HP 49G Advanced Users Guide

Volume 1

Part C

Other Commands: G to P

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

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}	$\rightarrow$	z_{get}
$[[\ matrix\ ]]$	$\{\ n_{\text{row}},\ m_{\text{column}}\ \}$	$\rightarrow$	z_{get}
$'name_{\text{matrix}}'$	n_{position}	$\rightarrow$	z_{get}
$'name_{\text{matrix}}'$	$\{\ n_{\text{row}},\ m_{\text{column}}\ \}$	$\rightarrow$	z_{get}
$[\ vector\ ]$	n_{position}	$\rightarrow$	z_{get}
$[\ vector\ ]$	$\{n_{\text{position}}\ \}$	$\rightarrow$	z_{get}
$'name_{\text{vector}}'$	n_{position}	$\rightarrow$	z_{get}
$'name_{\text{vector}}'$	$\{n_{\text{position}}\ \}$	$\rightarrow$	z_{get}
$\{\ list\ \}$	n_{position}	$\rightarrow$	obj_{get}
$\{\ list\ \}$	$\{n_{\text{position}}\ \}$	$\rightarrow$	obj_{get}
$'name_{\text{list}}'$	n_{position}	$\rightarrow$	obj_{get}
$'name_{\text{list}}'$	$\{n_{\text{position}}\ \}$	$\rightarrow$	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]]	$n_{\text{position1}}$	→	*[[matrix]]*	$n_{\text{position2}}$	z_{get}
[[matrix]]	$\{ n_{\text{row}}, m_{\text{column}} \}_1$	→	*[[matrix]]*	$\{ n_{\text{row}}, m_{\text{column}} \}_2$	z_{get}
$'name_{\text{matrix}}'$	$n_{\text{position1}}$	→	$'name_{\text{matrix}}'$	$n_{\text{position2}}$	z_{get}
$'name_{\text{matrix}}'$	$\{ n_{\text{row}}, m_{\text{column}} \}_1$	→	$'name_{\text{matrix}}'$	$\{ n_{\text{row}}, m_{\text{column}} \}_2$	z_{get}
[vector]	n_{position}	→	*[vector]*	$n_{\text{position2}}$	z_{get}
[vector]	$\{ n_{\text{position1}} \}$	→	*[vector]*	$\{ n_{\text{position2}} \}$	z_{get}
$'name_{\text{vector}}'$	$n_{\text{position1}}$	→	$'name_{\text{vector}}$	$n_{\text{position2}}$	z_{get}
$'name_{\text{vector}}'$	$\{ n_{\text{position1}} \}$	→	$'name_{\text{vector}}$	$\{ n_{\text{position2}} \}$	z_{get}
{ *list* }	$n_{\text{position1}}$	→	{ *list* }	$n_{\text{position2}}$	obj_{get}
{ *list* }	$\{ n_{\text{position1}} \}$	→	{ *list* }	$\{ n_{\text{position2}} \}$	obj_{get}
$'name_{\text{list}}'$	$n_{\text{position1}}$	→	$'name_{\text{list}}'$	$n_{\text{position2}}$	obj_{get}
$'name_{\text{list}}'$	$\{ n_{\text{position1}} \}$	→	$'name_{\text{list}}'$	$\{ n_{\text{position2}} \}$	obj_{get}

L = level; A = argument; I = item

See also: GET, PUT, PUTI

GOR

Type: Command

Description: Graphics OR Command: Superimposes $grob_1$ onto $grob_{\text{target}}$ or *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{\text{target}}$ 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 $grob_{\text{result}}$ 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 $grob_{\text{target}}$ 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
$grob_{\text{target}}$	{ #*n* #*m* }	$grob_1$	→	$grob_{\text{result}}$
$grob_{\text{target}}$	(*x*, *y*)	$grob_1$	→	$grob_{\text{result}}$
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*$_{\text{identity}}$ *]]*
[[matrix]]	→	*[[matrix*$_{\text{identity}}$ *]]*
'*name*'	→	*[[matrix*$_{\text{identity}}$ *]]*

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:

$$X_n = \frac{1}{N}\sum_{k=0}^{N-1} Y_k e^{\frac{2\pi ikn}{N}}, i = \sqrt{-1}$$

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:

$$X_{mn} = \frac{1}{MN}\sum_{k=0}^{M-1}\sum_{l=0}^{N-1} Y_{kl} e^{\frac{2\pi ikm}{M}} e^{\frac{2\pi iln}{N}}, i = \sqrt{-1}$$

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: ⓵ (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: ⓵ (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*'	→	$x_{\text{increment}}$

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*'	$\rightarrow$	
	{ *global* }	$\rightarrow$	
	{ *global* x_{start} x_{end} }	$\rightarrow$	
	{ x_{start} x_{end} }	$\rightarrow$	
x_{start}	x_{end}	$\rightarrow$	

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
$\{s_1\ s_2 \ldots s_n\}$	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. $\#n_{\text{function}}$ 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
$\#n_{\text{function}}$	→	

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	$m \ln(x) + \mathrm{b}$
EXPFIT	be^{mx}
PWRFIT	bx^{m}

Access: CAT ΔLINE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'$symb_{\mathrm{formula}}$'

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
$obj_1 \ldots obj_n$	*n*	→	{ $obj_1, \ldots, obj_n$ }

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.

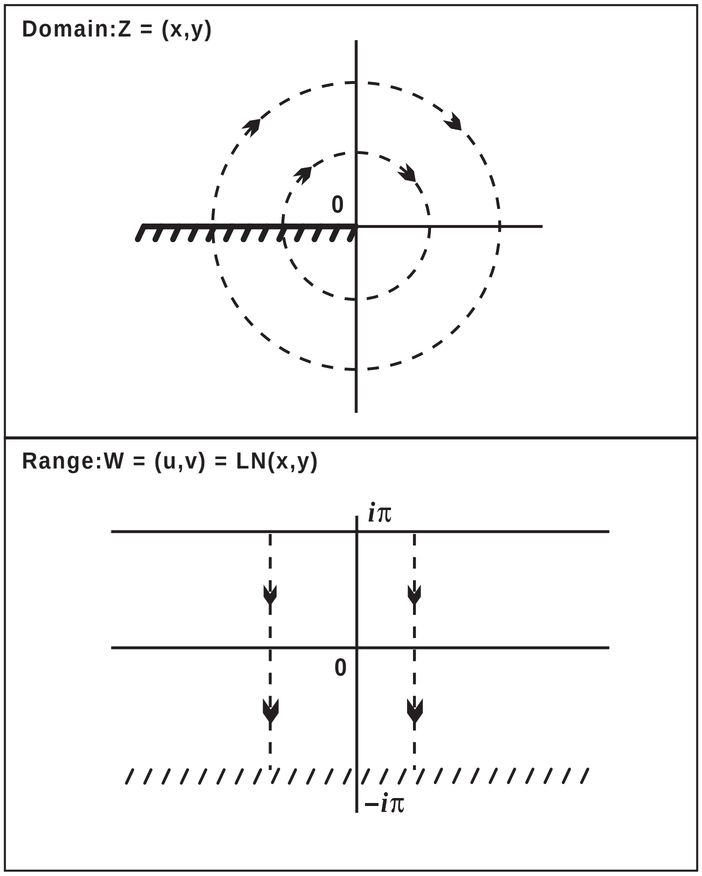
The points in this restricted domain of EXP are called the *principal values* of the inverse relation. LN in its entirety is called the *principal branch* of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the *branch cuts* of LN.

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**
	$\rightarrow$	*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 '$symb_{\text{pat}}$'. An optional condition, '$symb_{\text{cond}}$', 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 '$symb_{\text{pat}}$' and replacement '$symb_{\text{repl}}$' 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
	→	$x_{\max}$
	→	$[\, x_{\max 1}\ x_{\max 2} \ldots x_{\max m} \,]$

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}
	→	$[x_{\text{mean}1}, x_{\text{mean}2}, \ldots, x_{\text{mean}m}]$

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 ($name_{\text{definition}}$). 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}	→	
{ $list_{\text{definition}}$ }	→	
'$name_{\text{definition}}$'	→	
obj	→	

See also: RCLMENU, TMENU

MIN

Type: Function

Description: Minimum Function: Returns the lesser of two inputs.

Access: ⓵ (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
	→	$x_{\min}$
	→	$\{ x_{\min 1}\ x_{\min 2} \ldots x_{\min m} \}$

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: $x \bmod y = x - y \text{ floor } (x/y)$

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
z	→	$-z$
$\#n_1$	→	$\#n_2$
[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}), *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}	$\rightarrow$	

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:

$\{ x_{\text{left}}, x_{\text{right}}, y_{\text{near}}, y_{\text{far}}, z_{\text{low}}, z_{\text{high}}, x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}}, x_{\text{step}}, y_{\text{step}} \}$

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).
- $x_{\text{eye}}, y_{\text{eye}}$, 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:

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

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

- $(x_{\text{min}}, y_{\text{min}})$ is not used.
- $(x_{\text{max}}, y_{\text{max}})$ 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
$[\,array\,]_{\text{roots}}$	$\rightarrow$	$[\,array\,]_{\text{coefficients}}$

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:

$\{\ x_{\text{left}}\ x_{\text{right}}\ y_{\text{near}}\ y_{\text{far}}\ z_{\text{low}}\ z_{\text{high}}\ x_{\text{min}}\ x_{\text{max}}\ y_{\text{min}}\ y_{\text{max}}\ x_{\text{eye}}\ y_{\text{eye}}\ z_{\text{eye}}\ x_{\text{step}}\ y_{\text{step}}\ \}$

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:

$$\{\ (x_{\text{min}},\ y_{\text{min}})\ (x_{\text{max}},\ y_{\text{max}})\ \mathit{indep\ res\ axes\ ptype\ depend}\ \}$$

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

- $(x_{\text{min}}, y_{\text{min}})$ is not used.
- $(x_{\text{max}}, y_{\text{max}})$ 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 *k*th coordinate value in column n_1, x_{kn_2} is the *k*th 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
$(x_{\min}, y_{\min})$	$(x_{\max}, y_{\max})$	$\rightarrow$	
$\#n_{\text{width}}$	$\#m_{\text{height}}$	$\rightarrow$	

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*]$_{\text{coefficients}}$	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_i

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:

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

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 $y_{\text{dependent}}$, based on the independent-variable value $x_{\text{independent}}$, the currently selected statistical model, and the current regression coefficients in the reserved variable ΣPAR.

PREDV is the same as PREDY. See PREDY.

Access: CAT PREDV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$x_{\text{independent}}$	$\rightarrow$	$y_{\text{dependent}}$

PREDX

Type: Command

Description: Predicted x-Value Command: Returns the predicted independent-variable value $x_{\text{independent}}$, based on the dependent-variable value $y_{\text{dependent}}$, 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
$y_{\text{dependent}}$	$\rightarrow$	$x_{\text{independent}}$

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 $y_{\text{dependent}}$, based on the independent-variable value $x_{\text{independent}}$, 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
$x_{\text{independent}}$	→	$y_{\text{dependent}}$

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
$[\,array\,]_{\text{coefficients}}$	$\rightarrow$	$[\,array\,]_{\text{roots}}$

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}
	→	[x_{psdev1} x_{psdev2} ... 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} :$name_{\text{backup}}$	→	
: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}	$\to$	$[[\ matrix\ ]]_2$
$[[\ matrix\ ]]_1$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}$	z_{put}	$\to$	$[[\ matrix\ ]]_2$
$'name_{\text{matrix}}'$	n_{position}	z_{put}	$\to$	
$'name_{\text{matrix}}'$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}$	z_{put}	$\to$	
$[\ vector\ ]_1$	n_{position}	z_{put}	$\to$	$[\ vector\ ]_2$
$[\ vector\ ]_1$	$\{\ n_{\text{position}}\ \}$	z_{put}	$\to$	$[\ vector\ ]_2$
$'name_{\text{vector}}'$	n_{position}	z_{put}	$\to$	
$'name_{\text{vector}}'$	$\{\ n_{\text{position}}\ \}$	z_{put}	$\to$	
$\{\ list\ \}_1$	n_{position}	obj_{put}	$\to$	$\{\ list\ \}_2$
$\{\ list\ \}_1$	$\{\ n_{\text{position}}\ \}$	obj_{put}	$\to$	$\{\ list\ \}_2$
$'name_{\text{list}}'$	n_{position}	obj_{put}	$\to$	
$'name_{\text{list}}'$	$\{\ n_{\text{position}}\ \}$	obj_{put}	$\to$	

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
$[[\ matrix\ ]]_1$	$n_{\text{position1}}$	z_{put}	→	$[[\ matrix\ ]]_2$	$n_{\text{position2}}$
$[[\ matrix\ ]]_1$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}_1$	z_{put}	→	$[[\ matrix\ ]]_2$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}_2$
$'name_{\text{matrix}}'$	$n_{\text{position1}}$	z_{put}	→	$'name_{\text{matrix}}'$	$n_{\text{position2}}$
$'name_{\text{matrix}}'$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}_1$	z_{put}	→	$'name_{\text{matrix}}'$	$\{\ n_{\text{row}}\ m_{\text{col}}\ \}_2$
$[\ vector\ ]_1$	$n_{\text{position1}}$	z_{put}	→	$[\ vector\ ]_2$	$n_{\text{position2}}$
$[\ vector\ ]_1$	$\{\ n_{\text{position1}}\ \}$	z_{put}	→	$[\ vector\ ]_2$	$\{\ n_{\text{position2}}\ \}$
$'name_{\text{vector}}'$	$n_{\text{position1}}$	z_{put}	→	$'name_{\text{vector}}'$	$n_{\text{position2}}$
$'name_{\text{vector}}'$	$\{\ n_{\text{position1}}\ \}$	z_{put}	→	$'name_{\text{vector}}'$	$\{\ n_{\text{position2}}\ \}$
$\{\ list\ \}_1$	$n_{\text{position1}}$	obj_{put}	→	$\{\ list\ \}_2$	$n_{\text{position2}}$
$\{\ list\ \}_1$	$\{\ n_{\text{position1}}\ \}$	obj_{put}	→	$\{\ list\ \}_2$	$\{\ n_{\text{position2}}\ \}$
$'name_{\text{list}}'$	$n_{\text{position1}}$	obj_{put}	→	$'name_{\text{list}}'$	$n_{\text{position2}}$
$'name_{\text{list}}'$	$\{\ n_{\text{position1}}\ \}$	obj_{put}	→	$'name_{\text{list}}'$	$\{\ n_{\text{position2}}\ \}$

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 (Σ*DAT*).

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
	→	$x_{\text{pvariance}}$
	→	$[\, x_{\text{pvariance1}}, \ldots, x_{\text{pvariancem}} \,]$

See also: MEAN, PCOV, PSDEV, SDEV, VAR

PVARS

Type: Command

Description: Port-Variables Command: Returns a list of the backup objects ($:n_{\text{port}}:name$) and the library objects ($:n_{\text{port}}:n_{\text{library}}$) 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 $n_{\text{port}} = 0$, 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}	→	$\{ :n_{\text{port}} :name_{\text{backup}} \ldots \}$	*memory*
n_{port}	→	$\{ :n_{\text{port}} :n_{\text{library}} \ldots \}$	*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: ⓵ PRG PICT PX→C

Input/Output:

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

See also: C→PX