y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{x/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Access: CAT FANNING

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_{x/D}$	y_{Re}	→	$x_{fanning}$
$x_{x/D}$	*'symb'*	→	*'FANNING(*$x_{x/D}$*,symb)'*
'symb'	y_{Re}	→	*'FANNING(symb,*y_{Re}*)'*
'$symb_1$*'*	*'*$symb_2$*'*	→	*'FANNING(*$symb_1$*,*$symb_2$*)'*

See also: DARCY

FAST3D

Type: Command

Description: Fast 3D Plot Type Command: Sets the plot type to FAST 3D.

When plot type is set to FAST3D, the DRAW command plots an image graph of a 3-vector-valued function of two variables. FAST3D requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{left},x_{right},y_{near},y_{far},z_{low},z_{high},x_{min},x_{max},y_{min},y_{max},x_{eye},y_{eye},z_{eye},x_{step},y_{step} }

For plot type FAST3D, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye},y_{eye}, and z_{eye} are are not used.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}),(x_{max}, y_{max}),*indep*,*res*,*axes*,*ptype*,*depend* }

For plot type FAST3D, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is not used.

- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command FAST3D places the name FAST3D in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT FAST3D

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

FC?

Type: Command

Description: Flag Clear? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set.

Access: ← PRG TEST FC?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?C, FS? FS?C, SF

FC?C

Type: Command

Description: Flag Clear? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set. After testing, clears the flag.

Access: ← PRG TEST FC?C

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FS? FS?C, SF

FFT

Type: Command

Description: Discrete Fourier Transform Command: Computes the one- or two-dimensional discrete Fourier transform of an array.

If the argument is an N-vector or an $N \times 1$ or $1 \times N$ matrix, FFT computes the one-dimensional transform. If the argument is an $M \times N$ matrix, FFT computes the two-dimensional transform. M and N must be integral powers of 2.

The one-dimensional discrete Fourier transform of an N-vector X is the N-vector Y where:

$$Y_k = \sum_{n=0}^{N-1} X_n e^{-\frac{2\pi i k n}{N}}, i = \sqrt{-1}$$

for $k = 0, 1, \ldots, N-1$.

The two dimensional discrete Fourier transform of an $M \times N$ matrix X is the $M \times N$ matrix Y where:

$$Y_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{mn} e^{-\frac{2\pi i k m}{M}} e^{-\frac{2\pi i l n}{N}}, i = \sqrt{-1}$$

for $k = 0, 1, \ldots, M-1$ and $l = 0, 1, \ldots, N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The FFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Access: ↰ MTH FFT FFT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]$_1$	→	[*array*]$_2$

See also: IFFT

FILER

Type: Command

Description: Opens File Manager.

Access: ◁ FILES

CAT FILER

Input: None

Output: None

FINDALARM

Type: Command

Description: Find Alarm Command: Returns the alarm index n_{index} of the first alarm due after the specified time.

If the input is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the input is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the input is the real number 0, FINDALARM returns the first *past-due* alarm.

For any of the three arguments, FINDALARM returns 0 if no alarm is found.

Access: ▷ TIME TOOLS ALRM FINDALARM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

See also: DELALARM, RCLALARM, STOALARM

FINISH

Type: Command

Description: Finish Server Mode Command: Terminates Kermit Server mode in a device connected to an HP 49.

FINISH is used by a local Kermit device to tell a server Kermit (connected via the serial port) to exit Server mode.

Access: CAT FINISH

Input: None

Output: None

See also: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Type: Command

Description: Fix Mode Command: Sets the number display format to fix mode, which rounds the display to *n* decimal places.

Fix mode shows *n* digits to the right of the fraction mark (decimal point), where $0 \leq n \leq 11$. (Values for *n* outside this range are rounded to the nearest integer.) A number is displayed or printed as follows:

(*sign*) *mantissa*

where the mantissa can be of any form. However, the calculator automatically displays a number in scientific mode if either of the following is true:

- The number of digits to be displayed exceeds 12.
- A nonzero value rounded to *n* decimal places otherwise would be displayed as zero.

Access: CAT FIX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	

See also: SCI, STD

FLASHEVAL

Type: Command

Description: Evaluate Flash Function Command: Evaluates unnamed Flash functions.

Using FLASHEVAL with random addresses can corrupt memory. #n$_{function}$ is of the form *ffffbbbh*, where *bbb* is the bank ID, and *ffff* is the function number.

Access: CAT FLASHEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#$n_{function}$	→	

See also: EVAL,LIBEVAL, SYSEVAL

FLOOR

Type: Function

Description: Floor Function: Returns the greatest integer that is less than or equal to the argument.

Access: ↰ MTH REAL FLOOR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'*FLOOR(symb)*'

See also: CEIL, IP, RND, TRNC

FONT6

Type: Function

Description: Font Function: Returns the system FONT6 object. You use this in conjunction with the →FONT command to set the system font to type 6.

Access: CAT

See also: FONT7, FONT8, →FONT, FONT→

FONT7

Type: Function

Description: Font Function: Returns the system FONT7 object. You use this in conjunction with the →FONT command to set the system font to type 7.

Access: CAT

See also: FONT6, FONT8, →FONT, FONT→

FONT8

Type: Function

Description: Font Function: Returns the system FONT8 object. You use this in conjunction with the →FONT command to set the system font to type 8.

Access: CAT

See also: FONT6, FONT7, →FONT, FONT→

FONT→

Type: Function

Description: Returns the current system font.

Access: CAT

See also: FONT6, FONT7, FONT8

→FONT

Type: Function

Description: Set font Function: Sets the system font. You use this in conjunction with one of the three font commands to set the system font. for example, the following command sets the system font to type 6:

```
→FONT(FONT6)
```

Access: CAT

See also: FONT6, FONT7, FONT8, FONT→

FOR

Type: Command Operation

Description: FOR Definite Loop Structure Command: Starts FOR … NEXT and FOR … STEP definite loop structures.

Definite loop structures execute a command or sequence of commands a specified number of times.

- A FOR … NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The RPL syntax is this:

  x_{start} x_{finish} FOR counter loop-clause NEXT

  The algebraic syntax is this:

  FOR (counter,x_{start} ,x_{finish})loop-clause NEXT

  FOR takes x_{start} and x_{finish} as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. NEXT increments *counter* by one, and then tests whether *counter* is less than or equal to x_{finish}. If so, the loop clause is repeated (with the new value of *counter*).

  When the loop is exited, *counter* is purged.

- FOR … STEP works just like FOR … NEXT, except that it lets you specify an increment value other than 1. The syntax RPL is:

  x_{start} x_{finish} FOR counter loop-clause $x_{increment}$ STEP

  The algebraic syntax is:

  FOR(counter x_{start} x_{finish})loop-clause, STEP ($x_{increment}$)

  FOR takes x_{start} and x_{finish} as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. STEP takes $x_{increment}$ and increments *counter* by that value. If the argument of STEP is an algebraic expression or a name, it is automatically evaluated to a number.

  The increment value can be positive or negative. If the increment is positive, the loop is executed again when *counter* is less than or equal to x_{finish}. If the increment is negative, the loop is executed when *counter* is greater than or equal to x_{finish}.

  When the loop is exited, *counter* is purged.

Access: ⎋ PRG BRANCH FOR

Input/Output:

Level 2/	Level 1		Level 1/Item 1
FOR x_{start}	x_{finish}	→	
NEXT		→	
FOR x_{start}	x_{finish}	→	
STEP	$x_{increment}$	→	
STEP	'$symb_{increment}$'	→	

See also: NEXT, START, STEP

FP

Type: Function

Description: Fractional Part Function: Returns the fractional part of the argument.

The result has the same sign as the argument.

Access: ↰ MTH REAL FP

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*y*
x_unit	→	*y_unit*
'*symb*'	→	'*FP(symb)*'

See also: IP

FREEZE

Type: Command

Description: Freeze Display Command: Freezes the part of the display specified by $n_{display\ area}$, so that it is not updated until a key is pressed.

Normally, the stack display is updated as soon as the calculator is ready for data input. For example, when HALT stops a running program, or when a program ends, any displayed messages are cleared. The FREEZE command "freezes" a part or all of the display so that it is not updated until a key is pressed. This allows, for example, a prompting message to persist after a program halts to await data input.

$n_{\text{display area}}$ is the sum of the value codes for the areas to be frozen:

Display Area	Value Code
Status area	1
History/Stack/Command-line area	2
Menu area	4

So, for example, FREEZE(2) freezes the history/stack/command-line area, FREEZE(3) freezes the status area and the history/stack/command-line area, and FREEZE(7) freezes all three areas.

Values of $n_{\text{display area}} \geq 7$ or ≤ 0 freeze the entire display (are equivalent to value 7). To freeze the graphics display, you must freeze the status and stack/command-line areas (by entering 3), or the entire display (by entering 7).

Access: ⬅ PRG OUT FREEZE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{displayarea}}$	→	

See also: CLLCD, DISP, HALT

FS?

Type: Command

Description: Flag Set? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear.

Access: ⬅ PRG TEST FS?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FC?C, FS?C, SF

FS?C

Type: Command

Description: Flag Set? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear. After testing, clears the flag.

Access: ↰ PRG TEST FS?C

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FC?C, FS?, SF

FUNCTION

Type: Command

Description: Function Plot Type Command: Sets the plot type to FUNCTION.

When the plot type is FUNCTION, the DRAW command plots the current equation as a real-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

$$\{\ (x_{\min}, y_{\min})\ (x_{\max}, y_{\max})\ \textit{indep res axes ptype depend}\ \}$$

For plot type FUNCTION, the elements of *PPAR* are used as follows:

- $(x_{\min},y_{\min})$ is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- $(x_{\max},y_{\max})$ is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command FUNCTION places the name FUNCTION in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max})(the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation, as shown in the following table.

Form of Current Equation	Plotting Action
expr = *expr*	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.
name = *expr*	Only the expression is plotted.
indep = *constant*	A vertical line is plotted.

If flag –28 is set, all equations are plotted simultaneously.

If the independent variable in the current equation represents a unit object, you must specify the units by storing a unit object in the corresponding variable in the current directory. For example, if the current equation is X+3_m, and you want *X* to represent some number of inches, you would store 1_in (the number part of the unit object is ignored) in *X*. For each plotted point, the numerical value of the independent variable is combined with the specified unit (inches in this example) before the current equation is evaluated. If the result is a unit object, only the number part is plotted.

Access: CAT FUNCTION

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FAST3D, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

HP 49G Advanced Users Guide

Volume 1

Part C

Other Commands: G to P

Introduction

This volume details the HP 49G commands and functions that are not computer algebra-specific. See Volume 1, Computer algebra commands and functions, for information on computer algebra commands.

For each operation, the following details are provided:

Type: Function or command. Functions can be used as a part of an algebraic objects and commands cannot.

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys.

Input/Output: The input argument or arguments that the operation needs, and the outputs it produces.

See also: Related functions or commands

G to K

GET

Type: Command

Description: Get Element Command: Returns from the argument 1/level 2 array or list (or named array or list) the real or complex number z_{get} or object obj_{get} whose position is specified in argument 2/level 1.

For matrices, $n_{position}$ is incremented in *row* order.

Access: ↰ PRG LIST ELEMENTS GET

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[[matrix]]	$n_{position}$	→	z_{get}
[[matrix]]	{ n_{row}, m_{column} }	→	z_{get}
'$name_{matrix}$'	$n_{position}$	→	z_{get}
'$name_{matrix}$'	{ n_{row}, m_{column} }	→	z_{get}
[vector]	$n_{position}$	→	z_{get}
[vector]	{$n_{position}$}	→	z_{get}
'$name_{vector}$'	$n_{position}$	→	z_{get}
'$name_{vector}$'	{$n_{position}$}	→	z_{get}
{ *list* }	$n_{position}$	→	obj_{get}
{ *list* }	{$n_{position}$}	→	obj_{get}
'$name_{list}$'	$n_{position}$	→	obj_{get}
'$name_{list}$'	{$n_{position}$}	→	obj_{get}

See also: GETI, PUT, PUTI

GETI

Type: Command

Description: Get and Increment Index Command: Returns from the argument 1/level 2 array or list (or named array or list) the real or complex number z_{get} or object obj_{get} whose position is specified in argument 2/level 1, along with the first (level 2) argument and the next position in that argument.

For matrices, the position is incremented in *row* order.

Access: ↰ PRG LIST ELEMENTS GETI

Input/Output:

L_2/A_1	L_1/A_2		L_3/I_1	L_2/I_2	L_1/I_3
[[matrix]]	$n_{position1}$	→	*[[matrix]]*	$n_{position2}$	z_{get}
[[matrix]]	$\{ n_{row}, m_{column} \}_1$	→	*[[matrix]]*	$\{ n_{row}, m_{column} \}_2$	z_{get}
'$name_{matrix}$'	$n_{position1}$	→	'$name_{matrix}$'	$n_{position2}$	z_{get}
'$name_{matrix}$'	$\{ n_{row}, m_{column} \}_1$	→	'$name_{matrix}$'	$\{ n_{row}, m_{column} \}_2$	z_{get}
[vector]	$n_{position}$	→	*[vector]*	$n_{position2}$	z_{get}
[vector]	$\{n_{position1}\}$	→	*[vector]*	$\{n_{position2}\}$	z_{get}
'$name_{vector}$'	$n_{position1}$	→	'$name_{vector}$	$n_{position2}$	z_{get}
'$name_{vector}$'	$\{n_{position1}\}$	→	'$name_{vector}$	$\{n_{position2}\}$	z_{get}
{ *list* }	$n_{position1}$	→	{ *list* }	$n_{position2}$	obj_{get}
{ *list* }	$\{n_{position1}\}$	→	{ *list* }	$\{n_{position2}\}$	obj_{get}
'$name_{list}$'	$n_{position1}$	→	'$name_{list}$'	$n_{position2}$	obj_{get}
'$name_{list}$'	$\{n_{position1}\}$	→	'$name_{list}$'	$\{n_{position2}\}$	obj_{get}

L = level; A = argument; I = item

See also: GET, PUT, PUTI

GOR

Type: Command

Description: Graphics OR Command: Superimposes $grob_1$ onto $grob_{target}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*.

GOR uses a logical OR to determine the state (on or off) of each pixel in the overlapping portion of the argument graphics object.

If the first argument (stack level 3) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the first argument (level 3) is *PICT*, no result is returned to the stack.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

Access: ↰ PRG GROB GOR

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$grob_{target}$	{ #*n* #*m* }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(*x*, *y*)	$grob_1$	→	$grob_{result}$
PICT	{ #*n* #*m* }	$grob_1$	→	
PICT	(*x*, *y*)	$grob_1$	→	

Flags: None

See also: GXOR, REPL, SUB

GRAD

Type: Command

Description: Grads Mode Command: Sets Grads angle mode.

GRAD clears flag –17 and sets flag –18, and displays the GRD annunciator.

In Grads angle mode, real-number arguments that represent angles are interpreted as grads, and real-number results that represent angles are expressed in grads.

Access: MODE ANGLE MEASURE GRADS

CAT GRAD

Input: None

Output: None

See also: DEG, RAD

GRIDMAP

Type: Command

Description: GRIDMAP Plot Type Command: Sets the plot type to GRIDMAP.

When plot type is set GRIDMAP, the DRAW command plots a mapping grid representation of a 2-vector-valued function of two variables. GRIDMAP requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

$\{x_{left}, x_{right}, y_{near}, y_{far}, z_{low}, z_{high}, x_{min}, x_{max}, y_{min}, y_{max}, x_{eye}, y_{eye}, z_{eye}, x_{step}, y_{step}\}$

For plot type GRIDMAP, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are real numbers that specify the input region's width. The default value is (–1,1).
- y_{min} and y_{max} are real numbers that specify the input region's depth. The default value is (–1,1).
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which you view the graph.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted. These can be used instead of (or in combination with) RES.

The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$\{ (x_{min}, y_{min}), (x_{max}, y_{max}), indep, res, axes, ptype, depend \}$

For plot type GRIDMAP, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command GRIDMAP places the command name GRIDMAP in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT GRIDMAP

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

→GROB

Type: Command

Description: Stack to Graphics Object Command: Creates a graphics object from a specified object, where the argument $n_{\text{char size}}$ specifies the character size of the object.

$n_{\text{char size}}$ can be 0, 1 (small), 2 (medium), or 3 (large). $n_{\text{char size}} = 0$ is the same as $n_{\text{char size}} = 3$, except for unit objects and algebraic objects, where 0 specifies the Equation Writer application picture.

Access: CAT →GROB

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	n_{charsize}	→	*grob*

See also: →LCD, LCD→

GROBADD

Type: Command

Description: Combines two graphic objects.

Access: Catalog, CAT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$GROB_1$	$GROB_2$	→	$GROB_3$

GXOR

Type: Command

Description: Graphics Exclusive OR Command: Superimposes $grob_1$ onto $grob_{target}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*.

GXOR is used for creating cursors, for example, to make the cursor image appear dark on a light background and light on a dark background. Executing GXOR again with the same image restores the original picture.

GXOR uses a logical exclusive OR to determine the state of the pixels (on or off) in the overlapping portion of the argument graphics objects.

Any portion of $grob_1$ that extends past $grob_{target}$ or *PICT* is truncated.

If the first (level 3) argument (the target graphics object) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the first (level 3) argument is *PICT*, no result is returned to the stack.

Access: ← PRG GROB GXOR

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$grob_{target}$	{ #*n*, #*m* }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(*x*, *y*)	$grob_1$	→	$grob_{result}$
PICT	{ #*n*, #*m* }	$grob_1$	→	
PICT	(*x*, *y*)	$grob_1$	→	

See also: GOR, REPL, SUB

HALT

Type: Command

Description: Halt Program Command: Halts program execution.

Program execution is halted at the location of the HALT command in the program. The HLT annunciator is turned on. Program execution is resumed by executing CONT (that is, by pressing ↰ CONT). Executing KILL cancels all halted programs.

Access: ↰ PRG RUN & DEBUG HALT

Input: None

Output: None

See also: CONT, KILL

HEAD

Type: Command

Description: First Listed Element Command: Returns the first element of a list or string.

Access: ↰ PRG CHARS HEAD

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_1, ... ,obj_n }	→	obj_1
"string"	→	*"*$element_1$*"*

See also: TAIL

HEADER→

Type: Command

Description: Header size: Returns the current size of the header in lines.

Access: CAT

See also: →HEADER

→HEADER

Type: Command

Description: Header size: Sets the current size of the header in lines: 1, 2, 3, or 4 lines.

Access: CAT

See also: →HEADER

HEX

Type: Command

Description: Hexadecimal Mode Command: Selects hexadecimal base for binary integer operations. (The default base is decimal.)

Binary integers require the prefix #. Binary integers entered and returned in hexadecimal base automatically show the suffix h. If the current base is not hexadecimal, then you can enter a hexadecimal number by ending it with h. It will be displayed in the current base when it is entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: CAT HEX

Input: None

Output: None

See also: BIN, DEC, OCT, RCWS, STWS

HISTOGRAM

Type: Command

Description: Histogram Plot Type Command: Sets the plot type to HISTOGRAM.

When the plot type is HISTOGRAM, the DRAW command creates a histogram using data from one column of the current statistics matrix (reserved variable Σ*DAT*). The column is specified by the first parameter in the reserved variable Σ*PAR* (using the XCOL command). The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

$$\{ (x_{min}, y_{min}) (x_{max}, y_{max}) \textit{ indep res axes ptype depend } \}$$

For plot type HISTOGRAM, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers that specify the minimum and maximum values of the data to be plotted. The default value of *indep* is *X*.
- *res* is a real number specifying the bin size, in user-unit coordinates, or a binary integer specifying the bin size in pixels. The default value is 0, which specifies the bin size to be 1/13 of the difference between the specified minimum and maximum values of the data.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command HISTOGRAM places the command name HISTOGRAM in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The frequency of the data is plotted as bars, where each bar represents a collection of data points. The base of each bar spans the values of the data points, and the height indicates the number of data points. The width of each bar is specified by *res*. The overall maximum and minimum values for the data can be specified by *indep*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max}) are used.

Access: (CAT) HISTOGRAM

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

HISTPLOT

Type: Command

Description: Draw Histogram Plot Command: Plots a frequency histogram of the specified column in the current statistics matrix (reserved variable Σ*DAT*).

The data column to be plotted is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. If no data column is specified, column 1 is selected by default. The *y*-axis is autoscaled and the plot type is set to HISTOGRAM.

HISTPLOT plots *relative* frequencies, using 13 bins as the default number of partitions. The RES command lets you specify a different number of bins by specifying the bin width. To plot a frequency histogram with *numerical* frequencies, store the frequencies in ΣDAT and execute BINS and then BARPLOT.

When HISTPLOT is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Access: CAT HISTPLOT

Input: None

Output: None

See also: BARPLOT, BINS, FREEZE, PICTURE, PVIEW, RES, SCATRPLOT, XCOL

HMS–

Type: Command

Description: Hours-Minutes-Seconds Minus Command: Returns the difference of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number (hours or degrees).
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Access: → TIME TOOLS HMS–

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
HMS_1	HMS_2	→	$HMS_1 - HMS_2$

See also: HMS→, →HMS, HMS+

HMS+

Type: Command

Description: Hours-Minutes-Seconds Plus Command: Returns the sum of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number (hours or degrees).
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Access: ↱ TIME TOOLS HMS+

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
HMS_1	HMS_2	→	$HMS_1 + HMS_2$

See also: HMS→, →HMS, HMS–

HMS→

Type: Command

Description: Hours-Minutes-Seconds to Decimal Command: Converts a real number in hours-minutes-seconds format to its decimal form (hours or degrees with a decimal fraction).

The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number (hours or degrees).
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.

- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Access: ↱ TIME TOOLS HMS→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
HMS	→	*x*

See also: →HMS, HMS+, HMS–

→HMS

Type: Command

Description: Decimal to Hours-Minutes-Seconds Command: Converts a real number representing hours or degrees with a decimal fraction to hours-minutes-seconds format.

The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Access: ↱ TIME TOOLS →HMS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*HMS*

See also: HMS→, HMS+, HMS–

HOME

Type: Command

Description: HOME Directory Command: Makes the *HOME* directory the current directory.

Access: CAT HOME

Input: None

Output: None

See also: CRDIR, PATH, PGDIR, UPDIR

---

i

Type: Function

Description: *i* Function: Returns the symbolic constant *i* or its numerical representation, (0, 1).

Access: ↰ *i*

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*'i'*
	→	*(0,1)*

See also: *e*, MAXR, MINR, π

IDN

Type: Command

Description: Identity Matrix Command: Returns an identity matrix; that is, a square matrix with its diagonal elements equal to 1 and its off-diagonal elements equal to 0.

The result is either a new square matrix, or an existing square matrix with its elements replaced by the elements of the identity matrix, according to the argument.

- Creating a new matrix: If the argument is a real number *n*, a new real identity matrix is returned, with its number of rows and number of columns equal to *n*.
- Replacing the elements of an existing matrix: If the argument is a square matrix, an identity matrix of the same dimensions is returned. If the original matrix is complex, the resulting identity matrix will also be complex, with diagonal values (1,0).
- If the argument is a name, the name must identify a variable containing a square matrix. In this case, the elements of the matrix are replaced by those of the identity matrix (complex if the original matrix is complex).

Access: ↰ MATRICES CREATE IDN

↰ MTH MATRIX MAKE IDN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*[[R-matrix$_{identity}$]]*
[[matrix]]	→	*[[matrix$_{identity}$]]*
'*name*'	→	*[[matrix$_{identity}$]]*

See also: CON

IF

Type: Command Operation

Description: IF Conditional Structure Command: Starts IF … THEN … END and IF … THEN … ELSE … END conditional structures.

Conditional structures, used in combination with program tests, enable a program to make decisions.

- IF … THEN … END executes a sequence of commands only if a test returns a nonzero (true) result. The syntax is:

  IF *test-clause* THEN *true-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, program execution resumes following END.

- IF … THEN … ELSE … END executes one sequence of commands if a test returns a true (nonzero) result, or another sequence of commands if that test returns a false (zero) result. The syntax is:

  IF *test-clause* THEN *true-clause* ELSE *false-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is nonzero, the true clause is executed. Otherwise, the false clause is executed. After the appropriate clause is executed, execution resumes following END.

In RPL mode, the test clause can be a command sequence (for example, A B ≤) or an algebraic (for example, 'A ≤ B)'. If the test clause is an algebraic, it is *automatically evaluated* to a number (→NUM or EVAL isn't necessary).

Access: ↰ PRG BRANCH IF

Input/Output:

Level 1/Argument 1			Level 1/Item 1
IF		→	
THEN	*T/F*	→	
END			
IF			
THEN	*T/F*	→	
ELSE		→	
END		→	

See also: CASE, ELSE, END, IFERR, THEN

IFERR

Type: Command

Description: If Error Conditional Structure Command: Starts IFERR … THEN … END and IFERR … THEN … ELSE … END error trapping structures.

Error trapping structures enable program execution to continue after a "trapped" error occurs.

- IFERR … THEN … END executes a sequence of commands if an error occurs. The syntax of IFERR … THEN … END is:

  IFERR *trap-clause* THEN *error-clause* END

  If an error occurs during execution of the trap clause:

  1 The error is ignored.
  2 The remainder of the trap clause is discarded.
  3 The key buffer is cleared.
  4 If any or all of the display is "frozen" (by FREEZE), that state is canceled.
  5 If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack. Program execution jumps to the error clause.

  The commands in the error clause are executed only if an error is generated during execution of the trap clause.

- IFERR … THEN … ELSE … END executes one sequence of commands if an error occurs or another sequence of commands if an error does not occur. The syntax of IFERR … THEN … ELSE … END is:

  IFERR *trap-clause* THEN *error-clause* ELSE *normal-clause* END

If an error occurs during execution of the trap clause, the same six events listed above occur.

If no error occurs, execution jumps to the normal clause at the completion of the trap clause.

Access: ⮌ PRG ERROR IFERR IFERR

Input: None

Output: None

See also: CASE, ELSE, END, IF, THEN

IFFT

Type: Command

Description: Inverse Discrete Fourier Transform Command: Computes the one- or two-dimensional inverse discrete Fourier transform of an array.

If the argument is an N-vector or an $N \times 1$ or $1 \times N$ matrix, IFFT computes the one-dimensional inverse transform. If the argument is an $M \times N$ matrix, IFFT computes the two-dimensional inverse transform. M and N must be integral powers of 2.

The one-dimensional inverse discrete Fourier transform of an N-vector Y is the N-vector X where:

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} Y_k e^{\frac{2\pi i k n}{N}}, \; i = \sqrt{-1}$$

for $n = 0, 1, \ldots, N-1$.

The two-dimensional inverse discrete Fourier transform of an $M \times N$ matrix Y is the $M \times N$ matrix X where:

$$X_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} Y_{kl} e^{\frac{2\pi i k m}{M}} e^{\frac{2\pi i l n}{N}}, \; i = \sqrt{-1}$$

for $m = 0, 1, \ldots, M-1$ and $n = 0, 1, \ldots, N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The IFFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Access: ↰ MTH FFT IFFT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[array]$_1$	→	*[array]*$_2$

See also: FFT

IFT

Type: Command

Description: IF-THEN Command: Executes *obj* if *T/F* is nonzero. Discards *obj* if *T/F* is zero.

IFT lets you execute in stack syntax the decision-making process of the IF … THEN … END conditional structure. The "true clause" is *obj* in argument 2 (level 1).

Access: ↰ PRG BRANCH IFT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
T/F	*obj*	→	*It depends!*

See also: IFTE

IFTE

Type: Function

Description: IF-THEN-ELSE Function: Executes the *obj* in argument 2 or level 2 if *T/F* is nonzero. Executes the *obj* in argument 3 or level 1 if *T/F* is zero.

IFTE lets you execute in stack syntax the decision-making process of the IF … THEN … ELSE … END conditional structure. The "true clause" is obj_{true} in argument 2 or level 2. The "false clause" is obj_{false} in argument 3 or level 1.

IFTE is also allowed in algebraic expressions, with the following syntax:

IFTE(*test,true-clause,false-clause*)

When an algebraic containing IFTE is evaluated, its first argument *test* is evaluated to a test result. If it returns a nonzero real number, *true-clause* is evaluated. If it returns zero, *false-clause* is evaluated.

Access: ↰ PRG BRANCH IFTE

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
T/F	obj_{true}	obj_{false}	→	*It depends!*

See also: IFT

IM

Type: Function

Description: Imaginary Part Function: Returns the imaginary part of its complex argument.

If the argument is an array, IM returns a real array, the elements of which are equal to the imaginary parts of the corresponding elements of the argument array. If the argument array is real, all of the elements of the result array are zero.

Access: ↱ CMPLX IM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*0*
(x, y)	→	*y*
[R-array]	→	*[R-array]*
[C-array]	→	*[R-array]*
'symb'	→	*'IM(symb)'*

See also: C→R, RE, R→C

INCR

Type: Command

Description: Increment Command: Takes a variable, adds 1, stores the new value back into the original variable, and returns the new value.

The value in *name* must be a real number or an integer.

Access: ↰ PRG MEMORY ARITHMETIC INCR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	$x_{increment}$

See also: DECR

INDEP

Type: Command

Description: Independent Variable Command: Specifies the independent variable and its plotting range.

The specification for the independent variable name and its plotting range is stored as the third parameter in the reserved variable *PPAR*. If the argument to INDEP is a:

- Global variable name, that name replaces the independent variable entry in *PPAR*.
- List containing a global name, that name replaces the independent variable name but leaves unchanged any existing plotting range.
- List containing a global name and two real numbers, that list replaces the independent variable entry.
- List containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the independent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *X*.

Access: CAT INDEP

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
	'global'	→	
	{ *global* }	→	
	{ *global* x_{start} x_{end} }	→	
	{x_{start} x_{end} }	→	
x_{start}	x_{end}	→	

See also: DEPND

INFORM

Type: Command

Description: User-Defined Dialog Box Command: Creates a user-defined input form (dialog box). INFORM creates a standard dialog box based upon the following specifications:

Variable	Function
"title"	Title. This appears at the top of the dialog box.

Variable (Cont.)	Function
{s_1 s_2 … s_n}	Field definitions. A field definition (s_x) can have two formats: "label", a field label, or { "*label*" "*helpInfo*" $type_0$ $type_1$ … $type_n$ }, a field label with optional help text that appears near the bottom of the screen, and an optional list of valid object types for that field. If object types aren't specified, all object types are valid. For information about object types, see the TYPE command. When creating a multi-column dialog box, you can span columns by using an empty list as a field definition. A field that appears to the left of an empty field automatically expands to fill the empty space.
format	Field format information. This is the number *col* or a list of the form { *col tabs* }: *col* is the number of columns the dialog box has, and *tabs* optionally specifies the number of tab stops between the labels and the highlighted fields. This list can be empty. *col* defaults to 1 and *tabs* defaults to 3.
{ *resets* }	Default values displayed when RESET is selected. Specify reset values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.
{ *init* }	Initial values displayed when the dialog box appears. Specify initial values in the list in the same order as the fields were specified. To specify no value, use the NOVAL command as a place holder. This list can be empty.

If you exit the dialog box by selecting OK or ENTER, INFORM returns the field values { *vals* } in item 1 or level 2, and puts a 1 in item 2 or level 1. (If a field is empty, NOVAL is returned as a place holder.) If you exit the dialog box by selecting CANCEL or F2, INFORM returns 0.

Access: ↰ PRG IN INFORM

Input/Output:

L_5/A_1	L_4/A_2	L_3A_3	L_2/A_4	L_1/A_5		L_2/I_1	L_1/I_2
"title"	{ s_1 s_2 ... s_n }	*format*	{ *resets* }	{ *init* }	→	{ *vals* }	*1*
"title"	{ s_1 s_2 ... s_n }	*format*	{ *resets* }	{ *init* }	→		*0*

L = level; A = argument; I = item

See also: CHOOSE, INPUT, NOVAL, TYPE

INPUT

Type: Command

Description: Input Command: Prompts for data input to the command line and prevents the user access to stack operations.

When INPUT is executed, the stack or history area is blanked and program execution is suspended for data input to the command line. The contents of "*stack prompt*" are displayed at the top of the screen. Depending on the second argument (level 1), the command line may also contain the contents of a string, or it may be empty. Pressing ENTER resumes program execution and returns the contents of the command line in string form.

In its general form, the second argument (level 1) for INPUT is a list that specifies the content and interpretation of the command line. The list can contain *one or more* of the following parameters, *in any order*:

- "*command-line prompt*", whose contents are placed on the command line for prompting when the program pauses.
- Either a *real number*, or a *list containing two real numbers*, that specifies the initial cursor position on the command line:
  - A real number *n* at the *n*th character from the left end of the first row (line) of the command line. A *positive n* specifies the insert cursor; a *negative n* specifies the replace cursor. 0 specifies the end of the command-line string.

  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. 0 specifies the end of the command-line string in the specified row. A positive row number specifies the insert cursor; a negative row number specifies the replace cursor.

- One or more of the parameters ALG, α, or V, entered as unquoted names:
  - ALG activates Algebraic/Program-entry mode.
  - α specifies alpha lock.
  - V verifies if the characters in the result string "result", without the " delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the Invalid Syntax warning and prompts again for data.

You can choose to specify as few as one of the argument 2 (level1) list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the command-line prompt string.
- Program-entry mode.
- Result string not checked for invalid syntax.

If you specify *only* a command-line prompt string for the second argument (level 1), you don't need to put it in a list.

Access: ↰ PRG IN INPUT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"stack prompt"	*"command-line prompt"*	→	*"result"*
"stack prompt"	{ $list_{\text{command-line}}$ }	→	*"result"*

See also: PROMPT, STR→

INV

Type: Analytic function

Description: Inverse (1/*x*) Analytic Function: Returns the reciprocal or the matrix inverse.

For a *complex* argument (*x*, *y*), the inverse is the complex number:

$$\left(\frac{x}{x^2+y^2}, \frac{-y}{x^2+y^2}\right)$$

Matrix arguments must be square (real or complex). The computed inverse matrix A^{-1} satisfies $A \times A^{-1} = I_n$, where I_n is the $n \times n$ identity matrix.

Access: 1/x

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*1/z*
[[matrix]]	→	*[[matrix]]*$^{-1}$
'symb'	→	*'INV(symb)'*
x_unit	→	*1/x_1/unit*

See also: SINV, /

IP

Type: Function

Description: Integer Part Function: Returns the integer part of its argument.

The result has the same sign as the argument.

Access: ↰ MTH REAL IP

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*n*
x_unit	→	*n_unit*
'symb'	→	*'IP(symb)'*

See also: FP

ISOL

Type: Command

Description: Isolate Variable Command: Returns an algebraic $symb_2$ that rearranges $symb_1$ to "isolate" the first occurrence of variable *global*.

The result $symb_2$ is an equation of the form *global* = *expression*. If *global* appears more than once, then $symb_2$ is effectively the right side of an equation obtained by rearranging and solving $symb_1$ to isolate the first occurrence of *global* on the left side of the equation.

If $symb_1$ is an expression, it is treated as the left side of an equation $symb_1 = 0$.

If *global* appears in the argument of a function within $symb_1$, that function must be an *analytic* function, that is, a function for which the HP 49 provides an inverse. Thus ISOL cannot solve IP(x)=0 for x, since IP has no inverse.

ISOL is identical to SOLVE (see volume 1, CAS Commands).

Access: ↰ (S.SLV) ISOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_1$'	'*global*'	→	'$symb_2$'

See also: COLCT, EXPAN, QUAD, SHOW, SOLVE

KERRM

Type: Command

Description: Kermit Error Message Command: Returns the text of the most recent Kermit error packet.

If a Kermit transfer fails due to an error packet sent from the connected Kermit device to the HP 49, then executing KERRM retrieves and displays the error message. (Kermit errors not in packets are retrieved by ERRM rather than KERRM.)

Access: (CAT) KERRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*"error message"*

See also: FINISH, KGET, PKT, RECN, RECV, SEND, SERVER

KEY

Type: Command

Description: Key Command: Returns a test result and, if a key is pressed, returns the row-column location $x_{n\,m}$ of that key.

KEY returns a false result (0) to item 2 (stack level 1) until a key is pressed. When a key is pressed, it returns a true result (1) to item 2 (stack level 1) and $x_{n\,m}$ to item 1 (stack level 2). The result $x_{n\,m}$ is a two-digit number that identifies the row and column location of the key just pressed. Unlike WAIT, which returns a three-digit number that identifies alpha and shifted keyboard planes, KEY returns the row-column location of *any* key pressed, including ↰, ↱, and (ALPHA).

Access: ↰ (PRG) IN KEY

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	$x_{n\,m}$	*1*
	→		*0*

See also: WAIT, KEYEVAL

KEYEVAL

Type: Command

Description: Actions the specified key press.

You input a number, in the format ***ab.c***, that represents the key. In the number ***ab.c:***

- ***a*** is the row coordinate number, where row 1 is the left-most row.
- ***b*** is the column number, where column 1 is the top-most column.
- ***c*** is the shift state of the key, that is, whether it is normal, alpha-shifted, left shifted and so on. The shift state representations are as follows:

1: Normal function.
2: Left-shift function.
3. Right-shift function.
4. Alpha-function.
5. Alpha-left-shift function.
6. Alpha-right-shift function.

Access: Catalog, CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
nn.n	→	

Example: Turn the calculator off using a command.

Command: `KEYEVAL(101.3)`

Result: The calculator is turned off.

→KEYTIME

Type: Command

Description: Sets a new keytime value.

Keytime is the time after a keypress during which further keypresses will not be actioned. It is measured in ticks. If you experience key bounce, you can increase the value of keytime.

Access: CAT KEYTIME→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
time	→	

See Also: KEYTIME→

KEYTIME→

Type: Command

Description: Displays the current keytime value.

Keytime is the time after a keypress during which further keypresses will not be actioned. It is measured in milliseconds. If you experience key bounce, you can increase the value of keytime.

Access: CAT KEYTIME→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*time*

See Also: →KEYTIME

KGET

Type: Command

Description: Kermit Get Command: Used by a local Kermit to get a Kermit server to transmit the named object(s).

To rename an object when the local device gets it, include the old and new names in an embedded list. For example, {{ AAA BBB }} KGET gets the variable named *AAA* but changes its name to *BBB*. {{ AAA BBB } CCC } KGET gets *AAA* as *BBB* and gets *CCC* under its own name. (If the original name is not legal on the HP 49, enter it as a string.)

Access: CAT KGET

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	
"*name*"	→	
{ $name_{old}$ $name_{new}$ }	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } *name* ... }	→	

See also: BAUD, CKSM, FINISH, PARITY, RECN, RECV, SEND, SERVER, TRANSIO

KILL

Type: Command

Description: Cancel Halted Programs Command: Cancels all currently halted programs. If KILL is executed within a program, that program is also canceled.

Canceled programs cannot be resumed.

KILL cancels *only* halted programs and the program from which KILL was executed, if any. Commands that halt programs are HALT and PROMPT.

Suspended programs cannot be canceled. Commands that suspend programs are INPUT and WAIT.

Access: ⮌ PRG RUN & DEBUG KILL

Input: None

Output: None

See also: CONT, DOERR, HALT, PROMPT

L to N

LABEL

Type: Command

Description: Label Axes Command: Labels axes in *PICT* with *x*- and *y*-axis variable names and with the minimum and maximum values of the display ranges.

The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the *x-axis* element from that list is used.
2. If *axes* parameter is not a list, then the independent variable name in *PPAR* is used.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the *y-axis* element from that list is used.
2. If *axes* is not a list, then the dependent variable name from *PPAR* is used.

Access: CAT LABEL

Input: None

Output: None

See also: AXES, DRAW, DRAX

LABEL

Type: Command

Description: Label Axes Command: Labels axes in *PICT* with *x*- and *y*-axis variable names and with the minimum and maximum values of the display ranges.

The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the *x-axis* element from that list is used.
2. If *axes* parameter is not a list, then the independent variable name in *PPAR* is used.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the *y-axis* element from that list is used.
2. If *axes* is not a list, then the dependent variable name from *PPAR* is used.

Access: CAT LABEL

Input: None

Output: None

See also: AXES, DRAW, DRAX

LANGUAGE→

Type: Command

Description: Language: Sets the language for things such as error messages: 0 for English, 1 for French, and 2 for Spanish.

Access: CAT

See also: →LANGUAGE

→LANGUAGE

Type: Command

Description: Language: Returns the language that is currently set.

Access: CAT

See also: →LANGUAGE

LCD→

Type: Command

Description: LCD to Graphics Object Command: Returns the current stack and menu display as a 131 × 64 graphics object.

Access: ↰ PRG GROB LCD→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*grob*

Flags: None

See also: →GROB, →LCD

→LCD

Type: Command

Description: Graphics Object to LCD Command: Displays the specified graphics object with its upper left pixel in the upper left corner of the display.

If the graphics object is larger than 131 × 56, it is truncated.

Access: CAT →LCD

Input/Output:

Level 1/Argument 1		Level 1/Item 1
grob	→	

See also: BLANK, →GROB, LCD→

LIBEVAL

Type: Command

Description: Evaluate Library Function Command: Evaluates unnamed library functions.

Using LIBEVAL with random addresses can corrupt memory. $\#n_{function}$ is of the form *lllfff*h, where *lll* is the library number, and *fff* the function number.

Access: CAT LIBEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_{function}$	→	

See also: EVAL, SYSEVAL

LIBS

Type: Command

Description: Libraries Command: Lists the title, number, and port of each library attached to the current directory.

The title of a library often takes the form *LIBRARY-NAME* : *Description*. A library without a title is displayed as " ".

Access: CAT LIBS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ *"title"*, n_{lib}, n_{port}, ..., *"title"*, n_{lib}, n_{port} }

See also: ATTACH, DETACH

LINE

Type: Command Operation

Description: Draw Line Command: Draws a line in *PICT* between the input coordinates.

Access: ↰ PRG PICT LINE

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
(x_1, y_1)	(x_2, y_2)	→	
{ $\#n_1$, $\#m_1$ }	{ $\#n_2$, $\#m_2$ }	→	

Flags: None

See also: ARC, BOX, TLINE

ΣLINE

Type: Command

Description: Regression Model Formula Command: Returns an expression representing the best fit line according to the current statistical model, using *X* as the independent variable name, and explicit values of the slope and intercept taken from the reserved variable Σ*PAR*.

For each curve fitting model, the following table indicates the form of the expression returned by ΣLINE, where *m* is the slope, *x* is the independent variable, and *b* is the intercept.

Model	Form of Expression
LINFIT	$mx + b$
LOGFIT	$m \ln(x) + b$
EXPFIT	be^{mx}
PWRFIT	bx^{m}

Access: CAT ΔLINE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'$symb_{formula}$'

See also: BESTFIT, COLΣ, CORR, COV, EXPFIT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, XCOL, YCOL

LINFIT

Type: Command

Description: Linear Curve Fit Command: Stores LINFIT as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the linear curve fitting model.

LINFIT is the default specification in Σ*PAR*.

Access: CAT LINFIT

Input: None

Output: None

See also: BESTFIT, EXPFIT, LOGFIT, LR, PWRFIT

LININ

Type: Function

Description: Linear Test Function: Tests whether an algebraic is structurally linear for a given variable.

If any two subexpressions containing a variable (*name*) are combined only with addition and subtraction, and any subexpression containing the variable is at most multiplied or divided by another factor not containing the variable, the algebraic (*symb*) is determined to be linear for that variable.

LININ returns a 1 if the algebraic is linear for the variable, and a 0 if not.

Access: ⬅ PRG TEST LININ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'*symb*'	'*name*'	→	*0/1*

LIST→

Type: Command

Description: List to Stack Command: Takes a list of *n* objects and returns each object to a separate level, and returns the total number of objects to item *n*+1 (stack level 1).

The command OBJ→ also provides this function.

Access: CAT LIST→

Input/Output:

Level 1/Argument 1		Level$_{n+1}$/Item$_1$ ...	Level$_2$/Item$_n$	Level$_1$/Item$_{n+1}$
{ obj_1, ...,obj_n }	→	obj_1 ...	obj_n	*n*

See also: ARRY→, DTAG, EQ→, →LIST, OBJ→, STR→

→LIST

Type: Command

Description: Stack to List Command: Takes *n* specified objects and returns a list of those objects.

Access: CAT →LIST

Input/Output:

Level$_{n+1}$/Argument$_1$ ...Level$_2$/Argument$_n$	Level$_1$/Argument$_{n+1}$		Level 1/Item 1
obj_1 … obj_n	n	→	{ obj_1, … ,obj_n }

See also: →ARRY, LIST→, →STR, →TAG, →UNIT

ΔLIST

Type: Command

Description: List Differences Command: Returns the first differences of the elements in a list.
Adjacent elements in the list must be suitable for mutual subtraction.

Access: ↰ MTH LIST ΔLIST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	{ *differences* }

See also: ΣLIST, ΠLIST, STREAM

ΠLIST

Type: Command

Description: List Product Command: Returns the product of the elements in a list.

The elements in the list must be suitable for mutual multiplication.

Access: ↰ MTH LIST ΠLIST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	*product*

See also: ΣLIST, ΔLIST, STREAM

ΣLIST

Type: Command

Description: List Sum Command: Returns the sum of the elements in a list.

The elements in the list must be suitable for mutual addition.

Access: ↰ MTH LIST ΣLIST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *list* }	→	*sum*

See also: ΠLIST, STREAM

LN

Type: Analytic function

Description: Natural Logarithm Analytic Function: Returns the natural (base *e*) logarithm of the argument.

For $x = 0$ or (0, 0), an Infinite Result exception occurs, or, if flag –22 is set, –MAXR is returned.

The inverse of EXP is a *relation*, not a function, since EXP sends more than one argument to the same result. The inverse relation for EXP is expressed by ISOL as the *general solution*:

LN(Z)+2*π*i*n1

The function LN is the inverse of a *part* of EXP, a part defined by restricting the domain of EXP such that:

- each argument is sent to a distinct result, and
- each possible result is achieved.

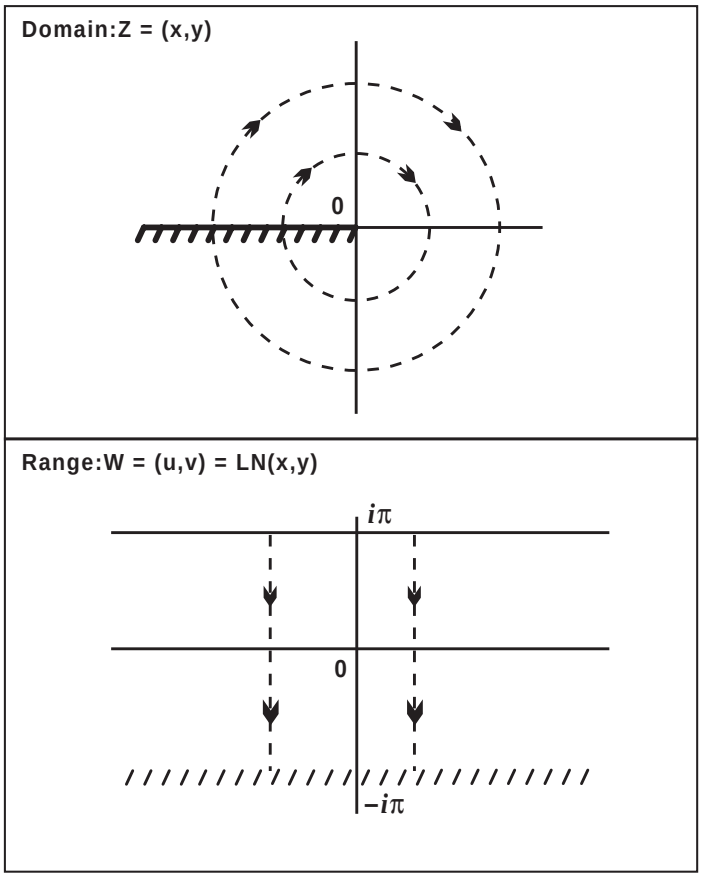
The points in this restricted domain of EXP are called the *principal values* of the inverse relation. LN in its entirety is called the *principal branch* of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the *branch cuts* of LN.

The principal branch used by the HP 49 for LN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued natural log function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of LN. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

These graphs show the inverse relation LN(Z)+2*π*i*n1 for the case *n1*=0. For other values of *n1*, the horizontal band in the lower graph is translated up (for *n1* positive) or down (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of EXP.

You can view these graphs with domain and range reversed to see how the domain of EXP



is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*,*y*). EXP sends this domain onto the whole complex plane in the range W = (*u*,*v*) = EXP(*x*,*y*) in the upper graph.

Access: ⇨LN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	*ln z*
'*symb*'	→	'*LN(symb)*'

See also: ALOG, EXP, ISOL, LNP1, LOG

LNP1

Type: Analytic function

Description: Natural Log of *x* Plus 1 Analytic Function: Returns ln (*x* + 1).

For values of *x* close to zero, LNP1(*x*) returns a more accurate result than does LN(*x*+1). Using LNP1 allows both the argument and the result to be near zero, and it avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.

For values of $x < -1$, an Undefined Result error results. For $x=-1$, an Infinite Result exception occurs, or, if flag –22 is set, LNP1 returns –MAXR.

Access: ↰ MTH HYPERBOLIC LNP1

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*ln (x + 1)*
'*symb*'	→	'*LNP1(symb)*'

See also: EXPM, LN

LOG

Type: Analytic function

Description: Common Logarithm Analytic Function: Returns the common logarithm (base 10) of the argument.

For $x=0$ or (0, 0), an Infinite Result exception occurs, or, if flag –22 is set (no error), LOG returns –MAXR.

The inverse of ALOG is a *relation*, not a function, since ALOG sends more than one argument to the same result. The inverse relation for ALOG is expressed by ISOL as the *general solution*:

LOG(Z)+2*π*i*n1/2.30258509299

The function LOG is the inverse of a *part* of ALOG, a part defined by restricting the domain of ALOG such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of ALOG are called the *principal values* of the inverse relation. LOG in its entirety is called the *principal branch* of the inverse relation, and the points sent by LOG to the boundary of the restricted domain of ALOG form the *branch cuts* of LOG.

The principal branch used by the HP 49 for LOG(z) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued LOG function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for LOG(z) from the graph for LN (see LN) and the relationship $\log z = \ln z / \ln 10$.

Access: ↱ log

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	$\log z$
'*symb*'	→	'*LOG(symb)*'

See also: ALOG, EXP, ISOL, LN

LOGFIT

Type: Command

Description: Logarithmic Curve Fit Command: Stores LOGFIT as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the logarithmic curve-fitting model.

LINFIT is the default specification in Σ*PAR*.

Access: CAT LOGFIT

Input: None

Output: None

See also: BESTFIT, EXPFIT, LINFIT, LR, PWRFIT

LQ

Type: Command

Description: LQ Factorization of a Matrix Command: Returns the LQ factorization of an $m \times n$ matrix.

LQ factors an $m \times n$ matrix A into three matrices:

- L is a lower $m \times n$ trapezoidal matrix.
- Q is an $n \times n$ orthogonal matrix.
- P is a $m \times m$ permutation matrix.

Where $P \times A = L \times Q$.

Access: ↰ MATRICES FACTORIZATION LQ

↰ MTH MATRIX FACTORS LQ

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
[[matrix]]$_A$	→	*[[matrix]]*$_L$	*[[matrix]]*$_Q$	*[[matrix]]*$_P$

See also: LSQ, QR

LR

Type: Command

Description: Linear Regression Command: Uses the currently selected statistical model to calculate the linear regression coefficients (intercept and slope) for the selected dependent and independent variables in the current statistics matrix (reserved variable Σ*DAT*).

The columns of independent and dependent data are specified by the first two elements in the reserved variable Σ*PAR*, set by XCOL and YCOL, respectively. (The default independent and dependent columns are 1 and 2.) The selected statistical model is the fifth element in Σ*PAR*. LR stores the intercept and slope (untagged) as the third and fourth elements, respectively, in Σ*PAR*.

The coefficients of the exponential (EXPFIT), logarithmic (LOGFIT), and power (PWRFIT) models are calculated using transformations that allow the data to be fitted by standard linear regression. The equations for these transformations appear in the table below, where b is the intercept and m is the slope. The logarithmic model requires positive x-values (XCOL), the

exponential model requires positive y-values (YCOL), and the power model requires positive x- and y-values.

Model	Transformation
Logarithmic	$y = b + m\ln(x)$
Exponential	$\ln(y) = \ln(b) + mx$
Power	$\ln(y) = \ln(b) + m\ln(x)$

Access: CAT LR

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	*Intercept:* x_1	*Slope:* x_2

See also: BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

LSQ

Type: Command

Description: Least Squares Solution Command: Returns the minimum norm least squares solution to any system of linear equations where $A \times X = B$.

If B is a vector, the resulting vector has a minimum Euclidean norm $||X||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X - B||$. If B is a matrix, each column of the resulting matrix, X_i, has a minimum Euclidean norm $||X_i||$ over all vector solutions that minimize the residual Euclidean norm $||A \times X_i - B_i||$.

If A has less than full row rank (the system of equations is underdetermined), an infinite number of solutions exist. LSQ returns the solution with the minimum Euclidean length.

If A has less than full column rank (the system of equations is overdetermined), a solution that satisfies all the equations may not exist. LSQ returns the solution with the minimum residuals of $A \times X - B$.

Access: ↰ MATRICES OPERATIONS LSQ

↰ MTH MATRIX LSQ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array]$_B$	*[[matrix]]*$_A$	→	*[array]*$_x$
[[matrix]]$_B$	*[[matrix]]*$_A$	→	*[[matrix]]*$_x$

See also: LQ, RANK, QR, /

LU

Type: Command

Description: LU Decomposition of a Square Matrix Command: Returns the LU decomposition of a square matrix.

When solving an exactly determined system of equations, inverting a square matrix, or computing the determinant of a matrix, the HP 49 factors a square matrix into its Crout LU decomposition using partial pivoting.

The Crout LU decomposition of A is a lower-triangular matrix L, an upper-triangular matrix U with ones on its diagonal, and a permutation matrix P, such that $P \times A = L \times U$. The results satisfy $P \times A \cong L \times U$.

Access: ↰ MATRICES FACTORIZATION LU

↰ MTH MATRIX FACTOR LU

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
[[matrix]]$_A$	→	*[[matrix]]*$_L$	*[[matrix]]*$_U$	*[[matrix]]*$_P$

See also: DET, INV, LSQ, /

MANT

Type: Function

Description: Mantissa Function: Returns the mantissa of the argument.

Access: ↰ MTH REAL MANT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	y_{mant}
'*symb*'	→	'*MANT(symb)*'

See also: SIGN, XPON

MAP

Type: Command

Description: Applies a specified program to a list of objects or values.

- Level 1/Argument 2 contains the list of objects or values
- Level 2/Argument 1 contains the program to apply to the objects or values.

Access: Catalog, CAT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\{list\}_1$	«*program*»	→	$\{list\}_2$

↓MATCH

Type: Command

Description: Match Pattern Down Command: Rewrites an expression that matches a specified pattern.

↓MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace .5 with 'SIN(π/6)'. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name

that begins with &, such as the name '&A', used in replacing 'SIN(&A+&B)' with 'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↓MATCH works from top down; that is, it checks the entire expression first. This approach works well for expansion. An expression expanded during one execution of ↓MATCH will contain additional subexpressions, and those subexpressions can be expanded by another execution of ↓MATCH. Several expressions can be expanded by one execution of ↓MATCH provided none is a subexpression of any other.

Access: CAT ↓MATCH

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$' '$symb_{repl}$' '$symb_{cond}$' }	→	'$symb_2$'	0/1

See also: ↑MATCH

↑MATCH

Type: Command

Description: Bottom-Up Match and Replace Command: Rewrites an expression.

↑MATCH rewrites expressions or subexpressions that match a specified pattern '$symb_{pat}$'. An optional condition, '$symb_{cond}$', can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; 1 if it did, 0 if it did not.

The pattern '$symb_{pat}$' and replacement '$symb_{repl}$' can be normal expressions; for example, you can replace 'SIN(π/6)' with '1/2'. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with &, such as the name '&A', used in replacing 'SIN(&A+π)' with '–SIN(&A)'. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↑MATCH works from bottom up; that is, it checks the lowest level (most deeply nested) subexpressions first. This approach works well for simplification. A subexpression simplified during one execution of ↑MATCH will be a simpler argument of its parent expression, so the parent expression can be simplified by another execution of ↑MATCH.

Several subexpressions can be simplified by one execution of ↑MATCH provided none is a subexpression of any other.

Access: CAT ↑MATCH

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
'$symb_1$'	{ '$symb_{pat}$', '$symb_{repl}$' }	→	'$symb_2$'	0/1
'$symb_1$'	{ '$symb_{pat}$', '$symb_{repl}$', '$symb_{cond}$' }	→	'$symb_2$'	0/1

See also: ↓MATCH

MATR

Type: Command

Description: Displays a menu of matrix commands.

Access: CAT MATR

Input: None

Output: None

See also: ARIT, BASE, CMPLX, DIFF, EXP&LN, SOLVER, TRIGO

MAX

Type: Function

Description: Maximum Function: Returns the greater of two inputs.

Access: ↰ MTH REAL MAX

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	*max(x,y)*
x	'*symb*'	→	'*MAX(x, symb)*'
'*symb*'	*x*	→	'*MAX(symb, x)*'
'$symb_1$'	'$symb_2$'	→	'*MAX*($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	*max*(x_unit_1, y_unit_2)

See also: MIN

MAXR

Type: Function

Description: Maximum Real Function: Returns the symbolic constant MAXR or its numerical representation 9.99999999999E499.

MAXR is the largest numerical value that can be represented by the HP 49.

Access: ↰ MTH CONSTANTS MAXR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'*MAXR*'
	→	*9.99999999999E499*

See also: *e*, *i*, MINR, π

MAXΣ

Type: Command

Description: Maximum Sigma Command: Finds the maximum coordinate value in each of the *m* columns of the current statistical matrix (reserved value ΣDAT).

The maxima are returned as a vector of *m* real numbers, or as a single real number if $m = 1$.

Access: CAT MAXΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{max}
	→	$[x_{max1}\ x_{max2} \cdots x_{maxm}]$

See also: BINS, MEAN, MINΣ, SDEV, TOT, VAR

MCALC

Type: Command

Description: Make Calculated Value Command: Designates a variable as a calculated variable for the multiple-equation solver.

MCALC designates a single variable, a list of variables, or all variables as calculated values.

Access: CAT MCALC

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	
{ *list* }	→	
"ALL"	→	

See also: MUSER

MEAN

Type: Command

Description: Mean Command: Returns the mean of each of the *m* columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

The mean is returned as a vector of *m* real numbers, or as a single real number if $m = 1$. The mean is computed from the formula:

$$\frac{1}{n}\sum_{i=1}^{n} x_i$$

where x_i is the *i*th coordinate value in a column, and *n* is the number of data points.

Access: CAT MEAN

↱ STAT SINGLE-VARIABLE STATISTICS MEAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{mean}
	→	$[x_{mean1}, x_{mean2}, \ldots, x_{meanm}]$

See also: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

MEM

Type: Command

Description: Memory Available Command: Returns the number of bytes of available RAM.

The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG, ↱ UNDO, and ↰ CMD) consume or release varying amounts of memory with each operation.

Before it can assess the amount of memory available, MEM must remove objects in temporary memory that are no longer being used. This clean-up process (also called "garbage collection") also occurs automatically at other times when memory is full. Since this process can slow down calculator operation at undesired times, you can force it to occur at a desired time by executing MEM. In a program, execute MEM DROP.

Access: ↰ PRG MEMORY MEM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x

See also: BYTES

MENU

Type: Command Operation

Description: Display Menu Command: Displays a built-in menu or a library menu, or defines and displays a custom menu.

A built-in menu is specified by a real number x_{menu}. The format of x_{menu} is *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu. If *pp* doesn't correspond to a page of the specified menu, the first page is displayed.

Library menus are specified in the same way as built-in menus, with the library number serving as the menu number.

Custom menus are specified by a list of the form { "*label-object*" *action-object* } or a name containing a list (*name*$_{definition}$). Either argument is stored in reserved variable *CST*, and the custom menu is subsequently displayed.

MENU takes *any* object as a valid argument and stores it in *CST*. However, the calculator can build a custom menu *only* if *CST* contains a list or a name containing a list. Thus, if an object other than a list or name containing a list is supplied to MENU, a Bad Argument Type error will occur when the calculator attempts to display the custom menu.

Access: CAT MENU

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{menu}	→	
{ *list*$_{definition}$ }	→	
'*name*$_{definition}$'	→	
obj	→	

See also: RCLMENU, TMENU

MIN

Type: Function

Description: Minimum Function: Returns the lesser of two inputs.

Access: ↰ MTH REAL MIN

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	*min(x,y)*
x	*'symb'*	→	*'MIN(x, symb)'*
'symb'	*x*	→	*'MIN(symb, x)'*
'symb$_1$'	*'symb$_2$'*	→	*'MIN(symb$_1$, symb$_2$)'*
x_unit$_1$	*y_unit$_2$*	→	*min(x_unit$_1$, y_unit$_2$)*

See also: MAX

MINIFONT→

Type: Command

Description: Minifont: Sets the font that is used as the minifont.

Access: CAT

See also: →MINIFONT

→MINIFONT

Type: Command

Description: Minifont: Returns the font that is set as the minifont.

Access: CAT

See also: MINFONT→

MINIT

Type: Command

Description: Multiple-equation Menu Initialization Command. Creates the reserved variable *MPAR*, which includes the equations in *EQ* and the variables in these equations.

Access: CAT MINIT

See also: MITM, MROOT, MSOLVER

MINR

Type: Function

Description: Minimum Real Function: Returns the symbolic constant MINR or its numerical representation, 1.00000000000E–499.

MINR is the smallest positive numerical value that can be represented by the HP 49.

Access: ← MTH CONSTANTS MINR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*'MINR'*
	→	*1.00000000000E–499*

See also: *e*, *i*, MAXR, π

MINΣ

Type: Command

Description: Minimum Sigma Command: Finds the minimum coordinate value in each of the *m* columns of the current statistics matrix (reserved variable Σ*DAT*).

The minima are returned as a vector of *m* real numbers, or as a single real number if *m* = 1.

Access: CAT MINΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{min}
	→	$\{ x_{min1}\ x_{min2} \ldots x_{minm} \}$

See also: BINS, MAXΣ, MEAN, SDEV, TOT, VAR

MITM

Type: Command

Description: Multiple-equation Menu Item OrderCommand. Changes multiple equation menu titles and order. The argument list contains the variable names in the order you want. Use "" to indicate a blank label. You must include all variables in the original menu and no others.

Access: CAT MITM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"*title*"	{ *list* }	→	

See also: MINIT

MOD

Type: Function

Description: Modulo Function: Returns a remainder defined by: x mod $y = x - y$ floor (x/y)

Mod (x, y) is periodic in x with period y. Mod (x, y) lies in the interval $[0, y)$ for $y > 0$ and in $(y, 0]$ for $y < 0$.

Algebraic syntax: *argument 1* MOD *argument 2*

Access: ↰ MTH REAL MOD

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	x *mod* y
x	'*symb*'	→	'*MOD(x, symb)*'
'*symb*'	x	→	'*MOD(symb, x)*'
'$symb_1$'	'$symb_2$'	→	'*MOD*($symb_1$, $symb_2$)'

See also: FLOOR, /

MROOT

Type: Command

Description: Multiple Roots Command: Uses the multiple-equation solver to solve for one or more variables using the equations in *EQ*. Given a variable name, MROOT returns the found value; with "ALL" MROOT stores a found value for each variable but returns nothing.

Access: CAT MROOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	x
"ALL"	→	

See also: MCALC, MUSER

MSGBOX

Type: Command

Description: Message Box Command: Creates a user-defined message box.

MSGBOX displays "*message*" in the form of a standard message box. Message text too long to appear on the screen is truncated. You can use spaces and new-line characters (↵ F5) to control word-wrapping and line breaks within the message.

Program execution resumes when the message box is exited by selecting OK or CANCEL.

Access: ← PRG OUT MSGBOX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"message"	→	

See also: CHOOSE, INFORM, PROMPT

MSOLVR

Type: Command

Description: Multiple Equation Solver Command: Gets the multiple-equation solver variable menu for the set of equations stored in *EQ*.

The multiple-equation solver application can solve a set of of two or more equations for unknown variables by finding the roots of each equation. The solver uses the list of equations stored in *EQ*.

Access: CAT MSOLVR

Input: None

Output: None

MUSER

Type: Command

Description: Make User-Defined Variable Command: Designates a variable as user-defined for the multiple-equation solver.

MUSER designates a single variable, a list of variables, or all variables as user-defined.

Access: CAT MUSER

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	
{ *list* }	→	
"ALL"	→	

See also: MCALC

→NDISP

Type: Command

Description: Sets the number of program lines displayed on the screen.

Access: CAT →NDISP

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	

NDIST

Type: Command

Description: Normal Distribution Command: Returns the normal probability distribution (bell curve) at x based on the mean m and variance v of the normal distribution.

NDIST is calculated using this formula:

$$ndist(m, v, x) = \frac{e^{-\frac{(x-m)^2}{2v}}}{\sqrt{2\pi v}}$$

Access: ↰ MTH PROBABILITY NDIST

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
m	*v*	*x*	→	*ndist(m, v, x)*

See also: UTPN

NDUPN

Type: RPL command

Description: Duplicates an object *n* times, and returns *n*.

Access: PRG STACK NDUPN

Input/Output:

Level 2	Level 1		$Level_{n+1}$ ... $Level_2$	$Level_1$
obj	*n*	→	*obj* … *obj*	*n*

See also: DUP, DUPDUP, DUPN, DUP2

NEG

Type: Analytic function

Description: Negate Analytic Function: Changes the sign or negates an object.

Negating an array creates a new array containing the negative of each of the original elements. Negating a binary number takes its two's complement (complements each bit and adds 1).

Negating a graphics object "inverts" it (toggles each pixel from on to off, or vice-versa). If the argument is *PICT*, the graphics object stored in *PICT* is inverted.

Access: ↱ CMPLX NEG

↰ MTH COMPLEX NEG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\tilde{z}$	→	$-\tilde{z}$
$\#n_1$	→	$\#n_2$
[array]	→	*[–array]*
'symb'	→	*'–(symb)'*
x_unit	→	*–x_unit*
$grob_1$	→	$grob_2$
$PICT_1$	→	$PICT_2$

See also: ABS, CONJ, NOT, SIGN

NEWOB

Type: Command

Description: New Object Command: Creates a new copy of the specified object.

NEWOB has two main uses:

- NEWOB enables the purging of a library or backup object that has been recalled from a port. NEWOB creates a new, separate copy of the object in memory, thereby allowing the original copy to be purged.
- Creating a new copy of an object that originated in a larger composite object (such as a list) allows you to recover the memory associated with the larger object when that larger object is no longer needed.

Access: ↰ PRG MEMORY NEWOB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*obj*

See also: MEM, PURGE

NEXT

Type: Command

Description: NEXT Command: Ends definite loop structures.

See the FOR and START keyword entries for more information.

Access: ↰ PRG BRANCH NEXT

Input: None

Output: None

See also: FOR, START, STEP

NIP

Type: RPL command

Description: Drops the $(n-1)^{th}$ argument, where n is the number of arguments or items on the stack. (that is, the object on level 2 of the stack). This is equivalent to executing SWAP folowed by DROP in RPN mode.

Access: PRG STACK NIP

Input/Output:

Level 2	Level 1		Level 1
obj_1	obj_2	→	obj_2

See also: DUP, DUPDUP, DUPN, DUP2

NOT

Type: Function

Description: NOT Command: Returns the one's complement or logical inverse of the argument.

When the argument is a binary integer or string, NOT complements each bit in the argument to produce the result.

- A binary integer is treated as a sequence of bits as long as the current wordsize.
- A string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code).

When the argument is a real number or symbolic, NOT does a true/false test. The result is 1 (true) if the argument is zero; it is 0 (false) if the argument is nonzero. This test is usually done on a test result (T/F).

If the argument is an algebraic object, then the result is an algebraic of the form NOT *symb*. Execute →NUM (or set flag –3 before executing NOT) to produce a numeric result from the algebraic result.

Access: ↰ PRG TEST NOT

↱ BASE logic not

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$
T/F	→	$0/1$
"$string_1$"	→	"$string_2$"
'$symb$'	→	'$NOT\ symb$'

See also: AND, OR, XOR

NOVAL

Type: Command

Description: INFORM Place Holder/Result Command: Place holder for reset and initial values in user-defined dialog boxes. NOVAL is returned when a field is empty.

NOVAL is used to mark an empty field in a user-defined dialog box created with the INFORM command. INFORM defines fields sequentially. If default values are used for those fields, the defaults must be defined in the same order as the fields were defined. To skip over (not provide defaults for) some of the fields, use the NOVAL command.

After INFORM terminates, NOVAL is returned if a field is empty and OK or ENTER is selected.

Access: ↰ PRG IN NOVAL

Input: None

Output: None

See also: INFORM

NΣ

Type: Command

Description: Number of Rows Command: Returns the number of rows in the current statistical matrix (reserved variable ΣDAT).

Access: CAT NΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	n_{rows}

See also: ΣX, ΣX*Y, ΣX^2, ΣY, ΣY^2

NSUB

Type: Command

Description: Number of Sublist Command: Provides a way to access the current sublist position during an iteration of a program or command applied using DOSUBS.

Returns an Undefined Local Name error if executed when DOSUBS is not active.

Access: ↰ PRG LIST PROCEDURES NSUB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	n_{position}

Input: None

Output: None

See also: DOSUBS, ENDSUB

NUM

Type: Command

Description: Character Number Command: Returns the character code *n* for the first character in the string.

The character codes are an extension of ISO 8859/1.

The number of a character can be found by accessing the Characters tool (↱ CHARS) and highlighting that character. The number appears near the bottom of the screen.

Access: ↰ PRG TYPE NUM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"string"	→	*n*

See also: CHR, POS, REPL, SIZE, SUB

NUMX

Type: Command

Description: Number of X-Steps Command: Sets the number of *x*-steps for each *y*-step in 3D perspective plots.

The number of *x*-steps is the number of independent variable points plotted for each dependent variable point plotted. This number must be 2 or more. This value is stored in the reserved variable VPAR. YSLICE is the only 3D plot type that does not use this value.

Access: CAT NUMX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_x	→	

See also: NUMY

NUMY

Type: Command

Description: Number of Y-Steps Command: Sets the number of *y*-steps across the view volume in 3D perspective plots.

The number of y-steps is the number of dependent variable points plotted across the view volume. This number must be 2 or more. This value is stored in the reserved variable VPAR.

Access: CAT NUMY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_y	→	

See also: NUMX

O to P

OBJ→

Type: Command

Description: Object to Stack Command: Separates an object into its components. For some object types, the *number* of components is returned as item *n*+1 (stack level 1).

If the argument is a complex number, list, array, or string, OBJ→ provides the same functions as C→R, LIST→, ARRY→, and STR→, respectively. For lists, OBJ→ also returns the number of list elements. If the argument is an array, OBJ→ also returns the dimensions { *m n* } of the array, where *m* is the number of rows and *n* is the number of columns.

For algebraic objects, OBJ→ returns the arguments of the top-level (least-nested) function (*arg*$_1$ … *arg*$_n$), the number of arguments of the top-level function (*n*), and the name of the top-level function (*function*).

If the argument is a string, the object sequence defined by the string is executed.

Access: ↰ PRG TYPE OBJ→

Input/Output:

Level 1/Argument 1		Level$_{n+1}$/Item$_1$		Level$_2$/Item$_n$	Level$_1$/Item$_{n+1}$
(x, y)	→		→	*x*	*y*
{ *obj*$_1$, … ,*obj*$_n$ }	→	*obj*$_1$	→	*obj*$_n$	*n*
[*x*$_1$, … ,*x*$_n$]	→	*x*$_1$	→	*x*$_n$	{ *n* }
[[*x*$_{1\ 1}$, … ,*x*$_{m\ n}$]]	→	*x*$_{1\ 1}$	→	*x*$_{m\ n}$	{ *m, n* }
"obj"	→		→		*evaluated object*
'symb'	→	*arg*$_1$ … *arg*$_n$	→	*n*	*'function'*
x_unit	→		→	*x*	*1_unit*
:tag:obj	→		→	*obj*	*"tag"*

See also: ARRY→, C→R, DTAG, EQ→, LIST→, R→C, STR→, →TAG

OCT

Type: Command

Description: Octal Mode Command: Selects octal base for binary integer operations. (The default base is decimal.)

Binary integers require the prefix #. Binary integers entered and returned in octal base automatically show the suffix o. If the current base is not octal, enter an octal number by ending it with o. It will be displayed in the current base when entered.

The current base does not affect the internal representation of binary integers as unsigned binary numbers.

Access: CAT OCT

Input: None

Output: None

See also: BIN, DEC, HEX, RCWS, STWS

OFF

Type: Command

Description: Off Command: Turns off the calculator.

When executed from a program, that program will resume execution when the calculator is turned on. This provides a programmable "autostart."

Access: ↱ OFF

Input: None

Output: None

See also: CONT, HALT, KILL

OPENIO

Type: Command

Description: Open I/O Port Command: Opens a serial port using the I/O parameters in the reserved variable *IOPAR*.

Since all HP 49 Kermit-protocol commands automatically effect an OPENIO first, OPENIO is not normally needed, but can be used if an I/O transmission does not work. OPENIO is necessary for interaction with devices that interpret a closed port as a break.

OPENIO is also necessary for the automatic reception of data into the input buffer using non-Kermit commands. If the port is closed, incoming characters are ignored. If the port is open, incoming characters are automatically placed in the input buffer. These characters can be detected with BUFLEN, and can be read out of the input buffer using SRECV.

If the port is already open, OPENIO does not affect the data in the input buffer. However, if the port is closed, executing OPENIO clears the data in the input buffer.

Access: (CAT) OPENIO

Input: None

Output: None

See also: BUFLEN, CLOSEIO, SBRK, SRECV, STIME, XMIT

OR

Type: Function

Description: OR Function: Returns the logical OR of two arguments.

When the arguments are binary integers or strings, OR does a bit-by-bit (base 2) logical comparison.

- An argument that is a binary integer is treated as a sequence of bits as long as the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments as shown in the following table:

bit_1	bit_2	bit_1 **OR** bit_2
0	0	0
0	1	1
1	0	1
1	1	1

- An argument that is a string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, OR simply does a true/false test. The result is 1 (true) if either or both arguments are nonzero; it is 0 (false) if both arguments are zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form $symb_1$ OR $symb_2$. Execute →NUM (or set flag –3 before executing OR) to produce a numeric result from the algebraic result.

Access: ⮡ (BASE) BASE LOGIC OR

⮢ (PRG) test or

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	$\#n_2$	→	$\#n_3$
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	*0/1*
T/F	'$symb$'	→	'T/F OR $symb$'
'$symb$'	T/F	→	'$symb$ OR T/F'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ OR $symb_2$'

See also: AND, NOT, XOR

ORDER

Type: Command

Description: Order Variables Command: Reorders the variables in the current directory (shown in the VAR menu) to the order specified.

The names that appear first in the list will be the first to appear in the VAR menu. Variables not specified in the list are placed after the reordered variables.

If the list includes the name of a large subdirectory, there may be insufficient memory to execute ORDER.

Access: ↰ PRG MEMORY DIRECTORY ORDER

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *global*$_1$ ... *global*$_n$ }	→	

Flags: None

See also: VARS

OVER

Type: RPL command

Description: Over Command: Returns a copy to stack level 1 of the object in level 2.

Access: ↰ PRG STACK OVER

Input/Output:

Level 2	Level 1		Level 3	Level 2	Level 1
obj$_1$	*obj*$_2$	→	*obj*$_1$	*obj*$_2$	*obj*$_1$

See also: PICK, ROLL, ROLLD, ROT, SWAP

PARAMETRIC

Type: Command

Description: Parametric Plot Type Command: Sets the plot type to PARAMETRIC.

When the plot type is PARAMETRIC, the DRAW command plots the current equation as a complex-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

$$\{ (x_{min}, y_{min}), (x_{max}, y_{max}), \textit{indep, res, axes, ptype, depend} \}$$

For plot type PARAMETRIC, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a list containing a name that specifies the independent variable, and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). Note that the default value is *X*. If *X* is not modified and included in a list with a plotting range, the values in (x_{min},y_{min}) and (x_{max},y_{max}) are used as the plotting range, which generally leads to meaningless results.
- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval equal to 1/130 of the difference between the maximum and minimum values in *indep* (the plotting range).
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command PARAMETRIC places the name PARAMETRIC in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The contents of *EQ* must be an expression or program; it cannot be an equation. It is evaluated for each value of the independent variable. The results, which must be complex numbers, give the coordinates of the points to be plotted. Lines are drawn between plotted points unless flag –31 is set.

Access: CAT PARAMETRIC

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PARITY

Type: Command

Description: Parity Command: Sets the parity value in the reserved variable *IOPAR*.

Legal values are shown below. A negative value means the HP 49 does not check parity on bytes received during Kermit transfers or with SRECV. Parity is still used during data transmission, however.

n-Value	Meaning
0	no parity (the default value)
1	odd parity
2	even parity
3	mark
4	space

Access: CAT PARITY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{parity}	→	

See also: BAUD, CKSM, TRANSIO

PARSURFACE

Type: Command

Description: PARSURFACE Plot Type Command: Sets plot type to PARSURFACE.

When plot type is set to PARSURFACE, the DRAW command plots an image graph of a 3-vector-valued function of two variables. PARSURFACE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

$\{ x_{left}, x_{right}, y_{near}, y_{far}, z_{low}, z_{high}, x_{min}, x_{max}, y_{min}, y_{max}, x_{eye}, y_{eye}, z_{eye}, x_{step}, y_{step} \}$

For plot type PARSURFACE, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are real numbers that specify the input region's width. The default value is (–1,1).
- y_{min} and y_{max} are real numbers that specify the input region's depth. The default value is (–1,1).
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

$\{ (x_{min}, y_{min}), (x_{max}, y_{max}), indep, res, axes, ptype, depend \}$

For plot type PARSURFACE, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.

- *ptype* is a command name specifying the plot type. Executing the command PARSURFACE places the name PARSURFACE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT PARSURFACE

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FAST3D, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PATH

Type: Command

Description: Current Path Command: Returns a list specifying the path to the current directory.

The first directory is always *HOME*, and the last directory is always the current directory.

If a program needs to switch to a specific directory, it can do so by evaluating a directory list, such as one created earlier by PATH.

Access: ↰ PRG MEMORY DIRECTORY PATH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ *HOME directory-name*$_1$ ... *directory-name*$_n$ }

See also: CRDIR, HOME, PGDIR, UPDIR

PCOEF

Type: Command

Description: Monic Polynomial Coefficients Command: Returns the coefficients of a monic polynomial (a polynomial with a leading coefficient of 1) having specific roots.

The argument must be a real or complex array of length *n* containing the polynomial's roots. The result is a real or complex vector of length *n*+1 containing the coefficients listed from highest order to lowest, with a leading coefficient of 1.

Access: ↰ ARITH POLYNOMIAL PCOEF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]$_{roots}$	→	[*array*]$_{coefficients}$

See also: PEVAL, PROOT

PCONTOUR

Type: Command

Description: PCONTOUR Plot Type Command: Sets the plot type to PCONTOUR.

When plot type is set PCONTOUR, the DRAW command plots a contour-map view of a scalar function of two variables. PCONTOUR requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

{ x_{left} x_{right} y_{near} y_{far} z_{low} z_{high} x_{min} x_{max} y_{min} y_{max} x_{eye} y_{eye} z_{eye} x_{step} y_{step} }

For plot type PCONTOUR, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min}, y_{min}) (x_{max}, y_{max}) *indep res axes ptype depend* }

For plot type PCONTOUR, the elements of *PPAR* are used as follows:

- (x_{min}, y_{min}) is not used.
- (x_{max}, y_{max}) is not used.

- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command PCONTOUR places the name PCONTOUR in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT PCONTOUR

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

PCOV

Type: Command

Description: Population Covariance Command: Returns the population covariance of the independent and dependent data columns in the current statistics matrix (reserved variable Σ*DAT*).

The columns are specified by the first two elements in reserved variable Σ*PAR*, set by XCOL and YCOL respectively. If Σ*PAR* does not exist, PCOV creates it and sets the elements to their default values (1 and 2).

The population covariance is calculated with the following formula:

$$\frac{1}{n}\sum_{k=1}^{n}(x_{kn_1}-\overline{x_{n_1}})(x_{kn_2}-\overline{x_{n_2}})$$

where x_{kn_1} is the kth coordinate value in column n_1, x_{kn_2} is the kth coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

Access: CAT PCOV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{\text{pcovariance}}$

See also: COLΣ, CORR, COV, PREDX, PREDY, XCOL, YCOL

PDIM

Type: Command

Description: PICT Dimension Command: Replaces *PICT* with a blank *PICT* of the specified dimensions.

If the arguments are complex numbers, PDIM changes the size of *PICT* and makes the arguments the new values of (x_{min}, y_{min}) and (x_{max}, y_{max}) in the reserved variable *PPAR*. Thus, the scale of a subsequent plot is not changed. If the arguments are binary integers, *PPAR* remains unchanged, so the scale of a subsequent plot *is* changed.

PICT cannot be smaller than 131 pixels wide × 64 pixels high, nor wider than 2048 pixels (height is unlimited).

Access: ⬅ PRG PICT PDIM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
(x_{min}, y_{min})	(x_{max}, y_{max})	→	
$\#n_{width}$	$\#m_{height}$	→	

See also: PMAX, PMIN

PERM

Type: Function

Description: Permutations Function: Returns the number of possible permutations of *n* items taken *m* at a time.

The formula used to calculate $P_{n,m}$ is:

$$P_{n,m} = \frac{n!}{(n-m)!}$$

The arguments *n* and *m* must each be less than 10^{12}. If $n < m$, zero is returned.

Access: ⬅ MTH PROBABILITY PERM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*m*	→	$P_{n,m}$
'$symb_n$'	*m*	→	'$PERM(symb_{n,m})$'
n	'$symb_m$'	→	'$PERM(n, symb_m)$'
'$symb_n$'	'$symb_m$'	→	'$PERM(symb_n,symb_m)$'

See also: COMB, !

PEVAL

Type: Command

Description: Polynomial Evaluation Command: Evaluates an *n*-degree polynomial at *x*.

The arguments must be an array of length $n+1$ containing the polynomial's coefficients listed from highest order to lowest, and the value *x* at which the polynomial is to be evaluated.

Access: CAT PEVAL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$[\ array\ ]_{coefficients}$	*x*	→	*p(x)*

See also: PCOEF, PROOT

PGDIR

Type: Command

Description: Purge Directory Command: Purges the named directory (whether empty or not).

Access: ↰ PRG MEMORY DIRECTORY PGDIR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*global*'	→	

See also: CLVAR, CRDIR, HOME, PATH, PURGE, UPDIR

PICK

Type: RPN command

Description: Pick Object Command: Copies the contents of a specified stack level to level 1.

Access: ↰ PRG STACK PICK

Input/Output:

L_{n+1}...	L_2	L_1		L_{n+1}	L_2	L_1
obj_n ...	obj_1	n	→	obj_n ...	obj_1	obj_1

L = level

See also: DUP, DUPN, DUP2, OVER, ROLL, ROLLD, ROT, SWAP

PICK3

Type: RPN command

Description: Duplicates the object on level 3 of the stack.

Access: PRG STACK PICK3

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
obj_1	obj_2	obj_3	→	obj_1	obj_2	obj_3	obj_3

L = Level; A = Argument; I =Item

See also: PICK, OVER, DUP

PICT

Type: Command

Description: PICT Command: Puts the name PICT on the stack.

PICT is the name of a storage location in calculator memory containing the current graphics object. The command PICT enables access to the contents of that memory location as if it were a variable. Note, however, that *PICT* is *not* a variable as defined in the HP 49: its name cannot be quoted, and only graphics objects may be stored in it.

If a graphics object smaller than 131 wide × 64 pixels high is stored in *PICT*, it is enlarged to 131 × 64. A graphics object of unlimited pixel height and up to 2048 pixels wide can be stored in *PICT*.

Access: ↰ PRG PICT PICT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*PICT*

See also: GOR, GXOR, NEG, PICTURE, PVIEW, RCL, REPL, SIZE, STO, SUB

PICTURE

Type: Command

Description: Picture Environment Command: Selects the Picture environment (that is, selects the graphics display and activates the graphics cursor and Picture menu).

When executed from a program, PICTURE suspends program execution until CANCEL is pressed.

Access: CAT PICTURE

Input: None

Output: None

See also: PICTURE, PVIEW, TEXT

PINIT

Type: Command

Description: Port Initialize Command: Initializes all currently active ports. It may affect data already stored in a port.

Access: CAT PINIT

Input: None

Output: None

PIX?

Type: Command

Description: Pixel On? Command: Tests whether the specified pixel in *PICT* is on; returns 1 (true) if the pixel is on, and 0 (false) if the pixel is off.

Access: ⮌ PRG PICT PIX?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	*0/1*
{ *#n #m* }	→	*0/1*

See also: PIXON, PIXOFF

PIXOFF

Type: Command

Description: Pixel Off Command: Turns off the pixel at the specified coordinate in *PICT*.

Access: ⮌ PRG PICT PIXOFF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	
{ *#n #m* }	→	

See also: PIXON, PIX?

PIXON

Type: Command

Description: Pixel On Command: Turns on the pixel at the specified coordinate in *PICT*.

Access: ↰ PRG PICT PIXON

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	
{ #*n* #*m* }	→	

See also: PIXOFF, PIX?

PKT

Type: Command

Description: Packet Command: Used to send command "packets" (and receive requested data) to a Kermit server.

To send HP 49 objects, use SEND.

PKT allows additional commands to be sent to a Kermit server.

The packet data, packet type, and the response to the packet transmission are all in string form. PKT first does an I (*initialization*) packet exchange with the Kermit server, then sends the server a packet constructed from the data and packet-type arguments supplied to PKT. The response to PKT will be either an acknowledging string (possibly blank) or an error packet (see KERRM).

For the *type* argument, only the first letter is significant.

Access: CAT PKT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"data"	*"type"*	→	*"response"*

See also: CLOSEIO, KERRM, SERVER

PLOTADD

Type: Function

Description: Adds a function to the existing plot function list, and opens the Plot Setup screen.

Access: Catalog, CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(symb)	→	

PMAX

Type: Command

Description: PICT Maximum Command: Specifies (*x*, *y*) as the coordinates at the upper right corner of the display.

The complex number (*x*, *y*) is stored as the second element in the reserved variable *PPAR*.

Access: CAT PMAX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	

See also: PDIM, PMIN, XRNG, YRNG

PMIN

Type: Command

Description: PICT Minimum Command: Specifies (*x*, *y*) as the coordinates at the lower left corner of the display.

The complex number (*x*, *y*) is stored as the first element in the reserved variable *PPAR*.

Access: CAT PMIN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	

See also: PDIM, PMAX, XRNG, YRNG

POLAR

Type: Command

Description: Polar Plot Type Command: Sets the plot type to POLAR.

When the plot type is POLAR, the DRAW command plots the current equation in polar coordinates, where the independent variable is the polar angle and the dependent variable is the radius. The current equation is specified in the reserved variable *EQ*.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

$$\{ (x_{min}, y_{min}) (x_{max}, y_{max}) \textit{ indep res axes ptype depend } \}$$

For plot type POLAR, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is 0, which specifies an interval of 2 degrees, 2 grads, or $\pi/90$ radians.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command POLAR places the name POLAR in *ptype*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the default minimum value is 0 and the default maximum value corresponds to one

full circle in the current angle mode (360 degrees, 400 grads, or 2π radians). Lines are drawn between plotted points unless flag –31 is set.

If flag –28 is set, all equations are plotted simultaneously.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation.

Form of Current Equation	Plotting Action
expr = expr	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal
name = expr	Only the expression is plotted

Access: (CAT) POLAR

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

POS

Type: Command

Description: Position Command: Returns the position of a substring within a string or the position of an object within a list.

If there is no match for *obj* or *substring*, POS returns zero.

Access: ↰ PRG CHARS POS

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"string"	*"substring"*	→	*n*
{ *list* }	*obj*	→	*n*

See also: CHR, NUM, REPL, SIZE, SUB

PR1

Type: Command

Description: Print Level 1 Command: Prints an object in multiline printer format.

All objects except strings are printed with their identifying delimiters. Strings are printed without the leading and trailing " delimiters.

Multiline printer format is similar to multiline display format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don't fit on the same line.
- Grobs are printed graphically.
- Arrays are printed with a numbered heading for each row and with a column number before each element.

For example, the 2 × 3 array

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

would be printed as follows:

Array (2 3)
Row 1
1] 1
2] 2
3] 3

Row 2
1] 4
2] 5
3] 6

Access: CAT PR1

Input: None

Output: None

See also: CR, DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR

PREDV

Type: Command

Description: Predicted y-Value Command: Returns the predicted dependent-variable value $y_{dependent}$, based on the independent-variable value $x_{independent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable Σ*PAR*.

PREDV is the same as PREDY. See PREDY.

Access: CAT PREDV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$x_{independent}$	→	$y_{dependent}$

PREDX

Type: Command

Description: Predicted x-Value Command: Returns the predicted independent-variable value $x_{independent}$, based on the dependent-variable value $y_{dependent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable Σ*PAR*.

The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable Σ*PAR*. For the linear statistical model, the equation used is this:

$$y_{dependent} = (mx_{independent}) + b$$

where m is the slope (the third element in Σ*PAR*) and b is the intercept (the fourth element in Σ*PAR*).

For the other statistical models, the equations used by PREDX are listed in the LR entry.

If PREDX is executed without having previously generated regression coefficients in Σ*PAR*, a default value of zero is used for both regression coefficients, and an error results.

Access: (CAT) PREDX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$y_{dependent}$	→	$x_{independent}$

See also: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDY, PWRFIT, XCOL, YCOL

PREDY

Type: Command

Description: Predicted y-Value Command: Returns the predicted dependent-variable value $y_{dependent}$, based on the independent-variable value $x_{independent}$, the currently selected statistical model, and the current regression coefficients in the reserved variable Σ*PAR*.

The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable Σ*PAR*. For the linear statistical model, the equation used is this:

$$y_{dependent} = (mx_{independent}) + b$$

where m is the slope (the third element in Σ*PAR*) and b is the intercept (the fourth element in Σ*PAR*).

For the other statistical models, the equations used by PREDY are listed in the LR entry.

If PREDY is executed without having previously generated regression coefficients in Σ*PAR*, a default value of zero is used for both regression coefficients–in this case PREDY will return 0 for statistical models LINFIT and LOGFIT, and error for statistical models EXPFIT and PWRFIT.

Access: CAT PREDY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$x_{independent}$	→	$y_{dependent}$

See also: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDX, PWRFIT, XCOL, YCOL

PRLCD

Type: Command

Description: Print LCD Command: Prints a pixel-by-pixel image of the current display (excluding the annunciators).

The width of the printed image of characters in the display is narrower using PRLCD than using a print command such as PR1. The difference results from the spacing between characters. On the display there is a single blank column between characters, and PRLCD prints this spacing. Print commands such as PR1 print two blank columns between adjacent characters.

Access: CAT PRLCD

Input: None

Output: None

See also: CR, DELAY, OLDPRT, PRST, PRSTC, PRVAR, PR1

PROMPT

Type: Command

Description: Prompt Command: Displays the contents of "*prompt*" in the status area, and halts program execution.

Access: ↰ PRG IN PROMPT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"prompt"	→	

See also: CONT, DISP, FREEZE, HALT, INFORM, INPUT, MSGBOX

PROMPTSTO

Type: Command

Description: Prompt Command: Creates a variable with the name supplied as an argument, prompts for a value, and stores the value you enter in the variable.

Access: CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"global"	→	

See also: PROMT, STO

PROOT

Type: Command

Description: Polynomial Roots Command: Returns all roots of an *n*-degree polynomial having real or complex coefficients.

For an n^{th}-order polynomial, the argument must be a real or complex array of length $n+1$ containing the coefficients listed from highest order to lowest. The result is a real or complex vector of length n containing the computed roots.

PROOT interprets leading coefficients of zero in a limiting sense. As a leading coefficient approaches zero, a root of the polynomial approaches infinity: therefore, if flag –22 is clear (the default), PROOT reports an Infinite Result error if a leading coefficient is zero. If flag –22 is set, PROOT returns a root of (MAXREAL,0) for each leading zero in an array containing real coefficients, and a root of (MAXREAL,MAXREAL) for each leading zero in an array containing complex coefficients.

Access: ↰ ARITH POLYNOMIAL PROOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]$_{\text{coefficients}}$	→	[*array*]$_{\text{roots}}$

See also: PCOEF, PEVAL

PRST

Type: Command

Description: Print Stack Command: Prints all objects in the stack, starting with the object on the highest level.

Objects are printed in multiline printer format. See the PR1 entry for a description of multiline printer format.

Access: CAT PRST

Input: None

Output: None

See also: CR, DELAY, OLDPRT, PRLCD, PRSTC, PRVAR, PR1

PRSTC

Type: Command

Description: Print Stack (Compact) Command: Prints in compact form all objects in the stack, starting with the object on the highest level.

Compact printer format is the same as compact display format. Multiline objects are truncated and appear on one line only.

Access: CAT PRSTC

Input: None

Output: None

See also: CR, DELAY, OLDPRT, PRLCD, PRST, PRVAR, PR1

PRVAR

Type: Command

Description: Print Variable Command: Searches the current directory path or port for the specified variables and prints the name and contents of each variable.

Objects are printed in multiline printer format. See the PR1 entry for a description of multiline printer format.

Access: CAT PRVAR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'name'	→	
{ *name*$_1$ *name*$_2$ ... }	→	
:n$_{port}$ *:'global'*	→	

See also: CR, DELAY, OLDPRT, PR1, PRLCD, PRST, PRSTC

PSDEV

Type: Command

Description: Population Standard Deviation Command: Calculates the population standard deviation of each of the *m* columns of coordinate values in the current statistics matrix (reserved variable Σ*DAT*).

PSDEV returns a vector of *m* real numbers, or a single real number if *m* = 1. The population standard deviation is computed using this formula:

$$\sqrt{\frac{1}{n}\sum_{k=1}^{n}(x_k-\bar{x})^2}$$

where x_k is the *k*th coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and *n* is the number of data points.

Access: CAT PSDEV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{psdev}
	→	$\lceil x_{psdev1}\ x_{psdev2} \cdots x_{psdevm}\ \}$

See also: MEAN, PCOV, PVAR, SDEV, TOT, VAR

PURGE

Type: Command

Description: Purge Command: Purges the named variables or empty subdirectories from the current directory.

PURGE executed in a program does not save its argument for recovery by LASTARG.

To empty a named directory before purging it, use PGDIR.

To help prepare a list of variables for purging, use VARS.

Purging *PICT* replaces the current graphics object with a 0 × 0 graphics object.

If a list of objects (with global names, backup objects, library objects, or *PICT*) for purging contains an invalid object, then the objects preceding the invalid object are purged, and the error Bad Argument Type occurs.

To purge a library or backup object, tag the library number or backup name with the appropriate port number ($:n_{port}$), which must be in the range from 0 to 2. For a backup object, the port number can be replaced with the wildcard character &, in which case the HP 49 will search ports 0 through 2, and then main memory for the named backup object.

A library object must be detached before it can be purged from the *HOME* directory.

Neither a library object nor a backup object can be purged if it is currently "referenced" internally by stack pointers (such as an object on the stack, in a local variable, on the LAST stack, or on an internal return stack). This produces the error Object in Use. To avoid these restrictions, use NEWOB before purging. (See NEWOB.)

Access: ↰ PRG MEMORY PURGE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*global*'	→	
{ $global_1$ ... $global_n$ }	→	
PICT	→	
$:n_{port}$ $:name_{backup}$	→	
$:n_{port}$ $:n_{library}$	→	

See also: CLEAR, CLVAR, NEWOB, PGDIR

PUT

Type: Command

Description: Put Element Command: Replaces the object at a specified position (second input) in a specified array or list (first input) with a specifed object (third input). If the array or list is unnamed, returns the new array or list.

For matrices, $n_{position}$ counts in row order.

Access: ↰ PRG LIST ELEMENTS PUT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$n_{position}$	z_{put}	→	$[[\ matrix\ ]]_2$
$[[\ matrix\ ]]_1$	$\{\ n_{row}\ m_{col}\ \}$	z_{put}	→	$[[\ matrix\ ]]_2$
$'name_{matrix}'$	$n_{position}$	z_{put}	→	
$'name_{matrix}'$	$\{\ n_{row}\ m_{col}\ \}$	z_{put}	→	
$[\ vector\ ]_1$	$n_{position}$	z_{put}	→	$[\ vector\ ]_2$
$[\ vector\ ]_1$	$\{\ n_{position}\ \}$	z_{put}	→	$[\ vector\ ]_2$
$'name_{vector}'$	$n_{position}$	z_{put}	→	
$'name_{vector}'$	$\{\ n_{position}\ \}$	z_{put}	→	
$\{\ list\ \}_1$	$n_{position}$	obj_{put}	→	$\{\ list\ \}_2$
$\{\ list\ \}_1$	$\{\ n_{position}\ \}$	obj_{put}	→	$\{\ list\ \}_2$
$'name_{list}'$	$n_{position}$	obj_{put}	→	
$'name_{list}'$	$\{\ n_{position}\ \}$	obj_{put}	→	

See also: GET, GETI, PUTI

PUTI

Type: Command

Description: Put and Increment Index Command: Replaces the object at a specified position (second input) in a specified array or list (first input) with a specifed object (third input), returning a new array or list together with the next position in the array or list.

For matrices, the position is incremented in *row* order.

Unlike PUT, PUTI returns a named array or list. This enables a subsequent execution of PUTI at the next position of a named array or list.

Access: ↩ PRG LIST ELEMENTS PUTI

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
$[[\ matrix\ ]]_1$	$n_{position1}$	$\tilde{z}_{put}$	→	$[[\ matrix\ ]]_2$	$n_{position2}$
$[[\ matrix\ ]]_1$	$\{\ n_{row}\ m_{col}\ \}_1$	$\tilde{z}_{put}$	→	$[[\ matrix\ ]]_2$	$\{\ n_{row}\ m_{col}\ \}_2$
$'name_{matrix}'$	$n_{position1}$	$\tilde{z}_{put}$	→	$'name_{matrix}'$	$n_{position2}$
$'name_{matrix}'$	$\{\ n_{row}\ m_{col}\ \}_1$	$\tilde{z}_{put}$	→	$'name_{matrix}'$	$\{\ n_{row}\ m_{col}\ \}_2$
$[\ vector\ ]_1$	$n_{position1}$	$\tilde{z}_{put}$	→	$[\ vector\ ]_2$	$n_{position2}$
$[\ vector\ ]_1$	$\{\ n_{position1}\ \}$	$\tilde{z}_{put}$	→	$[\ vector\ ]_2$	$\{\ n_{position2}\ \}$
$'name_{vector}'$	$n_{position1}$	$\tilde{z}_{put}$	→	$'name_{vector}'$	$n_{position2}$
$'name_{vector}'$	$\{\ n_{position1}\ \}$	$\tilde{z}_{put}$	→	$'name_{vector}'$	$\{\ n_{position2}\ \}$
$\{\ list\ \}_1$	$n_{position1}$	obj_{put}	→	$\{\ list\ \}_2$	$n_{position2}$
$\{\ list\ \}_1$	$\{\ n_{position1}\ \}$	obj_{put}	→	$\{\ list\ \}_2$	$\{\ n_{position2}\ \}$
$'name_{list}'$	$n_{position1}$	obj_{put}	→	$'name_{list}'$	$n_{position2}$
$'name_{list}'$	$\{\ n_{position1}\ \}$	obj_{put}	→	$'name_{list}'$	$\{\ n_{position2}\ \}$

L = level; A = argument; I = item

See also: GET, GETI, PUT

PVAR

Type: Command

Description: Population Variance Command: Calculates the population variance of the coordinate values in each of the m columns in the current statistics matrix (ΣDAT).

The population variance (equal to the square of the population standard deviation) is returned as a vector of m real numbers, or as a single real number if $m = 1$. The population variances are computed using this formula:

$$\frac{1}{n}\sum_{k=1}^{n}(x_k-\bar{x})^2$$

where x_k is the kth coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and n is the number of data points.

Access: CAT PVAR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{pvariance}$
	→	$[x_{pvariance1}, \ldots, x_{pvariancem}]$

See also: MEAN, PCOV, PSDEV, SDEV, VAR

PVARS

Type: Command

Description: Port-Variables Command: Returns a list of the backup objects ($:n_{port}:name$) and the library objects ($:n_{port}:n_{library}$) in the specified port. Also returns the available memory size (RAM) or the memory type.

The port number, n_{port}, must be in the range from 0 to 2.

If $n_{port} = 0$, then *memory* is bytes of available main RAM; otherwise *memory* is bytes of available RAM in the specified port.

Access: CAT PVARS

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
n_{port}	→	$\{ :n_{port} :name_{backup} \ldots \}$	*memory*
n_{port}	→	$\{ :n_{port} :n_{library} \ldots \}$	*memory*

See also: PVARS, VARS

PVIEW

Type: Command

Description: PICT View Command: Displays *PICT* with the specified coordinate at the upper left corner of the graphics display.

PICT must fill the entire display on execution of PVIEW. Thus, if a position other than the upper left corner of *PICT* is specified, *PICT* must be large enough to fill a rectangle that extends 131 pixels to the right and 64 pixels down.

If PVIEW is executed from a program with a coordinate argument (versus an empty list), the graphics display persists only until the keyboard is ready for input (for example, until the end

of program execution). However, the FREEZE command freezes the display until a key is pressed.

If PVIEW is executed with an *empty* list argument, *PICT* is centred in the graphics display with scrolling mode activated. In this case, the graphics display persists until CANCEL is pressed.

PVIEW does *not* activate the graphics cursor or the Picture menu. To activate the graphics cursor and Picture menu, execute PICTURE.

Access: ↰ PRG PICT PVIEW

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x,y)	→	
{ #*n*, #*m* }	→	
{ }	→	

See also: FREEZE, PICTURE, TEXT

PWRFIT

Type: Command

Description: Power Curve Fit Command: Stores PWRFIT as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the power curve fitting model.

LINFIT is the default specification in Σ*PAR*.

Access: CAT PWRFIT

Input: None

Output: None

See also: BESTFIT, EXPFIT, LINFIT, LOGFIT, LR

PX→C

Type: Command

Description: Pixel to Complex Command: Converts the specified pixel coordinates to user-unit coordinates.

The user-unit coordinates are derived from the (x_{min},y_{min}) and (x_{max},y_{max}) parameters in the reserved variable *PPAR*. The coordinates correspond to the geometrical center of the pixel.

Access: ↰ PRG PICT PX→C

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ #*n* #*m* }	→	(*x*,*y*)

See also: C→PX

HP 49G Advanced Users Guide

Volume 1

Part D: Other Commands: Q to S

Introduction

This volume details the HP 49G commands and functions that are not computer algebra-specific. See part A, Computer algebra commands and functions, for information on computer algebra commands.

For each operation, the following details are provided:

Type: Function or command. Functions can be used as a part of an algebraic objects and commands cannot.

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys.

Input/Output: The input argument or arguments that the operation needs, and the outputs it produces.

See also: Related functions or commands

Q to R

→Q

Type: Command

Description: To Quotient Command: Returns a rational form of the argument.

The rational result is a "best guess", since there might be more than one rational expression consistent with the argument. →Q finds a quotient of integers that agrees with the argument to within the number of decimal places specified by the display format mode.

→Q also acts on numbers that are part of algebraic expressions or equations.

Access: CAT →Q

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	'*a*/*b*'
(*x*,*y*)	→	'*a*/*b* + *c*/*d**i*'
'*symb*$_1$'	→	'*symb*$_2$'

See also: →Qπ, /

→Qπ

Type: Command

Description: To Quotient Times π Command: Returns a rational form of the argument, *or* a rational form of the argument with π factored out, whichever yields the smaller denominator.

→Qπ computes two quotients (rational expressions) and compares them: the quotient of the argument, and the quotient of the argument divided by π. It returns the fraction with the smaller denominator; if the argument was divided by π, then π is a factor in the result.

The rational result is a "best guess", since there might be more than one rational expression consistent with the argument. →Qπ finds a quotient of integers that agrees with the argument to the number of decimal places specified by the display format mode.

→Qπ also acts on numbers that are part of algebraic expressions or equations.

For a complex argument, the real or imaginary part (or both) can have π as a factor.

Access: CAT →Qπ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	'$a/b*\pi$'
x	→	'a/b'
'$symb_1$'	→	'$symb_2$'
(x,y)	→	'$a/b*\pi + c/d*\pi*i$'
(x,y)	→	'$a/b + c/d*i$'

See also: →Q, /, π

QR

Type: Command

Description: QR Factorization of a Matrix Command: Returns the QR factorization of an $m \times n$ matrix.

QR factors an $m \times n$ matrix A into three matrices:

- Q is an $m \times m$ orthogonal matrix.
- R is an $m \times n$ upper trapezoidal matrix.
- P is a $n \times n$ permutation matrix.

Where $A \times P = Q \times R$.

Access: ↰ MATRICES FACTORIZATION QR

↰ MTH MATRIX FACTORS QR

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
$[[\ matrix\ ]]_A$	→	$[[\ matrix\ ]]_Q$	$[[\ matrix\ ]]_R$	$[[\ matrix\ ]]_P$

See also: LQ, LSQ

QUAD

Type: Command

Description: Solve Quadratic Equation Command: This command is identical to the computer algebra command SOLVE, and is included for backward compatibility with the HP48G.

See volume 1, CAS Commands.

Access: CAT QUAD

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_1$'	'*global*'	→	'$symb_2$'

See also: COLCT, EXPAN, ISOL, SHOW, SOLVE

QUOTE

Type: Function

Description: Quote Argument Function: Returns its argument unevaluated.

When an algebraic expression is evaluated, the arguments to a function in the expression are evaluated before the function. For example, when SIN(X) is evaluated, the name *X* is evaluated first, and the result is placed on the stack as the argument for SIN.

This process creates a problem for functions that require symbolic arguments. For example, the integration function requires as one of its arguments a name specifying the variable of integration. If evaluating an integral expression caused the name to be evaluated, the result of evaluation would be left on the stack for the integral, rather than the name itself. To avoid this problem, the HP 49 automatically (and invisibly) quotes such arguments. When the quoted argument is evaluated, the unquoted argument is returned.

If a user-defined function takes symbolic arguments, quote the arguments using QUOTE.

Access: CAT QUOTE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*obj*

See also: APPLY, | (Where)

RAD

Type: Command

Description: Radians Mode Command: Sets Radians angle mode.

RAD sets flag –17 and clears flag –18, and displays the RAD annunciator.

In Radians angle mode, real-number arguments that represent angles are interpreted as radians, and real-number results that represent angles are expressed in radians.

Access: CAT RAD

Input: None

Output: None

See also: DEG, GRAD

RAND

Type: Command

Description: Random Number Command: Returns a pseudo-random number generated using a seed value, and updates the seed value.

The HP 49 uses a linear congruential method and a seed value to generate a random number x_{random} in the range $0 \leq x < 1$. Each succeeding execution of RAND returns a value computed from a seed value based upon the previous RAND value. (Use RDZ to change the seed.)

Access: ← MTH PROBABILITY RAND

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{random}

See also: COMB, PERM RDZ, !

RANK

Type: Command

Description: Matrix Rank Command: Returns the rank of a rectangular matrix.

Rank is computed by calculating the singular values of the matrix and counting the number of non-negligible values. If all computed singular values are zero, RANK returns zero. Otherwise RANK consults flag –54 as follows:

- If flag –54 is clear (the default), RANK counts all computed singular values that are less than or equal to 1.E–14 times the largest computed singular value.
- If flag –54 is set, RANK counts all nonzero computed singular values.

Access: ↰ MATRICES OPERATIONS RANK

↰ MTH MATRIX NORMALIZE RANK

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	N_{rank}

See also: LQ, LSQ, QR

RANM

Type: Command

Description: Random Matrix Command: Returns a matrix of specified dimensions that contains random integers in the range –9 through 9.

The probability of a nonzero digit occurring is 0.05; the probability of 0 occurring is 0.1.

Access: ↰ MATRICES CREATE RANM

↰ MTH MATRIX MAKE RANM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ *m n* }	→	[[*random matrix*]]$_{\text{m×n}}$
[[*matrix*]]$_{\text{m×n}}$	→	[[*random matrix*]]$_{\text{m×n}}$

See also: RAND, RDZ

RATIO

Type: Function

Description: Prefix Divide Function: Prefix form of / (divide).

RATIO is identical to / (divide), except that, in algebraic syntax, RATIO is a *prefix* function, while / is an *infix* function. For example, RATIO(A,2) is equivalent to A/2.

Access: CAT RATIO

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1/\tilde{z}_2$
[*array*]	{[*matrix*]}	→	[[*array* × *matrix*$^{-1}$]]
[*array*]	$\tilde{z}$	→	[*array*/$\tilde{z}$]
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$/*symb*'
'*symb*'	$\tilde{z}$	→	'*symb*/$\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1$/$symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit_1	y_unit_2	→	$(x/y)\_unit_1/unit_2$
x	y_unit	→	$(x/y)\_1/unit$
x_unit	y	→	$(x/y)\_unit$
'*symb*'	x_unit	→	'*symb*/x_unit'
x_unit	'*symb*'	→	'x_unit/*symb*'

RCEQ

Type: Command

Description: Recall from EQ Command: Returns the unevaluated contents of the reserved variable *EQ* from the current directory.

To recall the contents of *EQ* from a parent directory (when *EQ* doesn't exist in the current directory) evaluate the name *EQ*.

Access: CAT RCEQ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	obj_{EQ}

See also: STEQ

RCI

Type: Command

Description: Multiply Row by Constant Command: Multiplies row *n* of a matrix (or element *n* of a vector) by a constant x_{factor}, and returns the modified matrix.

RCI rounds the row number to the nearest integer, and treats vector arguments as column vectors.

Access: ↰ MATRICES CREATE ROW RCI

↰ MTH MATRIX ROW RCI

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	x_{factor}	$n_{row\ number}$	→	$[[\ matrix\ ]]_3$
$[\ vector\ ]_1$	x_{factor}	$n_{element\ number}$	→	$[\ vector\ ]_2$

See also: RCIJ

RCIJ

Type: Command

Description: Add Multiplied Row Command: Multiplies row *i* of a matrix by a constant x_{factor}, adds this product to row *j* of the matrix, and returns the modified matrix; or multiplies element *i* of a vector by a constant x_{factor}, adds this product to element *j* of the vector, and returns the modified vector.

RCIJ rounds the row numbers to the nearest integer, and treats vector arguments as column vectors.

Access: ↰ MATRICES CREATE ROW RCIJ

↰ MTH MATRIX ROW RCIJ

Input/Output:

Level 4/Argument 1	Level 3/Argument 2	Level 2/Argument 3	Level 1/Argument 4		Level 1/Item 1
$[[\ matrix\ ]]_1$	$\mathcal{X}_{\text{factor}}$	$n_{\text{row i}}$	$n_{\text{row j}}$	$\rightarrow$	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$\mathcal{X}_{\text{factor}}$	$n_{\text{element i}}$	$n_{\text{element j}}$	$\rightarrow$	$[\ vector\ ]_2$

See also: RCI

RCL

Type: Command Operation

Description: Recall Command: Returns the unevaluated contents of a specified variable.

RCL searches the entire current path, starting with the current directory. To search a different path, specify { *path name* }, where *path* is the new path to the variable *name*. The *path* subdirectory does not become the current subdirectory (unlike EVAL).

To recall a library or backup object, tag the library number or backup name with the appropriate port number (n_{port}), which must be an integer in the range 0 to 2. Recalling a backup object brings a copy of its *contents* to the stack, not the entire backup object.

To search for a backup object, replace the port number with the wildcard character &, in which case the HP 49 will search (in order) ports 0 through 2, and the main memory for the named backup object.

You can specify a port (that is, n_{port}) in one of three ways:

- H, 0, 1, or 2
- H, R, E, or F
- HOME, RAM, ERAM, IRAM

In each case, the ports are home, RAM, extended RAM, and Flash, respectively.

Access: ↰ RCL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*obj*'	→	*obj*
PICT	→	*grob*
:n_{port} :$n_{library}$	→	*obj*
:n_{port} :$name_{backup}$	→	*obj*
:n_{port} :{ *path* }	→	*obj*

See also: STO

RCLALARM

Type: Command

Description: Recall Alarm Command: Recalls a specified alarm.

obj_{action} is the alarm execution action. If an execution action was not specified, obj_{action} defaults to an empty string.

$\varkappa_{repeat}$ is the repeat interval in clock ticks, where 1 clock tick equals 1/8192 second. If a repeat interval was not specified, the default is 0.

Access: ⮌ TIME TOOLS ALRM RCLALARM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{index}	→	{ *date time* obj_{action} $\varkappa_{repeat}$ }

See also: DELALARM, FINDALARM, STOALARM

RCLF

Type: Command

Description: Recall Flags Command: Returns a list of integers representing the states of the system and user flags, respectively.

A bit with value 1 indicates that the corresponding flag is set; a bit with value 0 indicates that the corresponding flag is clear. The rightmost (least significant) bit of #n_{system} and #n_{user} indicate the states of system flag –1 and user flag +1, respectively.

Used with STOF, RCLF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status.

Access: CAT RCLF

Level 1/Argument 1		Level 1/Item 1
	→	{ #n_{system} #n_{user} #$n_{system2}$ #n_{user2} }

See also: STOF

RCLKEYS

Type: Command

Description: Recall Key Assignments Command: Returns the current user key assignments. This includes an S if the standard definitions are active (not suppressed) for those keys without user key assignments.

The argument x_{key} is a real number of the form *rc.p* specifying the key by its row number *r*, its column number *c*, and its plane (shift) *p*. (For a definition of plane, see the entry for ASN.)

Access: CAT RCLKEYS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ obj_1, $x_{key\ 1}$, ... ,objn, $x_{key\ n}$ }
	→	{ S, obj_1, $x_{key\ 1}$, ... objn, $x_{key\ n}$ }

See also: ASN, DELKEYS, STOKEYS

RCLMENU

Type: Command

Description: Recall Menu Number Command: Returns the menu number of the currently displayed menu.

x_{menu} has the form *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu.

Executing RCLMENU when the current menu is a user-defined menu (build by TMENU) returns 0.01 (in 2 Fix mode), indicating "Last menu".

Access: CAT RCLMENU

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{menu}

See also: MENU, TMENU

RCLΣ

Type: Command

Description: Recall Sigma Command: Returns the current statistical matrix (the contents of reserved variable ΣDAT) from the current directory.

To recall ΣDAT from the parent directory (when ΣDAT doesn't exist in the current directory), evaluate the name ΣDAT.

Access: CAT RCLΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*obj*

See also: CLΣ, STOΣ, Σ+, Σ–

RCWS

Type: Command

Description: Recall Wordsize Command: Returns the current wordsize in bits (1 through 64).

Access: ↱ BASE RCWS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*n*

See also: BIN, DEC, HEX, OCT, STWS

RDM

Type: Command

Description: Redimension Array Command: Rearranges the elements of the argument according to specified dimensions.

If the list contains a single number $n_{elements}$, the result is an n-element vector. If the list contains two numbers n_{rows} and m_{cols}, the result is an $n \times m$ matrix.

Elements taken from the argument vector or matrix preserve the same row order in the resulting vector or matrix. If the result is dimensioned to contain fewer elements than the argument vector or matrix, excess elements from the argument vector or matrix at the end of the row order are discarded. If the result is dimensioned to contain more elements than the argument vector or matrix, the additional elements in the result at the end of the row order are filled with zeros.

If the argument vector or matrix is specified by *global*, the result replaces the argument as the contents of the variable.

Access: ↰ MATRICES CREATE RDM

↰ MTH MATRIX MAKE RDM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[*vector*]$_1$	{ $n_{elements}$ }	→	[*vector*]$_2$
[*vector*]	{ n_{rows}, m_{cols} }	→	[[*matrix*]]
[[*matrix*]]	{ $n_{elements}$ }	→	[*vector*]
[[*matrix*]]$_1$	{ n_{rows}, m_{cols} }	→	[[*matrix*]]$_2$
'*global*'	{ $n_{elements}$ }	→	
'*global*'	{ n_{rows}, m_{cols} }	→	

See also: TRN

RDZ

Type: Command

Description: Randomize Command: Uses a real number x_{seed} as a seed for the RAND command.

If the argument is 0, a random value based on the system clock is used as the seed.

Access: ↰ MTH PROBABILITY RDZ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{seed}	→	

See also: COMB, PERM, RAND, !

RE

Type: Function

Description: Real Part Function: Returns the real part of the argument.

If the argument is a vector or matrix, RE returns a real array, the elements of which are equal to the real parts of the corresponding elements of the argument array.

Access: ↱ CMPLX RE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*x*
x_unit	→	*x*
(*x*,*y*)	→	*x*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'RE(*symb*')

See also: C→R, IM, R→C

RECN

Type: Command

Description: Receive Renamed Object Command: Prepares the HP 49 to receive a file from another Kermit server device, and to store the file in a specified variable.

RECN is identical to RECV except that the name under which the received data is stored is specified.

Access: CAT RECN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	
"*name*"	→	

See also: BAUD, CKSM, CLOSEIO, FINISH, KERRM, KGET, PARITY, RECV, SEND, SERVER, TRANSIO

RECT

Type: Command

Description: Rectangular Mode Command: Sets Rectangular coordinate mode.

RECT clears flags –15 and –16.

In Rectangular mode, vectors are displayed as rectangular components. Therefore, a 3D vector would appear as [*X Y Z*].

Access: CAT RECT

Input: None

Output: None

See also: CYLIN, SPHERE

RECV

Type: Command

Description: Receive Object Command: Instructs the HP 49 to look for a named file from another Kermit server device. The received file is stored in a variable named by the sender.

Since the HP 49 does not normally look for incoming Kermit files, you must use RECV to tell it to do so.

Access: CAT RECV

Input: None

Output: None

See also: BAUD, CKSM, FINISH, KGET, PARITY, RECN, SEND, SERVER, TRANSIO

RENAME

Type: Command

Description: Rename Object Command: Renames an object to the name that you specify.

Access: CAT RENAME

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
new 'name'	old 'name'	→	

See also: COPY

REPEAT

Type: Command

Description: REPEAT Command: Starts loop clause in WHILE … REPEAT … END indefinite loop structure.

See the WHILE entry for more information.

Access: ← PRG BRANCH REPEAT

Input: None

Output: None

See also: END, WHILE

REPL

Type: Command

Description: Replace Command: Replaces a portion of the target object (first input) with a specified object (third input), beginning at a specified position (second input).

For arrays, $n_{position}$ counts in row order. For matrices, $n_{position}$ specifies the new location of the upper left-hand element of the replacement matrix.

For graphics objects, the upper left corner of $grob_1$ is positioned at the user-unit or pixel coordinates (x,y) or { #*n* #*m* }. From there, it overwrites a rectangular portion of $grob_{target}$ or *PICT*. If $grob_1$ extends past $grob_{target}$ or *PICT* in either direction, it is truncated in that direction. If the specified coordinate is not on the target graphics object, the target graphics object does not change.

Access: ⮌ PRG LIST REPL

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[*matrix*]]$_1$	$n_{position}$	[[*matrix*]]$_2$	→	[[*matrix*]]$_3$
[[*matrix*]]$_1$	{ n_{row}, n_{column} }	[[*matrix*]]$_2$	→	[[*matrix*]]$_3$
[*vector*]$_1$	$n_{position}$	[*vector*]$_2$	→	[*vector*]$_3$
{ *list*$_{target}$ }	$n_{position}$	{ *list*$_1$ }	→	{ *list*$_{result}$ }
"*string*$_{target}$"	$n_{position}$	"*string*$_1$"	→	"*string*$_{result}$"
grob$_{target}$	(#*n*, #*m*)	*grob*$_1$	→	*grob*$_{result}$
grob$_{target}$	(*x*,*y*)	*grob*$_1$	→	*grob*$_{result}$
PICT	(#*n*, #*m*)	*grob*$_1$	→	
PICT	(*x*,*y*)	*grob*$_1$	→	

See also: CHR, GOR, GXOR, NUM, POS, SIZE, SUB

RES

Type: Command

Description: Resolution Command: Specifies the resolution of mathematical and statistical plots, where the resolution is the interval between values of the independent variable used to generate the plot.

A real number $n_{interval}$ specifies the interval in user units. A binary integer $\#n_{interval}$ specifies the interval in pixels.

The resolution is stored as the fourth item in *PPAR*, with default value 0. The interpretation of the default value is summarized in the following table.

Plot Type	Default Interval
BAR	10 pixels (bar width = 10 pixel columns)
DIFFEQ	unlimited: step size is not constrained

Plot Type	Default Interval (Continued)
FUNCTION	1 pixel (plots a point in every column of pixels)
CONIC	1 pixel (plots a point in every column of pixels)
TRUTH	1 pixel (plots a point in every column of pixels)
GRIDMAP	RES does not apply
HISTOGRAM	10 pixels (bin width = 10 pixel columns)
PARAMETRIC	[independent variable range in user units]/ 130
PARSURFACE	RES does not apply
PCONTOUR	RES does not apply
POLAR	2°, 2 grads, or π/90 radians
SCATTER	RES does not apply
SLOPEFIELD	RES does not apply
WIREFRAME	RES does not apply
YSLICE	1 pixel (plots a point in every column of pixels)

Access: CAT RES

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{interval}	→	
$\#n_{\text{interval}}$	→	

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

RESTORE

Type: Command

Description: Restore HOME Command: Replaces the current *HOME* directory with the specified backup copy (:n_{port}:$name_{backup}$) previously created by ARCHIVE.

The specified port number must be in the range 0 to 2.

To restore a *HOME* directory that was saved on a remote system using :IO:*name* ARCHIVE, put the backup object itself on the stack, execute RESTORE and then execute a warm start.

Access: CAT RESTORE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
:n_{port} :$name_{backup}$	→	
backup	→	

See also: ARCHIVE

REVLIST

Type: Command

Description: Reverse List Command: Reverses the order of the elements in a list.

Access: ↰ PRG LIST PROCEDURES REVLIST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_n ... obj_1 }	→	{ obj_1 ... obj_n }

See also: SORT

RKF

Type: Command

Description: Solve for Initial Values (Runge–Kutta–Fehlberg) Command: Computes the solution to an initial value problem for a differential equation, using the Runge-Kutta-Fehlberg (4,5) method.

RKF solves $y'(t) = f(t,y)$, where $y(t_0) = y_0$. The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (*t*) and solution (*y*) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- x_{tol} sets the absolute error tolerance. If a list is used, the first value is the absolute error tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.

RKF repeatedly calls RKFSTEP as it steps from the initial value to x_{Tfinal}.

Access: CAT RKF

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
{ *list* }	x_{tol}	$x_{T\ final}$	→	{ *list* }	x_{tol}
{ *list* }	{ x_{tol} x_{hstep} }	$x_{T\ final}$	→	{ *list* }	x_{tol}

L = level; A = argument; I = item

See also: RKFERR, RKFSTEP, RRK, RRKSTEP, RSBERR

RKFERR

Type: Command

Description: Error Estimate for Runge–Kutta–Fehlberg Method Command: Returns the absolute error estimate for a given step *h* when solving an initial value problem for a differential equation.

The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (*t*) and solution (*y*) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).
- *h* is a real number that specifies the step.
- y_{delta} displays the change in solution for the specified step.
- *error* displays the absolute error for that step. A zero error indicates that the Runge–Kutta–Fehlberg method failed and that Euler's method was used instead.

The absolute error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.)

Access: CAT RKFE

Input/Output:

L_2/A_1	L_1/A_2		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
{ *list* }	*h*	→	{ *list* }	*h*	y_{delta}	*error*

L = level; A = argument; I = item

See also: RKF, RKFSTEP, RRK, RRKSTEP, RSBERR

RKFSTEP

Type: Command

Description: Next Solution Step for RKF Command: Computes the next solution step (h_{next}) to an initial value problem for a differential equation.

The arguments and results are as follows:

- { *list* } contains three items in this order: the independent (*t*) and solution (*y*) variables, and the right-hand side of the differential equation (or a variable where the expression is stored).

- x_{tol} sets the tolerance value.
- h specifies the initial candidate step.
- h_{next} is the next candidate step.

The independent and solution variables must have values stored in them. RKFSTEP steps these variables to the next point upon completion.

Note that the actual step used by RKFSTEP will be less than the input value h if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RKFSTEP to reduce its stepsize to the point that the Runge–Kutta–Fehlberg methods fails, then RKFSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Runge-Kutta-Fehlberg method will fail if the current independent variable is zero and the stepsize $\leq 1.3 \times 10^{-498}$ or if the variable is nonzero and the stepsize is 1.3×10^{-10} times its magnitude.

Access: CAT RKFS

Input/Output:

L_3/A_1	L_2/A_n	L_1/A_{n+1}		L_3/I_1	L_2/I_2	L_1/I_3
{ *list* }	x_{tol}	h	→	{ *list* }	x_{tol}	h_{next}

L = level; A = argument; I = item

See also: RKF, RKFERR, RRK, RRKSTEP, RSBERR

RL

Type: Command

Description: Rotate Left Command: Rotates a binary integer one bit to the left.

The leftmost bit of $\#n_1$ becomes the rightmost bit of $\#n_2$.

Access: ← MTH BASE BIT RL

→ BASE BIT RL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: RLB, RR, RRB

RLB

Type: Command

Description: Rotate Left Byte Command: Rotates a binary integer one byte to the left.

The leftmost byte of #n_1 becomes the rightmost byte of #n_2. RLB is equivalent to executing RL eight times.

Access: ↰ MTH BASE BYTE RLB

↱ BASE BYTE RLB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2

See also: RL, RR, RRB

RND

Type: Function

Description: Round Function: Rounds an object to a specified number of decimal places or significant digits, or to fit the current display format.

n_{round} (or $symb_{\text{round}}$ if flag –3 is set) controls how the level 2 argument is rounded, as follows:

n_{round} or $symb_{\text{round}}$	Effect on Level 2 Argument
0 through 1	Rounded to *n* decimal places.
–1 through –11	Rounded to *n* significant digits.
12	Rounded to the current display format.

For complex numbers and arrays, each real number element is rounded. For unit objects, the numerical part of the object is rounded.

Access: ↰ MTH REAL RND

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	n_{round}	→	$\tilde{z}_2$
$\tilde{z}$	'$symb_{round}$'	→	'RND($symb_{round}$)'
'$symb$'	n_{round}	→	'RND($symb$,n_{round})'
'$symb_1$'	'$symb_{round}$'	→	'RND('$symb_1$, $symb_{round}$)'
[$array_1$]	n_{round}	→	[$array_2$]
x_unit	n_{round}	→	y_unit
x_unit	'$symb_{round}$'	→	'RND(x_unit, $symb_{round}$)'

See also: TRNC

RNRM

Type: Command

Description: Row Norm Command: Returns the row norm (infinity norm) of its argument array.

The row norm is the maximum (over all rows) of the sums of the absolute values of all elements in each row. For a vector, the row norm is the largest absolute value of any of its elements.

Access: ⮌ MATRICES OPERATIONS RNRM

⮌ MTH MATRIX NORMALIZE RNRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[$array$]	→	$x_{row\ norm}$

See also: CNRM, CROSS, DET, DOT

ROLL

Type: RPL command

Description: Roll Objects Command: Moves the contents of a specified level to level 1, and rolls upwards the portion of the stack beneath the specified level.

In RPN mode, 3 ROLL is equivalent to ROT.

Access: ↰ PRG STACK ROLL

Input/Output:

L_{n+1}... L_2	L_1		L_n ...	L_2	L_1
obj_n ... obj_1	n	→	obj_{n-1} ...	obj_1	obj_n

L = level

See also: OVER, PICK, ROLLD, ROT, SWAP

ROLLD

Type: RPL command

Description: Roll Down Command: Moves the contents of level 2 to a specified level, *n*, and rolls downward the portion of the stack beneath the specified level.

Access: ↰ PRG STACK ROLLD

Input/Output:

L_{n+1}... L_2	L_1		L_n	L_{n-1} ...	L_1
obj_n ... obj_2	n (obj_1)	→	obj_1	obj_n ...	obj_2

L = level

See also: OVER, PICK, ROLL, ROT, SWAP

ROMUPLOAD

Type: Command

Description: Upload the calculator's operating system to another calculator.. Use this command to upload your ROM to another calculator.

1. On the sending calculator, enter the ROMUPLOAD command and press ENTER. The calculator displays a screen with instructions on how to download the ROM.
2. On the receiving calculator, hold down the ON and press the F4. The calculator displays the Test menu.
3. On the receiving calculator, hold down ON and + and press ENTER. After a short time, the Terminal menu appears.
4. On the receiving calculator, press 4 to select the Download option.
5. On the sending calculator, press any key to start the download process. The process takes around 20 minutes.

Access: CAT

ROOT

Type: Command

Description: Root-Finder Command: Returns a real number x_{root} that is a value of the specified variable *global* for which the specified program or algebraic object most nearly evaluates to zero or a local extremum.

guess is an initial estimate of the solution. ROOT produces an error if it cannot find a solution, returning the message Bad Guess(es) if one or more of the guesses lie outside the domain of the equation, or returns the message Constant? if the equation returns the same value at every sample point. ROOT does *not* return interpretive messages when a root is found.

Access: CAT ROOT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
«*program*»	'*global*'	*guess*	→	x_{root}
«*program*»	'*global*'	{ *guesses* }	→	x_{root}
'*symb*'	'*global*'	*guess*	→	x_{root}
'*symb*'	'*global*'	{ *guesses* }	→	x_{root}

ROT

Type: RPL ommand

Description: Rotate Objects Command: Rotates the first three objects on the stack, moving the object on level 3 to level 1.

In RPN mode, ROT is equivalent to 3 ROLL.

Access: ◁ PRG STACK ROT

Input/Output:

L_3	L_2	L_1		L_3	L_2	L_1
obj_3	obj_2	obj_1	→	obj_2	obj_1	obj_3

L = level

See also: OVER, PICK, ROLL, ROLLD, SWAP, UNROT

ROW–

Type: Command

Description: Delete Row Command: Deletes row *n* of a matrix (or element *n* of a vector), and returns the modified matrix (or vector) and the deleted row (or element).

n_{row} or $n_{element}$ is rounded to the nearest integer.

Access: ◁ MATRICES CRREATE ROW ROW–

◁ MTH MATRIX ROW ROW–

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
$[[\ matrix\ ]]_1$	n_{row}	→	$[[\ matrix\ ]]_2$	$[\ vector\ ]_{row}$
$[\ vector\ ]_1$	$n_{element}$	→	$[\ vector\ ]_2$	$element_n$

See also: COL–, COL+, ROW+, RSWP

ROW+

Type: Command

Description: Insert Row Command: Inserts an array into a matrix (or one or more numbers into a vector) at the position indicated by n_{index}, and returns the modified matrix (or vector).

The inserted array must have the same number of columns as the target array.

n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and below the insertion point are shifted down.

Access: **Access:** ↰ MATRICES CRREATE ROW ROW+

↰ MTH MATRIX ROW ROW+

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$[[\ matrix\ ]]_2$	n_{index}	→	$[[\ matrix\ ]]_3$
$[[\ matrix\ ]]_1$	$[\ vector\ ]_{row}$	n_{index}	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{element}$	n_{index}	→	$[\ vector\ ]_2$

See also: COL–, COL+, ROW–, RSWP

ROW→

Type: Command

Description: Rows to Matrix Command: Transforms a series of row vectors and a row count into a matrix containing those rows, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

Access: MATR ROW ROW→

Input/Output:

L_{n+1}/A_1 ...	L_2/A_n	L_1/A_{n+1}		Level 1/Item 1
$[\ vector\ ]_{row\ 1}$ ...	$[\ vector\ ]_{row\ n}$	$n_{row\ count}$	→	$[[\ matrix\ ]]$
$element_1$ ...	$element_n$	$n_{element\ count}$	→	$[\ vector\ ]_{column}$

L = level; A = argument; I = item

See also: →COL, COL→, →ROW

→ROW

Type: Command

Description: Matrix to Rows Command: Transforms a matrix into a series of row vectors and returns the vectors and a row count, or transforms a vector into its elements and returns the elements and an element count.

Access: ↰ MATRICES CREATE ROW →ROW

↰ MTH MATRIX ROW →ROW

Input/Output:

L_1/Argument$_1$		L_{n+1}/I_1 … L_2/I_n	L_1/I_{n+1}
[[*matrix*]]	→	[*vector*]$_{\text{row n}}$ … [*vector*]$_{\text{row n}}$	n_{rowcount}
[*vector*]	→	$element_1$ … $element_n$	$n_{\text{elementcount}}$

L =Level; A = Argument; I = Item

See also: →COL, COL→, ROW→

RR

Type: Command

Description: Rotate Right Command: Rotates a binary integer one bit to the right.

The rightmost bit of $\#n_1$ becomes the leftmost bit of $\#n_2$.

Access: ↱ BASE BIT RR

↰ MTH BASE BIT RR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: RL, RLB, RRB

RRB

Type: Command

Description: Rotate Right Byte Command: Rotates a binary integer one byte to the right.

The rightmost byte of #n_1 becomes the leftmost byte of #n_2. RRB is equivalent to doing RR eight times.

Access: ↱ BASE BYTE RRB

↰ MTH BASE BYTE RRB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
#n_1	→	#n_2

See also: RL, RLB, RR

RRK

Type: Command

Description: Solve for Initial Values (Rosenbrock, Runge–Kutta) Command: Computes the solution to an initial value problem for a differential equation with known partial derivatives.

RRK solves y'(t) = f(t,y), where $y(t_0) = y_0$. The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (*t*).
  - The solution variable (*y*).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- x_{tol} sets the tolerance value. If a list is used, the first value is the tolerance and the second value is the initial candidate step size.
- x_{Tfinal} specifies the final value of the independent variable.

RRK repeatedly calls RKFSTEP as its steps from the initial value to x_{Tfinal}.

Access: CAT RRK

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
{ *list* }	x_{tol}	$x_{T\ final}$	→	{ *list* }	x_{tol}
{ *list* }	{ x_{tol} x_{hstep} }	$x_{T\ final}$		{ *list* }	x_{tol}

L = level; A = argument; I = item

See also: RKF, RKFERR, RKFSTEP, RRKSTEP, RSBERR

RRKSTEP

Type: Command

Description: Next Solution Step and Method (RKF or RRK) Command: Computes the next solution step (h_{next}) to an initial value problem for a differential equation, and displays the method used to arrive at that result.

The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (*t*).
  - The solution variable (*y*).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).
- x_{tol} is the tolerance value.
- *h* specifies the initial candidate step.
- *last* specifies the last method used (RKF = 1, RRK = 2). If this is the first time you are using RRKSTEP, enter 0.
- *current* displays the current method used to arrive at the next step.
- h_{next} is the next candidate step.

The independent and solution variables must have values stored in them. RRKSTEP steps these variables to the next point upon completion.

Note that the actual step used by RRKSTEP will be less than the input value *h* if the global error tolerance is not satisfied by that value. If a stringent global error tolerance forces RRKSTEP to reduce its stepsize to the point that the Runge–Kutta–Fehlberg or Rosenbrock methods fails, then RRKSTEP will use the Euler method to compute the next solution step and will consider the error tolerance satisfied. The Rosenbrock method will fail if the current independent variable is zero and the stepsize $\leq 2.5 \times 10^{-499}$ or if the variable is nonzero and the stepsize is 2.5×10^{-11} times its magnitude. The Runge–Kutta–Fehlberg method will fail if the current independent variable is zero and the stepsize $\leq 1.3 \times 10^{-498}$ or if the variable is nonzero and the stepsize is 1.3×10^{-10} times its magnitude.

Access: CAT RRKS

Input/Output:

L_4/A_1	L_3/A_2	L_2/A_3	L_1/A_4		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
{ *list* }	$\varkappa_{tol}$	*h*	*last*	→	{ *list* }	$\varkappa_{tol}$	h_{next}	*current*

L = level; A = argument; I = item

See also: RKF, RKFERR, RKFSTEP, RRK, RSBERR

RSBERR

Type: Command

Description: Error Estimate for Rosenbrock Method Command: Returns an error estimate for a given step *h* when solving an initial values problem for a differential equation.

The arguments and results are as follows:

- { *list* } contains five items in this order:
  - The independent variable (*t*).
  - The solution variable (*y*).
  - The right-hand side of the differential equation (or a variable where the expression is stored).
  - The partial derivative of y'(t) with respect to the solution variable (or a variable where the expression is stored).

– The partial derivative of y'(t) with respect to the independent variable (or a variable where the expression is stored).

- *h* is a real number that specifies the initial step.
- y_{delta} displays the change in solution.
- *error* displays the absolute error for that step. The *absolute* error is the absolute value of the estimated error for a scalar problem, and the row (infinity) norm of the estimated error vector for a vector problem. (The latter is a bound on the maximum error of any component of the solution.) A zero error indicates that the Rosenbrock method failed and Euler's method was used instead.

Access: CAT RSBER

Input/Output:

L_2/A_1	L_1/A_2		L_4/I_1	L_3/I_2	L_2/I_3	L_1/I_4
{ *list* }	*h*	→	{ *list* }	*h*	y_{delta}	*error*

L = level; A = argument; I = item

See also: RKF, RKFERR, RKFSTEP, RRK, RRKSTEP

RSD

Type: Command

Description: Residual Command: Computes the residual B – AZ of the arrays B, A, and Z.

A, B, and Z are restricted as follows:

- A must be a matrix.
- The number of columns of A must equal the number of elements of Z if Z is a vector, or the number of rows of Z if Z is a matrix.
- The number of rows of A must equal the number of elements of B if B is a vector, or the number of rows of B if B is a matrix.
- B and Z must both be vectors or both be matrices.
- B and Z must have the same number of columns if they are matrices.

RSD is typically used for computing a correction to Z, where Z has been obtained as an approximation to the solution X to the system of equations AX = B.

Access: ↰ MATRICES OPERATIONS RSD

↰ MTH MATRIX RSD

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[*vector*]$_B$	[[*matrix*]]$_A$	[*vector*]$_Z$	→	[*vector*]$_{B–AZ}$
[[*matrix*]]$_B$	[[*matrix*]]$_A$	[[*matrix*]]$_Z$	→	[[*matrix*]]$_{B–AZ}$

RSWP

Type: Command

Description: Row Swap Command: Swaps rows *i* and *j* of a matrix and returns the modified matrix, or swaps elements *i* and *j* of a vector and returns the modified vector.

Row numbers are rounded to the nearest integer. Vector arguments are treated as column vectors.

Access: ↰ MATRICES CREATE ROW RSWP

↰ MTH MATRIX ROW RSWP

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[*matrix*]]$_1$	$n_{\text{row i}}$	$n_{\text{row j}}$	→	[[*matrix*]]$_2$
[*vector*]$_1$	$n_{\text{element i}}$	$n_{\text{element j}}$	→	[*vector*]$_2$

See also: CSWP, ROW+, ROW–

R→B

Type: Command

Description: Real to Binary Command: Converts a positive real to its binary integer equivalent.

For any value of $n \leq 0$, the result is # 0. For any value of $n \geq 1.84467440738E19$ (base 10), the result is # FFFFFFFFFFFFFFFF (base 16).

Access: ↱ BASE R→B

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	#*n*

See also: B→R

R→C

Type: Command

Description: Real to Complex Command: Combines two real numbers or real arrays into a single complex number or complex array.

The firts input represents the real element(s) of the complex result. The second input represents the imaginary element(s) of the complex result.

Array arguments must have the same dimensions.

Access: ↰ PRG TYPE R→C

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	(*x*,*y*)
[*R-array*$_1$]	[*R-array*$_2$]	→	[*C-array*]

See also: C→R, IM, RE

R→D

Type: Function

Description: Radians to Degrees Function: Converts a real number expressed in radians to its equivalent in degrees.

This function operates independently of the angle mode.

Access: ↰ MTH REAL R→D

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	(180/π)*x*
'*symb*'	→	'R→D(*symb*)'

FSee also: D→R

S

SAME

Type: Command

Description: Same Object Command: Compares two objects, and returns a true result (1) if they are identical, and a false result (0) if they are not.

SAME is identical in effect to == for all object types except algebraics, names, and some units. (For algebraics and names, == returns an expression that can be evaluated to produce a test result based on numerical values.

Access: ↰ PRG TEST SAME

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	0/1

See also: TYPE, ==

SBRK

Type: Command

Description: Serial Break Command: Interrupts serial transmission or reception.

SBRK is typically used when a problem occurs in a serial data transmission.

Access: CAT SBRK

Input: None

Output: None

See also: BUFLEN, SRECV, STIME, XMIT

SCALE

Type: Command

Description: Scale Plot Command: Adjusts the first two parameters in *PPAR*, (x_{min},y_{min}) and (x_{max},y_{max}), so that x_{scale} and y_{scale} are the new plot horizontal and vertical scales, and the center point doesn't change.

The scale in either direction is the number of user units per tick mark. The default scale in both directions is 1 user-unit per tick mark.

Access: CAT SCALE

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{scale}	y_{scale}	→	

See also: AUTO, CENTR, SCALEH, SCALEW

SCALEH

Type: Command

Description: Multiply Height Command: Multiplies the vertical plot scale by x_{factor}.

Executing SCALEH changes the y-axis display range—the y_{min} and y_{max} components of the first two complex numbers in the reserved variable PPAR. The plot origin (the user-unit coordinate of the center pixel) is not changed.

Access: CAT SCALEH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{factor}	→	

See also: AUTO, SCALEW, YRING

SCALEW

Type: Command

Description: Multiply Width Command: Multiplies the horizontal plot scale by x_{factor}.

Executing SCALEH changes the x-axis display range—the x_{min} and x_{max} components of the first two complex numbers in the reserved variable PPAR. The plot origin (the user-unit coordinate of the center pixel) is not changed.

Access: CAT SCALEW

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{factor}	→	

See also: AUTO, SCALEH, YRING

SCATRPLOT

Type: Command

Description: Draw Scatter Plot Command: Draws a scatterplot of (x, y) data points from the specified columns of the current statistics matrix (reserved variable Σ*DAT*).

The data columns plotted are specified by XCOL and YCOL, and are stored as the first two parameters in the reserved variable Σ*PAR*. If no data columns are specified, columns 1 (independent) and 2 (dependent) are selected by default. The y-axis is autoscaled and the plot type is set to SCATTER.

When SCATRPLOT is executed from a program, the resulting display does not persist unless PICTURE or PVIEW is subsequently executed.

Access: CAT SCATTERPLOT

Input: None

Output: None

See also: BARPLOT, PICTURE, HISTPLOT, PVIEW, SCLΣ, XCOL, YCOL

SCATTER

Type: Command

Description: Scatter Plot Type Command: Sets the plot type to SCATTER.

When the plot type is SCATTER, the DRAW command plots points by obtaining x and y coordinates from two columns of the current statistics matrix (reserved variable ΣDAT). The columns are specified by the first and second parameters in the reserved variable ΣPAR (using the XCOL and YCOL commands). The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

$$\{ (x_{min},y_{min}), (x_{max},y_{max}), \textit{indep, res, axes, ptype, depend} \}$$

For plot type SCATTER, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command SCATTER places the name SCATTER in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT SCATTER

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE

SCHUR

Type: Command

Description: Schur Decomposition of a Square Matrix Command: Returns the Schur decomposition of a square matrix.

SCHUR decomposes A into two matrices Q and T:

- If A is a complex matrix, Q is a unitary matrix, and T is an upper-triangular matrix.
- If A is a real matrix, Q is an orthogonal matrix, and T is an upper quasi-triangular matrix (T is upper block triangular with 1×1 or 2×2 diagonal blocks where the 2×2 blocks have complex conjugate eigenvalues).

In either case, $A \cong Q \times T \times \text{TRN}(Q)$

Access: ↰ MATRICES FACTORIZATION SCHUR

↰ MTH MATRIX FACTORS SCHUR

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
[[*matrix*]]$_A$	→	[[*matrix*]]$_Q$	[[*matrix*]]$_T$

See also: LQ, LU, QR, SVD, SVL, TRN

SCI

Type: Command

Description: Scientific Mode Command: Sets the number display format to scientific mode, which displays one digit to the left of the fraction mark and n significant digits to the right.

Scientific mode is equivalent to scientific notation using $n + 1$ significant digits, where $0 \le n \le 11$. (Values for n outside this range are rounded to the nearest integer.) In scientific mode, numbers are displayed and printed like this:

(*sign*) *mantissa* E (*sign*) *exponent*

where the mantissa has the form $n.(n \ldots)$ and has zero to 11 decimal places, and the exponent has one to three digits.

Access: CAT SCI

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	

See also: ENG, FIX, STD

SCLΣ

Type: Command

Description: Scale Sigma Command: Adjusts (x_{min}, y_{min}) and (x_{max}, y_{max}) in *PPAR* so that a subsequent scatter plot exactly fills *PICT*.

When the plot type is SCATTER, the command AUTO incorporates the functions of SCLΣ. In addition, the command SCATTERPLOT automatically executes AUTO to achieve the same result.

Access: CAT SCLΣ

Input: None

Output: None

See also: AUTO, SCATRPLOT

SCONJ

Type: Command

Description: Store Conjugate Command: Conjugates the contents of a named object.

The named object must be a number, an array, or an algebraic object. For information on conjugation, see CONJ.

Access: ↰ PRG MEMORY ARITHMETIC SCONJ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: CONJ, SINV, SNEG

SCROLL

Type: Command

Description: .

Access: CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	

See also:

SDEV

Type: Command

Description: Standard Deviation Command: Calculates the sample standard deviation of each of the *m* columns of coordinate values in the current statistics matrix (reserved variable Σ*DAT*).

SDEV returns a vector of *m* real numbers, or a single real number if *m* = 1. The standard deviation (the square root of the variances) is computed using this formula:

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

where x_i is the *i*th coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and *n* is the number of data points.

Access: CAT SDEV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sdev}
	→	$[\, x_{sdev\,1}\; x_{sdev\,2} \ldots x_{sdev\,m} \,]$

See also: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, TOT, VAR

SEND

Type: Command

Description: Send Object Command: Sends a copy of the named objects to a Kermit device.

Data is always sent from a local Kermit, but can be sent either to another local Kermit (which must execute RECV or RECN) or to a server Kermit.

To rename an object when sending it, include the old and new names in an embedded list.

Access: CAT SEND

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } *name* ... }	→	

See also: BAUD, CLOSEIO, CKSM, FINISH, KERRM, KGET, PARITY, RECN, RECV, SERVER, TRANSIO

SEQ

Type: Command

Description: Sequential Calculation Command: Returns a list of results generated by repeatedly executing obj_{exec} using *index* over the range x_{start} to x_{end}, in increments of x_{incr}.

obj_{exec} is nominally a program or algebraic object that is a function of *index*, but can actually be an object. *index* must be a global or local name. The remaining objects can be anything that will evaluate to real numbers.

The action of SEQ for arbitrary inputs can be predicted exactly from this equivalent program.

$$x_{start}\ x_{end}\ \text{FOR}\ index\ obj_{exec}\ \text{EVAL}\ x_{incr}\ \text{STEP}\ n \rightarrow \text{LIST}$$

where *n* is the number of new objects left on the stack by the FOR … STEP loop. Notice that *index* becomes a local variable regardless of its original type.

Access: ↰ PRG LIST PROCEDURES SEQ

Input/Output:

L_5/A_1	L_4/A_2	L_3/A_3	L_2/A_4	L_1/A_5		L_i/I_1
obj_{exec}	*index*	x_{start}	x_{end}	x_{incr}	→	{ *list* }

L = level; A = argument; I = item

FSee also: DOSUBS, STREAM

SERVER

Type: Command

Description: Server Mode Command: Selects Kermit Server mode.

A Kermit server (a Kermit device in Server mode) passively processes requests sent to it by the local Kermit. The server receives data in response to SEND, transmits data in response to KGET, terminates Server mode in response to FINISH or LOGOUT, and transmits a directory listing in response to a generic directory request.

Access: (CAT) SERVER

Input: None

Output: None

See also: BAUD, CKSM, FINISH, KERRM, KGET, PARITY, PKT, RECN, RECV, SEND, TRANSIO

SF

Type: Command

Description: Set Flag Command: Sets a specified user or system flag.

User flags are numbered 1 through 128. System flags are numbered –1 through –128. See the HP 49G *Pocket Guide* for a listing of system flags and their flag numbers.

Access: ⮌ PRG TEST SF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flagnumber}}$	→	

See also: CF, FC?, FC?C, FS?, FS?C

SHOW

Type: Command

Description: Show Variable Command: Returns $symb_2$, which is equivalent to $symb_1$ except that all implicit references to a variable *name* are made explicit.

If the level 1 argument is a list, SHOW evaluates all global variables in $symb_1$ *not* contained in the list.

Access: CAT SHOW

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_1$'	'*name*'	→	'$symb_2$'
'$symb_1$'	{ $name_1$ $name_2$ ... }	→	'$symb_2$'

See also: COLCT, EXPAN, ISOL, QUAD

SIDENS

Type: Function

Description: Silicon Intrinsic Density Command: Calculates the intrinsic density of silicon as a function of temperature, x_T.

If x_T is a unit object, it must reduce to a pure temperature, and the density is returned as a unit object with units of $1/cm^3$.

If x_T is a real number, its units are assumed to be K, and the density is returned as a real number with implied units of $1/cm^3$.

x_T must be between 0 and 1685 K.

Access: CAT SIDEN

Input/Output:

Level 1/Argument 1	→	Level 1/Item 1
x_T		$x_{density}$
x_unit		x_1/cm^3
'*symb*'		'SIDENS(*symb*)'

SIGN

Type: Function

Description: Sign Function: Returns the sign of a real number argument, the sign of the numerical part of a unit object argument, or the unit vector in the direction of a complex number argument.

For real number and unit object arguments, the sign is defined as +1 for positive arguments, –1 for negative arguments, In exact mode, the sign for argument 0 is undefined (?). In approximate mode, the sign for argument 0 is 0.

For a complex argument:

$$\text{SIGN}(x+iy) = \frac{x}{\sqrt{x^2+y^2}} + \frac{iy}{\sqrt{x^2+y^2}}$$

Access: ↰ MTH REAL SIGN (returns the sign of a number)

↱ CMPLX SIGN (returns the unit vector of a complex number)

Input/Output:

Level 1/Argument 1	→	Level 1/Item 1
$\tilde{z}_1$		$\tilde{z}_2$
x_unit		x_{sign}
'*symb*'		'SIGN(*symb*)'

See also: ABS, MANT, XPON

SIN

Type: Analytic function

Description: Sine Analytic Function: Returns the sine of the argument.

For real arguments, the current angle mode determines the number's units, unless angular units are specified.

For complex arguments, $\sin(x + iy) = \sin x \cosh y + i \cos x \sinh y$.

If the argument for SIN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing SIN with a unit object, the angle mode must be set to radians (since this is a "neutral" mode).

Access: SIN

Input/Output:

Level 1/Argument 1	→	Level 1/Item 1
$\tilde{z}$		$\sin \tilde{z}$
x_unit_{angular}		$\sin(x\_unit_{\text{angular}})$
'*symb*'		'SIN(*symb*)'

See also: ASIN, COS, TAN

SINH

Type: Analytic function

Description: Hyperbolic Sine Analytic Function: Returns the hyperbolic sine of the argument.

For complex arguments, $\sinh(x + iy) = \sinh x \cos y + i \cosh x \sin y$.

Access: ↱ TRIG HYPERBOLIC SINH

↰ MTH HYPERBOLIC SINH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\tilde{z}$	→	sinh $\tilde{z}$
'*symb*'		'SINH(*symb*)'

See also: ASINH, COSH, TANH

SINV

Type: Command

Description: Store Inverse Command: Replaces the contents of the named variable with its inverse.

The named object must be a number, a matrix, an algebraic object, or a unit object. For information on reciprocals, see INV.

Access: ↰ PRG MEMORY ARITHMETIC SINV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: INV, SCONJ, SNEG

SIZE

Type: Command Operation

Description: Size Command: Returns the number of characters in a string, the number of elements in a list, the dimensions of an array, the number of objects in a unit object or algebraic object, or the dimensions of a graphics object.

The size of a unit is computed as follows: the scalar (+1), the underscore (+1), each unit name (+1), operator or exponent (+1), and each prefix (+2).

Any object type not listed above returns a value of 1.

Access: ↰ PRG CHARS SIZE

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
"*string*"	→		*n*
{ *list* }	→		*n*
[*vector*]	→		{ *n* }
[[*matrix*]]	→		{ *n m*}
'*symb*'	→		*n*
grob	→	$\#n_{\text{width}}$	$\#m_{\text{height}}$
PICT	→	$\#n_{\text{width}}$	$\#m_{\text{height}}$
x_unit	→		*n*

See also: CHR, NUM, POS, REPL, SUB

SL

Type: Command

Description: Shift Left Command: Shift a binary integer one bit to the left.

The most significant bit is shifted out to the left and lost, while the least significant bit is regenerated as a zero. SL is equivalent to binary multiplication by 2, truncated to the current wordsize.

Access: ↰ MTH BASE BIT SL

↱ BASE BIT SL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: ASR, SLB, SR, SRB

SLB

Type: Command

Description: Shift Left Byte Command: Shifts a binary integer one byte to the left.

The most significant byte is shifted out to the left and lost, while the least significant byte is regenerated as zero. SLB is equivalent to binary multiplication by 28 (or executing SL eight times), truncated to the current wordsize.

Access: ↰ MTH BASE BYTE SLB

↱ BASE BYTE SLB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: ASR, SL, SR, SRB

SLOPEFIELD

Type: Command

Description: SLOPEFIELD Plot Type Command: Sets the plot type to SLOPEFIELD.

When plot type is set to SLOPEFIELD, the DRAW command plots a slope representation of a scalar function with two variables. SLOPEFIELD requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

$\{ x_{left}\ x_{right}\ y_{near}\ y_{far}\ z_{low}\ z_{high}\ x_{min}\ x_{max}\ y_{min}\ y_{max}\ x_{eye}\ y_{eye}\ z_{eye}\ x_{step}\ y_{step} \}$

For plot type SLOPEFIELD, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.

- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min},y_{min}) (x_{max},y_{max}) *indep res axes ptype depend* }

For plot type SLOPEFIELD, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is not used.
- (x_{max},y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command SLOPEFIELD places the command name SLOPEFIELD in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT SLOPEFIELD

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, TRUTH, WIREFRAME, YSLICE

SNEG

Type: Command

Description: Store Negate Command: Replaces the contents of a variable with its negative.

The named object must be a number, an array, an algebraic object, a unit object, or a graphics object. For information on negation, see NEG.

Access: ↰ PRG MEMORY ARITHMETIC SNEG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: NEG, SCONJ, SINV

SNRM

Type: Command

Description: Spectral Norm Command: Returns the spectral norm of an array.

The spectral norm of a vector is its Euclidean length, and is equal to the largest singular value of a matrix.

Access: ↰ MATRICES OPERATIONS SNRM

↰ MTH MATRIX NORMALIZE SNRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]	→	$x_{\text{spectralnorm}}$

See also: ABS, CNRM, COND, RNRM, SRAD, TRACE

SOLVER

Type: Command

.Description: Displays a menu of commands used in solving equatons.

Access: CAT SOLVER

Input: None

Output: None

SORT

Type: Command

Description: Ascending Order Sort Command: Sorts the elements in a list in ascending order.

The elements in the list can be real numbers, strings, lists, names, binary integers, or unit objects. However, all elements in the list must all be of the same type. Strings and names are sorted by character code number. Lists of lists are sorted by the first element in each list.

To sort in reverse order, use SORT REVLIST.

Access: ↰ MTH LIST SORT

↰ PRG LIST PROCEDURES SORT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\{ list \}_1$	→	$\{ list \}_2$

See also: REVLIST

SPHERE

Type: Command

Description: Spherical Mode Command: Sets spherical coordinate mode.

SPHERE sets flags –15 and –16.

In spherical mode, vectors are displayed as polar components.

Access: CAT SPHERE

Input: None

Output: None

See also: CYLIN, RECT

SQ

Type: Analytic function

Description: Square Analytic Function: Returns the square of the argument.

The square of a complex argument (x, y) is the complex number $(x^2 - y^2, 2xy)$.

Matrix arguments must be square.

Access: ↰ x^2

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\tilde{z}$	→	$\tilde{z}^2$
x_unit	→	x^2*_unit*^{2}
[[*matrix*]]	→	[[*matrix* × *matrix*]]
'*symb*'	→	'SQ(*symb*)'

See also: √, ^

SR

Type: Command

Description: Shift Right Command: Shifts a binary integer one bit to the right.

The least significant bit is shifted out to the right and lost, while the most significant bit is regenerated as a zero. SR is equivalent to binary division by 2.

Access: ↰ MTH BASE BIT SR

↱ BASE BIT SR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: ASR, SL, SLB, SRB

SRAD

Type: Command

Description: Spectral Radius Command: Returns the spectral radius of a square matrix.

The spectral radius of a matrix is a measure of the size of the matrix, and is equal to the absolute value of the largest eigenvalue of the matrix.

Access: ↰ MATRICES OPERATIONS SRAD

↰ MTH MATRIX NORMALIZE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[\ matrix\ ]]_{n\times n}$	→	$x_{spectralradius}$

See also: COND, SNRM, TRACE

SRB

Type: Command

Description: Shift Right Byte Command: Shifts a binary integer one byte to the right.

The least significant byte is shifted out to the right and lost, while the most significant byte is regenerated as zero. SRB is equivalent to binary division by 2^8 (or executing SR eight times).

Access: ↰ MTH BASE BYTE SRB

↱ BASE BYTE SRB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_1$	→	$\#n_2$

See also: ASR, SL, SLB, SR

SRECV

Type: Command

Description: Serial Receive Command: Reads up to *n* characters from the serial input buffer and returns them as a string, along with a digit indicating whether errors occurred.

SRECV does not use Kermit protocol.

If *n* characters are not received within the time specified by STIME (default is 10 seconds), SRECV "times out", returning a 0 to level 1 and as many characters as were received to level 2.

If the level 2 output from BUFLEN is used as the input for SRECV, SRECV will not have to wait for more characters to be received. Instead, it returns the characters already in the input buffer.

If you want to accumulate bytes in the input buffer before executing SRECV, you must first open the port using OPENIO (if the port isn't already open).

SRECV can detect three types of error when reading the input buffer:

- Framing errors and UART overruns (both causing "Receive Error" in ERRM).
- Input-buffer overflows (causing "Receive Buffer Overflow" in ERRM).
- Parity errors (causing "Parity Error" in ERRM).

SRECV returns 0 if an error is detected when reading the input buffer, or 1 if no error is detected.

Parity errors do not stop data flow into the input buffer. However, if a parity error occurs, SRECV stops reading data after encountering a character with an error.

Framing, overrun, and overflow errors cause all subsequently received characters to be ignored until the error is cleared. SRECV does not detect and clear any of these types of errors until it tries to read the byte where the error occurred. Since these three errors cause the byte where the error occurred and all subsequent bytes to be ignored, the input buffer will be empty after all previously received good bytes have been read. Therefore, SRECV detects and clears these errors when it tries to read a byte from an empty input buffer.

Note that BUFLEN also clears the above-mentioned framing, overrun, and overflow errors. Therefore, SRECV cannot detect an input-buffer overflow after BUFLEN is executed, unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). SRECV also cannot detect framing and UART overrun errors cleared by BUFLEN. To find where the data error occurred, save the number of characters returned by BUFLEN (which gives the number of "good" characters received), because as soon as the error is cleared, new characters can enter the input buffer.

Access: CAT SRECV

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
n	→	'*string*'	0/1

See also: BUFLEN, CLOSEIO, OPENIO, SBRK, STIME, XMIT

SREPL

Type: Command

Description: Find and replace: Finds and replaces a string in a given text object. You supply the following inputs:
Level 3/argument 1: the string to search.
Level 2/argument 2: the string to find.
Level 1/argument 3: the string to replace it with.

Access: CAT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
'*string*'	'*string*'	'*string*'	→	'*string*'

START

Type: Command Operation

Description: START Definite Loop Structure Command: Begins START … NEXT and START … STEP definite loop structures.

Definite loop structures execute a command or sequence of commands a specified number of times.

- START … NEXT executes a portion of a program a specified number of times. The RPL syntax is this:

  x_{start} x_{finish} START *loop-clause* NEXT

  The algebraic syntax is this:

  START(x_{start} x_{finish}) *loop-clause* NEXT

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values for a loop counter. Then the loop clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to x_{finish}. If so, the loop clause is executed again. Notice that the loop clause is always executed at least once.

- START … STEP works just like START … NEXT, except that it can use an increment value other than 1. The RPL syntax is this:

  x_{start} x_{finish} START *loop-clause* $x_{increment}$ STEP

  The algebraic syntax is this:

  START (x_{start} x_{finish}) *loop-clause* STEP($x_{increment}$)

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values for the loop counter. Then the loop clause is executed. STEP takes $x_{increment}$ from the stack and increments the counter by that value. If the argument of STEP is an algebraic or a name, it is automatically evaluated to a number.

The increment value can be positive or negative:

– If positive, the loop is executed again when the counter is less than or equal to x_{finish}.

– If negative, the loop is executed when the counter is greater than or equal to x_{finish}.

Access: ⬅ PRG BRANCH START

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
START x_{start}	x_{finish}	→	
NEXT		→	
STEP	$x_{increment}$	→	
STEP	'$symb_{increment}$'	→	

See also: FOR, NEXT, STEP

STD

Type: Command

Description: Standard Mode Command: Sets the number display format to standard mode.

Executing STD has the same effect as clearing flags –49 and –50.

Standard format produces the following results when displaying or printing a number.

- Numbers that can be represented exactly as integers with 12 or fewer digits are displayed without a fraction mark or exponent. Zero is displayed as 0.
- Numbers that can be represented exactly with 12 or fewer digits, but not as integers, are displayed with a fraction mark but no exponent. Leading zeros to the left of the fraction mark and trailing zeros to the right of the fraction mark are omitted.
- All other numbers are displayed in scientific notation (see SCI) with both a fraction mark (with one number to the left) and an exponent (of one or three digits). There are no leading or trailing zeros.

In algebraic objects, integers less than 10^3 are always displayed in standard mode.

Access: CAT STD

Input: None

Output: None

See also: ENG, FIX, SCI

STEP

Type: Command Operation

Description: STEP Command: Defines the increment (step) value, and ends definite loop structure.

See the FOR and START keyword entries for more information.

Access: ↰ PRG BRANCH STEP

Input: None

Output: None

See also: FOR, NEXT, START

STEQ

Type: Command

Description: Store in EQ Command: Stores an object into the reserved variable *EQ* in the current directory.

Access: CAT STEQ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	

See also: RCEQ

STIME

Type: Command

Description: Serial Time-Out Command: Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

The value for *x* is interpreted as a positive value from 0 to 25.4 seconds. If no value is given, the default is 10 seconds. If *x* is 0, there is no time-out; that is, the device waits indefinitely, which can drain the batteries.

STIME is not used for Kermit time-out.

Access: CAT STIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\mathcal{X}_{\text{seconds}}$	→	
0	→	

See also: BUFLEN, CLOSEIO, SBRK, SRECV, XMIT

STO

Type: Command

Description: Store Command: Stores an object into a specified variable or object.

Storing a graphics object into *PICT* makes it the current graphics object.

To create a backup object, store the *obj* into the desired backup location (identified as :n_{port}:$name_{\text{backup}}$). STO will not overwrite an existing backup object.

To store backup objects and library objects, specify a port number (0 through 2).

After storing a library object in a port, it must then be attached to its directory before it can be used. The easiest way to do this is to execute a warm start (by pressing ON F3). This also causes the calculator to perform a *system halt*, which clears the stack, the LAST stack, and all local variables.

STO can also replace a single element of an array or list stored in a variable. Specify the variable in level 1 as *name*(*index*), which is a user function with *index* as the argument. The *index* can be *n* or *n*,*m*, where *n* specifies the row position in a vector or list, and *n*,*m* specifies the row-and-column position in a matrix.

Access: STO▸

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	
grob	*PICT*	→	
obj	:n_{port} :$name_{\text{backup}}$	→	
obj	'*name*(*index*)'	→	
backup	n_{port}	→	
library	n_{port}	→	
library	:n_{port} :n_{library}	→	

See also: DEFINE, RCL, →,

STO–

Type: Command

Description: Store Minus Command: Calculates the difference between a number (or other object) and the contents of a specified variable, and stores the new value in the specified variable.

The object on the stack and the object in the variable must be suitable for subtraction with each other. STO– can subtract any combination of objects suitable for stack subtraction.

Using STO– to subtract two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to subtract them.

Access: ↰ PRG MEMORY ARITHMETIC STO–

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	
'*name*'	*obj*	→	

See also: STO+, STO*, STO/, –

STO*

Type: Command

Description: Store Times Command: Multiplies the contents of a specified variable by a number or other object.

The object on the stack and the object in the variable must be suitable for multiplication with each other. When multiplying two arrays, the result depends on the order of the arguments. The new object of the named variable is the level 2 array times the level 1 array. The arrays must be conformable for multiplication.

Using STO* to multiply two arrays or to multiply a number and an array (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to multiply them.

Access: ↰ PRG MEMORY ARITHMETIC STO*

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	
'*name*'	*obj*	→	

See also: STO+, STO–, STO/, *

STO/

Type: Command

Description: Store Divide Command: Calculates the quotient of a number (or other object) and the contents of a specified variable, and stores the new value in the specified variable.

The new object of the specified variable is the level 2 object divided by the level 1 object.

The object on the stack and the object in the variable must be suitable for division with each other. If both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.

Using STO/ to divide one array by another array or to divide an array by a number (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to divide them.

Access: ↰ PRG MEMORY ARITHMETIC STO/

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	
'*name*'	*obj*	→	

See also: STO+, STO–, STO*, /

STO+

Type: Command

Description: Store Plus Command: Adds a number or other object to the contents of a specified variable.

The object on the stack and the object in the variable must be suitable for addition to each other. STO+ can add any combination of objects suitable for addition.

Using STO+ to add two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to add them.

Access: ↰ PRG MEMORY ARITHMETIC STO+

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	
'*name*'	*obj*	→	

FSee also: STO–, STO*, STO/, +

STOALARM

Type: Command

Description: Store Alarm Command: Stores an alarm in the system alarm list and returns its alarm index number.

If the argument is a real number x_{time}, the alarm date will be the current system date by default.

If obj_{action} is a string, the alarm is an appointment alarm, and the string is the alarm message. If obj_{action} is any other object type, the alarm is a control alarm, and the object is executed when the alarm comes due.

x_{repeat} is the repeat interval for the alarm in clock ticks, where 8192 ticks equals 1 second.

n_{index} is a real integer identifying the alarm based on its chronological position in the system alarm list.

Access: ⮌ TIME TOOLS ALRM STOALARM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{time}	→	n_{index}
{ *date time* }	→	n_{index}
{ *date time* obj_{action} }	→	n_{index}
{ *date time* obj_{action} x_{repeat} }	→	n_{index}

See also: DELALARM, FINDALARM, RCLALARM

STOF

Type: Command

Description: Store Flags Command: Sets the states of the system flags or the system and user flags.

With argument $\#n_{system}$, STOF sets the states of the system flags (–1 through –64) only. With argument { $\#n_{system}$, $\#n_{user}$, $\#n_{system2}$ $\#n_{user2}$ }, STOF sets the states of both the system and user flags.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bit of $\#n_{system}$ and $\#n_{user}$ correspond to the states of system flag –1 and user flag +1, respectively.

STOF can preserve the states of flags before a program executes and changes the states. RCLF can then recall the flag's states after the program is executed.

Access: CAT STOF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_{system}$	→	
{ $\#n_{system}$ $\#n_{user}$ $\#n_{system2}$ $\#n_{user2}$ }	→	

See also: RCLF, STWS, RCWS

STOKEYS

Type: Command

Description: Store Key Assignments Command: Defines multiple keys on the user keyboard by assigning objects to specified keys.

x_{key} is a real number of the form *rc.p* specifying the key by its row number *r*, its column number *c*, and its plane (shift) *p*. (For a definition of plane, see the entry for ASN).

The optional initial list parameter or argument S restores all keys without user assignments to their *standard* key assignments on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command S DELKEYS (see DELKEYS).

If the argument *obj* is the name SKEY, the specified key is restored to its *standard key* assignment.

Access: (CAT) STOKEYS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_1, $x_{key\ 1}$, ... obj_n, $x_{key\ n}$ }	→	
{ S, obj_1, $x_{key\ 1}$, ... obj_n, $x_{key\ n}$ }	→	
'S'	→	

See also: ASN, DELKEYS, RCLKEYS

STOΣ

Type: Command

Description: Store Sigma Command: Stores *obj* in the reserved variable Σ*DAT*.

STOΣ accepts any object anFd stores it in Σ*DAT*. However, if the object is not a matrix or the name of a variable containing a matrix,, an Invalid ΣDATA error occurs upon subsequent execution of a statistics command.

Access: CAT STOΣ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	

See also: CLΣ, RCLΣ, Σ+, Σ–

STR→

Type: Command

Description: Evaluate String Command: Evaluates the text of a string as if the text were entered from the command line.

OBJ→ also includes this function.

Access: CAT STRING→

Input/Output:

Level 1/Argument 1		Level 1/Item 1
"*obj*"	→	*evaluated-object*

See also: ARRY→, DTAG, EQ→, LIST→, OBJ→, →STR

→STR

Type: Command

Description: Object to String Command: Converts any object to string form.

The full-precision internal form of a number is not necessarily represented in the result string. To ensure that →STR preserves the full precision of a number, select Standard number display format or a wordsize of 64 bits (or both) before executing →STR.

The result string includes the entire object, even if the displayed form of the object is too large to fit in the display.

If the argument object is normally displayed in two or more lines, the result string will contain newline characters (character 10) at the end of each line. The newlines are displayed as the character ■.

If the argument object is already a string, →STR returns the string.

Access: CAT →STRING

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	"*obj*"

See also: →ARRY, →LIST, STR→, →TAG, →UNIT

STREAM

Type: Command

Description: Stream Execution Command: Moves the first two elements from the list onto the stack, and executes *obj*. Then moves the next element (if any) onto the stack, and executes *obj* again using the previous result and the new element. Repeats this until the list is exhausted, and returns the final result.

STREAM is nominally designed for *obj* to be a program or command that requires two arguments and returns one result.

Access: ◁ PRG LIST PROCEDURES STREAM

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ *list* }	*obj*	→	*result*

See also: DOSUBS

STWS

Type: Command

Description: Set Wordsize Command: Sets the current binary integer wordsize to *n* bits, where *n* is a value from 1 through 64 (the default is 64).

Values of *n* less than 1 or greater than 64 are interpreted as 1 or 64, respectively.

If the wordsize is smaller than an integer entered on the command line, then the *most* significant bits are not displayed upon entry. The truncated bits are still present internally (unless they exceed 64), but they are not used for calculations and they are lost when a command uses this binary integer as an argument.

Results that exceed the given wordsize are also truncated to the wordsize.

Access: ◁ MTH BASE STWS

▷ BASE STWS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	
$\#n$	→	

See also: BIN, DEC, HEX, OCT, RCWS

SUB

Type: Command Operation

Description: Subset Command: Returns the portion of a string or list defined by specified positions, or returns the rectangular portion of a graphics object or *PICT* defined by two corner pixel coordinates.

If $n_{\text{end position}}$ is less than $n_{\text{start position}}$, SUB returns an empty string or list. Values of n less than 1 are treated as 1; values of n exceeding the length of the string or list are treated as that length.

For graphics objects, a user-unit coordinate less than the minimum user-unit coordinate of the graphics object is treated as that minimum. A pixel or user-unit coordinate greater than the maximum pixel or user-unit coordinate of the graphics object is treated as that maximum.

Access: ↰ PRG LIST SUB

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	$[[\ matrix\ ]]_2$
$[[\ matrix\ ]]_1$	$\{n_{\text{row}}, n_{\text{column}}\}$	$n_{\text{endposition}}$	→	$[[\ matrix\ ]]_2$
$[[\ matrix\ ]]_1$	$n_{\text{startposition}}$	$\{n_{\text{row}}, n_{\text{column}}\}$	→	$[[\ matrix\ ]]_2$
$[[\ matrix\ ]]_1$	$\{n_{\text{row}}, n_{\text{column}}\}$	$\{n_{\text{row}}, n_{\text{column}}\}$	→	$[[\ matrix\ ]]_2$
$\text{“}string_{\text{target}}\text{”}$	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	$\text{“}string_{\text{result}}\text{”}$
$\{\ list_{\text{target}}\ \}$	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	$\{\ list_{\text{result}}\ \}$
$grob_{\text{target}}$	$\{\#n_1, \#m_1\}$	$\{\#n_2\ \#m_2\}$	→	$grob_{\text{result}}$
$grob_{\text{target}}$	(x_1, y_1)	(x_2, y_2)	→	$grob_{\text{result}}$
$PICT$	$\{\#n_1, \#m_1\}$	$\{\#n_2\ \#m_2\}$	→	$grob_{\text{result}}$
$PICT$	(x_1, y_1)	(x_2, y_2)	→	$grob_{\text{result}}$

See also: CHR, GOR, GXOR, NUM, POS, REPL, SIZE

SVD

Type: Command

Description: Singular Value Decomposition Command: Returns the singular value decomposition of an $m \times n$ matrix.

SVD decomposes A into 2 matrices and a vector. U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and S is a real vector, such that $A = U \times diag(S) \times V$. S has length MIN(m,n) and contains the singular values of A in nonincreasing order. The matrix $diag(S)$ is an $m \times n$ diagonal matrix containing the singular values S.

The computed results should minimize (within computational precision):

$$\frac{|A - U \cdot \mathrm{diag}(S) \cdot V|}{\min(m, n) \cdot |A|}$$

where diag(S) denotes the $m \times n$ diagonal matrix containing the singular values S.

Access: ↰ MATRICES FACTORIZATION SVD

↰ MTH MATRIX FACTORS SVD

Input/Output:

Level 1/Argument 1		Level 3/Item 1	Level 2/Item 2	Level 1/Item 3
[[*matrix*]]$_A$	→	[[*matrix*]]$_U$	[[*matrix*]]$_V$	[*vector*]$_S$

See also: DIAG→, MIN, SVL

SVL

Type: Command

Description: Singular Values Command: Returns the singular values of an $m \times n$ matrix.

SLV returns a real vector that contains the singular values of an $m \times n$ matrix in non-increasing order. The vector has length MIN(m,n).

Access: ↰ MATRICES FACTORIZATION SVL

↰ MTH MATRIX FACTORS SVL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[*vector*]

See also: MIN, SVD

SWAP

Type: RPL Command

Description: Swap Objects Command: Swaps the position of the two inputs.

Access: ↰ PRG STACK SWAP

Input/Output:

Level 2	Level 1		Level 2	Level 1
obj_1	obj_2	→	obj_2	obj_1

See also: DUP, DUPN, DUP2, OVER, PICK, ROLL, ROLLD, ROT

SYSEVAL

Type: Command

Description: Evaluate System Object Command: Evaluates unnamed operating system objects specified by their memory addresses.

Using SYSEVAL with random addresses can corrupt memory.

Access: CAT SYSEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_{\text{address}}$	→	

See also: EVAL, LIBEVAL, FLASHEVAL

HP 49G Advanced Users Guide

Volume 1

Part E: Other Commands: T to Z (and Symbols)

Introduction

This volume details the HP 49G commands and functions that are not computer algebra-specific. See part A, Computer algebra commands and functions, for information on computer algebra commands.

For each operation, the following details are provided:

Type: Function or command. Functions can be used as a part of an algebraic objects and commands cannot.

Description: A description of the operation.

Access: The menu or choose-list on which an operation can be found, and the keys that you press to access it. If the operation is on a sub-menu, the sub-menu name is in SMALL CAPITALS after the keys.

Input/Output: The input argument or arguments that the operation needs, and the outputs it produces.

See also: Related functions or commands

T to V

%T

Type: Function

Description: Percent of Total Function: Returns the percent of the first argument that is represented by the second argument.

If both arguments are unit objects, the units must be consistent with each other.

The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰ MTH REAL %T

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$100y/x$
x	'*symb*'	→	'%T(x,*symb*)'
'*symb*'	x	→	'%T(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%T($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	$100y\_unit_2/x\_unit_1$
x_unit	'*symb*'	→	'%T(x_unit,*symb*)'
'*symb*'	x_unit	→	'%T(*symb*,x_unit)'

See also: %, %CH

→TAG

Type: Command

Description: Stack to Tag Command: Combines objects in levels 1 and 2 to create tagged (labeled) object.

The "*tag*" argument is a string of fewer than 256 characters.

Access: CAT →TAG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	"*tag*"	→	:*tag*:*obj*
obj	'*name*'	→	:*name*:*obj*
obj	*x*	→	:*x*:*obj*

See also: →ARRY, DTAG, →LIST, OBJ→, →STR, →UNIT

TAIL

Type: Command

Description: Last Listed Elements Command: Returns all but the first element of a list or string.

Access: ↰ PRG CHARS TAIL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_1 ... obj_n }	→	{ obj_2 ... obj_n }
"$string_1$"	→	"$string_2$"

See also: HEAD

TAN

Type: Analytic function

Description: Tangent Analytic Function: Returns the tangent of the argument.

For real arguments, the current angle mode determines the number's interpretation as an angle, unless the angular units are specified.

For a real argument that is an odd-integer multiple of 90 in Degrees mode, an Infinite Result exception occurs. If flag –22 is set (no error), the sign of the result (MAXR) matches that of the argument.

For complex arguments:

$$\tan(x+iy) = \frac{(\sin x)(\cos x) + i(\sinh y)(\cosh y)}{\sinh^2 y + \cos^2 x}$$

If the argument for TAN is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing TAN with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Access: TAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	tan z
'*symb*'	→	'TAN(*symb*)'
x_unit_{angular}		tan (x_unit_{angular})

See also: ATAN, COS, SIN

TANH

Type: Analytic function

Description: Hyperbolic Tangent Analytic Function: Returns the hyperbolic tangent of the argument.

For complex arguments,

$$\tanh(x+iy) = \frac{\sinh 2x + i\sin 2y}{\cosh 2x + \cos 2y}$$

Access: → TRIG HYPERBOLIC TANH

← MTH HYPERBOLIC TANH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	tanh z
'*symb*'	→	'TANH(*symb*)'

See also: ATANH, COSH, SINH

TAYLR

Type: Command

Description: Taylor Polynomial Command: Calculates the *n*th order Taylor polynomial of *symb* in the variable *global*.

The polynomial is calculated at the point *global* = 0. The expression *symb* may have a removable singularity at 0. The order, *n*, is the *relative* order of the Taylor polynomial–the difference in order between the largest and smallest power of *global* in the polynomial.

TAYLR always returns a symbolic result, regardless of the state of the Numeric Results flag (–3).

Access: ↰ CALC LIMITS & SERIES TAYLR

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
'*symb*'	'*global*'	n_{order}	→	'$symb_{Taylor}$'

See also: ∂, ∫, Σ

TDELTA

Type: Function

Description: Temperature Delta Function: Calculates a temperature change.

TDELTA subtracts two points on a temperature scale, yielding a temperature *increment* (not an actual temperature). *x* or *x_unit1* is the final temperature, and *y* or *y_unit2* is the initial temperature. If unit objects are given, the increment is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the increment is returned as a real number.

Access: CAT TDELTA

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	x_{delta}
x_unit1	*x_unit2*	→	x_unit1_{delta}
x_unit	'*symb*'	→	'TDELTA(*x_unit, symb*)'
'*symb*'	*y_unit*	→	'TDELTA(*symb, y_unit*)'
'$symb_1$'	'$symb_2$'	→	'TDELTA($symb_1$, $symb_2$)'

See also: TINC

TEVAL

Type: Function

Description: For the specified operation, performs the same function as EVAL, and returns the the time taken to perform the evaluation as well as the result.

Access: Catalog, CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1	
Object	→	*result*	*time taken*

Input: The object to evaluate.

Output: Level 2/Item 1: The result of the evaluation.
Level 1/Item 2: The time taken in seconds to perform the evaluation.

TEXT

Type: Command

Description: Show Stack Display Command: Displays the stack display.

TEXT switches from the graphics display to the stack display. TEXT does not update the stack display.

Access: ← PRG OUT TEXT

Input: None

Output: None

See also: PICTURE, PVIEW

THEN

Type: Command

Description: THEN Command: Starts the true-clause in conditional or error-trapping structure.
See the IF and IFFER entries for more information.

Access: ↰ PRG BRANCH THEN

Input: None

Output: None

See also: CASE, ELSE, END, IF IFERR

TICKS

Type: Command

Description: Ticks Command: Returns the system time as a binary integer, in units of 1/8192 second.

Access: ↱ TIME TOOLS TICKS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	#*ntime*

Example: If the result from a previous invocation from TICKS is on level 1 of the stack, then the command:

```
TICKS SWAP - B→R8192 /
```

returns a real number whose value is the elapsed time in seconds between the two invocations.

See also: TIME

TIME

Type: Command

Description: Time Command: Returns the system time in the form HH.MMSSs.

time has the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* is always returned in 24-hour format, regardless of the state of the Clock Format flag (–41).

Access: ↱ TIME TOOLS TIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*time*

See also: DATE, TICKS, TSTR

→TIME

Type: Command

Description: Set System Time Command: Sets the system time.

time must have the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* must use 24-hour format.

Access: ↱ TIME TOOLS →TIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
time	→	

See also: CLKADJ, →DATE

TINC

Type: Function

Description: Temperature Increment Command: Calculates a temperature increment.

TINC adds a temperature *increment* (not an actual temperature) to a point on a temperature scale. Use a negative increment to subtract the increment from the temperature. $x_{initial}$ or *x_unit1* is the initial temperature, and y_{delta} or y_unit2_{delta} is the temperature increment. The returned temperature is the resulting final temperature. If unit objects are given, the final temperature is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the final temperature is returned as a real number.

Access: CAT TINC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_{initial}$	y_{delta}	→	x_{final}
x_unit1	y_unit2_{delta}	→	x_unit1_{final}
x_unit	'*symb*'	→	'TINC(*x_unit*, *symb*)'
'*symb*'	y_unit_{delta}	→	'TINC(*symb*, y_unit_{delta})'
'$symb_1$'	'$symb_2$'	→	'TINC($symb_1$, $symb_2$)'

See also: TDELTA

TLINE

Type: Command

Description: Toggle Line Command: For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off every pixel that is on, and turns on every pixel that is off.

Access: ↰ PRG PICT TLINE

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
(x_1,y_1)	(x_2,y_2)	→	
{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	

See also: ARC, BOX, LINE

TMENU

Type: Command

Description: Temporary Menu Command: Displays a built-in menu, library menu, or user-defined menu.

TMENU works just like MENU, except for user-defined menus (specified by a list or by the name of a variable that contains a list). Such menus are displayed like a custom menu and work like a custom menu, but are not stored in reserved variable *CST*. Thus, a menu defined and displayed by TMENU cannot be redisplayed by evaluating *CST*.

See the MENU entry for a list of the HP 49 built-in menus and the corresponding menu numbers (x_{menu}).

Access: CAT TMENU

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{menu}	→	
{ $list_{definition}$ }	→	
'$name_{definition}$'	→	

See also: MENU, RCLMENU

TOT

Type: Command

Description: Total Command: Computes the sum of each of the *m* columns of coordinate values in the current statistics matrix (reserved variable Σ*DAT*).

The sums are returned as a vector of *m* real numbers, or as a single real number if *m* = 1.

Access: CAT TOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sum}
	→	$[\, x_{sum\ 1}, x_{sum\ 2}, \cdots, x_{sum\ m} \,]$

See also: MAXΣ, MINΣ, MEAN, PSDEV, PVAR, SDEV, VAR

TRACE

Type: Command

Description: Matrix Trace Command: Returns the trace of a square matrix.

The trace of a square matrix is the sum of its diagonal elements.

Access: ↰ MATRICES OPERATIONS TRACE

↰ MTH MATRIX NORMALIZE TRACE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[\ matrix\ ]]_{n\times n}$	→	x_{trace}

TRAN

Type: Command

Description: Transpose Matrix Command: Returns the transpose of a matrix.

Same as TRN, but without conjugation of complex numbers.

Access: ↰ MATRICES OPERATIONS TRAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[[*matrix*]]$_{\text{transpose}}$
'*name*'	→	

See also: CONJ, TRN

TRANSIO

Type: Command

Description: I/O Translation Command: Specifies the character translation option. These translations affect only ASCII Kermit transfers and files printed to the serial port.

Legal values for *n* are as follows:

n	Effect
0	No translation
1	Translate character 10 (line feed only) to / from characters 10 and 13 (line feed with carriage return, the Kermit protocol) (the default value)
2	Translate characters 128 through 159 (80 through 9F hexadecimal)
3	Translate all characters (128 through 255)

Access: CAT TRANSIO

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{option}	→	

See also: BAUD, CKSM, PARITY

TRIGO

Type: Command

.Description: Displays a menu of trigonometry commands.

Access: CAT TRIGO

Input: None

Output: None

TRN

Type: Command

Description: Transpose Matrix Command: Returns the (conjugate) transpose of a matrix.

TRN replaces an $n \times m$ matrix A with an $m \times n$ matrix A^T, where:

$\mathbf{A}_{ij}{}^T = \mathbf{A}_{ji}$ for real matrices

$\mathbf{A}_{ij}{}^T = \text{CONJ}(\mathbf{A}_{ji})$ for complex matrices

If the matrix is specified by *name*, $\mathbf{A}^T$ replaces **A** in *name*.

Access: ↰ MTH MATRIX MAKE TRN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[[*matrix*]]$_{transpose}$
'*name*'	→	

See also: CONJ

TRNC

Type: Function

Description: Truncate Function: Truncates an object to a specified number of decimal places or significant digits, or to fit the current display format.

$n_{truncate}$ (or $symb_{truncate}$ if flag –3 is set) controls how the level 2 argument is truncated, as follows:

$n_{truncate}$	Effect on Level 2 Argument
0 through 11	truncated to *n* decimal places
–1 through –11	truncated to *n* significant digits
12	truncated to the current display format

For complex numbers and arrays, each real number element is truncated. For unit objects, the number part of the object is truncated.

Access: ↰ MTH REAL TRNC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	$n_{truncate}$	→	z_2
z_1	'$symb_{truncate}$'	→	'TRNC(z_1,$symb_{truncate}$)'
'$symb_1$'	$n_{truncate}$	→	'TRNC($symb_1$,$n_{truncate}$)'
'$symb_1$'	'$symb_{truncate}$'	→	'TRNC($symb_1$,$symb_{truncate}$)'
[*array*]$_1$	$n_{truncate}$	→	[*array*]$_2$
x_unit	$n_{truncate}$	→	*y_unit*
x_unit	'$symb_{truncate}$'	→	'TRNC(*x_unit*,$symb_{truncate}$)'

See also: RND

TRUTH

Type: Command

Description: Truth Plot Type Command: Sets the plot type to TRUTH.

When the plot type is TRUTH, the DRAW command plots the current equation as a truth-valued function of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min},y_{min}) (x_{max},y_{max}) *indep res axes ptype depend* }

For plot type TRUTH, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable on the horizontal axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the horizontal plotting range). The default value is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable on the *horizontal* axis, or a binary integer specifying that interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command TRUTH places the name TRUTH in *ptype*.
- *depend* is a name specifying the independent variable on the vertical axis, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable on the vertical axis (the vertical plotting range). The default value is *Y*.

The contents of *EQ* must be an expression or program, and cannot be an equation. It is evaluated for each pixel in the plot region. The minimum and maximum values of the independent variables (the plotting ranges) can be specified in *indep* and *depend*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max})(the display range) are used. The result of each evaluation

must be a real number. If the result is zero, the state of the pixel is unchanged. If the result is nonzero, the pixel is turned on (made dark).

Access: CAT TRUTH

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, WIREFRAME, YSLICE

TSTR

Type: Command

Description: Date and Time String Command: Returns a string derived from the date and time.

The string has the form "*DOW DATE TIME*", where *DOW* is a three-letter abbreviation of the day of the week corresponding to the argument *date* and *time*, *DATE* is the argument *date* in the current date format, and *TIME* is the argument *time* in the current time format.

Access: ↱ TIME TOOLS TSTR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
date	*time*	→	"*DOW DATE TIME*"

See also: DATE, TICKS, TIME

TVARS

Type: Command

Description: Typed Variables Command: Lists all global variables in the current directory that contain objects of the specified types.

If the current directory contains no variables of the specified types, TVARS returns an empty list.

For a table of the object-type numbers, see the entry for TYPE.

Access: ↰ PRG MEMORY DIRECTORY TVARS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{type}	→	{ *global* ... }
{ n_{type} ...}	→	{ *global* ... }

See also: PVARS, TYPE, VARS

TVM

Type: Command

Description: TVM Menu Command: Displays the TVM Solver menu.

Access: (CAT) TVM

Input: None

Output: None

See also: AMORT, TVMBEG, TVMEND, TVMROOT

TVMBEG

Type: Command

Description: Payment at Start of Period Command: Specifies that TVM calculations treat payments as being made at the begining of the compounding periods.

Access: (CAT) TVMBEG

Input: None

Output: None

See also: AMORT, TVM, TVMEND, TVMROOT

TVMEND

Type: Command

Description: Payment at End of Period Command: Specifies that TVM calculations treat payments as being made at the end of the compounding periods.

Access: (CAT) TVMEND

Input: None

Output: None

See also: AMORT, TVM, TVMBEG, TVMROOT

TVMROOT

Type: Command

Description: TVM Root Command: Solves for the specified TVM variable using values from the remaining TVM variables.

Access: (CAT) TVMROOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'TVM variable'	→	$x_{\text{TVM variable}}$

See also: AMORT, TVM, TVMBEG, TVMEND

TYPE

Type: Command

Description: Type Command: Returns the type number of an object.

The following table lists object types and their type numbers.

Object Type Numbers

Object Type	Number
User objects:	
Real number	0
Complex number	1
Character string	2
Real array	3
Complex array	4
List	5
Global name	6
Local name	7
Program	8
Algebraic object	9
Binary integer	10
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory	15

Object Type (Continued)	Number
Library	16
Backup object	17
Real integer	28
Font	30
Built-in Commands:	
Built-in function	18
Built-in command	19
System Objects:	
System binary	20
Extended real	21
Extended complex	22
Linked array	23
Character	24
Code object	25
Library data	26
Mini font	27
Integer number	28
Symbolic vector/matrix	29
Font	30
Extended object	31

T**Access:** ↰ PRG TEST TYPE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	n_{type}

See also: SAME, TVARS, VTYPE, ==

UBASE

Type: Function

Description: Convert to SI Base Units Function: Converts a unit object to SI base units.

Access: ↰ CONVERT UNITS TOOLS UBASE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	*y_base-units*
'*symb*'	→	'UBASE(*symb*)'

See also: CONVERT, UFACT, →UNIT, UVAL

UFACT

Type: Command

Description: Factor Unit Command: Factors the level 1 unit from the unit expression of the level 2 unit object.

Access: ↰ CONVERT UNITS TOOLS UFACT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_1_$unit_1$	x_2_$unit_2$	→	x_3_$unit_2$*$unit_3$

See also: CONVERT, UBASE, →UNIT, UVAL

UFL1→MINIF

Type: Command

.Description: Converts a UFL1 (universal font library) fontset to a HP 49G minifont.

You specify the fontset and give it an ID (0–255). The font must be a 6-by-4 font.

Access: CAT UFL1→MINIF

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_{fontset}	n_{ID}	→	The font converted to minifont.

→UNIT

Type: Command

Description: Stack to Unit Object Command: Creates a unit object from a real number and the unit part of a unit object.

→UNIT adds units to a real number, combining the number and the unit part of a unit object (the numerical part of the unit object is ignored). →UNIT is the reverse of OBJ→ applied to a unit object.

Access: CAT →UNIT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y_unit*	→	*x_unit*

See also: →ARRY, →LIST, →STR, →TAG

UNPICK

Type: RPN Command

Description: Replaces the object at level $n+2$ with the object at level 2 and deletes the objects at levels 1 and 2.

Access: ↰ PRG STACK UNPICK

Input/Output:

L_{n+2}	L_{n+1}	L_3	L_2	L_1		L_n	L_{n-1}	L_1
obj_n	obj_{n-1}	obj_1	obj	n	→	obj	obj_{n-1}	obj_1

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

UNROT

Type: RPN Command

Description: Changes the order of the first three objects on the stack. The order of the change is the opposite to that of the ROT command.

Access: ↰ PRG STACK UNROT

Input/Output:

L_3	L_2	L_1		L_3	L_2	L_1
obj_3	obj_2	obj_1	→	obj_1	obj_3	obj_2

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

UNTIL

Type: Command

Description: UNTIL Command: Startsthe test clause in a DO … UNTIL … END indefinite loop structure.

See the DO entry for more information.

Access: ↰ PRG BRANCH UNTIL

Input: None

Output: None

See also: DO, END

UPDIR

Type: Command

Description: Up Directory Command: Makes the parent of the current directory the new current directory. UPDIR has no effect if the current directory is *HOME*.

Access: ↰ UPDIR

Input: None

Output: None

See also: CRDIR, HOME, PATH, PGDIR

UTPC

Type: Command

Description: Upper Chi-Square Distribution Command: Returns the probability utpc(*n*, *x*) that a chi-square random variable is greater than *x*, where *n* is the number of degrees of freedom of the distribution.

The defining equations are these:

- For x ≥ 0:

$$utpc(n,x) = \left[\frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)}\right]\int_x^{\infty} t^{\frac{n}{2}-1}\cdot e^{-\frac{t}{2}}dt$$

- For x < 0:

$$utpc(n,x)) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value *n* is rounded to the nearest integer and, when rounded, must be positive.

Access: ↰ MTH PROBABILITY UTPC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*x*	→	*utpc*(*n*,*x*)

See also: UTPF, UTPN, UTPT

UTPF

Type: Command

Description: Upper Snedecor's F Distribution Command: Returns the probability utpf(n_1, n_2, x) that a Snedecor's F random variable is greater than x, where n_1 and n_2 are the numerator and denominator degrees of freedom of the F distribution.

The defining equations for utpf(n_1, n_2, x) are these:

- For x ≥ 0:

$$\left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}}\left[\frac{\Gamma\left(\frac{n_1+n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)}\right]\int_x^{\infty} t^{\frac{n_1-2}{2}}\left[1+\left(\frac{n_1}{n_2}\right)t\right]^{-\frac{(n_1+n_2)}{2}} dt$$

- For x < 0:

$$utpf(n_1, n_2, x) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the HP 49 factorial command.

The values n_1 and n_2 are rounded to the nearest integers and, when rounded, must be positive.

Access: ↰ MTH PROBABILITY UTPF

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
n_1	n_2	x	→	$utpf(n_1,n_2,x)$

See also: UTPC, UTPN, UTPT

UTPN

Type: Command

Description: Upper Normal Distribution Command: Returns the probability utpn(m, v, x) that a normal random variable is greater than x, where m and v are the mean and variance, respectively, of the normal distribution.

For all x and m, and for $v > 0$, the defining equation is this:

$$utpn(m, v, x) = \left[\frac{1}{\sqrt{2\pi v}}\right]\int_{x}^{\infty} e^{-\frac{(t-m)^2}{2v}}\, dt$$

For $v = 0$, UTPN returns 0 for $x \geq m$, and 1 for $x < m$.

Access: ↰ MTH PROBABILITY UTPN

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
m	v	x	→	$utpn(m,v,x)$

See also: UTPC, UTPF, UTPT

UTPT

Type: Command

Description: Upper Student's t Distribution Command: Returns the probability utpt(n, x) that a Student's t random variable is greater than x, where n is the number of degrees of freedom of the distribution.

The following is the defining equation for all x:

$$utpt(n,x) = \left[\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right]\int_x^\infty \left(1+\frac{t^2}{n}\right)^{-\frac{n+1}{2}} dt$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value n is rounded to the nearest integer and, when rounded, must be positive.

Access: ◁ MTH PROBABILITY UTPT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	x	→	$utpt(n,x)$

See also: UTPC, UTPF, UTPN

UVAL

Type: Function

Description: Unit Value Function: Returns the numerical part of a unit object.

Access: ▷ UNITS TOOLS UVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	x
'$symb$'	→	'UVAL($symb$)'

See also: CONVERT, UBASE, UFACT, →UNIT

V→

Type: Command

Description: Vector/Complex Number to Stack Command: Separates a vector or complex number into its component elements.

For vectors with four or more elements, V→ executes *independently* of the coordinate system mode, and always returns the elements of the vector to the stack as they are stored internally (in rectangular form). Thus, V→ is equivalent to OBJ→ for vectors with four or more elements.

Access: ↰ MTH VECTOR V→

Input/Output:

L_1/A_1		L_n/I_1 ... L_3/I_{n-2}	L_2/I_{n-1}	L_1/I_n
$[\,x\,y\,]$	→		x	y
$[\,x_r, y_{theta}\,]$	→		x_r	y_{theta}
$[\,x_1, x_2, x_3\,]$	→	x_1	x_2	x_3
$[\,x_1, x_{theta}, x_z\,]$	→	x_1	x_{theta}	x_z
$[\,x_1, x_{theta}, x_{phi}\,]$	→	x_1	x_{theta}	x_{phi}
$[\,x_1, x_2, ..., x_n\,]$	→	x_1 ... x_{n-2}	x_{n-1}	x_n
(x, y)	→		x	y
(x_r, y_{theta})	→		x_r	y_{theta}

L = level; A = argument; I = item

See also: →V2, →V3

→V2

Type: Command

Description: Stack to Vector/Complex Number Command: Converts two specified numbers into a 2-element vector or a complex number.

The result returned depends on the setting of flags –16 and –19, as shown in the following table:

	Flag –19 clear	Flag –19 set
Flag –16 clear (Rectangular mode)	[x y]	(x, y)
Flag –16 set (Polar mode)	[x ∠ y]	(x, ∠ y)

Access: ↰ MTH VECTOR →V2

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	[x y]
x	y	→	[x ∠ y]
x	y	→	(x, y)
x	y	→	(x, ∠ y)

See also: V→, →V3

→V3

Type: Command

Description: Stack to 3-Element Vector Command: Converts three numbers into a 3-element vector.

The result returned depends on the coordinate mode used, as shown in the following table:

Mode	Result
Rectangular (flag –16 clear)	[x_1 x_2 x_3]
Polar/Cylindrical (flag –15 clear and –16 set)	[x_1 x$\angle_{theta}$ x_z]
Polar/Spherical (flag –15 and –16 set)	[x_1 x$\angle_{theta}$ x$\angle_{phi}$]

Access: ↰ MTH VECTOR →V3

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
x_1	x_2	x_3	→	[x_1 x_2 x_3]
x_1	x_{theta}	x_z	→	[x_1 $\angle x_{theta}$ x_z]
x_1	x_{theta}	x_{phi}	→	[x_1 $\angle x_{theta}$ x_{phi}]

See also: V→, →V2

VAR

Type: Command

Description: Variance Command: Calculates the sample variance of the coordinate values in each of the *m* columns in the current statistics matrix (Σ*DAT*).

The variance (equal to the square of the standard deviation) is returned as a vector of *m* real numbers, or as a single real number if $m = 1$. The variances are computed using this formula:

$$\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$$

where x_i is the *i*th coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and *n* is the number of data points.

Access: (CAT) VAR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{variance}$
	→	$[\, x_{variance1}, \ldots, x_{variancem} \,]$

See also: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, SDEV, TOT

VARS

Type: Command

Description: Variables Command: Returns a list of the names of all variables in the VAR menu for the current directory.

Access: (←) (PRG) MEMORY DIRECTORY VARS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$\{\, global_1 \ldots global_n \,\}$

See also: ORDER, PVARS, TVARS

VERSION

Type: Command

Description: Software Version Command: Displays the software version and copyright message.

Access: (CAT) VERSION

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	"*version number*"	"*copyright message*"

Flags: None

VISIT

Type: Command

.Description: For a specified variable, opens the contents in the command-line editor.

Access: CAT VISIT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	The contents opened in the command line editor.

See also: VISITB, EDIT, EDITB

VISITB

Type: Command

.Description: For a specified variable, opens the contents in the most suitable editor for the object type. For example, if the specified variable holds an equation, the equation is opened in Equation Writer.

Access: CAT VISITB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	The contents opened in the most suitable editor.

See also: VISITB, EDIT, EDITB

VTYPE

Type: Command

Description: Variable Type Command: Returns the type number of the object contained in the named variable.

If the named variable does not exist, VTYPE returns –1.

For a table of the objects' type numbers, see the entry for TYPE.

Access: ↰ PRG TYPE VTYPE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$'name'$	$\rightarrow$	n_{type}
$:n_{port} : name_{backup}$	$\rightarrow$	n_{type}
$:n_{port} : n_{library}$	$\rightarrow$	n_{type}

See also: TYPE

W to Z

WAIT

Type: Command

Description: Wait Command: Suspends program execution for specified time, or until a key is pressed.

The function of WAIT depends on the argument, as follows:

- Argument x interrupts program execution for x seconds.
- Argument 0 suspends program execution until a valid key is pressed (see below). WAIT then returns x_{key}, which defines where the pressed key is on the keyboard, and resumes program execution.

  x_{key} is a three-digit number that identifies a key's location on the keyboard. See the entry for ASN for a description of the format of x_{key}.
- Argument –1 works as with argument 0, except that the currently specified menu is also displayed.

◁, ▷, ALPHA, ALPHA◁, and ALPHA▷ are not by themselves valid keys.

Arguments 0 and –1 do not affect the display, so that messages persist even though the keyboard is ready for input (FREEZE is not required).

Normally, the MENU command does not update the menu keys until a program halts or ends. WAIT with argument –1 enables a previous execution of MENU to display that menu while the program is suspended for a key press.

Access: ◁ PRG IN WAIT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	
0	→	x_{key}
–1	→	x_{key}

See also: KEY

WHILE

Type: Command Operation

Description: WHILE Indefinite Loop Structure Command: Starts the WHILE … REPEAT … END indefinite loop structure.

WHILE … REPEAT … END repeatedly evaluates a test and executes a loop clause if the test is true. Since the test clause occurs before the loop-clause, the loop clause is never executed if the test is initially false. The syntax is this:

WHILE *test-clause* REPEAT *loop-clause* END

The test clause is executed and must return a test result to the stack. REPEAT takes the value from the stack. If the value is not zero, execution continues with the loop clause; otherwise, execution resumes following END.

Access: ↰ PRG BRANCH WHILE

Input/Output:

Level 1/Argument 1			Level 1/Item 1
WHILE		→	
REPEAT	*T/F*	→	
END		→	

See also: DO, END, REPEAT

WIREFRAME

Type: Command

Description: WIREFRAME Plot Type Command: Sets the plot type to WIREFRAME.

When the plot type is set to WIREFRAME, the DRAW command plots a perspective view of the graph of a scalar function of two variables. WIREFRAME requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

$\{ x_{left}, x_{right}, y_{near}, y_{far}, z_{low}, z_{high}, x_{min}, x_{max}, y_{min}, y_{max}, x_{eye}, y_{eye}, z_{eye}, x_{step}, y_{step} \}$

For plot type WIREFRAME, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min},y_{min}) (x_{max},y_{max}) *indep res axes ptype depend* }

For plot type WIREFRAME, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is not used.
- (x_{max},y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a name specifying the plot type. Executing the command WIREFRAME places the command name WIREFRAME in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT WIREFRAME

Input: None

Output: None

See also: BAR, CONIC DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, YSLICE

WSLOG

Type: Command

Description: Warmstart Log Command: Returns four strings recording the date, time, and cause of the four most recent warmstart events.

Each string "log_n" has the form "*code–date time*". The following table summarizes the legal values of *code* and their meanings.

Code	Description
0	The warmstart log was cleared by pressing ON SPC.
1	The interrupt system detected a very low battery condition at the battery contacts (not the same as a low system voltage), and put the calculator in "Deep Sleep mode" (*with the system clock running*). When ON is pressed after the battery voltage is restored, the system warmstarts and puts a 1 in the log.
2	Hardware failed during transmission (timeout).
3	Run through address 0.
4	System time is corrupt
5	A Deep Sleep wakeup (for example, ON, Alarm).
6	Not used
7	A 5-nibble word (CMOS test word) in RAM was corrupt. (This word is checked on every interrupt, but it is used only as an indicator of potentially corrupt RAM.)
8	Not used
9	The alarm list is corrupt.
A	System RPL jump to #0.

Code	Description (Continued)
B	The card module was removed (or card bounce).
C	Hardware reset occurred (for example, an electrostatic discharge or user reset)
D	An expected System (RPL) error handler was not found in runstream

The date and time stamp (*date time*) part of the log may be displayed as 00…0000 for one of three reasons:

- The system time was corrupt when the stamp was recorded.
- The date and time stamp itself is corrupt (bad checksum).
- Fewer than four warmstarts have occurred since the log was last cleared.

Access: CAT WSLOG

Input/Output:

Level 1/Argument 1		Level 4/Item 1 ... Level 1/Item 4
	→	"$\log_4$" ... "$\log_1$"

ΣX

Type: Command

Description: Sum of *x*-Values Command: Sums the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

Access: CAT ΣX

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sum}

See also: NΣ, XCOL, ΣXY, ΣX2, ΣY, ΣY2

ΣX2

Type: Command

Description: Sum of Squares of x-Values Command: Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

Access: CAT ΣX2

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sum}

See also: NΣ, ΣX, XCOL, ΣX*Y, ΣY, ΣY^2

XCOL

Type: Command

Description: Independent Column Command: Specifies the independent-variable column of the current statistics matrix (reserved variable Σ*DAT*).

The independent-variable column number is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

XCOL will accept a noninteger real number and store it in Σ*PAR*, but subsequent commands that utilize the XCOL specification in Σ*PAR* will cause an error.

Access: CAT XCOL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{col}	→	

See also: BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, YCOL

XGET

Type: Command

Description: XModem Get Command: Retrieves a specified filename via XMODEM from another calculator. The other calculator needs to be in server mode for the operation to work (APPS I/O FUNCTIONS START SERVER.

Access: CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: BAUD, RECN, RECV, SEND XRECV, XSERV, XPUT

XMIT

Type: Command

Description: Serial Transmit Command: Sends a string serially without using Kermit protocol, and returns a single digit that indicates whether the transmission was successful.

XMIT is useful for communicating with non-Kermit devices such as RS-232 printers.

If the transmission is successful, XMIT returns a 1. If the transmission is not successful, XMIT returns the unsent portion of the string and a 0. Use ERRM to get the error message.

After receiving an XOFF command (with *transmit pacing* in the reserved variable *IOPAR* set), XMIT stops transmitting and waits for an XON command. XMIT resumes transmitting if an XON is received before the time-out set by STIME elapses; otherwise, XMIT terminates, returns a 0, and stores "Timeout" in ERRM.

Access: CAT XMIT

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
"*string*"	→		1
"*string*"	→	"$substring_{unsent}$"	0

See also: BUFLEN, SBRK, SRECV, STIME

XOR

Type: Function

Description: Exclusive OR Function: Returns the logical exclusive OR of two arguments.

When the arguments are binary integers or strings, XOR does a bit-by-bit (base 2) logical comparison:

- Binary integer arguments are treated as sequences of bits with length equal to the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments, as shown in the following table:

bit_1	bit_2	bit_1 **XOR** bit_2
0	0	0
0	1	1
1	0	1
1	1	0

- String arguments are treated as sequences of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, XOR simply does a true/false test. The result is 1 (true) if either, but not both, arguments are nonzero; it is 0 (false) if both arguments are nonzero or zero. This test is usually done to compare two test results.

If either or both of the arguments are algebraic objects, then the result is an algebraic of the form $symb_1$ XOR $symb_2$. Execute →NUM (or set flag –3 before executing XOR) to produce a numeric result from the algebraic result.

Access: ▶ BASE LOGIC XOR

◀ MTH BASE LOGIC XOR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	$\#n_2$	→	$\#n_3$

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"*string*$_1$"	"*string*$_2$"	→	"*string*$_3$"
T/F$_1$	*T/F*$_2$	→	0/1
T/F	'*symb*'	→	'*T/F* XOR *symb*'
'*symb*'	*T/F*	→	'*symb* XOR *T/F*'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ XOR *symb*$_2$'

See also: AND, NOT, OR

XPON

Type: Function

Description: Exponent Function: Returns the exponent of the argument.

Access: ↰ MTH REAL XPON

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	n_{expon}
'*symb*'	→	'XPON(*symb*)'

See also: MANT, SIGN

XPUT

Type: Command

Description: XModem Send Command: Sends a specified filename via XMODEM to a calculator. The receiving calculator needs to be in Server mode (APPS I/O FUNCTIONS START SERVER).

Access: CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: BAUD, RECN, RECV, SEND XRECV, XSERV, XGET

XRECV

Type: Command

Description: XModem Receive Command: Prepares the HP 49 to receive an object via XModem. The received object is stored in the given variable name.

The transfer will start more quickly if you start the XModem sender *before* executing XRECV.

Invalid object names cause an error. If flag –36 is clear, object names that are already in use also cause an error.

If you are transferring data between two HP 49s, executing {AAA BBB CCC} XRECV receives *AAA*, *BBB*, and *CCC*. You also need to use a list on the sending end ({AAA BBB CCC} XSEND, for example).

Access: CAT XRECV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: BAUD, RECV, RECN, SEND, XSEND

XRNG

Type: Command

Description: x-Axis Display Range Command: Specifies the *x*-axis display range.

The *x*-axis display range is stored in the reserved variable *PPAR* as x_{min} and x_{max} in the complex numbers (x_{min},y_{min}) and (x_{max},y_{max}). These complex numbers are the first two elements of *PPAR* and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of x_{min} and x_{max} are –6.5 and 6.5, respectively.

Access: CAT XRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{min}	x_{max}	→	

See also: AUTO, PDIM, PMAX, PMIN, YRNG

XROOT

Type: Analytic function

Description: *x*th Root of *y* Command: Computes the *x*th root of a real number.

XROOT is equivalent to $y^{1/x}$, but with greater accuracy.

If $y < 0$, x must be an integer.

Access: ↱ $\sqrt[x]{y}$

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y	*x*	→	$\sqrt[x]{y}$
'*symb*$_1$'	'*symb*$_2$'	→	'XROOT(*symb*$_2$,*symb*$_1$)'
'*symb*'	*x*	→	'XROOT(*x*,*symb*)'
y	'*symb*'	→	'XROOT(*symb*,*y*)'
y_unit	*x*	→	$\sqrt[x]{y}$ *_unit*$^{1/x}$
y_unit	'*symb*'	→	'XROOT(*symb*,*y_unit*)'

XSEND

Type: Command

Description: XModem Send Command: Sends a copy of the named object via XModem.

A receiving HP 49 must execute XRECV to receive an object via XModem.

The transfer occurs more quickly if you start the receiving XModem *after* executing XSEND. Also, configuring the receiving modem *not* to do CRC checksums (if possible) will avoid a 30 to 60-second delay when starting the transfer.

If you are transferring data between two HP 49s, executing {AAA BBB CCC} XSEND sends *AAA*, *BBB*, and *CCC*. You also need to use a list on the receiving end ({AAA BBB CCC} XRECV, for example).

Access: CAT XSEND

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	

See also: BAUD, RECN, RECV, SEND XRECV

XSERV

Type: Command

Description: XModem Server Command: Puts the calculator in XMODEM server mode. When in server mode, the following commands are available:
P: Put a file in the calculator
G: Get a file from the calculator
E: Execute a command line
M Get the calculator memory
L: List the files in the current directory

Access: CAT XSERV

See also: BAUD, RECN, RECV, SEND XRECV, XGET, XPUT

XVOL

Type: Command

Description: X Volume Coordinates Command: Sets the width of the view volume in the reserved variable *VPAR*.

x_{left} and x_{right} set the x-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable *VPAR*.

Access: CAT XVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{left}	x_{right}	→	

See also: EYEPT, XXRNG, YVOL, YYRNG, ZVOL

XXRNG

Type: Command

Description: X Range of an Input Plane (Domain) Command: Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

x_{min} and x_{max} are real numbers that set the x-coordinates for the input plane. These values are stored in the reserved variable *VPAR*.

Access: CAT XXRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{min}	x_{max}	→	

See also: EYEPT, NUMX, NUMY, XVOL, YVOL, YYRNG, ZVOL

ΣXY

Type: Command

Description: Sum of X times Y command: Sums the products of each of the corresponding values in the independent- and dependent-variable columns of the current statistical matrix (reserved variable ΣDAT). The independent column is the column designated as XCOL and the dependent column is the column designated as YCOL.

Access: ↱ STAT SUMMARY STATS

Output: The sum of the corresponding products.

See also: ΣY

ΣY

Type: Command

Description: Sum of y-Values Command: Sums the values in the dependent variable column of the current statistical matrix (reserved variable Σ*DAT*).

The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable Σ*PAR*. The default dependent variable column number is 2.

Access: CAT ΣY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sum}

See also: NΣ, ΣX, XCOL, ΣXY, ΣX2, YCOL, ΣY2

ΣY2

Type: Command

Description: Sum of squares of Y-value command: Sums the squares of the values in the dependent-variable columns of the current statistical matrix (reserved variable ΣDAT). The dependent column is the column designated as YCOL

Access: ▷ STAT SUMMARY STATS

Output: The sum of the dependent variables.

See also: ΣXY

YCOL

Type: Command

Description: Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable Σ*DAT*).

The dependent variable column number is stored as the second parameter in the reserved variable Σ*PAR*. The default dependent variable column number is 2.

YCOL will accept a noninteger real number and store it in Σ*PAR*, but subsequent commands that utilize the YCOL specification in Σ*PAR* will cause an error.

Access: CAT YCOL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{col}	→	

See also: BARPLOT, BESTFIT, COLΣ, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL

YRNG

Type: Command

Description: y-Axis Display Range Command: Specifies the y-axis display range.

The y-axis display range is stored in the reserved variable *PPAR* as y_{min} and y_{max} in the complex numbers (x_{min},y_{min}) and (x_{max},y_{max}). These complex numbers are the first two elements of *PPAR* and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of y_{min} and y_{max} are –3.1 and 3.2, respectively.

Access: CAT YRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{min}	y_{max}	→	

See also: AUTO, PDIM, PMAX, PMIN, XRNG

YSLICE

Type: Command

Description: Y-Slice Plot Command: Sets the plot type to YSLICE.

When plot type is set YSLICE, the DRAW command plots a slicing view of a scalar function of two variables. YSLICE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

$\{ x_{left}, x_{right}, y_{near}, y_{far}, z_{low}, z_{high}, x_{min}, x_{max}, y_{min}, y_{max}, x_{eye}, y_{eye}, z_{eye}, x_{step}, y_{step} \}$

For plot type YSLICE, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} determines the interval between plotted x-values within each "slice".

- y_{step} determines the number of slices to draw.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

{ (x_{min},y_{min}), (x_{max},y_{max}), *indep*, *res*, *axes*, *ptype*, *depend* }

For plot type YSLICE, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is not used.
- (x_{max},y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is a real number specifying the interval, in user-unit coordinates, between plotted values of the independent variable; or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command YSLICE places YSLICE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: (CAT) YSLICE

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME

YVOL

Type: Command

Description: Y Volume Coordinates Command: Sets the depth of the view volume in the reserved variable *VPAR*.

The variables y_{near} and y_{far} are real numbers that set the y-coordinates for the view volume used in 3D plots. y_{near} must be less than y_{far}. These values are stored in the reserved variable *VPAR*.

Access: (CAT) YVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YYRNG, ZVOL

YYRNG

Type: Command

Description: Y Range of an Input Plane (Domain) Command: Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

The variables $y_{\text{y near}}$ and $y_{\text{y far}}$ are real numbers that set the y-coordinates for the input plane. These values are stored in the reserved variable *VPAR*.

Access: CAT YYRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YVOL, ZVOL

ZFACTOR

Type: Function

Description: Gas Compressibility Z Factor Function: Calculates the gas compressibility correction factor for non-ideal behavior of a hydrocarbon gas.

x_{Tr} is the reduced temperature: the ratio of the actual temperature (T) to the pseudocritical temperature (T_c). (Calculate the ratio using absolute temperatures.) x_{Tr} must be between 1.05 and 3.0.

y_{Pr} is the reduced pressure: the ratio of the actual pressure (P) to the pseudocritical pressure (P_c). y_{Pr} must be between 0 and 30.

x_{Tr} and y_{Pr} must be real numbers or unit objects that reduce to dimensionless numbers.

Access: CAT ZFACTOR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{Tr}	y_{Pr}	→	$x_{ZFactor}$
x_{Tr}	'*symb*'	→	'ZFACTOR(x_{Tr},*symb*)'
'*symb*'	y_{Pr}	→	'ZFACTOR(*symb*,y_{Pr})'
'$symb_1$'	'$symb_2$'	→	'ZFACTOR($symb_1$,$symb_2$)'

ZVOL

Type: Command

Description: Z Volume Coordinates Command: Sets the height of the view volume in the reserved variable *VPAR*.

x_{low} and x_{high} are real numbers that set the z-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable *VPAR*.

Access: CAT ZVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{low}	x_{high}	→	

See also: EYEPT, XVOL, XXRNG, YVOL, YYRNG

---

^

Type: Function

Description: Power Analytic Function: Returns the value of the level 2 object raised to the power of the level 1 object.

If either argument is complex, the result is complex.

The branch cuts and inverse relations for w^z are determined by this relationship:

$$w^z = \exp(z(\ln w))$$

Access: y^x

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
w	*z*	→	*wz*
z	'*symb*'	→	'*z*^(*symb*)'
'*symb*'	*z*	→	'(*symb*)^*z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$^('$symb_2$)'
x_unit	*y*	→	*x^y_unit*
x_unit	'*symb*'	→	'(*x_unit*)^(*symb*)'

See also: EXP, ISOL, LN, XROOT

| (Where)

Type: Function

Description: Where Function: Substitutes values for names in an expression.

| is used primarily in algebraic objects, where its syntax is:

'$symb_{old}$ | ($name_1$ = $symb_1$, $name_2$ = $symb_2$ …)'

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as ∫ and ∂) can then extract substitution information from local variables and include that information in the expression, avoiding the problem that would occur if the local variables no longer existed when the local names were finally evaluated.

Access:

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_{old}$'	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	'$symb_{new}$'
x	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	*x*
(*x*,*y*)	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	(*x*,*y*)

See also: APPLY, QUOTE

√

Type: Function

Description: Square Root Analytic Function: Returns the (positive) square root of the argument.

For a complex number (x_1, y_1), the square root is this complex number:

$$(x_2, y_2) = \left(\sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\right)$$

where r = ABS (x_1, y_1), and θ = ARG (x_1, y_1).

If $(x_1, y_1) = (0,0)$, then the square root is (0, 0).

The inverse of SQ is a *relation*, not a function, since SQ sends more than one argument to the same result. The inverse relation for SQ is expressed by ISOL as this *general solution*:

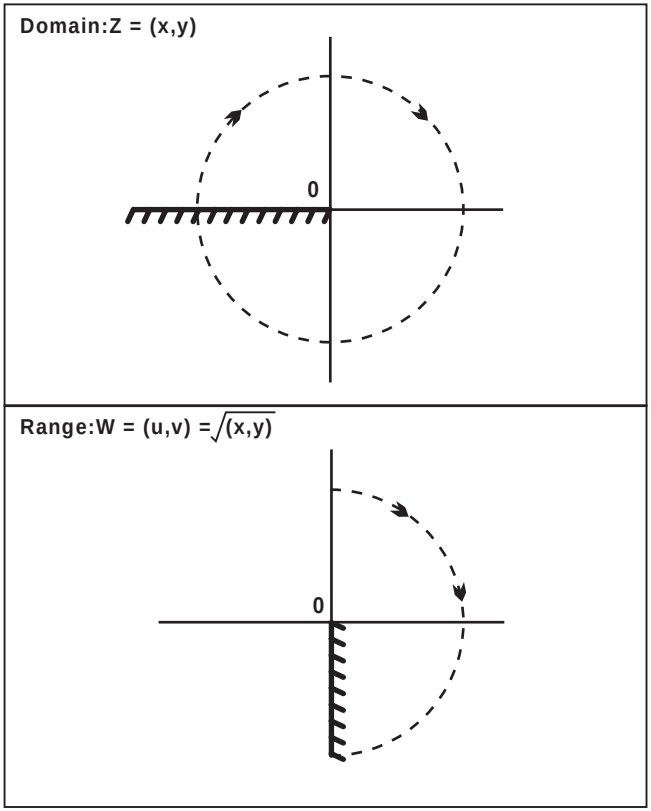
's1*√Z'

The function √ is the inverse of a *part* of SQ, a part defined by restricting the domain of SQ such that:

1. each argument is sent to a distinct result, and
2. each possible result is achieved. The points in this restricted domain of SQ are called the *principal values* of the inverse relation. The √ function in its entirety is called the *principal branch* of the inverse relation, and the points sent by √ to the boundary of the restricted domain of SQ form the *branch cuts* of √.

The principal branch used by the HP 49G for √ was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued square root function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

The graphs below show the domain and range of √. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.



These graphs show the inverse relation 's1*√Z' for the case *s1*=1. For the other value of *s1*, the half-plane in the lower graph is rotated. Taken together, the half-planes cover the whole complex plane, which is the domain of SQ.

View these graphs with domain and range reversed to see how the domain of SQ is restricted to make an inverse *function* possible. Consider the half-plane in the lower graph as the restricted domain Z = (*x*, *y*). SQ sends this domain onto the whole complex plane in the range W = (*u*, *v*) = SQ(*x*, *y*) in the upper graph.

Access: $\boxed{\sqrt{x}}$

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	$\sqrt{z}$
x_unit	→	$\sqrt{x}\ unit^{1/2}$
'*symb*'	→	'$\sqrt{(symb)}$'

See also: SQ, ^, ISOL

∫

Type: Function

Description: Integral Function: Integrates an *integrand* from *lower limit* to *upper limit* with respect to a specified variable of integration.

The algebraic syntax for ∫ parallels its stack syntax:

∫(*lower limit*, *upper limit*, *integrand*, *name*)

where *lower limit*, *upper limit*, and *integrand* can be real or complex numbers, unit objects, names, or algebraic expressions.

Evaluating ∫ in Symbolic Results mode (flag –3 *clear*) returns a symbolic result. Some functions that the The HP 49G can integrate are as follows:

- All built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, SIN can be integrated since its antiderivative, COS, is a built-in function. The arguments for these functions must be linear.
- Sums, differences, and negations of built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, 'SIN(X)–COS(X)'.
- Derivatives of all built-in functions—for example, 'INV(1+X^2)' can be integrated because it is the derivative of the built-in function ATAN.
- Polynomials whose base term is linear—for example, 'X^3+X^2–2*X+6' can be integrated since X is a linear term. '(X^2–6)^3+(X^2–6)^2' cannot be integrated since X^2–6 is not linear.
- Selected patterns composed of functions whose antiderivatives can be expressed in terms of other built-in functions—for example, '1/(COS(X)*SIN(X))' returns 'LN(TAN(X))'.

If the result of the integration is an expression with no integral sign in the result, the symbolic integration was successful. If, however, the result still contains an integral sign, try rearranging the expression and evaluating again, or estimate the answer using numerical integration.

Evaluating ∫ in Numerical Results mode (flag –3 *set*) returns a numerical approximation, and stores the error of integration in variable *IERR*. ∫ consults the number format setting to determine how accurately to compute the result.

Access: ↱ ∫

Input/Output:

L4/A1	L3/A2	L2/A3	L1/A4		L1/I1
lower limit	*upper limit*	*integrand*	'*name*'	→	'$symb_{integral}$'

L = level; A = Argument; I = Item

See also: TAYLR, ∂, Σ

Σ

Type: Function

Description: Summation Function: Calculates the value of a finite series.

The summand argument *smnd* can be a real number, a complex number, a unit object, a local or global name, or an algebraic object.

The algebraic syntax for Σ differs from the stack syntax. The algebraic syntax is:

'Σ(*indx*=*initial*,*final*,*summand*)'

Access: ↱ Σ

nput/Output:

L4/A1	L3/A2	L2/A3	L1/A4		L1/I1
'*indx*'	x_{init}	x_{final}	*smnd*	→	x_{sum}
'*indx*'	'*init*'	x_{final}	*smnd*	→	'$\Sigma(indx = init, x_{final}, smnd)$'
'*indx*'	x_{init}	'*final*'	*smnd*	→	'$\Sigma(indx = x_{init}, final, smnd)$'
'*indx*'	'*init*'	'*final*'	*smnd*	→	'$\Sigma(indx = init, final, smnd)$'

L = level; A = Argument; I = Item

See also: TAYLR, ∫, ∂

Σ+

Type: Command

Description: Sigma Plus Command: Adds one or more data points to the current statistics matrix (reserved variable Σ*DAT*).

For a statistics matrix with *m* columns, arguments for Σ+ can be entered several ways:

- To enter one data point with a single coordinate value, the argument for Σ+ must be a real number.
- To enter one data point with multiple coordinate values, the argument for Σ+ must be a vector with *m* real coordinate values.
- To enter several data points, the argument for Σ+ must be a matrix of *n* rows of *m* real coordinate values.

In each case, the coordinate values of the data point(s) are added as new rows to the current statistics matrix (reserved variable Σ*DAT*). If Σ*DAT* does not exist, Σ+ creates an *n* x *m* matrix and stores the matrix in Σ*DAT*. If Σ*DAT* does exist, an error occurs if it does not contain a real matrix, or if the number of coordinate values in each data point entered with Σ+ does not match the number of columns in the current statistics matrix.

Once Σ*DAT* exists, individual data points of *m* coordinates can be entered as *m* separate real numbers or an *m*-element vector.

LASTARG returns the *m*-element vector in either case.

Access: CAT Σ+

Input/Output:

$L_m/A_1 \ldots L_2/A_{m-1}$	L_1/A_m		L_1/I_1
	x	→	
	$[x_1, x_2, \ldots, x_m]$	→	
	$[[x_{1\,1}, \ldots, x_{1\,m}]\ [x_{n\,1}, \ldots, x_{n\,m}]]$	→	
$x_1 \ldots x_{m-1}$	x_m	→	

L = level; A = Argument; I = Item

See also: CLΣ, RCLΣ, STOΣ, Σ–

Σ–

Type: Command

Description: Sigma Minus Command: Returns a vector of m real numbers (or one number x if $m = 1$) corresponding to the coordinate values of the last data point entered by Σ+ into the current statistics matrix (reserved variable Σ*DAT*).

The last row of the statistics matrix is deleted.

The vector returned by Σ– can be edited or replaced, then restored to the statistics matrix by Σ+.

Access: CAT Σ–

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x
	→	$[\, x_1\ x_2\ \ldots\ x_m\,]$

See also: CLΣ, RCLΣ, STOΣ, Σ+

π

Type: Function

Description: π Function: Returns the symbolic constant 'π' or its numerical representation, 3.14159265359.

The number returned for π is the closest approximation of the constant π to 12-digit accuracy.

In Radians mode with flag –2 and –3 clear (to return symbolic results), trigonometric functions of π and π/2 are automatically simplified. For example, evaluating 'SIN(π)' returns zero. However, if flag –2 or flag –3 is set (to return numerical results), then evaluating 'SIN(π)' returns the numerical approximation –2.06761537357E–13.

Access: ↰ π

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'π'
	→	3.14159265359…

See also: e, i, MAXR, MINR, →Qπ

∂

Type: Function

Description: Derivative Function: Takes the derivative of an expression, number, or unit object with respect to a specified variable of differentiation.

When executed in stack syntax, ∂ executes a *complete* differentiation: the expression '$symb_1$' is evaluated repeatedly until it contains no derivatives. As part of this process, if the variable of differentiation *name* has a value, the final form of the expression substitutes that value substituted for all occurrences of the variable.

The algebraic syntax for ∂ is '∂*name*(*symb*$_1$'). When executed in algebraic syntax, ∂ executes a *stepwise* differentiation of *symb*$_1$, invoking the chain rule of differentiation— the result of one evaluation of the expression is the derivative of the argument expression *symb*$_1$, multiplied by a new subexpression representing the derivative of *symb*$_1$'s argument.

If ∂ is applied to a function for which theHP 49G does not provide a derivative, ∂ returns a new function whose name is *der* followed by the original function name.

Access: ⮡ ∂

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_1$'	'*name*'	→	'$symb_2$'
z	'*name*'	→	0
x_unit	'*name*'	→	0

See also: TAYLOR, ∫, Σ

≤

Type: Function

Description: Less Than or Equal Function: Tests whether one object is less than or equal to another object.

The function ≤ returns a true test result (1) if the first argument is less than or equal to the lsecond argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≤ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, “less than or equal” means numerically equal or smaller (1 is less than 2). For real numbers, “less than or equal” also means equally or more negative (–2 is less than –1).

For strings, “less than or equal” means alphabetically equal or previous (“ABC” is less than or equal to “DEF”; “AAA” is less than or equal to “AAB”; “A” is less than or equal to “AA”). In general, characters are ordered according to their character codes. This means, for example, that “B” is less than “a”, since “B” is character code 66, and “a” is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperature and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰≤

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
“$string_1$”	“$string_2$”	→	0/1
x	'$symb$'	→	'$x \le symb$'
'$symb$'	x	→	'$symb \le x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \le symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit \le symb$'
'$symb$'	x_unit	→	'$symb \le x\_unit$'

See also: <, >, ≥, ==, ≠

≥

Type: Function

Description: Greater Than or Equal Function: Tests whether one object is greater than or equal to another object.

The function ≥ returns a true test result (1) if the first argument is greater than or equal to the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≥ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than or equal to" means numerically equal or greater (2 is greater than or equal to 1). For real numbers, "greater than or equal to" also means equally or less negative (–1 is greater than or equal to –2).

For strings, "greater than or equal to" means alphabetically equal or subsequent ("DEF" is greater than or equal to "ABC"; "AAB" is greater than or equal to "AAA"; "AA" is greater than or equal to "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than or equal to "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰ ≥

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x \geq symb$'
'$symb$'	x	→	'$symb \geq x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \geq symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit \geq symb$'
'$symb$'	x_unit	→	'$symb \geq x\_unit$'

See also: <, ≤, >, ==, ≠

≠

Type: Function

Description: Not Equal Function: Tests if two objects are not equal.

The function ≠ returns a true result (1) if the two objects have different values, or a false result (0) otherwise. (Lists and programs are considered to have the same values if the objects they contain are identical.)

If one object is algebraic or a name, and the other is a number, a name, or algebraic, ≠ returns a symbolic comparison expression that can be evaluated to return a test result.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number, so, for example, 6 and (6,0) and considered to be equal..

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰ ≠

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	0/1
$(x,0)$	x	→	0/1
x	$(x,0)$	→	0/1
z	'*symb*'	→	'z ≠ *symb*'
'*symb*'	z		'*symb* ≠ z'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \neq symb_2$'

See also: SAME, TYPE, <, ≤, >,≥, ==, ≠

→

Type: Command

Description: Create Local Variables Command: Creates local variables.

Local variable structures specify one or more local variables and a defining procedure.

A local variable structure consists of the → command, followed by one or more names, followed by a defining procedure—either a program or an algebraic. The → command stores objects into local variables with the specified names. The resultant *local variables* exist only while the defining procedure is being executed. The syntax of a local variable structure is one of the following:

- → $name_1$ $name_2$ … $name_n$ « *program* »
- → $name_1$ $name_2$ … $name_n$ '*algebraic expression*'

Access: ⮌⮎

IInput/Output:

$Level_n$/$Argument_1$ … $Level_1$/$Argument_n$		Level 1/Item 1
obj_1 … obj_n	→	

See also: DEFINE, STO

!

Type: Function

Description: Factorial (Gamma) Function: Returns the factorial $n!$ of a positive integer argument n, or the gamma function $\Gamma(x+1)$ of a non-integer argument x.

For $x \geq 253.1190554375$ or $n < 0$, ! causes an overflow exception (if flag –21 is set, the exception is treated as an error). For non-integer $x \leq -254.1082426465$, ! causes an underflow exception (if flag –20 is set, the exception is treated as an error).

In algebraic syntax, ! follows its argument. Thus the algebraic syntax for the factorial of 7 is 7!.

For non-integer arguments x, $x! = \Gamma(x + 1)$, defined for $x > -1$ as:

$$\Gamma(x+1) = \int_0^\infty e^{-t} t^x dt$$

and defined for other values of x by analytic continuation:

$$\Gamma(x + 1) = x\,\Gamma(x)$$

Access: ↰ MTH PROBABILITY !

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	$n!$
x	→	$\Gamma(x + 1)$
'*symb*'	→	'(*symb*!)'

See also: COMB, PERM

%

Type: Function

Description: Percent Function: Returns x percent of y.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, –, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

For more information on using temperature units with arithmetic functions, see the entry for +.

Access: ↰ MTH REAL %

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$xy/100$
x	'*symb*'	→	'%(x,*symb*)'
'*symb*'	x	→	'%(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%($symb_1$, $symb_2$)'
x	*y_unit*	→	(xy/100)_*unit*
x_unit	y	→	(xy/100)_*unit*
'*symb*'	*x_unit*	→	'%(*symb*,*x_unit*)'
x_unit	'*symb*'	→	'%(*x_unit*,*symb*)'

See also: %CH, %T

*****

Type: Function

Description: Multiply Analytic Function: Returns the product of the arguments.

The product of a real number a and a complex number (x, y) is the complex number (xa, ya).

The product of two complex numbers (x_1, y_1) and (x_2, y_2) is the complex number $(x_1 x_2 - y_1 y_2, x_1 y_2 + x_2 y_1)$.

The product of a real array and a complex array or number is a complex array. Each element x of the real array is treated as a complex element $(x, 0)$.

Multiplying a matrix by an array returns a matrix product. The matrix must have the same number of columns as the array has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a *row* of numbers, the HP 49G treats a vector as an $n \times 1$ matrix when multiplying matrices or computing matrix norms.

Multiplying a binary integer by a real number returns a binary integer that is the product of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the multiplication.)

The product of two binary integers is truncated to the current binary integer wordsize.

When multiplying two unit objects, the scaler parts and the unit parts are multiplied separately.

Access: CAT *

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 \tilde{z}_2$
[[*matrix*]]	[*array*]	→	[[*matrix* × *array*]]
$\tilde{z}$	[*array*]	→	[$\tilde{z}$ × *array*]
[*array*]	$\tilde{z}$	→	[*array* × $\tilde{z}$]
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ * *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* * $\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ *$symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit	*y_unit*	→	$xy\_unit_x \times unit_y$
x	*y_unit*	→	*xy_unit*
x_unit	*y*	→	*xy_unit*
'*symb*'	*x_unit*	→	'*symb* * *x_unit*'
x_unit	'*symb*'	→	'*x_unit* * *symb*'

See also: +, –, /, =

+

Type: Function

Description: Add Analytic Function: Returns the sum of the arguments.

The sum of a real number *a* and a complex number (x, y) is the complex number $(x+a, y)$.

The sum of two complex numbers (x_1, y_1) and (x_2, y_2) is the complex number (x_1+x_2, y_1+y_2).

The sum of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The arrays must have the same dimensions.

The sum of a binary integer and a real number is a binary integer that is the sum of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the addition.)

The sum of two binary integers is truncated to the current binary integer wordsize.

The sum of two unit objects is a unit object with the same dimensions as the second argument. The units of the two arguments must be consistent.

The sum of two graphics objects is the same as the result of performing a logical OR, except that the two graphics objects *must* have the same dimensions.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units x_°C and x_°F represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠. This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to Kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as x_°C/min, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, –, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ⊞

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	z_2	→	$z_1 + z_2$
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_3$
z	'*symb*'	→	'z +*symb*'
'*symb*'	z	→	'*symb* +z'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ + *symb*$_2$'
{ *list*$_1$ }	{ *list*$_2$	→	{ *list*$_1$ *list*$_2$ }
obj$_A$	{ *obj*$_1$ … *obj*$_n$ }	→	{ *obj*$_A$ *obj*$_1$ … *obj*$_n$ }
{ *obj*$_1$ … *obj*$_n$ }	*obj*$_A$	→	{*obj*$_1$ … *obj*$_n$ *obj*$_A$ }
"*string*$_1$"	"*string*$_2$"	→	"*string*$_1$ *string*$_2$"
obj	"*string*"	→	"*obj string*"
"*string*"	*obj*	→	"*string obj*"

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit_1	y_unit_2	→	$(x_2 + y)\_unit_2$
'*symb*'	x_unit	→	'*symb* + x_unit'
x_unit	'*symb*'	→	'x_unit + *symb*'
$grob_1$	$grob_2$	→	$grob_3$

See also: –, *, /, =

–

Type: Function

Description: Subtract Analytic Function: Returns the difference of the arguments.

The difference of a real number a and a complex number (x, y) is $(x–a, y)$ or $(a–x, –y)$. The difference of two complex numbers (x_1, y_1) and (x_2, y_2) is $(x_1 – x_2, y_1 – y_2)$.

The difference of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The two array arguments must have the same dimensions.

The difference of a binary integer and a real number is a binary integer that is the sum of the first argument and the two's complement of the second argument. (The real number is converted to a binary integer before the subtraction.)

The difference of two binary integers is a binary integer that is the sum of the first argument and the two's complement of the second argument.

The difference of two unit objects is a unit object with the same dimensions as the second argument. The units of the two arguments must be consistent.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units x_°C and x_°F represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠.

This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to Kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as x_°C/min, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ⊖

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 - \tilde{z}_2$
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1-2}$
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ – *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* – $\tilde{z}$'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ – *symb*$_2$'
#n_1	n_2	→	#n_3
n_1	#n_2	→	#n_3
#n_1	#n_2	→	#n_3
x_1_*unit*$_1$	y_*unit*$_2$	→	$(x_2 - y)$_*unit*$_2$
'*symb*'	x_*unit*	→	'*symb* – x_*unit*'
x_*unit*	'*symb*'	→	'x_*unit* – *symb*'

See also: +, *, /, =

/

Type: Function

Description: Divide Analytic Function: Returns the quotient of the arguments: the first argument is divided by the second argument.

A real number *a* divided by a complex number (x, y) returns:

$$\left(\frac{ax}{x^2+y^2}, \frac{ay}{x^2+y^2}\right)$$

. A complex number (x, y) divided by a real number *a* returns the complex number $(x/a, y/a)$.

A complex number (x_1, y_1) divided by another complex number (x_2, y_2) returns this complex quotient:

$$\left(\frac{x_1x_2+y_1y_2}{x_2^2+y_2^2}, \frac{y_1x_2-x_1y_2}{x_2^2+y_2^2}\right)$$

An array **B** divided by a matrix **A** solves the system of equations **AX**=**B** for **X**; that is, **X** = $\mathbf{A}^{-1}\,\mathbf{B}$. This operation uses 15-digit internal precision, providing a more precise result than the calculation INV(A)*B. The matrix must be square, and must have the same number of columns as the array has rows (or elements, if the array is a vector).

A binary integer divided by a real or binary number returns a binary integer that is the integral part of the quotient. (The real number is converted to a binary integer before the division.) A divisor of zero returns # 0.

When dividing two unit objects, the scaler parts and the unit parts are divided separately.

Access: CAT /

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1$ / $\tilde{z}_2$
[*array*]	[[*matrix*]]	→	[[$matrix^{-1} \times array$]]
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ / *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* / $\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ / $symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_unit_1	y_unit_2	→	$(x/y)\_unit_1/unit_2$
x	y_unit	→	$(x/y)\_1/unit$
x_unit	y	→	$(x/y)\_unit$
'*symb*'	x_unit	→	'*symb* / x_unit'
x_unit	'*symb*'	→	'x_unit / *symb*'

See also: +, –, *, =

<

Type: Function

Description: Less Than Function: Tests whether one object is less than another object.

The function < returns a true test result (1) if the first argument is less than the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, < returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "less than" means numerically smaller (1 is less than 2). For real numbers, "less than" also means more negative (–2 is less than –1).

For strings, "less than" means alphabetically previous ("ABC" is less than "DEF"; "AAA" is less than "AAB"; "A" is less than "AA"). In general, characters are ordered according to their character codes. This means, for example, that "B" is less than "a", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent, and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↱<

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x < symb$'
'$symb$'	x	→	'$symb < x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 < symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit < symb$'
'$symb$'	x_unit	→	'$symb < x\_unit$'

See also: ≤, >, ≥, ==, ≠

=

Type: Function

Description: Equals Analytic Function: Returns an equation formed from the two arguments.

The equals sign equates two expressions such that the difference between the two expressions is zero.

In Symbolic Results mode, the result is an algebraic equation. In Numerical Results mode, the result is the difference of the two arguments because = acts equivalent to –. This allows expressions and equations to be used interchangeably as arguments for symbolic and numerical rootfinders.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, –, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ↱=

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 = \tilde{z}_2$
$\tilde{z}$	'*symb*'	→	'$\tilde{z} = symb$'
'*symb*'	$\tilde{z}$	→	'$symb = \tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 = symb_2$'
y_unit	x	→	$y\_unit_1 = x$
y_unit	*x_unit*	→	$y\_unit_1 = x\_unit$
'*symb*'	*x_unit*	→	'$symb = x\_unit$'
x_unit	'*symb*'	→	'$x\_unit = symb$'

See also: DEFINE, EVAL, –

==

Type: Function

Description: Logical Equality Function: Tests if two objects are equal.

The function == returns a true result (1) if the two objects are the same type and have the same value, or a false result (0) otherwise. Lists and programs are considered to have the same values if the objects they contain are identical.

If one object is algebraic (or a name), and the other is a number (real or complex) or an algebraic, == returns a symbolic comparison expression that can be evaluated to return a test result.

Note that == is used for comparisons, while = separates two sides of an equation.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: CAT ==

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	0/1
(x,0)	x	→	0/1
x	(x,0)	→	0/1
z	'symb'	→	'z == symb'
'symb'	z	→	'symb == z'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ == $symb_2$'

See also: SAME, TYPE, <, ≤, >, ≥, ≠

>

Type: Function

Description: Greater Than Function: Tests whether one object is greater than another object.

The function > returns a true test result (1) if the first argument is greater than the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, > returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than" means numerically greater (2 is greater than 1). For real numbers, "greater than" also means less negative (–1 is greater than –2).

For strings, "greater than" means alphabetically subsequent ("DEF" is greater than "ABC"; "AAB" is greater than "AAA"; "AA" is greater than "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↱ >

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x > symb$'
'$symb$'	x	→	'$symb > x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 > symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit > symb$'
'$symb$'	x_unit	→	'$symb > x\_unit$'

See also: <, ≤, ≥, ==, ≠

▶

Type: Command

Description: Store Command: Stores an object into a specified variable.

To create a backup object, store the *obj* into the desired backup location (identified as $:n_{port}:name_{backup}$). ▶ will not overwrite an existing backup object.

To replace an element of an array or list, use STO. Also use STO to store a graphic object into PICT or a library or backup object into a port.

Access: CAT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	*obj*
obj	$:n_{port}:name_{backup}$	→	*obj*

See also: DEFINE, RCL, →, STO

Command Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Symbols

E

F

G

H

T

U

V

W

X

Y

Z

Symbols