HP 49G Advanced Users Guide

Volume 1

Part E: Other Commands: T to Z (and Symbols)

Introduction

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

For each operation, the following details are provided:

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

Description: A description of the operation.

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

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

See also: Related functions or commands

T to V

%T

Type: Function

Description: Percent of Total Function: Returns the percent of the first argument that is represented by the second argument.

If both arguments are unit objects, the units must be consistent with each other.

The dimensions of a unit object are dropped from the result, *but units are part of the calculation.*

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

Access: ↰ MTH REAL %T

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$100y/x$
x	'*symb*'	→	'%T(x,*symb*)'
'*symb*'	x	→	'%T(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%T($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	$100y\_unit_2/x\_unit_1$
x_unit	'*symb*'	→	'%T(x_unit,*symb*)'
'*symb*'	x_unit	→	'%T(*symb*,x_unit)'

See also: %, %CH

→TAG

Type: Command

Description: Stack to Tag Command: Combines objects in levels 1 and 2 to create tagged (labeled) object.

The "*tag*" argument is a string of fewer than 256 characters.

Access: CAT →TAG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	"*tag*"	→	:*tag*:*obj*
obj	'*name*'	→	:*name*:*obj*
obj	*x*	→	:*x*:*obj*

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

TAIL

Type: Command

Description: Last Listed Elements Command: Returns all but the first element of a list or string.

Access: ↰ PRG CHARS TAIL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
{ obj_1 ... obj_n }	→	{ obj_2 ... obj_n }
"$string_1$"	→	"$string_2$"

See also: HEAD

TAN

Type: Analytic function

Description: Tangent Analytic Function: Returns the tangent of the argument.

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

For a real argument that is an odd-integer multiple of 90 in Degrees mode, an Infinite Result exception occurs. If flag –22 is set (no error), the sign of the result (MAXR) matches that of the argument.

For complex arguments:

$$\tan(x+iy) = \frac{(\sin x)(\cos x) + i(\sinh y)(\cosh y)}{\sinh^2 y + \cos^2 x}$$

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

Access: TAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	tan z
'*symb*'	→	'TAN(*symb*)'
x_unit_{angular}		tan (x_unit_{angular})

See also: ATAN, COS, SIN

TANH

Type: Analytic function

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

For complex arguments,

$$\tanh(x+iy) = \frac{\sinh 2x + i\sin 2y}{\cosh 2x + \cos 2y}$$

Access: → TRIG HYPERBOLIC TANH

← MTH HYPERBOLIC TANH

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	tanh z
'*symb*'	→	'TANH(*symb*)'

See also: ATANH, COSH, SINH

TAYLR

Type: Command

Description: Taylor Polynomial Command: Calculates the *n*th order Taylor polynomial of *symb* in the variable *global*.

The polynomial is calculated at the point *global* = 0. The expression *symb* may have a removable singularity at 0. The order, *n*, is the *relative* order of the Taylor polynomial–the difference in order between the largest and smallest power of *global* in the polynomial.

TAYLR always returns a symbolic result, regardless of the state of the Numeric Results flag (–3).

Access: ↰ CALC LIMITS & SERIES TAYLR

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
'*symb*'	'*global*'	n_{order}	→	'$symb_{Taylor}$'

See also: ∂, ∫, Σ

TDELTA

Type: Function

Description: Temperature Delta Function: Calculates a temperature change.

TDELTA subtracts two points on a temperature scale, yielding a temperature *increment* (not an actual temperature). *x* or *x_unit1* is the final temperature, and *y* or *y_unit2* is the initial temperature. If unit objects are given, the increment is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the increment is returned as a real number.

Access: CAT TDELTA

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y*	→	x_{delta}
x_unit1	*x_unit2*	→	x_unit1_{delta}
x_unit	'*symb*'	→	'TDELTA(*x_unit, symb*)'
'*symb*'	*y_unit*	→	'TDELTA(*symb, y_unit*)'
'$symb_1$'	'$symb_2$'	→	'TDELTA($symb_1$, $symb_2$)'

See also: TINC

TEVAL

Type: Function

Description: For the specified operation, performs the same function as EVAL, and returns the the time taken to perform the evaluation as well as the result.

Access: Catalog, CAT

Input/Output:

Level 1/Argument 1		Level 1/Item 1	
Object	→	*result*	*time taken*

Input: The object to evaluate.

Output: Level 2/Item 1: The result of the evaluation.
Level 1/Item 2: The time taken in seconds to perform the evaluation.

TEXT

Type: Command

Description: Show Stack Display Command: Displays the stack display.

TEXT switches from the graphics display to the stack display. TEXT does not update the stack display.

Access: ↰ PRG OUT TEXT

Input: None

Output: None

See also: PICTURE, PVIEW

THEN

Type: Command

Description: THEN Command: Starts the true-clause in conditional or error-trapping structure.
See the IF and IFFER entries for more information.

Access: ↰ PRG BRANCH THEN

Input: None

Output: None

See also: CASE, ELSE, END, IF IFERR

TICKS

Type: Command

Description: Ticks Command: Returns the system time as a binary integer, in units of 1/8192 second.

Access: ↱ TIME TOOLS TICKS

Input/Output:

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

Example: If the result from a previous invocation from TICKS is on level 1 of the stack, then the command:

```
TICKS SWAP - B→R8192 /
```

returns a real number whose value is the elapsed time in seconds between the two invocations.

See also: TIME

TIME

Type: Command

Description: Time Command: Returns the system time in the form HH.MMSSs.

time has the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* is always returned in 24-hour format, regardless of the state of the Clock Format flag (–41).

Access: ↱ TIME TOOLS TIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*time*

See also: DATE, TICKS, TSTR

→TIME

Type: Command

Description: Set System Time Command: Sets the system time.

time must have the form *HH.MMSSs*, where *HH* is hours, *MM* is minutes, *SS* is seconds, and *s* is zero or more digits (as many as allowed by the current display mode) representing fractional seconds. *time* must use 24-hour format.

Access: ↱ TIME TOOLS →TIME

Input/Output:

Level 1/Argument 1		Level 1/Item 1
time	→	

See also: CLKADJ, →DATE

TINC

Type: Function

Description: Temperature Increment Command: Calculates a temperature increment.

TINC adds a temperature *increment* (not an actual temperature) to a point on a temperature scale. Use a negative increment to subtract the increment from the temperature. $x_{initial}$ or *x_unit1* is the initial temperature, and y_{delta} or y_unit2_{delta} is the temperature increment. The returned temperature is the resulting final temperature. If unit objects are given, the final temperature is returned as a unit object with the same units as *x_unit1*. If real numbers are given, the final temperature is returned as a real number.

Access: CAT TINC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$x_{initial}$	y_{delta}	→	x_{final}
x_unit1	y_unit2_{delta}	→	x_unit1_{final}
x_unit	'*symb*'	→	'TINC(*x_unit*, *symb*)'
'*symb*'	y_unit_{delta}	→	'TINC(*symb*, y_unit_{delta})'
'$symb_1$'	'$symb_2$'	→	'TINC($symb_1$, $symb_2$)'

See also: TDELTA

TLINE

Type: Command

Description: Toggle Line Command: For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off every pixel that is on, and turns on every pixel that is off.

Access: ↰ PRG PICT TLINE

Input/Output:

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

See also: ARC, BOX, LINE

TMENU

Type: Command

Description: Temporary Menu Command: Displays a built-in menu, library menu, or user-defined menu.

TMENU works just like MENU, except for user-defined menus (specified by a list or by the name of a variable that contains a list). Such menus are displayed like a custom menu and work like a custom menu, but are not stored in reserved variable *CST*. Thus, a menu defined and displayed by TMENU cannot be redisplayed by evaluating *CST*.

See the MENU entry for a list of the HP 49 built-in menus and the corresponding menu numbers (x_{menu}).

Access: CAT TMENU

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_{menu}	→	
{ $list_{definition}$ }	→	
'$name_{definition}$'	→	

See also: MENU, RCLMENU

TOT

Type: Command

Description: Total Command: Computes the sum of each of the *m* columns of coordinate values in the current statistics matrix (reserved variable Σ*DAT*).

The sums are returned as a vector of *m* real numbers, or as a single real number if *m* = 1.

Access: CAT TOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	x_{sum}
	→	$[\, x_{sum\ 1}, x_{sum\ 2}, \cdots, x_{sum\ m} \,]$

See also: MAXΣ, MINΣ, MEAN, PSDEV, PVAR, SDEV, VAR

TRACE

Type: Command

Description: Matrix Trace Command: Returns the trace of a square matrix.

The trace of a square matrix is the sum of its diagonal elements.

Access: ↰ MATRICES OPERATIONS TRACE

↰ MTH MATRIX NORMALIZE TRACE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$[[\ matrix\ ]]_{n \times n}$	→	x_{trace}

TRAN

Type: Command

Description: Transpose Matrix Command: Returns the transpose of a matrix.

Same as TRN, but without conjugation of complex numbers.

Access: ↰ MATRICES OPERATIONS TRAN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[[*matrix*]]$_{\text{transpose}}$
'*name*'	→	

See also: CONJ, TRN

TRANSIO

Type: Command

Description: I/O Translation Command: Specifies the character translation option. These translations affect only ASCII Kermit transfers and files printed to the serial port.

Legal values for *n* are as follows:

n	Effect
0	No translation
1	Translate character 10 (line feed only) to / from characters 10 and 13 (line feed with carriage return, the Kermit protocol) (the default value)
2	Translate characters 128 through 159 (80 through 9F hexadecimal)
3	Translate all characters (128 through 255)

Access: CAT TRANSIO

Input/Output:

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

See also: BAUD, CKSM, PARITY

TRIGO

Type: Command

.Description: Displays a menu of trigonometry commands.

Access: CAT TRIGO

Input: None

Output: None

TRN

Type: Command

Description: Transpose Matrix Command: Returns the (conjugate) transpose of a matrix.

TRN replaces an $n \times m$ matrix A with an $m \times n$ matrix A^T, where:

$\mathbf{A}_{ij}{}^T = \mathbf{A}_{ji}$ for real matrices

$\mathbf{A}_{ij}{}^T = \text{CONJ}(\mathbf{A}_{ji})$ for complex matrices

If the matrix is specified by *name*, $\mathbf{A}^T$ replaces **A** in *name*.

Access: ↰ MTH MATRIX MAKE TRN

Input/Output:

Level 1/Argument 1		Level 1/Item 1
[[*matrix*]]	→	[[*matrix*]]$_{transpose}$
'*name*'	→	

See also: CONJ

TRNC

Type: Function

Description: Truncate Function: Truncates an object to a specified number of decimal places or significant digits, or to fit the current display format.

$n_{truncate}$ (or $symb_{truncate}$ if flag –3 is set) controls how the level 2 argument is truncated, as follows:

$n_{truncate}$	Effect on Level 2 Argument
0 through 11	truncated to *n* decimal places
–1 through –11	truncated to *n* significant digits
12	truncated to the current display format

For complex numbers and arrays, each real number element is truncated. For unit objects, the number part of the object is truncated.

Access: ↰ MTH REAL TRNC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$n_{truncate}$	→	$\tilde{z}_2$
$\tilde{z}_1$	'$symb_{truncate}$'	→	'TRNC($\tilde{z}_1$,$symb_{truncate}$)'
'$symb_1$'	$n_{truncate}$	→	'TRNC($symb_1$,$n_{truncate}$)'
'$symb_1$'	'$symb_{truncate}$'	→	'TRNC($symb_1$,$symb_{truncate}$)'
[*array*]$_1$	$n_{truncate}$	→	[*array*]$_2$
x_unit	$n_{truncate}$	→	*y_unit*
x_unit	'$symb_{truncate}$'	→	'TRNC(*x_unit*,$symb_{truncate}$)'

See also: RND

TRUTH

Type: Command

Description: Truth Plot Type Command: Sets the plot type to TRUTH.

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

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

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

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

The contents of *EQ* must be an expression or program, and cannot be an equation. It is evaluated for each pixel in the plot region. The minimum and maximum values of the independent variables (the plotting ranges) can be specified in *indep* and *depend*; otherwise, the values in (x_{min},y_{min}) and (x_{max},y_{max})(the display range) are used. The result of each evaluation

must be a real number. If the result is zero, the state of the pixel is unchanged. If the result is nonzero, the pixel is turned on (made dark).

Access: CAT TRUTH

Input: None

Output: None

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

TSTR

Type: Command

Description: Date and Time String Command: Returns a string derived from the date and time.

The string has the form "*DOW DATE TIME*", where *DOW* is a three-letter abbreviation of the day of the week corresponding to the argument *date* and *time*, *DATE* is the argument *date* in the current date format, and *TIME* is the argument *time* in the current time format.

Access: ↱ TIME TOOLS TSTR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
date	*time*	→	"*DOW DATE TIME*"

See also: DATE, TICKS, TIME

TVARS

Type: Command

Description: Typed Variables Command: Lists all global variables in the current directory that contain objects of the specified types.

If the current directory contains no variables of the specified types, TVARS returns an empty list.

For a table of the object-type numbers, see the entry for TYPE.

Access: ↰ PRG MEMORY DIRECTORY TVARS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
n_{type}	→	{ *global* ... }
{ n_{type} ...}	→	{ *global* ... }

See also: PVARS, TYPE, VARS

TVM

Type: Command

Description: TVM Menu Command: Displays the TVM Solver menu.

Access: (CAT) TVM

Input: None

Output: None

See also: AMORT, TVMBEG, TVMEND, TVMROOT

TVMBEG

Type: Command

Description: Payment at Start of Period Command: Specifies that TVM calculations treat payments as being made at the begining of the compounding periods.

Access: (CAT) TVMBEG

Input: None

Output: None

See also: AMORT, TVM, TVMEND, TVMROOT

TVMEND

Type: Command

Description: Payment at End of Period Command: Specifies that TVM calculations treat payments as being made at the end of the compounding periods.

Access: CAT TVMEND

Input: None

Output: None

See also: AMORT, TVM, TVMBEG, TVMROOT

TVMROOT

Type: Command

Description: TVM Root Command: Solves for the specified TVM variable using values from the remaining TVM variables.

Access: CAT TVMROOT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
'TVM variable'	→	$x_{\text{TVM variable}}$

See also: AMORT, TVM, TVMBEG, TVMEND

TYPE

Type: Command

Description: Type Command: Returns the type number of an object.

The following table lists object types and their type numbers.

Object Type Numbers

Object Type	Number
User objects:	
Real number	0
Complex number	1
Character string	2
Real array	3
Complex array	4
List	5
Global name	6
Local name	7
Program	8
Algebraic object	9
Binary integer	10
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory	15

Object Type (Continued)	Number
Library	16
Backup object	17
Real integer	28
Font	30
Built-in Commands:	
Built-in function	18
Built-in command	19
System Objects:	
System binary	20
Extended real	21
Extended complex	22
Linked array	23
Character	24
Code object	25
Library data	26
Mini font	27
Integer number	28
Symbolic vector/matrix	29
Font	30
Extended object	31

T**Access:** ↰ PRG TEST TYPE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
obj	→	n_{type}

See also: SAME, TVARS, VTYPE, ==

UBASE

Type: Function

Description: Convert to SI Base Units Function: Converts a unit object to SI base units.

Access: ↰ CONVERT UNITS TOOLS UBASE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	*y_base-units*
'*symb*'	→	'UBASE(*symb*)'

See also: CONVERT, UFACT, →UNIT, UVAL

UFACT

Type: Command

Description: Factor Unit Command: Factors the level 1 unit from the unit expression of the level 2 unit object.

Access: ↰ CONVERT UNITS TOOLS UFACT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_1_$unit_1$	x_2_$unit_2$	→	x_3_$unit_2$*$unit_3$

See also: CONVERT, UBASE, →UNIT, UVAL

UFL1→MINIF

Type: Command

.Description: Converts a UFL1 (universal font library) fontset to a HP 49G minifont.

You specify the fontset and give it an ID (0–255). The font must be a 6-by-4 font.

Access: CAT UFL1→MINIF

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_{fontset}	n_{ID}	→	The font converted to minifont.

→UNIT

Type: Command

Description: Stack to Unit Object Command: Creates a unit object from a real number and the unit part of a unit object.

→UNIT adds units to a real number, combining the number and the unit part of a unit object (the numerical part of the unit object is ignored). →UNIT is the reverse of OBJ→ applied to a unit object.

Access: CAT →UNIT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	*y_unit*	→	*x_unit*

See also: →ARRY, →LIST, →STR, →TAG

UNPICK

Type: RPN Command

Description: Replaces the object at level $n+2$ with the object at level 2 and deletes the objects at levels 1 and 2.

Access: ↰ PRG STACK UNPICK

Input/Output:

L_{n+2}	L_{n+1}	L_3	L_2	L_1		L_n	L_{n-1}	L_1
obj_n	obj_{n-1}	obj_1	obj	n	→	obj	obj_{n-1}	obj_1

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

UNROT

Type: RPN Command

Description: Changes the order of the first three objects on the stack. The order of the change is the opposite to that of the ROT command.

Access: ↰ PRG STACK UNROT

Input/Output:

L_3	L_2	L_1		L_3	L_2	L_1
obj_3	obj_2	obj_1	→	obj_1	obj_3	obj_2

See also: OVER, PICK, ROLL, ROLLD, SWAP, ROT

UNTIL

Type: Command

Description: UNTIL Command: Startsthe test clause in a DO … UNTIL … END indefinite loop structure.

See the DO entry for more information.

Access: ↰ PRG BRANCH UNTIL

Input: None

Output: None

See also: DO, END

UPDIR

Type: Command

Description: Up Directory Command: Makes the parent of the current directory the new current directory. UPDIR has no effect if the current directory is *HOME*.

Access: ↰ UPDIR

Input: None

Output: None

See also: CRDIR, HOME, PATH, PGDIR

UTPC

Type: Command

Description: Upper Chi-Square Distribution Command: Returns the probability utpc(*n*, *x*) that a chi-square random variable is greater than *x*, where *n* is the number of degrees of freedom of the distribution.

The defining equations are these:

- For x ≥ 0:

$$utpc(n,x) = \left[\frac{1}{2^{\frac{n}{2}}\Gamma(\frac{n}{2})}\right]\int_x^{\infty} t^{\frac{n}{2}-1}\cdot e^{-\frac{t}{2}}dt$$

- For x < 0:

$$utpc(n,x)) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value *n* is rounded to the nearest integer and, when rounded, must be positive.

Access: ↰ MTH PROBABILITY UTPC

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	*x*	→	*utpc*(*n*,*x*)

See also: UTPF, UTPN, UTPT

UTPF

Type: Command

Description: Upper Snedecor's F Distribution Command: Returns the probability utpf(n_1, n_2, x) that a Snedecor's F random variable is greater than x, where n_1 and n_2 are the numerator and denominator degrees of freedom of the F distribution.

The defining equations for utpf(n_1, n_2, x) are these:

- For x ≥ 0:

$$\left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}}\left[\frac{\Gamma\left(\frac{n_1+n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)}\right]\int_x^{\infty} t^{\frac{n_1-2}{2}}\left[1+\left(\frac{n_1}{n_2}\right)t\right]^{-\frac{(n_1+n_2)}{2}} dt$$

- For x < 0:

$$utpf(n_1, n_2, x) = 1$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the HP 49 factorial command.

The values n_1 and n_2 are rounded to the nearest integers and, when rounded, must be positive.

Access: ↰ MTH PROBABILITY UTPF

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
n_1	n_2	x	→	$utpf(n_1,n_2,x)$

See also: UTPC, UTPN, UTPT

UTPN

Type: Command

Description: Upper Normal Distribution Command: Returns the probability utpn(m, v, x) that a normal random variable is greater than x, where m and v are the mean and variance, respectively, of the normal distribution.

For all x and m, and for $v > 0$, the defining equation is this:

$$utpn(m, v, x) = \left[\frac{1}{\sqrt{2\pi v}}\right]\int_{x}^{\infty} e^{-\frac{(t-m)^2}{2v}}\, dt$$

For $v = 0$, UTPN returns 0 for $x \geq m$, and 1 for $x < m$.

Access: ↰ MTH PROBABILITY UTPN

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
m	v	x	→	$utpn(m,v,x)$

See also: UTPC, UTPF, UTPT

UTPT

Type: Command

Description: Upper Student's t Distribution Command: Returns the probability utpt(n, x) that a Student's t random variable is greater than x, where n is the number of degrees of freedom of the distribution.

The following is the defining equation for all x:

$$utpt(n,x) = \left[\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\right]\int_x^{\infty}\left(1+\frac{t^2}{n}\right)^{-\frac{n+1}{2}}dt$$

For any value z, $\Gamma\left(\frac{z}{2}\right) = \left(\frac{z}{2}-1\right)!$, where ! is the factorial command.

The value n is rounded to the nearest integer and, when rounded, must be positive.

Access: ◁ MTH PROBABILITY UTPT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
n	x	→	$utpt(n,x)$

See also: UTPC, UTPF, UTPN

UVAL

Type: Function

Description: Unit Value Function: Returns the numerical part of a unit object.

Access: ▷ UNITS TOOLS UVAL

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x_unit	→	x
'$symb$'	→	'UVAL($symb$)'

See also: CONVERT, UBASE, UFACT, →UNIT

V→

Type: Command

Description: Vector/Complex Number to Stack Command: Separates a vector or complex number into its component elements.

For vectors with four or more elements, V→ executes *independently* of the coordinate system mode, and always returns the elements of the vector to the stack as they are stored internally (in rectangular form). Thus, V→ is equivalent to OBJ→ for vectors with four or more elements.

Access: ↰ MTH VECTOR V→

Input/Output:

L_1/A_1		L_n/I_1 ... L_3/I_{n-2}	L_2/I_{n-1}	L_1/I_n
$[\,x\,y\,]$	→		x	y
$[\,x_r, y_{theta}\,]$	→		x_r	y_{theta}
$[\,x_1, x_2, x_3\,]$	→	x_1	x_2	x_3
$[\,x_1, x_{theta}, x_z\,]$	→	x_1	x_{theta}	x_z
$[\,x_1, x_{theta}, x_{phi}\,]$	→	x_1	x_{theta}	x_{phi}
$[\,x_1, x_2, ..., x_n\,]$	→	x_1 ... x_{n-2}	x_{n-1}	x_n
(x, y)	→		x	y
(x_r, y_{theta})	→		x_r	y_{theta}

L = level; A = argument; I = item

See also: →V2, →V3

→V2

Type: Command

Description: Stack to Vector/Complex Number Command: Converts two specified numbers into a 2-element vector or a complex number.

The result returned depends on the setting of flags –16 and –19, as shown in the following table:

	Flag –19 clear	Flag –19 set
Flag –16 clear (Rectangular mode)	[x y]	(x, y)
Flag –16 set (Polar mode)	[x ∠ y]	(x, ∠ y)

Access: ↰ MTH VECTOR →V2

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	[x y]
x	y	→	[x ∠ y]
x	y	→	(x, y)
x	y	→	(x, ∠ y)

See also: V→, →V3

→V3

Type: Command

Description: Stack to 3-Element Vector Command: Converts three numbers into a 3-element vector.

The result returned depends on the coordinate mode used, as shown in the following table:

Mode	Result
Rectangular (flag –16 clear)	[x_1 x_2 x_3]
Polar/Cylindrical (flag –15 clear and –16 set)	[x_1 x$\angle_{theta}$ x_z]
Polar/Spherical (flag –15 and –16 set)	[x_1 x$\angle_{theta}$ x$\angle_{phi}$]

Access: ↰ MTH VECTOR →V3

Input/Output:

Level 3/Argument 1	Level 2/Argument 2	Level 1/Argument 3		Level 1/Item 1
x_1	x_2	x_3	→	[x_1 x_2 x_3]
x_1	x_{theta}	x_z	→	[x_1 $\angle x_{theta}$ x_z]
x_1	x_{theta}	x_{phi}	→	[x_1 $\angle x_{theta}$ x_{phi}]

See also: V→, →V2

VAR

Type: Command

Description: Variance Command: Calculates the sample variance of the coordinate values in each of the *m* columns in the current statistics matrix (Σ*DAT*).

The variance (equal to the square of the standard deviation) is returned as a vector of *m* real numbers, or as a single real number if $m = 1$. The variances are computed using this formula:

$$\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$$

where x_i is the *i*th coordinate value in a column, $\bar{x}$ is the mean of the data in this column, and *n* is the number of data points.

Access: (CAT) VAR

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	$x_{variance}$
	→	$[\, x_{variance1}, \ldots, x_{variancem} \,]$

See also: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, SDEV, TOT

VARS

Type: Command

Description: Variables Command: Returns a list of the names of all variables in the VAR menu for the current directory.

Access: (←) (PRG) MEMORY DIRECTORY VARS

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	{ *global*$_1$ ... *global*$_n$ }

See also: ORDER, PVARS, TVARS

VERSION

Type: Command

Description: Software Version Command: Displays the software version and copyright message.

Access: (CAT) VERSION

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
	→	"*version number*"	"*copyright message*"

Flags: None

VISIT

Type: Command

.Description: For a specified variable, opens the contents in the command-line editor.

Access: CAT VISIT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	The contents opened in the command line editor.

See also: VISITB, EDIT, EDITB

VISITB

Type: Command

.Description: For a specified variable, opens the contents in the most suitable editor for the object type. For example, if the specified variable holds an equation, the equation is opened in Equation Writer.

Access: CAT VISITB

Input/Output:

Level 1/Argument 1		Level 1/Item 1
A variable name	→	The contents opened in the most suitable editor.

See also: VISITB, EDIT, EDITB

VTYPE

Type: Command

Description: Variable Type Command: Returns the type number of the object contained in the named variable.

If the named variable does not exist, VTYPE returns –1.

For a table of the objects' type numbers, see the entry for TYPE.

Access: ↰ PRG TYPE VTYPE

Input/Output:

Level 1/Argument 1		Level 1/Item 1
$'name'$	$\rightarrow$	n_{type}
$:n_{\text{port}} : name_{\text{backup}}$	$\rightarrow$	n_{type}
$:n_{\text{port}} : n_{\text{library}}$	$\rightarrow$	n_{type}

See also: TYPE

W to Z

WAIT

Type: Command

Description: Wait Command: Suspends program execution for specified time, or until a key is pressed.

The function of WAIT depends on the argument, as follows:

- Argument x interrupts program execution for x seconds.
- Argument 0 suspends program execution until a valid key is pressed (see below). WAIT then returns x_{key}, which defines where the pressed key is on the keyboard, and resumes program execution.

  x_{key} is a three-digit number that identifies a key's location on the keyboard. See the entry for ASN for a description of the format of x_{key}.
- Argument –1 works as with argument 0, except that the currently specified menu is also displayed.

◁, ▷, ALPHA, ALPHA◁, and ALPHA▷ are not by themselves valid keys.

Arguments 0 and –1 do not affect the display, so that messages persist even though the keyboard is ready for input (FREEZE is not required).

Normally, the MENU command does not update the menu keys until a program halts or ends. WAIT with argument –1 enables a previous execution of MENU to display that menu while the program is suspended for a key press.

Access: ◁ PRG IN WAIT

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	
0	→	x_{key}
–1	→	x_{key}

See also: KEY

WHILE

Type: Command Operation

Description: WHILE Indefinite Loop Structure Command: Starts the WHILE … REPEAT … END indefinite loop structure.

WHILE … REPEAT … END repeatedly evaluates a test and executes a loop clause if the test is true. Since the test clause occurs before the loop-clause, the loop clause is never executed if the test is initially false. The syntax is this:

WHILE *test-clause* REPEAT *loop-clause* END

The test clause is executed and must return a test result to the stack. REPEAT takes the value from the stack. If the value is not zero, execution continues with the loop clause; otherwise, execution resumes following END.

Access: ↰ PRG BRANCH WHILE

Input/Output:

Level 1/Argument 1			Level 1/Item 1
WHILE		→	
REPEAT	*T/F*	→	
END		→	

See also: DO, END, REPEAT

WIREFRAME

Type: Command

Description: WIREFRAME Plot Type Command: Sets the plot type to WIREFRAME.

When the plot type is set to WIREFRAME, the DRAW command plots a perspective view of the graph of a scalar function of two variables. WIREFRAME requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

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

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

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

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

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

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

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

Access: CAT WIREFRAME

Input: None

Output: None

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

WSLOG

Type: Command

Description: Warmstart Log Command: Returns four strings recording the date, time, and cause of the four most recent warmstart events.

Each string "log_n" has the form "*code–date time*". The following table summarizes the legal values of *code* and their meanings.

Code	Description
0	The warmstart log was cleared by pressing ON SPC.
1	The interrupt system detected a very low battery condition at the battery contacts (not the same as a low system voltage), and put the calculator in "Deep Sleep mode" (*with the system clock running*). When ON is pressed after the battery voltage is restored, the system warmstarts and puts a 1 in the log.
2	Hardware failed during transmission (timeout).
3	Run through address 0.
4	System time is corrupt
5	A Deep Sleep wakeup (for example, ON, Alarm).
6	Not used
7	A 5-nibble word (CMOS test word) in RAM was corrupt. (This word is checked on every interrupt, but it is used only as an indicator of potentially corrupt RAM.)
8	Not used
9	The alarm list is corrupt.
A	System RPL jump to #0.

Code	Description (Continued)
B	The card module was removed (or card bounce).
C	Hardware reset occurred (for example, an electrostatic discharge or user reset)
D	An expected System (RPL) error handler was not found in runstream

The date and time stamp (*date time*) part of the log may be displayed as 00…0000 for one of three reasons:

- The system time was corrupt when the stamp was recorded.
- The date and time stamp itself is corrupt (bad checksum).
- Fewer than four warmstarts have occurred since the log was last cleared.

Access: CAT WSLOG

Input/Output:

Level 1/Argument 1		Level 4/Item 1 ... Level 1/Item 4
	→	"$\log_4$" ... "$\log_1$"

ΣX

Type: Command

Description: Sum of *x*-Values Command: Sums the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

Access: CAT ΣX

Input/Output:

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

See also: NΣ, XCOL, ΣXY, ΣX2, ΣY, ΣY2

ΣX2

Type: Command

Description: Sum of Squares of x-Values Command: Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

Access: CAT ΣX2

Input/Output:

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

See also: NΣ, ΣX, XCOL, ΣX*Y, ΣY, ΣY^2

XCOL

Type: Command

Description: Independent Column Command: Specifies the independent-variable column of the current statistics matrix (reserved variable Σ*DAT*).

The independent-variable column number is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1.

XCOL will accept a noninteger real number and store it in Σ*PAR*, but subsequent commands that utilize the XCOL specification in Σ*PAR* will cause an error.

Access: CAT XCOL

Input/Output:

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

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

XGET

Type: Command

Description: XModem Get Command: Retrieves a specified filename via XMODEM from another calculator. The other calculator needs to be in server mode for the operation to work (APPS I/O FUNCTIONS START SERVER.

Access: CAT

Input/Output:

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

See also: BAUD, RECN, RECV, SEND XRECV, XSERV, XPUT

XMIT

Type: Command

Description: Serial Transmit Command: Sends a string serially without using Kermit protocol, and returns a single digit that indicates whether the transmission was successful.

XMIT is useful for communicating with non-Kermit devices such as RS-232 printers.

If the transmission is successful, XMIT returns a 1. If the transmission is not successful, XMIT returns the unsent portion of the string and a 0. Use ERRM to get the error message.

After receiving an XOFF command (with *transmit pacing* in the reserved variable *IOPAR* set), XMIT stops transmitting and waits for an XON command. XMIT resumes transmitting if an XON is received before the time-out set by STIME elapses; otherwise, XMIT terminates, returns a 0, and stores "Timeout" in ERRM.

Access: CAT XMIT

Input/Output:

Level 1/Argument 1		Level 2/Item 1	Level 1/Item 2
"*string*"	→		1
"*string*"	→	"$substring_{unsent}$"	0

See also: BUFLEN, SBRK, SRECV, STIME

XOR

Type: Function

Description: Exclusive OR Function: Returns the logical exclusive OR of two arguments.

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

- Binary integer arguments are treated as sequences of bits with length equal to the current wordsize. Each bit in the result is determined by comparing the corresponding bits (bit_1 and bit_2) in the two arguments, as shown in the following table:

bit_1	bit_2	bit_1 **XOR** bit_2
0	0	0
0	1	1
1	0	1
1	1	0

- String arguments are treated as sequences of bits, using 8 bits per character (that is, using the binary version of the character code). The two string arguments must be the same length.

When the arguments are real numbers or symbolics, XOR simply does a true/false test. The result is 1 (true) if either, but not both, arguments are nonzero; it is 0 (false) if both arguments are nonzero or zero. This test is usually done to compare two test results.

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

Access: ▶ BASE LOGIC XOR

◀ MTH BASE LOGIC XOR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	$\#n_2$	→	$\#n_3$

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
"*string*$_1$"	"*string*$_2$"	→	"*string*$_3$"
T/F$_1$	*T/F*$_2$	→	0/1
T/F	'*symb*'	→	'*T/F* XOR *symb*'
'*symb*'	*T/F*	→	'*symb* XOR *T/F*'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ XOR *symb*$_2$'

See also: AND, NOT, OR

XPON

Type: Function

Description: Exponent Function: Returns the exponent of the argument.

Access: ↰ MTH REAL XPON

Input/Output:

Level 1/Argument 1		Level 1/Item 1
x	→	n_{expon}
'*symb*'	→	'XPON(*symb*)'

See also: MANT, SIGN

XPUT

Type: Command

Description: XModem Send Command: Sends a specified filename via XMODEM to a calculator. The receiving calculator needs to be in Server mode (APPS I/O FUNCTIONS START SERVER).

Access: CAT

Input/Output:

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

See also: BAUD, RECN, RECV, SEND XRECV, XSERV, XGET

XRECV

Type: Command

Description: XModem Receive Command: Prepares the HP 49 to receive an object via XModem. The received object is stored in the given variable name.

The transfer will start more quickly if you start the XModem sender *before* executing XRECV.

Invalid object names cause an error. If flag –36 is clear, object names that are already in use also cause an error.

If you are transferring data between two HP 49s, executing {AAA BBB CCC} XRECV receives *AAA*, *BBB*, and *CCC*. You also need to use a list on the sending end ({AAA BBB CCC} XSEND, for example).

Access: CAT XRECV

Input/Output:

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

See also: BAUD, RECV, RECN, SEND, XSEND

XRNG

Type: Command

Description: x-Axis Display Range Command: Specifies the *x*-axis display range.

The *x*-axis display range is stored in the reserved variable *PPAR* as x_{min} and x_{max} in the complex numbers (x_{min},y_{min}) and (x_{max},y_{max}). These complex numbers are the first two elements of *PPAR* and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of x_{min} and x_{max} are –6.5 and 6.5, respectively.

Access: CAT XRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{min}	x_{max}	→	

See also: AUTO, PDIM, PMAX, PMIN, YRNG

XROOT

Type: Analytic function

Description: *x*th Root of *y* Command: Computes the *x*th root of a real number.

XROOT is equivalent to $y^{1/x}$, but with greater accuracy.

If $y < 0$, x must be an integer.

Access: ↱ $\sqrt[x]{y}$

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y	*x*	→	$\sqrt[x]{y}$
'*symb*$_1$'	'*symb*$_2$'	→	'XROOT(*symb*$_2$,*symb*$_1$)'
'*symb*'	*x*	→	'XROOT(*x*,*symb*)'
y	'*symb*'	→	'XROOT(*symb*,*y*)'
y_unit	*x*	→	$\sqrt[x]{y}$ *_unit*$^{1/x}$
y_unit	'*symb*'	→	'XROOT(*symb*,*y_unit*)'

XSEND

Type: Command

Description: XModem Send Command: Sends a copy of the named object via XModem.

A receiving HP 49 must execute XRECV to receive an object via XModem.

The transfer occurs more quickly if you start the receiving XModem *after* executing XSEND. Also, configuring the receiving modem *not* to do CRC checksums (if possible) will avoid a 30 to 60-second delay when starting the transfer.

If you are transferring data between two HP 49s, executing {AAA BBB CCC} XSEND sends *AAA*, *BBB*, and *CCC*. You also need to use a list on the receiving end ({AAA BBB CCC} XRECV, for example).

Access: CAT XSEND

Input/Output:

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

See also: BAUD, RECN, RECV, SEND XRECV

XSERV

Type: Command

Description: XModem Server Command: Puts the calculator in XMODEM server mode. When in server mode, the following commands are available:
P: Put a file in the calculator
G: Get a file from the calculator
E: Execute a command line
M Get the calculator memory
L: List the files in the current directory

Access: CAT XSERV

See also: BAUD, RECN, RECV, SEND XRECV, XGET, XPUT

XVOL

Type: Command

Description: X Volume Coordinates Command: Sets the width of the view volume in the reserved variable *VPAR*.

x_{left} and x_{right} set the x-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable *VPAR*.

Access: CAT XVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{left}	x_{right}	→	

See also: EYEPT, XXRNG, YVOL, YYRNG, ZVOL

XXRNG

Type: Command

Description: X Range of an Input Plane (Domain) Command: Specifies the x range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

x_{min} and x_{max} are real numbers that set the x-coordinates for the input plane. These values are stored in the reserved variable *VPAR*.

Access: CAT XXRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{min}	x_{max}	→	

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

ΣXY

Type: Command

Description: Sum of X times Y command: Sums the products of each of the corresponding values in the independent- and dependent-variable columns of the current statistical matrix (reserved variable ΣDAT). The independent column is the column designated as XCOL and the dependent column is the column designated as YCOL.

Access: ↱ STAT SUMMARY STATS

Output: The sum of the corresponding products.

See also: ΣY

ΣY

Type: Command

Description: Sum of y-Values Command: Sums the values in the dependent variable column of the current statistical matrix (reserved variable Σ*DAT*).

The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable Σ*PAR*. The default dependent variable column number is 2.

Access: CAT ΣY

Input/Output:

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

See also: NΣ, ΣX, XCOL, ΣXY, ΣX2, YCOL, ΣY2

ΣY2

Type: Command

Description: Sum of squares of Y-value command: Sums the squares of the values in the dependent-variable columns of the current statistical matrix (reserved variable ΣDAT). The dependent column is the column designated as YCOL

Access: ▷ STAT SUMMARY STATS

Output: The sum of the dependent variables.

See also: ΣXY

YCOL

Type: Command

Description: Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable Σ*DAT*).

The dependent variable column number is stored as the second parameter in the reserved variable Σ*PAR*. The default dependent variable column number is 2.

YCOL will accept a noninteger real number and store it in Σ*PAR*, but subsequent commands that utilize the YCOL specification in Σ*PAR* will cause an error.

Access: CAT YCOL

Input/Output:

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

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

YRNG

Type: Command

Description: y-Axis Display Range Command: Specifies the y-axis display range.

The y-axis display range is stored in the reserved variable *PPAR* as y_{min} and y_{max} in the complex numbers (x_{min},y_{min}) and (x_{max},y_{max}). These complex numbers are the first two elements of *PPAR* and specify the coordinates of the lower left and upper right corners of the display ranges.

The default values of y_{min} and y_{max} are –3.1 and 3.2, respectively.

Access: CAT YRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{min}	y_{max}	→	

See also: AUTO, PDIM, PMAX, PMIN, XRNG

YSLICE

Type: Command

Description: Y-Slice Plot Command: Sets the plot type to YSLICE.

When plot type is set YSLICE, the DRAW command plots a slicing view of a scalar function of two variables. YSLICE requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

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

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

- x_{left} and x_{right} are real numbers that specify the width of the view space.
- y_{near} and y_{far} are real numbers that specify the depth of the view space.
- z_{low} and z_{high} are real numbers that specify the height of the view space.
- x_{min} and x_{max} are not used.
- y_{min} and y_{max} are not used.
- x_{eye}, y_{eye}, and z_{eye} are real numbers that specify the point in space from which the graph is viewed.
- x_{step} determines the interval between plotted x-values within each "slice".

- y_{step} determines the number of slices to draw.

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

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

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

- (x_{min},y_{min}) is not used.
- (x_{max},y_{max}) is not used.
- *indep* is a name specifying the independent variable. The default value of *indep* is *X*.
- *res* is a real number specifying the interval, in user-unit coordinates, between plotted values of the independent variable; or a binary integer specifying the interval in pixels. The default value is 0, which specifies an interval of 1 pixel.
- *axes* is not used.
- *ptype* is a command name specifying the plot type. Executing the command YSLICE places YSLICE in *ptype*.
- *depend* is a name specifying the dependent variable. The default value is *Y*.

Access: (CAT) YSLICE

Input: None

Output: None

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

YVOL

Type: Command

Description: Y Volume Coordinates Command: Sets the depth of the view volume in the reserved variable *VPAR*.

The variables y_{near} and y_{far} are real numbers that set the *y*-coordinates for the view volume used in 3D plots. y_{near} must be less than y_{far}. These values are stored in the reserved variable *VPAR*.

Access: (CAT) YVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YYRNG, ZVOL

YYRNG

Type: Command

Description: Y Range of an Input Plane (Domain) Command: Specifies the y range of an input plane (domain) for GRIDMAP and PARSURFACE plots.

The variables $y_{\text{y near}}$ and $y_{\text{y far}}$ are real numbers that set the y-coordinates for the input plane. These values are stored in the reserved variable *VPAR*.

Access: CAT YYRNG

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
y_{near}	y_{far}	→	

See also: EYEPT, XVOL, XXRNG, YVOL, ZVOL

ZFACTOR

Type: Function

Description: Gas Compressibility Z Factor Function: Calculates the gas compressibility correction factor for non-ideal behavior of a hydrocarbon gas.

x_{Tr} is the reduced temperature: the ratio of the actual temperature (T) to the pseudocritical temperature (T_c). (Calculate the ratio using absolute temperatures.) x_{Tr} must be between 1.05 and 3.0.

y_{Pr} is the reduced pressure: the ratio of the actual pressure (P) to the pseudocritical pressure (P_c). y_{Pr} must be between 0 and 30.

x_{Tr} and y_{Pr} must be real numbers or unit objects that reduce to dimensionless numbers.

Access: CAT ZFACTOR

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{Tr}	y_{Pr}	→	$x_{ZFactor}$
x_{Tr}	'*symb*'	→	'ZFACTOR(x_{Tr},*symb*)'
'*symb*'	y_{Pr}	→	'ZFACTOR(*symb*,y_{Pr})'
'$symb_1$'	'$symb_2$'	→	'ZFACTOR($symb_1$,$symb_2$)'

ZVOL

Type: Command

Description: Z Volume Coordinates Command: Sets the height of the view volume in the reserved variable *VPAR*.

x_{low} and x_{high} are real numbers that set the z-coordinates for the view volume used in 3D plots. These values are stored in the reserved variable *VPAR*.

Access: CAT ZVOL

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_{low}	x_{high}	→	

See also: EYEPT, XVOL, XXRNG, YVOL, YYRNG

---

^

Type: Function

Description: Power Analytic Function: Returns the value of the level 2 object raised to the power of the level 1 object.

If either argument is complex, the result is complex.

The branch cuts and inverse relations for w^z are determined by this relationship:

$$w^z = \exp(z(\ln w))$$

Access: y^x

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
w	*z*	→	*wz*
z	'*symb*'	→	'*z*^(*symb*)'
'*symb*'	*z*	→	'(*symb*)^*z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$^('$symb_2$)'
x_unit	*y*	→	*x^y_unit*
x_unit	'*symb*'	→	'(*x_unit*)^(*symb*)'

See also: EXP, ISOL, LN, XROOT

| (Where)

Type: Function

Description: Where Function: Substitutes values for names in an expression.

| is used primarily in algebraic objects, where its syntax is:

'$symb_{old}$ | ($name_1$ = $symb_1$, $name_2$ = $symb_2$ …)'

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as ∫ and ∂) can then extract substitution information from local variables and include that information in the expression, avoiding the problem that would occur if the local variables no longer existed when the local names were finally evaluated.

Access:

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
'$symb_{old}$'	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	'$symb_{new}$'
x	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	*x*
(*x*,*y*)	{ $name_1$, '$symb_1$', $name_2$, '$symb_2$' … }	→	(*x*,*y*)

See also: APPLY, QUOTE

√

Type: Function

Description: Square Root Analytic Function: Returns the (positive) square root of the argument.

For a complex number (x_1, y_1), the square root is this complex number:

$$(x_2, y_2) = \left(\sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\right)$$

where r = ABS (x_1, y_1), and θ = ARG (x_1, y_1).

If $(x_1, y_1) = (0,0)$, then the square root is (0, 0).

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

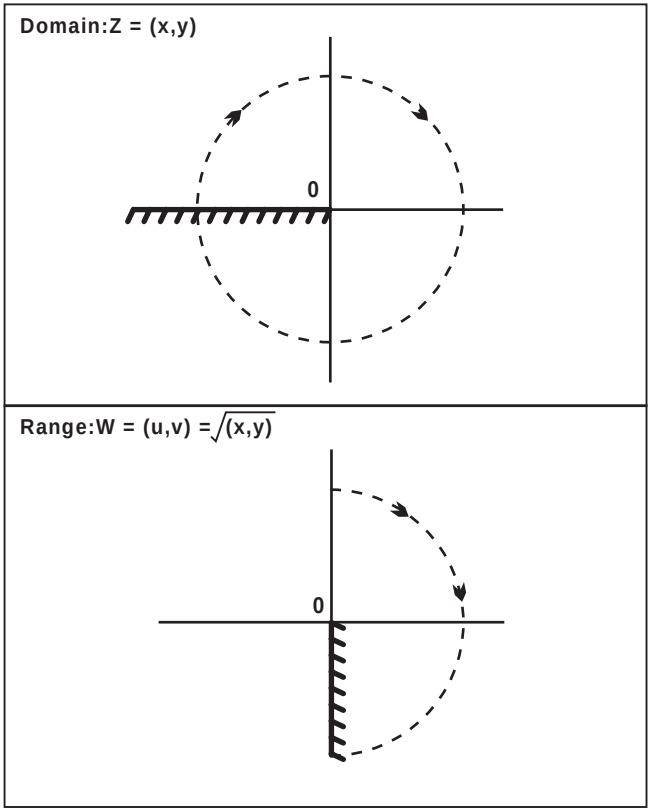
's1*√Z'

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

1. each argument is sent to a distinct result, and
2. each possible result is achieved. The points in this restricted domain of SQ are called the *principal values* of the inverse relation. The √ function in its entirety is called the *principal branch* of the inverse relation, and the points sent by √ to the boundary of the restricted domain of SQ form the *branch cuts* of √.

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

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



These graphs show the inverse relation 's1*√Z' for the case *s1*=1. For the other value of *s1*, the half-plane in the lower graph is rotated. Taken together, the half-planes cover the whole complex plane, which is the domain of SQ.

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

Access: $\boxed{\sqrt{x}}$

Input/Output:

Level 1/Argument 1		Level 1/Item 1
z	→	$\sqrt{z}$
x_unit	→	$\sqrt{x}\ unit^{1/2}$
'*symb*'	→	'$\sqrt{(symb)}$'

See also: SQ, ^, ISOL

∫

Type: Function

Description: Integral Function: Integrates an *integrand* from *lower limit* to *upper limit* with respect to a specified variable of integration.

The algebraic syntax for ∫ parallels its stack syntax:

∫(*lower limit*, *upper limit*, *integrand*, *name*)

where *lower limit*, *upper limit*, and *integrand* can be real or complex numbers, unit objects, names, or algebraic expressions.

Evaluating ∫ in Symbolic Results mode (flag –3 *clear*) returns a symbolic result. Some functions that the The HP 49G can integrate are as follows:

- All built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, SIN can be integrated since its antiderivative, COS, is a built-in function. The arguments for these functions must be linear.
- Sums, differences, and negations of built-in functions whose antiderivatives can be expressed in terms of other built-in functions—for example, 'SIN(X)–COS(X)'.
- Derivatives of all built-in functions—for example, 'INV(1+X^2)' can be integrated because it is the derivative of the built-in function ATAN.
- Polynomials whose base term is linear—for example, 'X^3+X^2–2*X+6' can be integrated since X is a linear term. '(X^2–6)^3+(X^2–6)^2' cannot be integrated since X^2–6 is not linear.
- Selected patterns composed of functions whose antiderivatives can be expressed in terms of other built-in functions—for example, '1/(COS(X)*SIN(X))' returns 'LN(TAN(X))'.

If the result of the integration is an expression with no integral sign in the result, the symbolic integration was successful. If, however, the result still contains an integral sign, try rearranging the expression and evaluating again, or estimate the answer using numerical integration.

Evaluating ∫ in Numerical Results mode (flag –3 *set*) returns a numerical approximation, and stores the error of integration in variable *IERR*. ∫ consults the number format setting to determine how accurately to compute the result.

Access: ↱ ∫

Input/Output:

L4/A1	L3/A2	L2/A3	L1/A4		L1/I1
lower limit	*upper limit*	*integrand*	'*name*'	→	'$symb_{integral}$'

L = level; A = Argument; I = Item

See also: TAYLR, ∂, Σ

Σ

Type: Function

Description: Summation Function: Calculates the value of a finite series.

The summand argument *smnd* can be a real number, a complex number, a unit object, a local or global name, or an algebraic object.

The algebraic syntax for Σ differs from the stack syntax. The algebraic syntax is:

'Σ(*indx*=*initial*,*final*,*summand*)'

Access: ↱ Σ

nput/Output:

L4/A1	L3/A2	L2/A3	L1/A4		L1/I1
'*indx*'	x_{init}	x_{final}	*smnd*	→	x_{sum}
'*indx*'	'*init*'	x_{final}	*smnd*	→	'$\Sigma(indx = init, x_{final}, smnd)$'
'*indx*'	x_{init}	'*final*'	*smnd*	→	'$\Sigma(indx = x_{init}, final, smnd)$'
'*indx*'	'*init*'	'*final*'	*smnd*	→	'$\Sigma(indx = init, final, smnd)$'

L = level; A = Argument; I = Item

See also: TAYLR, ∫, ∂

Σ+

Type: Command

Description: Sigma Plus Command: Adds one or more data points to the current statistics matrix (reserved variable Σ*DAT*).

For a statistics matrix with *m* columns, arguments for Σ+ can be entered several ways:

- To enter one data point with a single coordinate value, the argument for Σ+ must be a real number.
- To enter one data point with multiple coordinate values, the argument for Σ+ must be a vector with *m* real coordinate values.
- To enter several data points, the argument for Σ+ must be a matrix of *n* rows of *m* real coordinate values.

In each case, the coordinate values of the data point(s) are added as new rows to the current statistics matrix (reserved variable Σ*DAT*). If Σ*DAT* does not exist, Σ+ creates an *n* x *m* matrix and stores the matrix in Σ*DAT*. If Σ*DAT* does exist, an error occurs if it does not contain a real matrix, or if the number of coordinate values in each data point entered with Σ+ does not match the number of columns in the current statistics matrix.

Once Σ*DAT* exists, individual data points of *m* coordinates can be entered as *m* separate real numbers or an *m*-element vector.

LASTARG returns the *m*-element vector in either case.

Access: CAT Σ+

Input/Output:

$L_m/A_1 \ldots L_2/A_{m-1}$	L_1/A_m		L_1/I_1
	x	→	
	$[\, x_1, x_2, \ldots, x_m \,]$	→	
	$[[\, x_{1\,1}, \ldots, x_{1\,m} \,]\, [\, x_{n\,1}, \ldots, x_{n\,m} \,]]$	→	
$x_1 \ldots x_{m-1}$	x_m	→	

L = level; A = Argument; I = Item

See also: CLΣ, RCLΣ, STOΣ, Σ–

Σ–

Type: Command

Description: Sigma Minus Command: Returns a vector of *m* real numbers (or one number *x* if *m* = 1) corresponding to the coordinate values of the last data point entered by Σ+ into the current statistics matrix (reserved variable Σ*DAT*).

The last row of the statistics matrix is deleted.

The vector returned by Σ– can be edited or replaced, then restored to the statistics matrix by Σ+.

Access: CAT Σ–

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	*x*
	→	[x_1 x_2 … x_m]

See also: CLΣ, RCLΣ, STOΣ, Σ+

π

Type: Function

Description: π Function: Returns the symbolic constant 'π' or its numerical representation, 3.14159265359.

The number returned for π is the closest approximation of the constant π to 12-digit accuracy.

In Radians mode with flag –2 and –3 clear (to return symbolic results), trigonometric functions of π and π/2 are automatically simplified. For example, evaluating 'SIN(π)' returns zero. However, if flag –2 or flag –3 is set (to return numerical results), then evaluating 'SIN(π)' returns the numerical approximation –2.06761537357E–13.

Access: ↰ π

Input/Output:

Level 1/Argument 1		Level 1/Item 1
	→	'π'
	→	3.14159265359…

See also: e, i, MAXR, MINR, →Qπ

∂

Type: Function

Description: Derivative Function: Takes the derivative of an expression, number, or unit object with respect to a specified variable of differentiation.

When executed in stack syntax, ∂ executes a *complete* differentiation: the expression '$symb_1$' is evaluated repeatedly until it contains no derivatives. As part of this process, if the variable of differentiation *name* has a value, the final form of the expression substitutes that value substituted for all occurrences of the variable.

The algebraic syntax for ∂ is '∂*name*(*symb*$_1$'). When executed in algebraic syntax, ∂ executes a *stepwise* differentiation of *symb*$_1$, invoking the chain rule of differentiation— the result of one evaluation of the expression is the derivative of the argument expression *symb*$_1$, multiplied by a new subexpression representing the derivative of *symb*$_1$'s argument.

If ∂ is applied to a function for which theHP 49G does not provide a derivative, ∂ returns a new function whose name is *der* followed by the original function name.

Access: ⮡ ∂

Input/Output:

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

See also: TAYLOR, ∫, Σ

≤

Type: Function

Description: Less Than or Equal Function: Tests whether one object is less than or equal to another object.

The function ≤ returns a true test result (1) if the first argument is less than or equal to the lsecond argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≤ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, “less than or equal” means numerically equal or smaller (1 is less than 2). For real numbers, “less than or equal” also means equally or more negative (–2 is less than –1).

For strings, “less than or equal” means alphabetically equal or previous (“ABC” is less than or equal to “DEF”; “AAA” is less than or equal to “AAB”; “A” is less than or equal to “AA”). In general, characters are ordered according to their character codes. This means, for example, that “B” is less than “a”, since “B” is character code 66, and “a” is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperature and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰≤

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
“$string_1$”	“$string_2$”	→	0/1
x	'$symb$'	→	'$x \le symb$'
'$symb$'	x	→	'$symb \le x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \le symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit \le symb$'
'$symb$'	x_unit	→	'$symb \le x\_unit$'

See also: <, >, ≥, ==, ≠

≥

Type: Function

Description: Greater Than or Equal Function: Tests whether one object is greater than or equal to another object.

The function ≥ returns a true test result (1) if the first argument is greater than or equal to the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, ≥ returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than or equal to" means numerically equal or greater (2 is greater than or equal to 1). For real numbers, "greater than or equal to" also means equally or less negative (–1 is greater than or equal to –2).

For strings, "greater than or equal to" means alphabetically equal or subsequent ("DEF" is greater than or equal to "ABC"; "AAB" is greater than or equal to "AAA"; "AA" is greater than or equal to "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than or equal to "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰ ≥

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x \geq symb$'
'$symb$'	x	→	'$symb \geq x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \geq symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit \geq symb$'
'$symb$'	x_unit	→	'$symb \geq x\_unit$'

See also: <, ≤, >, ==, ≠

≠

Type: Function

Description: Not Equal Function: Tests if two objects are not equal.

The function ≠ returns a true result (1) if the two objects have different values, or a false result (0) otherwise. (Lists and programs are considered to have the same values if the objects they contain are identical.)

If one object is algebraic or a name, and the other is a number, a name, or algebraic, ≠ returns a symbolic comparison expression that can be evaluated to return a test result.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number, so, for example, 6 and (6,0) and considered to be equal..

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↰ ≠

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	0/1
$(x,0)$	x	→	0/1
x	$(x,0)$	→	0/1
$\tilde{z}$	'*symb*'	→	'z ≠ *symb*'
'*symb*'	$\tilde{z}$		'*symb* ≠ $\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \neq symb_2$'

See also: SAME, TYPE, <, ≤, >,≥, ==, ≠

→

Type: Command

Description: Create Local Variables Command: Creates local variables.

Local variable structures specify one or more local variables and a defining procedure.

A local variable structure consists of the → command, followed by one or more names, followed by a defining procedure—either a program or an algebraic. The → command stores objects into local variables with the specified names. The resultant *local variables* exist only while the defining procedure is being executed. The syntax of a local variable structure is one of the following:

- → $name_1$ $name_2$ … $name_n$ « *program* »
- → $name_1$ $name_2$ … $name_n$ '*algebraic expression*'

Access: ⮡⮕

IInput/Output:

$Level_n$/$Argument_1$ … $Level_1$/$Argument_n$		Level 1/Item 1
obj_1 … obj_n	→	

See also: DEFINE, STO

!

Type: Function

Description: Factorial (Gamma) Function: Returns the factorial $n!$ of a positive integer argument n, or the gamma function $\Gamma(x+1)$ of a non-integer argument x.

For $x \geq 253.1190554375$ or $n < 0$, ! causes an overflow exception (if flag –21 is set, the exception is treated as an error). For non-integer $x \leq -254.1082426465$, ! causes an underflow exception (if flag –20 is set, the exception is treated as an error).

In algebraic syntax, ! follows its argument. Thus the algebraic syntax for the factorial of 7 is 7!.

For non-integer arguments x, $x! = \Gamma(x + 1)$, defined for $x > -1$ as:

$$\Gamma(x+1) = \int_0^\infty e^{-t} t^x dt$$

and defined for other values of x by analytic continuation:

$$\Gamma(x + 1) = x\,\Gamma(x)$$

Access: ↰ MTH PROBABILITY !

Input/Output:

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

See also: COMB, PERM

%

Type: Function

Description: Percent Function: Returns x percent of y.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, –, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

For more information on using temperature units with arithmetic functions, see the entry for +.

Access: ◁ MTH REAL %

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	$xy/100$
x	'*symb*'	→	'%(x,*symb*)'
'*symb*'	x	→	'%(*symb*,x)'
'$symb_1$'	'$symb_2$'	→	'%($symb_1$, $symb_2$)'
x	*y_unit*	→	($xy/100$)_*unit*
x_unit	y	→	($xy/100$)_*unit*
'*symb*'	*x_unit*	→	'%(*symb*,*x_unit*)'
x_unit	'*symb*'	→	'%(*x_unit*,*symb*)'

See also: %CH, %T

*****

Type: Function

Description: Multiply Analytic Function: Returns the product of the arguments.

The product of a real number a and a complex number (x, y) is the complex number (xa, ya).

The product of two complex numbers (x_1, y_1) and (x_2, y_2) is the complex number $(x_1 x_2 - y_1 y_2, x_1 y_2 + x_2 y_1)$.

The product of a real array and a complex array or number is a complex array. Each element x of the real array is treated as a complex element $(x, 0)$.

Multiplying a matrix by an array returns a matrix product. The matrix must have the same number of columns as the array has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a *row* of numbers, the HP 49G treats a vector as an $n \times 1$ matrix when multiplying matrices or computing matrix norms.

Multiplying a binary integer by a real number returns a binary integer that is the product of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the multiplication.)

The product of two binary integers is truncated to the current binary integer wordsize.

When multiplying two unit objects, the scaler parts and the unit parts are multiplied separately.

Access: CAT *

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 \tilde{z}_2$
[[*matrix*]]	[*array*]	→	[[*matrix* × *array*]]
$\tilde{z}$	[*array*]	→	[$\tilde{z}$ × *array*]
[*array*]	$\tilde{z}$	→	[*array* × $\tilde{z}$]
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ * *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* * $\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ *$symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit	*y_unit*	→	$xy\_unit_x \times unit_y$
x	*y_unit*	→	*xy_unit*
x_unit	*y*	→	*xy_unit*
'*symb*'	*x_unit*	→	'*symb* * *x_unit*'
x_unit	'*symb*'	→	'*x_unit* * *symb*'

See also: +, –, /, =

+

Type: Function

Description: Add Analytic Function: Returns the sum of the arguments.

The sum of a real number *a* and a complex number (x, y) is the complex number $(x+a, y)$.

The sum of two complex numbers (x_1, y_1) and (x_2, y_2) is the complex number (x_1+x_2, y_1+y_2).

The sum of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The arrays must have the same dimensions.

The sum of a binary integer and a real number is a binary integer that is the sum of the two arguments, truncated to the current wordsize. (The real number is converted to a binary integer before the addition.)

The sum of two binary integers is truncated to the current binary integer wordsize.

The sum of two unit objects is a unit object with the same dimensions as the second argument. The units of the two arguments must be consistent.

The sum of two graphics objects is the same as the result of performing a logical OR, except that the two graphics objects *must* have the same dimensions.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units x_°C and x_°F represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠. This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to Kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as x_°C/min, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, –, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ⊞

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
z_1	z_2	→	$z_1 + z_2$
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_3$
z	'*symb*'	→	'z +*symb*'
'*symb*'	z	→	'*symb* +z'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ + *symb*$_2$'
{ *list*$_1$ }	{ *list*$_2$	→	{ *list*$_1$ *list*$_2$ }
obj$_A$	{ *obj*$_1$ … *obj*$_n$ }	→	{ *obj*$_A$ *obj*$_1$ … *obj*$_n$ }
{ *obj*$_1$ … *obj*$_n$ }	*obj*$_A$	→	{*obj*$_1$ … *obj*$_n$ *obj*$_A$ }
"*string*$_1$"	"*string*$_2$"	→	"*string*$_1$ *string*$_2$"
obj	"*string*"	→	"*obj string*"
"*string*"	*obj*	→	"*string obj*"

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$
x_unit_1	y_unit_2	→	$(x_2 + y)\_unit_2$
'*symb*'	x_unit	→	'*symb* + x_unit'
x_unit	'*symb*'	→	'x_unit + *symb*'
$grob_1$	$grob_2$	→	$grob_3$

See also: –, *, /, =

–

Type: Function

Description: Subtract Analytic Function: Returns the difference of the arguments.

The difference of a real number a and a complex number (x, y) is $(x–a, y)$ or $(a–x, –y)$. The difference of two complex numbers (x_1, y_1) and (x_2, y_2) is $(x_1 – x_2, y_1 – y_2)$.

The difference of a real array and a complex array is a complex array, where each element x of the real array is treated as a complex element $(x, 0)$. The two array arguments must have the same dimensions.

The difference of a binary integer and a real number is a binary integer that is the sum of the first argument and the two's complement of the second argument. (The real number is converted to a binary integer before the subtraction.)

The difference of two binary integers is a binary integer that is the sum of the first argument and the two's complement of the second argument.

The difference of two unit objects is a unit object with the same dimensions as the second argument. The units of the two arguments must be consistent.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The calculator assumes that the simple temperature units x_°C and x_°F represent thermometer temperatures when used as arguments to the functions <, >, ≤, ≥, ==, and ≠.

This means that, in order to do the calculation, the calculator will first convert any Celsius temperature to Kelvins and any Fahrenheit temperature to Rankines. (For other functions or *compound* temperature units, such as x_°C/min, the calculator assumes temperature units represent temperature differences, so there is no additive constant involved, and hence no conversion.)

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant, but require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ⊖

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 - \tilde{z}_2$
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1-2}$
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ – *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* – $\tilde{z}$'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ – *symb*$_2$'
#n_1	n_2	→	#n_3
n_1	#n_2	→	#n_3
#n_1	#n_2	→	#n_3
x_1_*unit*$_1$	y_*unit*$_2$	→	$(x_2 - y)$_*unit*$_2$
'*symb*'	x_*unit*	→	'*symb* – x_*unit*'
x_*unit*	'*symb*'	→	'x_*unit* – *symb*'

See also: +, *, /, =

/

Type: Function

Description: Divide Analytic Function: Returns the quotient of the arguments: the first argument is divided by the second argument.

A real number *a* divided by a complex number (x, y) returns:

$$\left(\frac{ax}{x^2+y^2}, \frac{ay}{x^2+y^2}\right)$$

. A complex number (x, y) divided by a real number *a* returns the complex number $(x/a, y/a)$.

A complex number (x_1, y_1) divided by another complex number (x_2, y_2) returns this complex quotient:

$$\left(\frac{x_1x_2+y_1y_2}{x_2^2+y_2^2}, \frac{y_1x_2-x_1y_2}{x_2^2+y_2^2}\right)$$

An array **B** divided by a matrix **A** solves the system of equations **AX**=**B** for **X**; that is, **X** = $\mathbf{A}^{-1}\,\mathbf{B}$. This operation uses 15-digit internal precision, providing a more precise result than the calculation INV(A)*B. The matrix must be square, and must have the same number of columns as the array has rows (or elements, if the array is a vector).

A binary integer divided by a real or binary number returns a binary integer that is the integral part of the quotient. (The real number is converted to a binary integer before the division.) A divisor of zero returns # 0.

When dividing two unit objects, the scaler parts and the unit parts are divided separately.

Access: CAT /

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1$ / $\tilde{z}_2$
[*array*]	[[*matrix*]]	→	[[$matrix^{-1} \times array$]]
$\tilde{z}$	'*symb*'	→	'$\tilde{z}$ / *symb*'
'*symb*'	$\tilde{z}$	→	'*symb* / $\tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ / $symb_2$'
$\#n_1$	n_2	→	$\#n_3$
n_1	$\#n_2$	→	$\#n_3$
$\#n_1$	$\#n_2$	→	$\#n_3$

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x_unit_1	y_unit_2	→	$(x/y)\_unit_1/unit_2$
x	y_unit	→	$(x/y)\_1/unit$
x_unit	y	→	$(x/y)\_unit$
'*symb*'	x_unit	→	'*symb* / x_unit'
x_unit	'*symb*'	→	'x_unit / *symb*'

See also: +, –, *, =

<

Type: Function

Description: Less Than Function: Tests whether one object is less than another object.

The function < returns a true test result (1) if the first argument is less than the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, < returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "less than" means numerically smaller (1 is less than 2). For real numbers, "less than" also means more negative (–2 is less than –1).

For strings, "less than" means alphabetically previous ("ABC" is less than "DEF"; "AAA" is less than "AAB"; "A" is less than "AA"). In general, characters are ordered according to their character codes. This means, for example, that "B" is less than "a", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent, and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↱<

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x < symb$'
'$symb$'	x	→	'$symb < x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 < symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit < symb$'
'$symb$'	x_unit	→	'$symb < x\_unit$'

See also: ≤, >, ≥, ==, ≠

=

Type: Function

Description: Equals Analytic Function: Returns an equation formed from the two arguments.

The equals sign equates two expressions such that the difference between the two expressions is zero.

In Symbolic Results mode, the result is an algebraic equation. In Numerical Results mode, the result is the difference of the two arguments because = acts equivalent to –. This allows expressions and equations to be used interchangeably as arguments for symbolic and numerical rootfinders.

Common usage is ambiguous about some units of temperature. When °C or °F represents a thermometer reading, then the temperature is a unit with an additive constant: 0 °C = 273.15 K, and 0 °F = 459.67 °R. But when °C or °F represents a *difference* in thermometer readings, then the temperature is a unit with no additive constant: 1 °C = 1 K and 1 °F = 1 °R.

The arithmetic operators +, –, %, %CH, and %T treat temperatures as differences, without any additive constant. However, +, –, %CH, and %T require both arguments to be either absolute (K and °R), both °C, or both °F. No other combinations are allowed.

Access: ↱=

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
$\tilde{z}_1$	$\tilde{z}_2$	→	$\tilde{z}_1 = \tilde{z}_2$
$\tilde{z}$	'*symb*'	→	'$\tilde{z} = symb$'
'*symb*'	$\tilde{z}$	→	'$symb = \tilde{z}$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 = symb_2$'
y_unit	x	→	$y\_unit_1 = x$
y_unit	*x_unit*	→	$y\_unit_1 = x\_unit$
'*symb*'	*x_unit*	→	'$symb = x\_unit$'
x_unit	'*symb*'	→	'$x\_unit = symb$'

See also: DEFINE, EVAL, –

==

Type: Function

Description: Logical Equality Function: Tests if two objects are equal.

The function == returns a true result (1) if the two objects are the same type and have the same value, or a false result (0) otherwise. Lists and programs are considered to have the same values if the objects they contain are identical.

If one object is algebraic (or a name), and the other is a number (real or complex) or an algebraic, == returns a symbolic comparison expression that can be evaluated to return a test result.

Note that == is used for comparisons, while = separates two sides of an equation.

If the imaginary part of a complex number is 0, it is ignored when the complex number is compared to a real number.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: CAT ==

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj_1	obj_2	→	0/1
(x,0)	x	→	0/1
x	(x,0)	→	0/1
z	'symb'	→	'z == symb'
'symb'	z	→	'symb == z'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ == $symb_2$'

See also: SAME, TYPE, <, ≤, >, ≥, ≠

>

Type: Function

Description: Greater Than Function: Tests whether one object is greater than another object.

The function > returns a true test result (1) if the first argument is greater than the second argument, or a false test result (0) otherwise.

If one object is a symbolic (an algebraic or a name), and the other is a number or symbolic or unit object, > returns a symbolic comparison expression that can be evaluated to return a test result.

For real numbers and binary integers, "greater than" means numerically greater (2 is greater than 1). For real numbers, "greater than" also means less negative (–1 is greater than –2).

For strings, "greater than" means alphabetically subsequent ("DEF" is greater than "ABC"; "AAB" is greater than "AAA"; "AA" is greater than "A"). In general, characters are ordered according to their character codes. This means, for example, that "a" is greater than "B", since "B" is character code 66, and "a" is character code 97.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison. If you use simple temperature units, the calculator assumes the values represent temperatures and not differences in temperatures. For compound temperature units, the calculator assumes temperature units represent temperature differences. For more information on using temperature units with arithmetic functions, refer to the entry for +.

Access: ↱ >

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
x	y	→	0/1
$\#n_1$	$\#n_2$	→	0/1
"$string_1$"	"$string_2$"	→	0/1
x	'$symb$'	→	'$x > symb$'
'$symb$'	x	→	'$symb > x$'
'$symb_1$'	'$symb_2$'	→	'$symb_1 > symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'$symb$'	→	'$x\_unit > symb$'
'$symb$'	x_unit	→	'$symb > x\_unit$'

See also: <, ≤, ≥, ==, ≠

▶

Type: Command

Description: Store Command: Stores an object into a specified variable.

To create a backup object, store the *obj* into the desired backup location (identified as $:n_{port}:name_{backup}$). ▶ will not overwrite an existing backup object.

To replace an element of an array or list, use STO. Also use STO to store a graphic object into PICT or a library or backup object into a port.

Access: CAT

Input/Output:

Level 2/Argument 1	Level 1/Argument 2		Level 1/Item 1
obj	'*name*'	→	*obj*
obj	$:n_{port}:name_{backup}$	→	*obj*

See also: DEFINE, RCL, →, STO
