HP 49G Advanced Users Guide

Volume 1

Part B

Other Commands: A to F

Introduction

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

For each operation, the following details are provided:

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

Description: A description of the operation.

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

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

See also: Related functions or commands

A to B

ABS

Type: Function

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

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

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

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

Access: ↰ ABS

Input/Output:

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

See also: NEG, SIGN

ACK

Type: Command

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

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

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

Access: ↱ TIME TOOLS ALRM ACK

See also: ACKALL

ACKALL

Type: Command

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

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

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

Access: ↱ TIME TOOLS ALRM ACK

See also: ACK

ACOS

Type: Analytic Function

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

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

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

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

s1*ACOS(Z)+2*π*n1

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

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

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

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

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

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

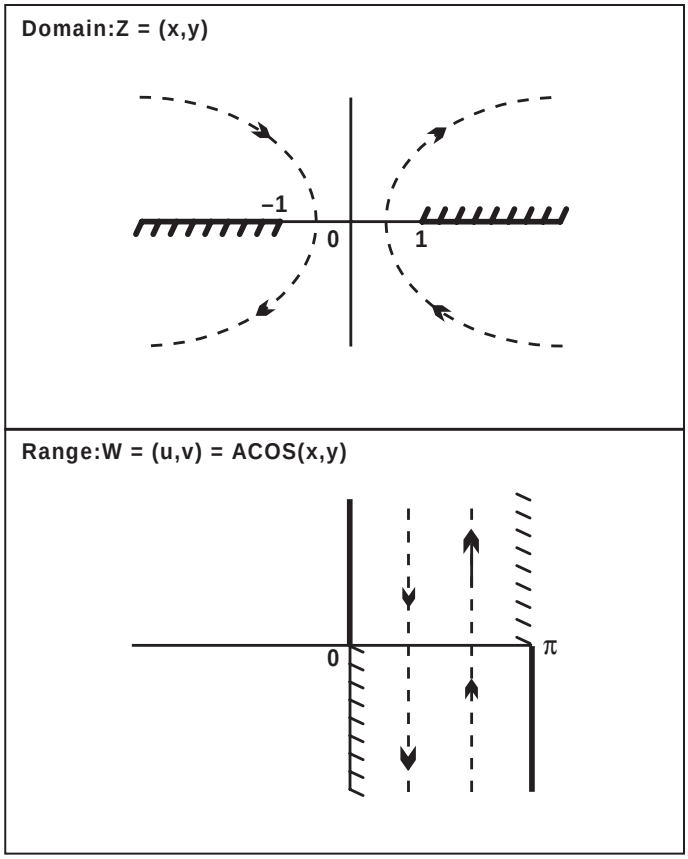
View these graphs with domain and range reversed to see how the domain of COS is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). COS sends this domain onto the whole complex plane in the range W = (*u*, *v*) = COS(*x*,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

Access: ↰ ACOS

Input/Output:

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

See also: ASIN, ATAN, COS, ISOL.



ACOSH

Type: Analytic Function

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

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

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

s1*ACOSH(Z)+2*π*i*n1

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

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

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

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

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

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

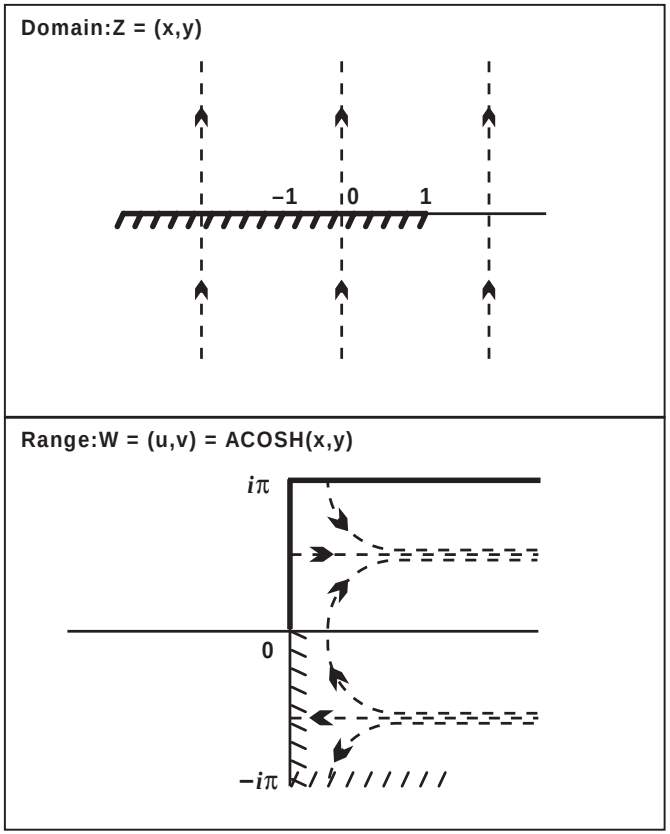
View these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse *function* possible. Consider the horizontal half-band in the lower graph as the restricted domain Z = (*x*, *y*). COSH sends this domain onto the whole complex plane in the range W = (*u*,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

Access: → TRIG HYPERBOLIC ACOSH

Input/Output:

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

See also: ASINH, ATANH, COSH, ISOL



ADD

Type: Command

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

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

Access: CAT ADD

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ *list*$_1$ }	{ *list*$_2$ }	→	{ *list*$_{result}$ }
{ *list* }	*obj*$_{non\text{-}list}$	→	{ *list*$_{result}$ }
obj$_{non\text{-}list}$	{ *list* }	→	{ *list*$_{result}$ }

See also: ΔLIST, ΠLIST, ΣLIST

ADDTOREAL

Type: Command

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

Access: CAT ADDTOREAL

Input/Output:

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

ALOG

Type: Analytic Function

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

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

Access: ↰ 10^x

Input/Output:

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

See also: EXP, LN, LOG

AMORT

Type: Command

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

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

Access: ↰ FINANCE AMOR

Input/Output:

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

See also: TVM, TVMBEG, TVMEND, TVMROOT

AND

Type: Function

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

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

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

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

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

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

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

Access: ↱ BASE LOGIC AND

test and

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	$\#n_2$	→	$\#n_3$
"string$_1$*"*	*"string*$_2$*"*	→	*"string*$_3$*"*
T/F_1	T/F_2	→	*0/1*
T/F	*'symb'*	→	*'T/F AND symb'*
'symb'	*T/F*	→	*'symb AND T/F'*
'symb$_1$*'*	*'symb*$_2$*'*	→	*'symb*$_1$ *AND symb*$_2$*'*

See also: NOT, OR , XOR

ANIMATE

Type: Command

Description: Animate Command: Displays graphic objects in sequence.

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

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

Access: ↰ PRG GROB ANIMATE

Input/Output:

$L_{n+1}.../A_1$	L_1/A_{n+1}		L_1/I_1
grob$_n$*...grob*$_1$	n_{grobs}	→	*same stack*
grob$_n$*...grob*$_1$	{ *n* {#*X*#*Y* } *delay* *rep* }	→	*same stack*

L = level, A = argument, I = item

ANS

Type: Command

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

Access: ↩ ANS

Input/Output:

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

See also: LASTARG

APPLY

Type: Function

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

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

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

Access: CAT APPLY

Input/Output:

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

See also: QUOTE, |

ARC

Type: Command

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

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

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

Access: ↰ PRG PICT ARC

Input/Output:

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

See also: BOX, LINE, TLINE

ARCHIVE

Type: Command

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

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

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

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

Access: ↰ PRG MEMORY ARCHIVE

Input/Output:

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

See also: RESTORE

ARG

Type: Function

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

The polar angle θ is equal to:

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

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

Access: ↱ ARG

Input/Output:

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

ARIT

Type: Command

Description: Displays a menu of arithmetic commands.

Access: CAT ARIT

Input: None

Output: None

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

ARRY→

Type: Command

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

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

Access: CAT ARRY→

Input/Output:

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

L = level; I = item

Flags: None

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

→ARRY

Type: Command

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

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

Access: CAT →ARRY

Input/Output:

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

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

ASIN

Type: Analytic Function

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

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

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

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

```
ASIN(Z)*(-1)^n1+π*n1
```

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

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

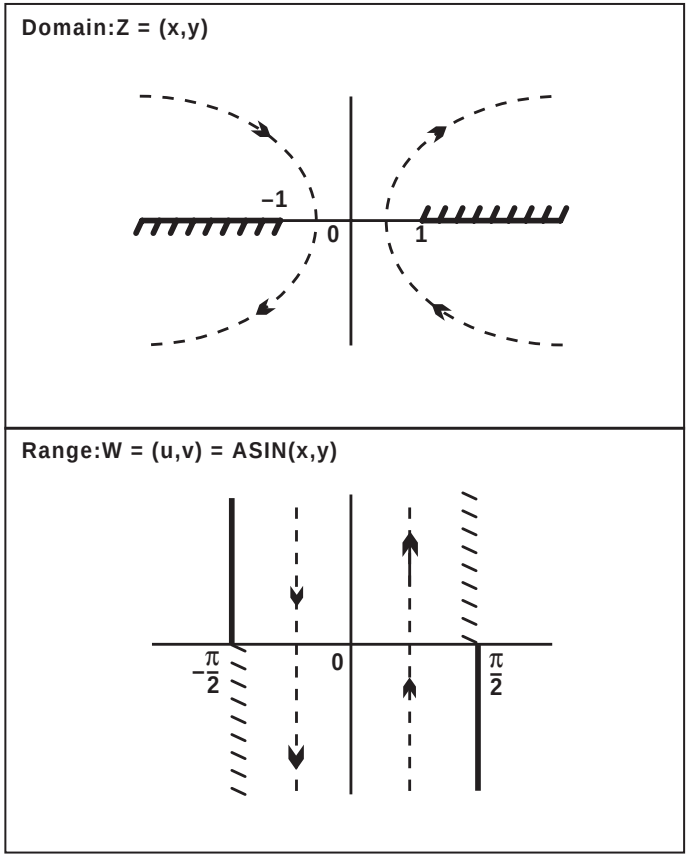
The points in this restricted domain of SIN are called the *principal values* of the inverse relation. ASIN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASIN to the boundary of the restricted domain of SIN form the *branch cuts* of ASIN.

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

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

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

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



Access: ↰ ASIN

Input/Output:

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

See also: ACOS, ATAN, ISOL, SIN

ASINH

Type: Analytic Function

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

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

```
ASINH(Z)*(-1)^n1+π*i*n1
```

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

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

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

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

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

Access: ↱ TRIG HYPERBOLIC ASINH

Input/Output:

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

See also: ACOSH, ATANH, ISOL, SINH

ASN

Type: Command

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

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

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

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

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

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

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

Access: CAT ASN

Input/Output:

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

Output: None

See also: DELKEYS, RCLKEYS, STOKEYS

ASR

Type: Command

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

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

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

Access: ↱ BASE BIT ASR

Input/Output:

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

See also: SL, SLB, SR, SRB

ATAN

Type: Analytic Function

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

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

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

ATAN(Z)+π*n1

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

- each argument is sent to a distinct result, and

- each possible result is achieved.

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

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

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

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

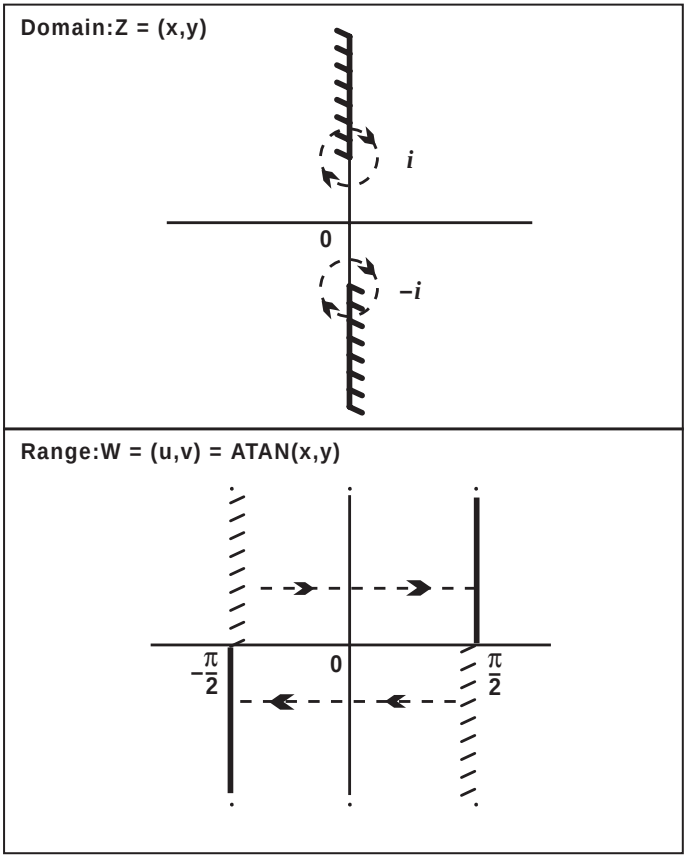
View these graphs with domain and range reversed to see how the domain of TAN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (x, y). TAN sends this domain onto the whole complex plane in the range W = (u, v) = TAN(x, y)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

Access: ↰ ATAN

Input/Output:

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

See also: ACOS, ASIN, ISOL, TAN



ATANH

Type: Analytic Function

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

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

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

ATANH(Z)+π*i*n1

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

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

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

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

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

Access: ↱ TRIG HYPERBOLIC ATAN

Input/Output:

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

See also: ACOSH, ASINH, ISOL, TANH

ATICK

Type: Command

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

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

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

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

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

Access: CAT ATICK

Input/Output:

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

See also: AXES, DRAX

ATTACH

Type: Command

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

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

Some libraries require you to ATTACH them.

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

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

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

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

Access: CAT ATTACH

Input/Output:

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

See also: DETACH, LIBS

AUTO

Type: Command

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

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

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

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

AUTO does not affect 3D plots.

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

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

Access: CAT AUTO

Input: None

Output: None

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

AXES

Type: Command

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

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

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

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

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

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

Access: CAT AXES.

Input/Output:

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

See also: ATICK, DRAW, DRAX, LABEL

BAR

Type: Command

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

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

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

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

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

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

Access: CAT BAR

Input: None

Output: None

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

BARPLOT

Type: Command

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

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

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

Access: CAT BARPLOT

Input: None

Output: A bar chart based on ΣDAT.

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

BASE

Type: Command

Description: Displays a menu of basic algebra commands.

Access: CAT BASE

Input: None

Output: None

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

BAUD

Type: Command

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

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

Access: CAT BAUD

Input/Output:

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

See also: CKSM, PARITY, TRANSIO

BEEP

Type: Command

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

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

Access: ↰ PRG OUT BEEP

Input/Output:

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

See also: HALT, INPUT, PROMPT, WAIT

BESTFIT

Type: Command

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

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

Access: CAT BESTFIT

Input: None

Output: None

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

BIN

Type: Command

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

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

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

Access: (CAT) BIN

Input: None

Output: None

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

BINS

Type: Command

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

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

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

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

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

Access: (CAT) BINS

Input/Output:

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

L = level; A = argument; I = item

See also: BARPLOT, XCOL

BLANK

Type: Command

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

Access: ⮌ PRG GROB BLANK

Input/Output:

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

See also: →GROB, LCD→

BOX

Type: Command Operation

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

Access: ⮌ PRG PICT BOX

Input/Output:

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

See also: ARC, LINE, TLINE

BUFLEN

Type: Command

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

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

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

Access: (CAT) BUFLEN.

Input/Output:

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

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

BYTES

Type: Command

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

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

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

Access: (↰) (PRG) MEMORY BYTES

Input/Output:

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

See also: MEM

B→R

Type: Command

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

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

Access: ↱ BASE B→R

Input/Output:

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

See also: R→B

C to D

CASE

Type: Command

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

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

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

The CASE … END structure has this syntax:

CASE
test-clause$_1$ THEN *true-clause*$_1$ END
test-clause$_2$ THEN *true-clause*$_2$ END
.
.
.
test-clause$_n$ THEN *true-clause*$_n$ END
default-clause (*optional*)
END

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

Access: ↰ PRG BRCH CASE

Input/Output:

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

See also: END, IF, IFERR, THEN

CEIL

Type: Function

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

Access: ↰ MTH REAL CEIL

Input/Output:

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

See also: FLOOR, IP, RND, TRNC

CENTR

Type: Command

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

If the argument is a real number x, CENTR makes the point (x,0) the plot center.

Access: CAT CENTR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
(x, y)	→	
x	→	

See also: SCALE

CF

Type: Command

Description: Clear Flag Command: Clears the specified user or system flag.

User flags are numbered 1 through 128. System flags are numbered –1 through –128. See the *Pocket Guide* for a listing of HP 49 system flags and their flag numbers.

Access: ↰ PRG TEST CF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flagnumber}}$	→	

See also: FC?, FC?C, FS?, FS?C, SF

%CH

Type: Function

Description: Percent Change Function: Returns the percent change from x to y as a percentage of x.

If both arguments are unit objects, the units must be consistent with each other. The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

For more information on using temperature units with arithmetic functions, refer to the keyword entry of +.

Access: ↰ MTH REAL %CH

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$100(y-x)/x$
x	'*symb*'	→	'%CH(x,*symb*)'
'*symb*'	x	→	'%CH(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%CH($symb_1$, $symb_2$)'
x_unit	y_unit	→	$100(y\_unit - x\_unit)/x\_unit$
x_unit	'*symb*'	→	'%CH(x_unit,*symb*)'
'*symb*'	x_unit	→	'%CH(*symb*,x_unit)'

See also: %, %T

CHOOSE

Type: Command

Description: Create User-Defined Choose Box Command: Creates a user-defined choose box.

CHOOSE creates a standard single-choice choose box based on the following specifications:

Variable	Function
"*prompt*"	A message that appears at the top of choose box. If "*prompt*" is an empty string (""), no message is displayed.
$\{c_1 \ldots c_n\}$	Definitions that appear within the choose box. A choice definition (c_x) can have two formats. • *obj*, any object • { $obj_{display}$ obj_{result} }, the object to be displayed followed by the result returned to the stack if that object is selected.
n_{pos}	The position number of an item definition. This item is highlighted when the choose box appears. If $n_{pos} = 0$, no item is highlighted, and the choose box can be used to view items only.

If you choose an item from the choose box and press OK, CHOOSE returns the *result* (or the object itself if no result is specified) to level 2 and 1 to level 1. If you press CANCEL, CHOOSE returns 0. Also, if $n_{pos} = 0$, CHOOSE returns 0.

Access: ↰ PRG IN CHOOS

Input/Output:

L_3/A_1	L_2/A_2	L_1/A_3		L_2/I_1	L_1/I_2
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→	*obj or result*	"*1*"
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→		"*0*"

L = level; A = argument; I = item

See also: INFORM, NOVAL

CHR

Type: Command

Description: Character Command: Returns a string representing the character corresponding to the character code *n*.

The character codes are an extension of ISO 8859/1. Codes 128 through 160 are unique to the HP 49.

The default character ■ is supplied for all character codes that are *not* part of the normal HP 49 display character set.

Character code 0 is used for the special purpose of marking the end of the command line. Attempting to edit a string containing this character causes the error Can't Edit CHR(0).

You can use the CHARS application to find the character code for any character used by the HP 49. See "Entering Special Characters" in chapter 2 of the *HP 49 User's Guide*.

Access: ↰ PRG TYPE CHR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*"string"*

See also: NUM, POS, REPL, SIZE, SUB

CKSM

Type: Command

Description: Checksum Command: Specifies the error-detection scheme.

Legal values for $n_{checksum}$ are as follows:

- 1-digit arithmetic checksum.
- 2-digit arithmetic checksum.
- 3-digit cyclic redundancy check (default).

The CKSM specified is the error-detection scheme that will be requested by KGET, PKT, or SEND. If the sender and receiver disagree about the request, however, then a 1-digit arithmetic checksum will be used.

Access: CAT CKSM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{checksum}$	→	

See also: BAUD, PARITY, TRANSIO

CLEAR

Type: Command

Description: Clear Command: Removes all objects from the stack or history.

To recover a cleared stack or history, press ↱ UNDO before executing any other operation. There is no programmable command to recover the stack or history.

Access: ↱ CLEAR

Input/Output:

$Level_n$/Argument 1 ... Level 1/$Argument_n$		$Level_n$/Item 1 ... Level 1/$Item_n$
$obj_n ... obj_1$	→	

See also: CLVAR, PURGE

CLKADJ

Type: Command

Description: Adjust System Clock Command: Adjusts the system time by x clock ticks, where 8192 clock ticks equal 1 second.

If x is positive, x clock ticks are added to the system time. If x is negative, x clock ticks are subtracted from the system time.

Access: ↱ TIME TOOLS CLKADJ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	

See also: →TIME

CLLCD

Type: Command

Description: Clear LCD Command: Clears (blanks) the stack display.

The menu labels continue to be displayed after execution of CLLCD.

When executed from a program, the blank display persists only until the keyboard is ready for input. To cause the blank display to persist until a key is pressed, execute FREEZE after executing CLLCD. (When executed from the keyboard, CLLCD *automatically* freezes the display.)

Access: ↰ (PRG) OUT CLLCD

Input: None

Output: None

See also: DISP, FREEZE

CLOSEIO

Type: Command

Description: Close I/O Port Command: Closes the serial port, and clears the input buffer and any error messages for KERRM.

When the HP 49 turns off, it automatically closes the serial, but does not clear KERRM. Therefore, CLOSEIO is not needed to close the port, but can conserve power without turning off the calculator.

Executing HP 49 Kermit protocol commands automatically clears the input buffer; however, executing non-Kermit commands (such as SRECV and XMIT) does not.

CLOSEIO also clears error messages from KERRM. This can be useful when debugging.

Access: (CAT) CLOSEIO

Input: None

Output: None

See also: BUFLEN, OPENIO

CLΣ

Type: Command

Description: Purges the current statistics matrix (reserved variable ΣDAT).

Access: CAT CLΣ

Input: None

Output: None

See also: RCLΣ, STOΣ, Σ+, Σ–

CLVAR

Type: Command

Description: Clear Variables Command: Purges all variables and empty subdirectories in the current directory.

Access: CAT CLVAR

Input: None

Output: None

See also: PGDIR, PURGE

CMPLX

Type: Command

Description: Displays a menu of commands pertaining to complex numbers.

Access: CAT CMPLX

Input: None

Output: None

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

CNRM

Type: Command

Description: Column Norm Command: Returns the column norm (one-norm) of the array argument.

The column norm of a matrix is the maximum (over all columns) of the sum of the absolute values of all elements in each column. For a vector, the column norm is the sum of the absolute values of the vector elements. For complex arrays, the absolute value of a given element (x, y) is $\sqrt{x^2 + y^2}$.

Access: ↰ MATRICES OPERATIONS CNRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[array]	→	$x_{\text{columnnorm}}$

See also: CROSS, DET, DOT, RNRM

→COL

Type: Command

Description: Transforms a matrix into a series of column vectors and returns the vectors and a column count, or transforms a vector into its elements and returns the elements and an element count.

→COL introduces no rounding error.

Access: ↰ MTH MATRIX COL →COL

↰ MATRICES CREATE COLUMN →COL

Input/Output:

Level 1/Argument 1		Level$_{n+1}$/Item 1 ...	Level 2/Item 2	Level 1/Item 3
[[matrix]]	→	*[vector]*$_{\text{coll}}$	*[vector]*$_{\text{coln}}$	n_{colcount}
[vector]	→	$element_1$	$element_n$	$n_{\text{elementcount}}$

See also: COL→, →ROW, ROW→

COL→

Type: Command

Description: Transforms a series of column vectors and a column count into a matrix containing those columns, or transforms a sequence of numbers and an element count into a vector with those numbers as elements.

All vectors must have the same length. The column or element count is rounded to the nearest integer.

Access: ↰ MTH MATRIX COL COL→

↰ MATRICES CREATE COLUMN COL→

Input/Output:

L_{n+1}/A_1 ...	L_2/A_2	L_1/A_{n+1}		Level 1/Item 1
$[\ vector\ ]_{col1}$	$[\ vector\ ]_{coln}$	$n_{colcount}$	→	$[[\ matrix\ ]]$
$element_1$	$element_n$	$n_{elementcount}$	→	$[\ vector\ ]$

L = level; A = argument; I = item

See also: →COL, →ROW, ROW→

COL–

Type: Command

Description: Delete Column Command: Deletes column *n* of a matrix (or element *n* of a vector), and returns the modified matrix (or vector) and the deleted column (or element).

n is rounded to the nearest integer.

Access: ↰ MTH MATRIX COL COL–

↰ MATRICES CREATE COLUMN COL–

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 2/Item 1	Level 1/Item 2
$[[\ matrix\ ]]_1$	n_{column}	→	$[[\ matrix\ ]]_2$	$[\ vector\ ]_{column}$
$[\ vector\ ]_1$	$n_{element}$	→	$[\ vector\ ]_2$	$element_n$

See also: COL+, CSWP, ROW+, ROW–

COL+

Type: Command

Description: Insert Column Command: Inserts an array (vector or matrix) into a matrix (or one or more elements into a vector) at the position indicated by n_{index}, and returns the modified array.

The inserted array must have the same number of rows as the target array.

n_{index} is rounded to the nearest integer. The original array is redimensioned to include the new columns or elements, and the elements at and to the right of the insertion point are shifted to the right.

Access: ↰ MTH MATRIX COL COL+

↰ MATRICES CREATE COLUMN COL+

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$[[\ matrix\ ]]_2$	n_{index}	→	$[[\ matrix\ ]]_3$
$[[\ matrix\ ]]_1$	$[\ vector\ ]_{column}$	n_{index}	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{element}$	n_{index}	→	$[\ vector\ ]_2$

See also: COL–, CSWP, ROW+, ROW–

COLCT

Type: Command

Description: Factorizes a polynomial or an integer. Identical to FACTOR (see volume 1, CAS Commands).

Access: CAT COLCT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$'symb_1'$	→	$'symb_2'$
x	→	x
(x, y)	→	(x, y)

See also: EXPAN, FACTOR, ISOL, QUAD, SHOW

COLΣ

Type: Command

Description: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable ΣDAT).

COLΣ combines the functionality of XCOL and YCOL. The independent-variable column number x_{xcol} is stored as the first parameter in the reserved variable ΣPAR (the default is 1). The dependent-variable column number x_{ycol} is stored as the second parameter in the reserved variable ΣPAR (the default is 2).

COLΣ accepts and stores noninteger values, but subsequent commands that use these two parameters in ΣPAR will cause errors.

Access: CAT COLΣ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{col}	y_{col}	→	

See also: BARPLOT, BESTFIT, CORR, COV, EXPFIT, HISTPLOT, LINFIT, LOGFIT, LR, PREDX, PREDY, PWRFIT, SCATRPLOT, XCOL, YCOL

COMB

Type: Function

Description: Combinations Function: Returns the number of possible combinations of *n* items taken *m* at a time.

The following formula is used:

$$C_{n,m} = \frac{n!}{m! \cdot (n-m)!}$$

The arguments *n* and *m* must each be less than 10^{12}. If $n<m$, zero is returned.

Access: ↰ MTH PROBABILITY COMB

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*m*	→	$C_{n;m}$
'$symb_n$'	*m*	→	'$COMB(symb_n,m)$'
n	'$symb_m$'	→	'$COMB(n, symb_m)$'

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_n$'	'$symb_m$'	→	'$COMB(symb_n,symb_m)$'

See also: PERM, !

CON

Type: Command

Description: Constant Array Command: Returns a constant array, defined as an array whose elements all have the same value.

The constant value is a real or complex number taken from argument 2/level 1. The resulting array is either a new array, or an existing array with its elements replaced by the constant, depending on the object in argument 1/level 2.

- Creating a new array: If argument 1/level 2 contains a list of one or two integers, CON returns a new array. If the list contains a single integer $n_{columns}$, CON returns a constant vector with n elements. If the list contains two integers n_{rows} and $m_{columns}$, CON returns a constant matrix with n rows and m columns.
- Replacing the elements of an existing array: If argument 1/level 2 contains an array, CON returns an array of the same dimensions, with each element equal to the constant. If the constant is a complex number, the original array must also be complex.
  If argument 1/level 2 contains a name, the name must identify a variable that contains an array. In this case, the elements of the array are replaced by the constant. If the constant is a complex number, the original array must also be complex.

Access: ↰ (MTH) MATRIX MAKE CON

↰ (MATRICES) CREATE ON

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
{ $n_{columns}$ }	$z_{constant}$	→	[$vector_{constant}$]
{ n_{rows} $m_{columns}$ }	$z_{constant}$	→	[[$matrix_{constant}$]]
[R-*array*]	$x_{constant}$	→	[R-$array_{constant}$]
[C-*array*]	$z_{constant}$	→	[C-$array_{constant}$]
'*name*'	$z_{constant}$	→	

See also: IDN

COND

Type: Command

Description: Condition Number Command: Returns the 1-norm (column norm) condition number of a square matrix.

The condition number of a matrix is the product of the norm of the matrix and the norm of the inverse of the matrix. COND uses the 1-norm and computes the condition number of the matrix without computing the inverse of the matrix.

The condition number expresses the sensitivity of the problem of solving a system of linear equations having coefficients represented by the elements of the matrix (this includes inverting the matrix). That is, it indicates how much an error in the inputs may be magnified in the outputs of calculations using the matrix.

In many linear algebra computations, the base 10 logarithm of the condition number of the matrix is an estimate of the number of digits of precision that might be lost in computations using that matrix. A reasonable rule of thumb is that the number of digits of accuracy in the result is approximately MIN(12,15–$\log_{10}$(COND)).

Access: ↰ MTH MATRIX NORMALIZE COND

↰ MATRICES OPERATIONS COND

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[\ matrix\ ]]_{m\times n}$	→	$\varkappa_{conditionnumber}$

See also: SNRM, SRAD, TRACE

CONIC

Type: Command

Description: Conic Plot Type Command: Sets the plot type to CONIC.

When the plot type is CONIC, the DRAW command plots the current equation as a second-order polynomial of two real variables. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

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

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

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5, –3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5, 3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes, or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

The current equation is used to define a pair of functions of the independent variable. These functions are derived from the second-order Taylor's approximation to the current equation. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max}) (the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

Access: (CAT) CONIC

Input: None

Output: None

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

CONJ

Type: Function

Description: Conjugate Analytic Function: Conjugates a complex number or a complex array.

Conjugation is the negation (sign reversal) of the imaginary part of a complex number. For real numbers and real arrays, the conjugate is identical to the original argument.

Access: ↱ CMPLX CONJ

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*x*
(x, y)	→	*(x, –y)*
[R-array]	→	*[R-array]*
[C-array]$_1$	→	*[C-array]*$_2$
'symb'	→	*'CONJ(symb)'*

See also: ABS, IM, RE, SCONJ, SIGN

CONLIB

Type: Command

Description: Open Constants Library Command: Opens the Constants Library catalog.

Access: APPS CONSTANTS LIBRARY

Input: None

Output: None

See also: CONST

CONST

Type: Function

Description: Constant Value Command: Returns the value of a constant.

CONST returns the value of the specified constant. It chooses the unit type depending on flag –60: SI if clear, English if set, and uses the units depending on flag –61: units if clear, no units if set.

Access: CAT CONST

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	*x*

See also: CONLIB

CONT

Type: Command

Description: Continue Program Execution Command: Resumes execution of a halted program.

Since CONT is a command, it can be assigned to a key or to a custom menu.

Access: ↩ CONT

Input: None

Output: None

See also: HALT, KILL, PROMPT

CONVERT

Type: Command

Description: Convert Units Command: Converts a source unit object to the dimensions of a target unit.

The source and target units must be compatible. The number part x_2 of the target unit object is ignored.

Access: ↩ CONVERT UNITS TOOLS CONVERT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_1\_units_{source}$	$x_2\_units_{target}$	→	$x_3\_units_{target}$

See also: UBASE, UFACT, →UNIT, UVAL

CORR

Type: Command

Description: Correlation Command: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable Σ*DAT*).

The columns are specified by the first two elements in the reserved variable Σ*PAR*, set by XCOL and YCOL, respectively. If Σ*PAR* does not exist, CORR creates it and sets the elements to their default values (1 and 2).

The correlation is computed with the following formula:

$$\frac{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})(x_{in_2}-\bar{x}_{n_2})}{\sqrt{\sum_{i=1}^{n}(x_{in_1}-\bar{x}_{n_1})^2\sum_{i=1}^{n}(x_{in_2}-\bar{x}_{n_2})^2}}$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\bar{x}_{n_1}$ is the mean of the data in column n_1, $\bar{x}_{n_2}$ is the mean of the data in column n_2, and n is the number of data points.

Access: CAT CORR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{correlation}$

Flags: None

See also: COLΣ, COV, PREDX, PREDY, XCOL, YCOL

COS

Type: Analytic Function

Description: Cosine Analytic Function: Returns the cosine of the argument.

For real arguments, the current angle mode determines the number's interpretation as an angle, unless the angular units are specified.

For complex arguments, $\cos(x + iy) = \cos x \cosh y - i \sin x \sinh y$.

If the argument for COS is a unit object, then the specified angular unit overrides the angle mode to determine the result. Integration and differentiation, on the other hand, always observe the angle mode. Therefore, to correctly integrate or differentiate expressions containing COS with a unit object, the angle mode must be set to Radians (since this is a "neutral" mode).

Access: COS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	cos z
'*symb*'	→	'*COS(symb)*'
x_unit_{angular}	→	$cos\ (x\_unit_{\text{angular}})$

See also: ACOS, SIN, TAN

COSH

Type: Analytic Function

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

For complex arguments, $\cosh(x + iy) = \cosh x \cos y + i \sinh x \sin y$.

Access: ↱ TRIG HYPERBOLIC COSH

↰ MTH HYPERBOLIC COSH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	cosh z
'*symb*'	→	'*COSH(symb)*'

See also: ACOSH, SINH, TANH

COV

Type: Command

Description: Covariance Command: Returns the sample covariance of the independent and dependent data columns in the current statistics matrix (reserved variable Σ*DAT*).

The columns are specified by the first two elements in reserved variable Σ*PAR*, set by XCOL and YCOL respectively. If Σ*PAR* does not exist, COV creates it and sets the elements to their default values (1 and 2).

The covariance is calculated with the following formula:

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_{in_1}-\overline{x_{n_1}})(x_{in_2}-\overline{x_{n_2}})$$

where x_{in_1} is the ith coordinate value in column n_1, x_{in_2} is the ith coordinate value in the column n_2, $\overline{x_{n_1}}$ is the mean of the data in column n_1, $\overline{x_{n_2}}$ is the mean of the data in column n_2, and n is the number of data points.

Access: CAT COV

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{\text{covariance}}$

See also: COLΣ, CORR, PCOV, PREDX, PREDY, XCOL, YCOL

CR

Type: Command

Description: Carriage Right Command: Prints the contents, if any, of the printer buffer.

CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable *PRTPAR*.

Access: CAT CR

Input: None

Output: None

See also: DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

CRDIR

Type: Command

Description: Create Directory Command: Creates an empty subdirectory with the specified name in the current directory.

CRDIR does not change the current directory; evaluate the name of the new subdirectory to make it the current directory.

Access: ↰ PRG MEMORY DIRECTORY CRDIR

Input/Output:

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

See also: HOME, PATH, PGDIR, UPDIR

CROSS

Type: Command

Description: Cross Product Command: CROSS returns the cross product C = A × B of vectors A and B.

The arguments must be vectors having two or three elements, and need not have the same number of elements. (The HP 49 automatically converts a two-element argument [d_1, d_2] to a three-element argument [d_1, d_2, 0].)

Access: ↰ MTH VECTOR CROSS

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\lceil$ *vector* $\rceil_A$	$\lceil$ *vector* $\rceil_B$	→	$\lceil$ *vector* $\rceil_{A \times B}$

See also: CNRM, DET, DOT, RNRM

CSWP

Type: Command

Description: Column Swap Command: Swaps columns *i* and *j* of the argument matrix and returns the modified matrix, or swaps elements *i* and *j* of the argument vector and returns the modified vector.

Column numbers are rounded to the nearest integer. Vector arguments are treated as row vectors.

Access: ↰ MATRICES CREATE COLUMN CSWP

↰ MTH MATRIX COL CSWP

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
$[[\ matrix\ ]]_1$	$n_{columni}$	$n_{columnj}$	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{elementi}$	$n_{elementj}$	→	$[\ vector\ ]_2$

See also: COL+, COL–, RSWP

CYLIN

Type: Command

Description: Cylindrical Mode Command: Sets Cylindrical coordinate mode.

CYLIN clears flag –15 and sets flag –16.

In Cylindrical mode, vectors are displayed as polar components.

Access: CAT CYLIN

Input: None

Output: None

See also: RECT, SPHERE

C→PX

Type: Command

Description: Complex to Pixel Command: Converts the specified user-unit coordinates to pixel coordinates.

The user-unit coordinates are derived from the (x_{min}, y_{min}) and (x_{max}, y_{max}) parameters in the reserved variable *PPAR*.

Access: ↰ PRG PICT C→PX

Input/Output:

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

See also: PX→C

C→R

Type: Command

Description: Complex to Real Command: Separates the real and imaginary parts of a complex number or complex array.

The result in item 1/level 2 represents the real part of the complex argument. The result in item 2/ level 1 represents the imaginary part of the complex argument.

Access: ↰ PRG TYPE C→R

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
(x, y)	→	*x*	*y*
[C-array]	→	*[R-array]*$_1$	*[R-array]*$_2$

See also: R→C, RE, IM

DARCY

Type: Function

Description: Darcy Friction Factor Function: Calculates the Darcy friction factor of certain fluid flows.

DARCY calculates the Fanning friction factor and multiplies it by 4. $x_{e/D}$ is the relative roughness – the ratio of the conduit roughness to its diameter. y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{e/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Access: CAT DARCY

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_e / D	y_{Re}	→	x_{Darcy}

See also: FANNING

DATE

Type: Command

Description: Date Command: Returns the system date.

Access: ↱ TIME TOOLS DATE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*date*

See also: DATE+, DDAYS, TIME, TSTR

→DATE

Type: Command

Description: Set Date Command: Sets the system date to *date*.

date has the form *MM.DDYYYY* or *DD.MMYYYY*, depending on the state of flag –42. *MM* is month, *DD* is day, and *YYYY* is year. If *YYYY* is not supplied, the current specification for the year is used. The range of allowable dates is January 1, 1991 to December 31, 2090.

Access: ↱ TIME TOOLS →DATE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
date	→	

See also: →TIME

DATE+

Type: Command

Description: Date Addition Command: Returns a past or future date, given a date in argument 1/level 2 and a number of days in argument 2/level 1.

If x_{days} is negative, DATE+ calculates a past date. The range of allowable dates is October 15, 1582, to December 31, 9999.

Access: ↱ TIME TOOLS DATE+

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$date_1$	x_{days}	→	$date_{new}$

See also: DATE, DDAYS

DBUG

Type: Operation

Description: Debug Operation: Starts program execution, then suspends it as if HALT were the first program command.

DBUG is not programmable.

Access: CAT DEBUG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
« *program* » or '*program name*'	→	

See also: HALT, NEXT

DDAYS

Type: Command

Description: Delta Days Command: Returns the number of days between two dates.

If the argument 1/level 2 date is chronologically later than the argument 2/ level 1 date, the result is negative. The range of allowable dates is October 15, 1582, to December 31, 9999.

Access: ↱ TIME TOOLS DDAYS

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$date_1$	$date_2$	→	x_{days}

See also: DATE, DATE+

DEC

Type: Command

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

Binary integers require the prefix #. Binary integers entered and returned in decimal base automatically show the suffix *d*. If the current base is not decimal, then you can enter a decimal number by ending it with d. It will be displayed in the current base when it is entered.

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

Access: CAT DEC

Input: None

Output: None

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

DECR

Type: Command

Description: Decrement Command: Takes a variable, subtracts 1, stores the new value back into the original variable, and returns the new value.

The contents of *name* must be a real number or an integer.

Access: ← PRG MEMORY ARITHMETIC DECR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name*'	→	x_{new}

See also: INCR, STO+, STO–

DEFINE

Type: Command

Description: Define Variable or Function Command: Stores the expression on the right side of the = in the variable specified on the left side, or creates a user-defined function.

If the left side of the equation is *name* only, DEFINE stores *exp* in the variable *name*.

If the left side of the equation is *name* followed by parenthetical arguments $name_1$ … $name_n$, DEFINE creates a user-defined function and stores it in the variable *name*.

Access: ↰ DEF

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'*name=exp*'	→	
'$name(name_1 \ldots name_n)=exp(name_1 \ldots name_n)$'	→	

See also: STO

DEG

Type: Command

Description: Degrees Command: Sets Degrees angle mode.

DEG clears flags –17 and –18, and clears the RAD and GRAD annunciators.

In Degrees angle mode, real-number arguments that represent angles are interpreted as degrees, and real-number results that represent angles are expressed in degrees.

Access: CAT DEG

Input: None

Output: None

See also: GRAD, RAD

DELALARM

Type: Command

Description: Delete Alarm Command: Deletes the specified alarm.

If n_{index} is 0, all alarms in the system alarm list are deleted.

Access: ↱ TIME TOOLS ALRM DELALARM

Input/Output:

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

Flags: None

See also: FINDALARM, RCLALARM, STOALARM

DELAY

Type: Command

Description: Delay Command: Specifies how many seconds the HP 49 waits between sending lines of information to the printer.

x_{delay} specifies the delay time in seconds. The default delay is 1.8 seconds. The maximum delay is 6.9 seconds. (The sign of x_{delay} is ignored, so –4 DELAY is equivalent to 4 DELAY.)

The delay setting is the first parameter in the reserved variable *PRTPAR*.

A shorter delay setting can be useful when the HP 49 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay *shorter* than the time to print one line, you may lose information.

Access: CAT DELAY

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{delay}	→	

See also: CR, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

DELKEYS

Type: Command

Description: Delete Key Assignments Command: Clears user-defined key assignments.

The argument x_{key} is a real number *rc.p* specifying the key by its *r*ow number, its *c*olumn number, and its *p*lane (shift). For a definition of plane, see ASN.

Specifying 0 for x_{key} clears *all* user key assignments and restores the standard key assignments.

Specifying S as the argument for DELKEYS suppresses all standard key assignments on the user keyboard. This makes keys without user key assignments inactive on the user keyboard. (You can make exceptions using ASN, or restore them all using STOKEYS.) If you are stuck in User mode – probably with a "locked" keyboard – because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt ("warm start"): press and hold ON and F3 simultaneously, releasing F3 first. This cancels User mode.

Deleted user key assignments still take up from 2.5 to 15 bytes of memory each. You can free this memory by packing your user key assignments by executing RCLKEYS 0 DELKEYS STOKEYS.

Access: (CAT) DELKEYS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{key}	→	
$\{ x_{key1}, \ldots, x_{key\ n} \}$	→	
0	→	
'*S*'	→	

See also: ASN, RCLKEYS, STOKEYS

DEPND

Type: Command

Description: Dependent Variable Command: Specifies the dependent variable (and its plotting range for TRUTH plots).

The specification for the dependent variable name and its plotting range is stored in the reserved variable *PPAR* as follows:

- If the argument is a global variable name, that name replaces the dependent variable entry in *PPAR*.
- If the argument is a list containing a global name, that name replaces the dependent variable name but leaves unchanged any existing plotting range.
- If the argument is a list containing a global name and two real numbers, or a list containing a name, array, and real number, that list replaces the dependent variable entry.
- If the argument is a list containing two real numbers, or two real numbers from levels 1 and 2, those two numbers specify a new plotting range, leaving the dependent variable name unchanged. (LASTARG returns a list, even if the two numbers were entered separately.)

The default entry is *Y*.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Access: (CAT) DEPND

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
	'global'	→	
	{ *global* }	→	
	{ *global*, y_{start}, y_{end} }	→	
	{y_{start}, y_{end} }	→	
y_{start}	y_{end}	→	

See also: INDEP

DEPTH

Type: RPN Command

Description: Depth Command: Returns a real number representing the number of objects present on the stack (before DEPTH was executed).

Access: (←) (PRG) STACK DEPTH

Input/Output:

$Level_n$...Level 1		Level 1
	→	*n*

DET

Type: Command

Description: Determinant Function: Returns the determinant of a square matrix.

The argument matrix must be square. DET computes the determinant of 1 × 1 and 2 × 2 matrices directly from the defining expression for the determinant. DET computes the determinant of a larger matrix by computing the Crout LU decomposition of the matrix and accumulating the product of the decomposition's diagonal elements.

Since floating-point division is used to do this, the computed determinant of an integer matrix is often not an integer, even though the actual determinant of an integer matrix must be an integer. DET corrects this by rounding the computed determinant to an integer value. This technique is also used for noninteger matrices with determinants having fewer than 15 nonzero digits: the computed determinant is rounded at the appropriate digit position to restore some or all of the accuracy lost to floating-point arithmetic.

This refining technique can cause the computed determinant to exhibit discontinuity. To avoid this, you can disable the refinement by setting flag –54.

Access: ↰ MATRICES OPERATIONS DET

↰ MTH NORMALIZE DET

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[matrix]]	→	$x_{\text{determinant}}$

See also: CNRM, CROSS, DOT, RNRM

DETACH

Type: Command

Description: Detach Library Command: Detaches the library with the specified number from the current directory. Each library has a unique number. If a port number is specified, it is ignored.

A RAM-based library object attached to the HOME directory must be detached before it can be purged, whereas a library attached to any other directory does not. Also, a library object attached to a non-*HOME* directory is *automatically* detached (without using DETACH) whenever a new library object is attached there.

Access: CAT DETACH

Input/Output:

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

See also: ATTACH, LIBS, PURGE

DIAG→

Type: Command

Description: Vector to Matrix Diagonal Command: Takes an array and a specified dimension and returns a matrix whose major diagonal elements are the elements of the array.

Real number dimensions are rounded to integers. If a single dimension is given, a square matrix is returned. If two dimensions are given, the proper order is { *number of rows, number of columns* }. No more than two dimensions can be specified.

If the main diagonal of the resulting matrix has more elements than the array, additional diagonal elements are set to zero. If the main diagonal of the resulting matrix has fewer elements than the array, extra array elements are dropped.

Access: ↰ MATRICES CREATE DIAG→

↰ MTH MATRIX MAKE DIAG→

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array]$_{\text{diagonals}}$	{ *dim* }	→	*[[matrix]]*

→DIAG

Type: Command

Description: Matrix Diagonal to Array Command: Returns a vector that contains the major diagonal elements of a matrix.

The input matrix does not have to be square.

Access: ↰ MATRICES CREATE →DIAG

↰ MTH MATRIX MAKE →DIAG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[*vector*]$_{\text{diagonals}}$

See also: DIAG→

DIFF

Type: Command

Description: Displays a menu of calculus commands.

Access: CAT DIFF

Input: None

Output: None

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

DIFFEQ

Type: Command

Description: Differential Equation Plot Type Command: Sets the plot type to DIFFEQ.

When the plot type is DIFFEQ and the reserved variable *EQ* does not contain a list, the initial value problem is solved and plotted over an interval using the Runge–Kutta–Fehlberg (4,5) method. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

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

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

- (x_{min},y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- (x_{max},y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a list, $\{ X\ x_0\ x_f \}$, containing a name that specifies the independent variable, and two numbers that specify the initial and final values for the independent variable. The default values for these elements are $\{ X\ 0\ x_{max} \}$.
- *res* is a real number specifying the maximum interval, in user-unit coordinates, between values of the independent variable. The default value is 0. If *res* does not equal zero, then the maximum interval is *res*. If *res* equals zero, the maximum interval is unlimited.
- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. If

the solution is real-valued, these strings can specify the dependent or the independent variable; if the solution is vector valued, the strings can specify a solution component:

- 0 specifies the dependent variable (X)
- 1 specifies the dependent variable (Y)
- *n* specifies a solution component Y_n

If *axes* contains any strings other than 0, 1 or *n*, the DIFFEQ plotter uses the default strings 0 and 1, and plots the independent variable on the horizontal axis and the dependent variable on the vertical.

- *ptype* is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in *PPAR*.
- *depend* is a list, { Y y_0 $\varkappa_{ErrTol}$ }, containing a name that specifies the dependent variable (the solution), and two numbers that specify the initial value of Y and the global absolute error tolerance in the solution Y. The default values for these elements are { Y 0 .0001 }

EQ must define the right-hand side of the initial value problem Y'(XF(X,Y). Y can return a real value or real vector when evaluated.

The DIFFEQ-plotter attempts to make the interval between values of the independent variable as large as possible, while keeping the computed solution within the specified error tolerance $\varkappa_{ErrTol}$. This tolerance may hold only at the computed points. Straight lines are drawn between computed step endpoints, and these lines may not accurately represent the actual shape of the solution. *res* limits the maximum interval size to provide higher plot resolution.

On exit from DIFFEQ plot, the first elements of *indep* and *depnd* (identifiers) contain the final values of X and Y, respectively.

If *EQ* contains a list, the initial value problem is solved and plotted using a combination of Rosenbrock (3,4) and Runge-Kutta-Fehlberg (4,5) methods. In this case DIFFEQ uses RRKSTEP to calculate y_f, and *EQ* must contain two additional elements:

- The second element of *EQ* must evaluate to the partial derivative of Y' with respect to X, and can return a real value or real vector when evaluated.
- The third element of *EQ* must evaluate to the partial derivative of Y' with respect to Y, and can return a real value or a real matrix when evaluated.

Access: CAT DIFFEQ

Input: None

Output: None

See also: AXES, CONIC, FUNCTION, PARAMETRIC, POLAR, RKFSTEP, RRKSTEP, TRUTH

DISP

Type: Command

Description: Display Command: Displays *obj* in the *n*th display line.

$n \leq 1$ indicates the top line of the display; $n \geq 7$ indicates the bottom line.

To facilitate the display of messages, strings are displayed without the surrounding " " delimiters. All other objects are displayed in the same form as would be used if the object were in level 1 in the multiline display format. If the object display requires more than one display line, the display starts in line *n*, and continues down the display either to the end of the object or the bottom of the display.

The object displayed by DISP persists in the display only until the keyboard is ready for input. The FREEZE command can be used to cause the object to persist in the display until a key is pressed.

Access: ↰ PRG OUT DISP

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	*n*	→	

See also: FREEZE, HALT, INPUT, PROMPT

DO

Type: Command Operation

Description: DO Indefinite Loop Structure Command: Starts DO … UNTIL … END indefinite loop structure.

DO … UNTIL … END executes a loop repeatedly until a test returns a true (nonzero) result. Since the test clause is executed after the loop clause, the loop is always executed at least once. The syntax is:

DO *loop-clause* UNTIL *test-clause* END

DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from

the stack. If its value is zero, the loop clause is executed again; otherwise, execution resumes following END.

Access: ⮌ PRG BRANCH DO

Input/Output:

Level 1/Argument 1			Level 1/Item 1
DO		→	
UNTIL		→	
END	*T/F*	→	

See also: END, UNTIL, WHILE

DOERR

Type: Command

Description: Do Error Command: Executes a "user-specified" error, causing a program to behave exactly as if a normal error had occurred during program execution.

DOERR causes a program to behave exactly as if a normal error has occurred during program execution. The error message depends on the argument provided to DOERR:

- n_{error} or $\#n_{error}$ display the corresponding built-in error message.
- "*error*" displays the contents of the string. (A subsequent execution of ERRM returns "*error*". ERRN returns # 70000h.)
- 0 abandons program execution without displaying a message. 0 DOERR is equivalent to pressing CANCEL.

Access: ⮌ PRG ERROR DOERR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{error}	→	
$\#n_{error}$	→	
"error"	→	
0	→	

See also: ERRM, ERRN, ERR0

DOLIST

Type: Command

Description: Do to List Command: Applies commands, programs, or user-defined functions to lists.

The number of lists, *n*, can be omitted when the first or level 1 argument is any of the following:

- A command.
- A program containing exactly one command (e.g. «DUP»)
- A program conforming to the structure of a user-defined function.

The final argument 1 (or level 1 object) can be a local or global name that refers to a program or command.

All lists must be the same length *l*. The program is executed *l* times: on the *i*th iteration, *n* objects each taken from the *i*th position in each list are entered on the stack in the same order as in their original lists, and the program is executed. The results from each execution are left on the stack. After the final iteration, any new results are combined into a single list.

Access: ↰ PRG LIST PROCEDURES DOLIST

Input/Output:

L_{n+2}/A_1 ... L_3/A_{n-2}	L_2/A_{n+1}	L_1/A_{n+2}		Level 1/Item 1
{ *list* }$_1$ ... { *list* }$_n$	*n*	*« program »*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*command*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*name*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*« program »*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*command*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_{n+1}$	*name*	→	{ *results* }

L = level; A = argument

See also: DOSUBS, ENDSUB, NSUB, STREAM

DOSUBS

Type: Command

Description: Do to Sublist Command: Applies a program or command to groups of elements in a list.

The real number *n* can be omitted when the first argument is one of the following:

- A command.
- A user program containing a single command.
- A program with a user-defined function structure.
- A global or local name that refers to one of the above.

The first iteration uses elements 1 through *n* from the list; the second iteration uses elements 2 through *n* + 1; and so on. In general, the m^{th} iteration uses the elements from the list corresponding to positions *m* through *m* + *n* – 1.

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the command ENDSUB. Both of these commands return an Undefined Local Name error if executed when DOSUBS is not active.

DOSUBS returns the Invalid User Function error if the object at level 1/argument 3 is a user program that does not contain only one command and does not have a user-defined function structure. DOSUBS also returns the Wrong Argument Count error if the object at level 1/argument 3 is a command that does not accept 1 to 5 arguments of specific types (DUP, ROT, or →LIST, for example).

Access: ⬅ PRG LIST PROCEDURES DOSUBS

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
{ *list* }$_1$	*n*	« *program* »	→	{ *list* }$_2$
{ *list* }$_1$	*n*	*command*	→	{ *list* }$_2$
{ *list* }$_1$	*n*	*name*	→	{ *list* }$_2$
	{ *list* }$_1$	« *program* »	→	{ *list* }$_2$
	{ *list* }$_1$	*command*	→	{ *list* }$_2$
	{ *list* }$_1$	*name*	→	{ *list* }$_2$

See also: DOLIST, ENDSUB, NSUB, STREAM

DOT

Type: Command

Description: Dot Product Command: Returns the dot product A·B of two arrays A and B, calculated as the sum of the products of the corresponding elements of the two arrays.

Both arrays must have the same dimensions.

Some authorities define the dot product of two complex arrays as the sum of the products of the conjugated elements of one array with their corresponding elements from the other array. The HP 49 uses the ordinary products without conjugation. If you prefer the alternative definition, apply CONJ to one array before using DOT.

Access: ↰ MATRICES VECTOR DOT

↰ MTH VECTOR DOT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
[array A]	*[array B]*	→	*x*

See also: CNRM, CROSS, DET, RNRM

DRAW

Type: Command Operation

Description: Draw Plot Command: Plots the mathematical data in the reserved variable *EQ* or the statistical data in the reserved variable Σ*DAT*, using the specified *x*- and *y*-axis display ranges.

The plot type determines if the data in the reserved variable *EQ* or the data in the reserved variable Σ*DAT* is plotted.

DRAW does not erase *PICT* before plotting; execute ERASE to do so. DRAW does not draw axes; execute DRAX to do so.

When DRAW is executed from a program, the graphics display, which shows the resultant plot, does not persist unless PICTURE, PVIEW (with an empty list argument), or FREEZE is subsequently executed.

Access: CAT DRAW

Input: None

Output: None

See also: AUTO, AXES, DRAX, ERASE, FREEZE, PICTURE, LABEL, PVIEW

DRAW3DMATRIX

Type: Command

Description: Draws a 3D plot from the values in a specified matrix.

The number of rows indicates the number of units along the x axis, the number of columns indicates the number of units along the y axis, and the values in the matrix give the magnitudes of the plotted points along the z axis. In other words, the coordinates of a plotted point are (r, c, v) where r is the row number, c the column number and v the value in the corresponding cell of the matrix.

You can limit the points that are plotted by specifing a minimum value (v_{min}) and a maximum value (v_{max}). Values in the matrix outside this range are not plotted. If all values are included, the total number of points plotted is $r \times c$.

Once the plot has been drawn, you can rotate it in various ways by pressing the following keys:

- ▶ and ◀ rotate the plot around the x axis (in different directions)
- ▲ and ▼ rotate the plot around the y axis (in different directions)
- TOOL and NXT rotate the plot around the z axis (in different directions)

Access: CAT FAST3DMATRIX

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
[[*matrix*]]	v_{min}	v_{max}	→	

See also: FAST3D

DRAX

Type: Command

Description: Draw Axes Command: Draws axes in *PICT*.

The coordinates of the axes intersection are specified by AXES. Axes tick-marks are specified in PPAR with the ATICK, or AXES command. DRAX does not draw axes labels; execute LABEL to do so.

Access: CAT DRAX

Input: None

Output: None

See also: AXES, DRAW, LABEL

DROP

Type: RPN Command

Description: Drop Object Command: Removes the level 1 object from the stack.

Access: ↰ PRG STACK DROP

Input/Output:

Level 1		Level 1
obj	→	

See also: CLEAR, DROPN, DROP2

DROP2

Type: RPN Command

Description: Drop 2 Objects Command: Removes the first two objects from the stack.

Access: ↰ PRG STACK DROP2

Input/Output:

Level 2	Level 1		Level 1
obj_1	obj_2	→	

See also: CLEAR, DROP, DROPN

DROPN

Type: RPN Command

Description: Drop n Objects Command: Removes the first $n + 1$ objects from the stack (the first n objects excluding the integer n itself).

Access: ↰ PRG STACK DROPN

Input/Output:

$Level_{n+1}$ ... Level 2	Level 1		Level 1
obj1 ... obj_n	*n*	→	

See also: CLEAR, DROP, DROP2

DTAG

Type: Command

Description: Delete Tag Command: DTAG removes all tags (labels) from an object.

The leading colon is not shown for readability when the tagged object is on the stack.

DTAG has no effect on an untagged object.

Access: ⮌ PRG TYPE DTAG

Input/Output:

Level 1/Argument 1		Level 1/Item 1
tag:obj	→	*obj*

See also: LIST→, →TAG

DUP

Type: RPN Command

Description: Duplicate Object Command: DUP returns a copy of the argument (or the object on level 1).

Access: ⮌ PRG STACK DUP

Input/Output:

Level 1		Level 2	Level 1
obj	→	*obj*	*obj*

See also: DUPN, DUP2, PICK

DUP2

Type: RPN Command

Description: Duplicate 2 Objects Command: DUP2 returns copies of the two objects on levels 1 and 2 of the stack.

Access: ⮌ PRG STACK DUP2

Input/Output:

L_2	L_1		L_4	L_3	L_2	L_1
obj_2	obj_1	→	obj_2	obj_1	obj_2	obj_1

L = level

See also: DUP, DUPN, PICK

DUPDUP

Type: RPL Command

Description: Duplicates an object twice.

Access: PRG STACK DUPDUP

Input/Output:

Level 1		Level 3	Level 2	Level 1
obj	→	*obj*	*obj*	*obj*

See also: DUP, NDUPN, DUPN, DUP2

DUPN

Type: RPN Command

Description: Duplicate n Objects Command: Takes an integer *n* from level 1 of the stack, and returns copies of the objects on stack levels 2 through *n* + 1.

Access: ↰ PRG STACK DUPN

Input/Output:

L_{i+1}	L_i ...L_3	L_2	L_1		L_{i+n}	L_{i+n-1} ... L_2	L_1
obj_1	obj_2 ... obj_{i-1}	obj_i	n	→	obj_1	obj_2 ... obj_{i-1}	obj_i

L = level

See also: DUP, DUP2, PICK

D→R

Type: Function

Description: Degrees to Radians Function: Converts a real number representing an angle in degrees to its equivalent in radians.

This function operates independently of the angle mode.

Access: ↰ MTH REAL D→R

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	*(π/180)x*
'symb'	→	*'D→R(symb)'*

See also: R→D

E to F

e

Type: Function

Description: e Function: Returns the symbolic constant *e* or its numerical representation, 2.71828182846.

The number returned for *e* is the closest approximation to 12-digit accuracy. For exponentiation, use the expression EXP(*x*) rather than *e*^*x*, since the function EXP uses a special algorithm to compute the exponential to greater accuracy.

Access: ALPHA ↰ E

↰ MTH CONSTANTS *e*

↰ MTH CONSTANTS 2.718281828…

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*'e'*
	→	*2.71828182846*

See also: EXP, EXPM, *i*, LN, LNP1, MAXR, MINR, π

EDIT

Type: Command

Description: Edit Command: Moves specified object to the command line where it can be edited.

Access: CAT EDIT

See also: VISIT

EDITB

Type: Command

Description: Edit Command: Opens the specified object in the most suitable editor. For example, if you use a matrix as the specified object, the command opens it in Matrix Writer.

Access: CAT EDIT

See also: VISIT

EGV

Type: Command

Description: Eigenvalues and Eigenvectors Command: Computes the eigenvalues and right eigenvectors for a square matrix.

The resulting vector EVal contains the computed eigenvalues. The columns of matrix EVec contain the right eigenvectors corresponding to the elements of vector EVal.

The computed results should minimize (within computational precision):

$$\frac{|A \cdot EVec - EVec \cdot diag(EVal)|}{n \cdot |A|}$$

where diag (*EVal*) denotes the $n \times n$ diagonal matrix containing the eigenvalues *EVal*.

Access: ↰ MATRICES EIGENVECTOR EGV

↰ MTH MATRIX EGV

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
$[[matrix]]_{A}$	→	$[[matrix]]_{EVec}$	$[vector]_{EVal}$

See also: EGVL

EGVL

Type: Command

Description: Eigenvalues Command: Computes the eigenvalues of a square matrix.

The resulting vector L contains the computed eigenvalues.

Access: ↰ MATRICES EIGENVECTOR EGVL

↰ MTH MATRIX EGVL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[matrix]]_{A}$	→	$[vector]_{EVal}$

See also: EGV

ELSE

Type: Command

Description: ELSE Command: Starts false clause in conditional or error-trapping structure.

See the IF and IFERR keyword entries for more information.

Access: ↰ PRG BRANCH ELSE

Input: None

Output: None

See also: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE

END

Type: Command

Description: END Command: Ends conditional, error-trapping, and indefinite loop structures.

See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

Access: ↰ PRG BRANCH END

Input: None

Output: None

See also: IF, CASE, DO, ELSE, IFERR, REPEAT, THEN, UNTIL, WHILE

ENDSUB

Type: Command

Description: Ending Sublist Command: Provides a way to access the total number of sublists contained in the list used by DOSUBS.

Returns an Undefined Local Name error if executed when DOSUBS is not active.

Access: ↰ PRG LIST PROCEDURES ENDSUB

Input: None

Output: None

See also: DOSUBS, NSUB

ENG

Type: Command

Description: Engineering Mode Command: Sets the number display format to engineering mode, which displays one to three digits to the left of the fraction mark (decimal point) and an exponent that is a multiple of three. The total number of significant digits displayed is $n + 1$.

Engineering mode uses $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for n outside this range are rounded up or down.) A number is displayed or printed as follows:

(*sign*) *mantissa* E (*sign*) *exponent*

where the mantissa is of the form $(nn)n.(n\ldots)$ (with up to 12 digits total) and the exponent has one to three digits.

A number with an exponent of –499 is displayed automatically in scientific mode.

Access: CAT ENG

Input/Output:

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

See also: FIX, SCI, STD

EQW

Type: Command

Description: Opens Equation Writer, where you can edit an expression.

If the object you specify—or the object on level 1—is not the type of object that can be edited in Equation Writer, EQW places the object on the command line, where you edit it.

Access: CAT EQW

Input/Output:

Level 1/Argument 1		Level 1/Item 1
exp_1	→	exp_2

See also: EDIT, EDITB, VISIT, VISITB

EQ→

Type: Command

Description: Equation to Stack Command: Separates an equation into its left and right sides.
If the argument is an expression, it is treated as an equation whose right side equals zero.

Access: ↰ PRG TYPE EQ→

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
'$symb_1$=$symb_2$'	→	'$symb_1$'	'$symb_2$'
z	→	z	*0*
'*name*'	→	'*name*'	*0*
'*x_unit*'	→	'*x_unit*'	*0*
'*symb*'	→	'*symb*'	*0*

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

ERASE

Type: Command

Description: Erase PICT Command: Erases *PICT*, leaving a blank *PICT* of the same dimensions.

Access: CAT ERASE

Input: None

Output: None

See also: DRAW

ERR0

Type: Command

Description: Clear Last Error Number Command: Clears the last error number so that a subsequent execution of ERRN returns # 0h, and clears the last error message.

Access: ⮌ PRG ERROR ERR0

Input: None

Output: None

See also: DOERR, ERRM, ERRN

ERRM

Type: Command

Description: Error Message Command: Returns a string containing the error message of the most recent calculator error.

ERRM returns the string for an error generated by DOERR. If the argument to DOERR was 0, the string returned by ERRM is empty.

Access: ⮌ PRG ERROR ERRM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*"error message"*

See also: DOERR, ERRN, ERR0

ERRN

Type: Command

Description: Error Number Command: Returns the error number of the most recent calculator error.

If the most recent error was generated by DOERR with a string argument, ERRN returns #70000h. If the most recent error was generated by DOERR with a binary integer argument, ERRN returns that binary integer. (If the most recent error was generated by DOERR with a real number argument, ERRN returns the binary integer conversion of the real number.)

Access: ⮌ PRG ERROR ERRN

Input/Output:

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

See also: DOERR, ERRM, ERR0

EVAL

Type: Command

Description: Evaluate Object Command: Evaluates the object.

The following table describes the effect of the evaluation on different object types.

Obj. Type	Effect of Evaluation
Local Name	Recalls the contents of the variable.
Global Name	*Calls* the contents of the variable: • A name is evaluated. • A program is evaluated. • A directory becomes the current directory. • Other objects are put on the stack. If no variable exists for a given name, evaluating the name returns the name to the stack.
Program	*Enters* each object in the program: • Names are evaluated (unless quoted). • Commands are evaluated. • Other objects are put on the stack.

Obj. Type	Effect of Evaluation (Continued)
List	*Enters* each object in the list: • Names are evaluated. • Commands are evaluated • Programs are evaluated. • Other objects are put on the stack.
Tagged	If the tag specifies a port, recalls and evaluates the specified object. Otherwise, puts the untagged object on the stack.
Algebraic	*Enters* each object in the algebraic expression: • Names are evaluated. • Commands are evaluated. • Other objects are put on the stack.
Command, Function, XLIB Name	Evaluates the specified object.
Other Objects	Puts the object on the stack.

To evaluate a symbolic argument to a numerical result, evaluate the argument in Numerical Result mode (flag –3 set) or execute →NUM on that argument.

Access: ↱ EVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	*(see above)*

See also: →NUM, SYSEVAL

EXP

Type: Analytic Function

Description: Exponential Analytic Function: Returns the exponential, or natural antilogarithm, of the argument; that is, *e* raised to the given power.

EXP uses extended precision constants and a special algorithm to compute its result to full 12-digit precision for all arguments that do not trigger an underflow or overflow error.

EXP provides a more accurate result for the exponential than can be obtained by using *e* y^x. The difference in accuracy increases as *z* increases. For example:

z	EXP(*z*)	e^z
3	20.0855369232	20.0855369232
10	22026.4657948	22026.4657949
100	2.68811714182E43	2.68811714191E43
500	1.40359221785E217	1.40359221809E217
1000	1.9707111402E434	1.9707111469E434

For complex arguments:

$$e^{(x,y)} = e^x \cos y + ie^x \sin y$$

Access: ↰ e^x

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	e^z
'*symb*'	→	'*EXP(symb)*'

See also: ALOG, EXPM, LN, LOG

EXPAN

Type: Command

Description: Expand Products Command: Rewrites an algebraic expression or equation by expanding products and powers. Same as the EXPAND command (see volume 1, CAS Commands).

Access: CAT EXPAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	x
'$symb_1$'	→	'$symb_2$'
(x, y)	→	(x, y)

See also: COLCT, EXPLAND, ISOL, QUAD, SHOW

EXPFIT

Type: Command

Description: Exponential Curve Fit Command: Stores EXPFIT as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the exponential curve fitting model.

LINFIT is the default specification in Σ*PAR*.

Access: CAT EXPFIT

Input: None

Output: None

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

EXPM

Type: Analytic Function

Description: Exponential Minus 1 Analytic Function: Returns $e^x - 1$.

For values of x close to zero, EXPM(x) returns a more accurate result than does EXP(x)–1. (Using EXPM allows both the argument and the result to be near zero, and avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.)

Access: ↰ (MTH) HYPERBOLIC EXPM

↰ (EXP&LN) EXPM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	$e^x - 1$
'*symb*'	→	'*EXPM(symb)*'

See also: EXP, LNP1

EYEPT

Type: Command

Description: Eye Point Command: Specifies the coordinates of the eye point in a perspective plot.

x_{point}, y_{point}, and z_{point} are real numbers that set the x-, y-, and z-coordinates as the eye-point from which to view a 3D plot's view volume. The y-coordinate must always be 1 unit less than the view volume's nearest point (y_{near} of YVOL). These coordinates are stored in the reserved variable *VPAR*.

Access: (CAT) EYEPT

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
x_{point}	y_{point}	z_{point}	→	

See also: NUMX, NUMY, XVOL, XXRNG, YVOL, YYRNG, ZVOL

F0λ

Type: Function

Description: Black Body Emissive Power Function: Returns the fraction of total black-body emissive power at temperature x_T between wavelengths 0 and y_{lambda}. If units are not specified, y_{lambda} has implied units of meters and x_T has implied units of K.

F0λ returns a dimensionless fraction.

Access: (CAT) F0λ

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{lambda}	x_T	→	x_{power}
y_{lambda}	'*symb*'	→	'*F0λ(*y_{lambda}*,symb)*'
'*symb*'	x_T	→	'*F0λ(symb,*x_T*)*'
'$symb_1$'	'$symb_2$'	→	'*F0λ(*$symb_1$*,*$symb_2$*)*'

FACT

Type: Command

Description: Factorial (Gamma) Function: FACT is the same as !. See !.

Access: None. Must be typed in.

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n	→	*n!*
x	→	$\Gamma(x + 1)$
'*symb*'	→	'*(symb)!*'

FANNING

Type: Function

Description: Fanning Friction Factor Function: Calculates the Fanning friction factor of certain fluid flows.

FANNING calculates the Fanning friction factor, a correction factor for the frictional effects of fluid flows having constant temperature, cross-section, velocity, and viscosity (a typical pipe flow, for example). $x_{x/D}$ is the relative roughness (the ratio of the conduit roughness to its diameter). y_{Re} is the Reynolds number. The function uses different computation routines for laminar flow ($Re \leq 2100$) and turbulent flow ($Re > 2100$). $x_{x/D}$ and y_{Re} must be real numbers or unit objects that reduce to dimensionless numbers, and both numbers must be greater than 0.

Access: CAT FANNING

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_{x/D}$	y_{Re}	→	$x_{fanning}$
$x_{x/D}$	*'symb'*	→	*'FANNING(*$x_{x/D}$*,symb)'*
'symb'	y_{Re}	→	*'FANNING(symb,*y_{Re}*)'*
'$symb_1$*'*	*'*$symb_2$*'*	→	*'FANNING(*$symb_1$*,*$symb_2$*)'*

See also: DARCY

FAST3D

Type: Command

Description: Fast 3D Plot Type Command: Sets the plot type to FAST 3D.

When plot type is set to FAST3D, the DRAW command plots an image graph of a 3-vector-valued function of two variables. FAST3D requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR is made up of the following elements:

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

For plot type FAST3D, the elements of *VPAR* are used as follows:

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye},y_{eye}, and z_{eye} are are not used.
- x_{step} and y_{step} are real numbers that set the number of x-coordinates versus the number of y-coordinates plotted.

The plotting parameters are specified in the reserved variable *PPAR*, which has this form:

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

For plot type FAST3D, the elements of *PPAR* are used as follows:

- (x_{min},y_{min}) is not used.

- (x_{max}, y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is not used.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command FAST3D places the name FAST3D in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: CAT FAST3D

Input: None

Output: None

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

FC?

Type: Command

Description: Flag Clear? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set.

Access: ← PRG TEST FC?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?C, FS? FS?C, SF

FC?C

Type: Command

Description: Flag Clear? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is clear, and returns a corresponding test result: 1 (true) if the flag is clear or 0 (false) if the flag is set. After testing, clears the flag.

Access: ← PRG TEST FC?C

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FS? FS?C, SF

FFT

Type: Command

Description: Discrete Fourier Transform Command: Computes the one- or two-dimensional discrete Fourier transform of an array.

If the argument is an N-vector or an $N \times 1$ or $1 \times N$ matrix, FFT computes the one-dimensional transform. If the argument is an $M \times N$ matrix, FFT computes the two-dimensional transform. M and N must be integral powers of 2.

The one-dimensional discrete Fourier transform of an N-vector X is the N-vector Y where:

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

for $k = 0, 1, \ldots, N-1$.

The two dimensional discrete Fourier transform of an $M \times N$ matrix X is the $M \times N$ matrix Y where:

$$Y_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{mn} e^{-\frac{2\pi i k m}{M}} e^{-\frac{2\pi i l n}{N}}, i = \sqrt{-1}$$

for $k = 0, 1, \ldots, M-1$ and $l = 0, 1, \ldots, N-1$.

The discrete Fourier transform and its inverse are defined for any positive sequence length. However, the calculation can be performed very rapidly when the sequence length is a power of two, and the resulting algorithms are called the fast Fourier transform (FFT) and inverse fast Fourier transform (IFFT).

The FFT command uses truncated 15-digit arithmetic and intermediate storage, then rounds the result to 12-digit precision.

Access: ↰ MTH FFT FFT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[*array*]$_1$	→	[*array*]$_2$

See also: IFFT

FILER

Type: Command

Description: Opens File Manager.

Access: ↰ FILES

CAT FILER

Input: None

Output: None

FINDALARM

Type: Command

Description: Find Alarm Command: Returns the alarm index n_{index} of the first alarm due after the specified time.

If the input is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the input is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the input is the real number 0, FINDALARM returns the first *past-due* alarm.

For any of the three arguments, FINDALARM returns 0 if no alarm is found.

Access: ↱ TIME TOOLS ALRM FINDALARM

Input/Output:

Level 1/Argument 1		Level 1/Item 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

See also: DELALARM, RCLALARM, STOALARM

FINISH

Type: Command

Description: Finish Server Mode Command: Terminates Kermit Server mode in a device connected to an HP 49.

FINISH is used by a local Kermit device to tell a server Kermit (connected via the serial port) to exit Server mode.

Access: CAT FINISH

Input: None

Output: None

See also: BAUD, CKSM, KGET, PARITY, PKT, RECN, RECV, SEND, SERVER

FIX

Type: Command

Description: Fix Mode Command: Sets the number display format to fix mode, which rounds the display to *n* decimal places.

Fix mode shows *n* digits to the right of the fraction mark (decimal point), where $0 \leq n \leq 11$. (Values for *n* outside this range are rounded to the nearest integer.) A number is displayed or printed as follows:

(*sign*) *mantissa*

where the mantissa can be of any form. However, the calculator automatically displays a number in scientific mode if either of the following is true:

- The number of digits to be displayed exceeds 12.
- A nonzero value rounded to *n* decimal places otherwise would be displayed as zero.

Access: CAT FIX

Input/Output:

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

See also: SCI, STD

FLASHEVAL

Type: Command

Description: Evaluate Flash Function Command: Evaluates unnamed Flash functions.

Using FLASHEVAL with random addresses can corrupt memory. $\#n_{function}$ is of the form *ffffbbbh*, where *bbb* is the bank ID, and *ffff* is the function number.

Access: CAT FLASHEVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$\#n_{function}$	→	

See also: EVAL,LIBEVAL, SYSEVAL

FLOOR

Type: Function

Description: Floor Function: Returns the greatest integer that is less than or equal to the argument.

Access: ← MTH REAL FLOOR

Input/Output:

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

See also: CEIL, IP, RND, TRNC

FONT6

Type: Function

Description: Font Function: Returns the system FONT6 object. You use this in conjunction with the →FONT command to set the system font to type 6.

Access: CAT

See also: FONT7, FONT8, →FONT, FONT→

FONT7

Type: Function

Description: Font Function: Returns the system FONT7 object. You use this in conjunction with the →FONT command to set the system font to type 7.

Access: CAT

See also: FONT6, FONT8, →FONT, FONT→

FONT8

Type: Function

Description: Font Function: Returns the system FONT8 object. You use this in conjunction with the →FONT command to set the system font to type 8.

Access: CAT

See also: FONT6, FONT7, →FONT, FONT→

FONT→

Type: Function

Description: Returns the current system font.

Access: CAT

See also: FONT6, FONT7, FONT8

→FONT

Type: Function

Description: Set font Function: Sets the system font. You use this in conjunction with one of the three font commands to set the system font. for example, the following command sets the system font to type 6:

```
→FONT(FONT6)
```

Access: CAT

See also: FONT6, FONT7, FONT8, FONT→

FOR

Type: Command Operation

Description: FOR Definite Loop Structure Command: Starts FOR … NEXT and FOR … STEP definite loop structures.

Definite loop structures execute a command or sequence of commands a specified number of times.

- A FOR … NEXT loop executes a program segment a specified number of times using a local variable as the loop counter. You can use this variable within the loop. The RPL syntax is this:

  x_{start} x_{finish} FOR counter loop-clause NEXT

  The algebraic syntax is this:

  FOR (counter,x_{start} ,x_{finish})loop-clause NEXT

  FOR takes x_{start} and x_{finish} as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Then, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. NEXT increments *counter* by one, and then tests whether *counter* is less than or equal to x_{finish}. If so, the loop clause is repeated (with the new value of *counter*).

  When the loop is exited, *counter* is purged.

- FOR … STEP works just like FOR … NEXT, except that it lets you specify an increment value other than 1. The syntax RPL is:

  x_{start} x_{finish} FOR counter loop-clause $x_{increment}$ STEP

  The algebraic syntax is:

  FOR(counter x_{start} x_{finish})loop-clause, STEP ($x_{increment}$)

  FOR takes x_{start} and x_{finish} as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop clause is executed; *counter* can be referenced or have its value changed within the loop clause. STEP takes $x_{increment}$ and increments *counter* by that value. If the argument of STEP is an algebraic expression or a name, it is automatically evaluated to a number.

  The increment value can be positive or negative. If the increment is positive, the loop is executed again when *counter* is less than or equal to x_{finish}. If the increment is negative, the loop is executed when *counter* is greater than or equal to x_{finish}.

  When the loop is exited, *counter* is purged.

Access: ⎋ PRG BRANCH FOR

Input/Output:

Level 2/	Level 1		Level 1/Item 1
FOR x_{start}	x_{finish}	→	
NEXT		→	
FOR x_{start}	x_{finish}	→	
STEP	$x_{increment}$	→	
STEP	'$symb_{increment}$'	→	

See also: NEXT, START, STEP

FP

Type: Function

Description: Fractional Part Function: Returns the fractional part of the argument.

The result has the same sign as the argument.

Access: ↰ MTH REAL FP

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	y
x_unit	→	*y_unit*
'*symb*'	→	'*FP(symb)*'

See also: IP

FREEZE

Type: Command

Description: Freeze Display Command: Freezes the part of the display specified by $n_{display\ area}$, so that it is not updated until a key is pressed.

Normally, the stack display is updated as soon as the calculator is ready for data input. For example, when HALT stops a running program, or when a program ends, any displayed messages are cleared. The FREEZE command "freezes" a part or all of the display so that it is not updated until a key is pressed. This allows, for example, a prompting message to persist after a program halts to await data input.

$n_{\text{display area}}$ is the sum of the value codes for the areas to be frozen:

Display Area	Value Code
Status area	1
History/Stack/Command-line area	2
Menu area	4

So, for example, FREEZE(2) freezes the history/stack/command-line area, FREEZE(3) freezes the status area and the history/stack/command-line area, and FREEZE(7) freezes all three areas.

Values of $n_{\text{display area}} \geq 7$ or ≤ 0 freeze the entire display (are equivalent to value 7). To freeze the graphics display, you must freeze the status and stack/command-line areas (by entering 3), or the entire display (by entering 7).

Access: ⬅ PRG OUT FREEZE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{displayarea}}$	→	

See also: CLLCD, DISP, HALT

FS?

Type: Command

Description: Flag Set? Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear.

Access: ⬅ PRG TEST FS?

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FC?C, FS?C, SF

FS?C

Type: Command

Description: Flag Set? Clear Command: Tests whether the system or user flag specified by $n_{\text{flag number}}$ is set, and returns a corresponding test result: 1 (true) if the flag is set or 0 (false) if the flag is clear. After testing, clears the flag.

Access: ↰ PRG TEST FS?C

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$n_{\text{flag number}}$	→	*0/1*

See also: CF, FC?, FC?C, FS?, SF

FUNCTION

Type: Command

Description: Function Plot Type Command: Sets the plot type to FUNCTION.

When the plot type is FUNCTION, the DRAW command plots the current equation as a real-valued function of one real variable. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

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

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

- $(x_{\min},y_{\min})$ is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is (–6.5,–3.1).
- $(x_{\max},y_{\max})$ is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is (6.5,3.2).
- *indep* is a name specifying the independent variable, or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval (in user-unit coordinates) between plotted values of the independent variable, or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.

- *axes* is a list containing one or more of the following, in the order listed: a complex number specifying the user-unit coordinates of the plot origin, a list specifying the tick-mark annotation, and two strings specifying labels for the horizontal and vertical axes. The default value is (0,0).
- *ptype* is a command name specifying the plot type. Executing the command FUNCTION places the name FUNCTION in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specified in *indep*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max})(the display range) are used. Lines are drawn between plotted points unless flag –31 is set.

If *EQ* contains an expression or program, the expression or program is evaluated in Numerical Results mode for each value of the independent variable to give the values of the dependent variable. If *EQ* contains an equation, the plotting action depends on the form of the equation, as shown in the following table.

Form of Current Equation	Plotting Action
expr = *expr*	Each expression is plotted separately. The intersection of the two graphs shows where the expressions are equal.
name = *expr*	Only the expression is plotted.
indep = *constant*	A vertical line is plotted.

If flag –28 is set, all equations are plotted simultaneously.

If the independent variable in the current equation represents a unit object, you must specify the units by storing a unit object in the corresponding variable in the current directory. For example, if the current equation is X+3_m, and you want *X* to represent some number of inches, you would store 1_in (the number part of the unit object is ignored) in *X*. For each plotted point, the numerical value of the independent variable is combined with the specified unit (inches in this example) before the current equation is evaluated. If the result is a unit object, only the number part is plotted.

Access: CAT FUNCTION

Input: None

Output: None

See also: BAR, CONIC, DIFFEQ, FAST3D, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, SLOPEFIELD, TRUTH, WIREFRAME, YSLICE
