SCATRPLOT automatically executes AUTO to achieve the same result. SCLΣ is included in the HP 48 for compatibility with the HP 28.

Related Commands: AUTO, SCATRPLOT

SCONJ

Store Conjugate Command: Conjugates the contents of a named object.

{}

Level 1	→	Level 1
'name'	→	

Keyboard Access: (↰)(MEMORY) ARITH (NXT) SCON

Affected by Flags: None

Remarks: The named object must be a number, an array, or an algebraic object. For information on conjugation, see CONJ.

Related Commands: CONJ, SINV, SNEG

SDEV

Standard Deviation Command: Calculates the sample standard deviation of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sdev}
	→	[x_{sdev1} x_{sdev2} ... x_{sdevm}]

Keyboard Access: (↰)(STAT) 1VAR SDEV

Affected by Flags: None

Remarks: SDEV returns a vector of m real numbers, or a single real number if $m = 1$. The standard deviations (the square root of the variances) are computed using this formula:

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

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

Related Commands: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, TOT, VAR

SEND

Send Object Command: Sends a copy of the named objects to a Kermit device.

Level 1	→	Level 1
'name'	→	
{ $name_1$ ... $name_n$ }	→	
{{ $name_{old}$ $name_{new}$ } *name* ... }	→	

Keyboard Access: (↰)(I/O) SEND

Affected by Flags: I/O Device (−33), I/O Data Format (−35), I/O Messages (−39)

If flag −35 is clear (ASCII transfer), the translation setting also has an effect.

Remarks: Data is always sent from a local Kermit, but can be sent either to another local Kermit (which must execute RECV or RECN) or to a server Kermit.

To rename an object when sending it, include the old and new names in an embedded list.

Examples: Executing `{{ AAA BBB }} SEND` sends the variable named *AAA* but changes its name to *BBB*.

Executing `{{ AAA BBB } CCC } SEND` sends *AAA* as *BBB* and sends *CCC* under its own name. (If the new name is not legal on the HP 48, just enter it as a string.)

Related Commands: BAUD, CLOSEIO, CKSM, FINISH, KERRM, KGET, PARITY, RECN, RECV, SERVER, TRANSIO

SEQ

Sequential Calculation Command: Returns a list of results generated by repeatedly executing obj_{exec} using *index* over the range x_{start} to x_{end}, in increments of x_{incr}.

{}

Level 5	Level 4	Level 3	Level 2	Level 1	→	Level 1
obj_{exec}	*index*	x_{start}	x_{end}	x_{incr}	→	{ *list* }

Keyboard Access: (PRG) LIST PROC (NXT) SEQ

Affected by Flags: None

Remarks: obj_{exec} is nominally a program or algebraic object that is a function of *index*, but can actually be any object. *index* must be a global or local name. The remaining objects can be anything that will evaluate to real numbers.

The action of SEQ for arbitrary inputs can be predicted exactly from this equivalent program:

x_{start} x_{end} FOR *index* obj_{exec} EVAL x_{incr} STEP n → LIST

where n is the number of new objects left on the stack by the FOR ... STEP loop. Notice that *index* becomes a local variable regardless of its original type.

Example: `'n^2' 'n' 1 4 1` returns `{ 1 4 9 16 }`.

Related Commands: DOSUBS, STREAM

SERVER

Server Mode Command: Selects Kermit Server mode.

Keyboard Access:

(↰)(I/O) SERVR SERVE

(↱)(▶)

Affected by Flags: I/O Device (−33), I/O Data Format (−35), RECV Overwrite (−36), I/O Messages (−39)

Remarks: A Kermit server (a Kermit device in Server mode) passively processes requests sent to it by the local Kermit. The server receives data in response to SEND, transmits data in response to KGET, terminates Server mode in response to FINISH or LOGOUT, and transmits a directory listing in response to a generic directory request.

Server mode supports Kermit Host Command packets. This allows you, for instance, to use a PC to type into the HP 48's command line. (This is especially convenient while testing programs.) Do this as follows:

1. Set up the HP 48 for data transfer to a computer, as described in "Transferring Data Between the HP 48 and a Computer" in chapter 27 of the *HP 48 User's Guide*.
2. Execute SERVER to set the HP 48 to Server mode.
3. On the PC, type `REMOTE HOST` followed by up to 89 characters to be entered into the HP 48 command line.
4. Press (Return) to transmit and execute the commands. The HP 48 executes the transmitted commands, then sends back to the PC's display the resulting contents of the stack as the HP 48 would normally display them.

If you use a PC to write programs for the HP 48, you should include the *%%HP...;* header in the program. See the discussion of ASCII mode in chapter 27 of the *HP 48 User's Guide.*

Related Commands: BAUD, CKSM, FINISH, KERRM, KGET, PARITY, PKT, RECN, RECV, SEND, TRANSIO

SF

Set Flag Command: Sets a specified user or system flag.

{ }

Level 1	→	Level 1
$n_{\text{flagnumber}}$	→	

Keyboard Access:

(PRG) TEST (NXT) (NXT) SF

(↰)(MODES) FLAG SF

Affected by Flags: None

Remarks: User flags are numbered 1 through 64. System flags are numbered −1 through −64. See appendix C, "System Flags," for a listing of HP 48 system flags and their flag numbers.

Related Commands: CF, FC?, FC?C, FS?, FS?C

SHOW

Show Variable Command: Returns '$symb_2$', which is equivalent to '$symb_1$' except that all implicit references to a variable *name* are made explicit.

Level 2	Level 1	→	Level 1
'$symb_1$'	'*name*'	→	'$symb_2$'
'$symb_1$'	{ $name_1$ $name_2$ ... }	→	'$symb_2$'

Keyboard Access: (↰)(SYMBOLIC) SHOW

Affected by Flags: Numerical Results (−3)

Remarks: If the level 1 argument is a list, SHOW evaluates all global variables in '$symb_1$' *not* contained in the list.

Example: If 7 is stored in C and 5 is stored in D, then

```
'X-Y+2*C+3*D' { X Y } SHOW
```

returns 'X-Y+14+15'

Related Commands: COLCT, EXPAN, ISOL, QUAD

SIDENS

Silicon Intrinsic Density Command: Calculates the intrinsic density of silicon as a function of temperature, x_T

{ }

Level 1	→	Level 1
x_T	→	x_{density}
x_unit	→	*x_1/cm*^{3}
'*symb*'	→	'SIDENS(*symb*)'

SIDENS

Keyboard Access: (←)(EQ LIB) UTILS SIDEN

Affected by Flags: Numerical Results (−3)

Remarks: If x_T is a unit object, it must reduce to a pure temperature, and the density is returned as a unit object with units of $1/\text{cm}^3$.

If x_T is a real number, its units are assumed to be K, and the density is returned as a real number with implied units of $1/\text{cm}^3$

x_T must be between 0 and 1685 K.

SIGN

Sign Function: Returns the sign of a real number argument, the sign of the numerical part of a unit object argument, or the unit vector in the direction of a complex number argument.

{ }

Level 1	→	Level 1
z_1	→	z_2
x_unit	→	x_{sign}
'*symb*'	→	'SIGN(*symb*)'

Keyboard Access:

(MTH) REAL (NXT) SIGN (returns the sign of a number)

(MTH) (NXT) CMPL (NXT) SIGN (returns the unit vector of a complex number)

Affected by Flags: Numerical Results (−3)

Remarks: For real number and unit object arguments, the sign is defined as +1 for positive arguments, −1 for negative arguments, and 0 for argument 0.

For a complex argument:

$$\text{SIGN}(x+iy)=\frac{x}{\sqrt{x^2+y^2}}+\frac{iy}{\sqrt{x^2+y^2}}$$

Examples: `32_ft SIGN` returns `1`.

`(1,1) SIGN` returns `(.707106781187,.707106781187)`.

Related Commands: ABS, MANT, XPON

SIMU

Simultaneous Plotting Command: Enables and disables simultaneous plotting.

Keyboard Access: (←)(PLOT) (NXT) FLAG SIMU

Affected by Flags: Simultaneous Plotting (−28)

Remarks: SIMU changes to SIMU■ when flag −28 is enabled (and simultaneous plotting is enabled).

If the calculator is in program entry mode, pressing the menu key echoes AXES, CNCT, and SIMU flag numbers to the command line. Pressing (←) or (→) first echoes the flag numbers and SF or CF to the command line.

Related Commands: AXES, CF, SF

SIN

Sine Analytic Function: Returns the sine of the argument.

{ }

Level 1	→	Level 1
z	→	sin *z*
'*symb*'	→	'SIN(*symb*)'
$x\_unit_{angular}$	→	sin ($x\_unit_{angular}$)

SIN

Keyboard Access: (SIN)

Affected by Flags: Numerical Results (−3), Angle Mode (−17, −18)

Remarks: For real arguments, the current angle mode determines the number's units, unless angular units are specified.

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

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

Related Commands: ASIN, COS, TAN

SINH

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

{}

Level 1	→	Level 1
z	→	sinh *z*
'*symb*'	→	'SINH(*symb*)'

Keyboard Access: (MTH) HYP SINH

Affected by Flags: Numerical Results (−3)

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

Related Commands: ASINH, COSH, TANH

SINV

Store Inverse Command: Replaces the contents of the named variable with its inverse.

{}

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: (↰)(MEMORY) ARITH (NXT) SINV

Affected by Flags: None

Remarks: The named object must be a number, a matrix, an algebraic object, or a unit object. For information on reciprocals, see INV.

Related Commands: INV, SCONJ, SNEG

SIZE

Size Command: Returns the number of characters in a string, the number of elements in a list, the dimensions of an array, the number of objects in a unit object or algebraic object, or the dimensions of a graphics object.

Level 1	→	Level 2	Level 1
"string"	→		*n*
{ *list* }	→		*n*
[*vector*]	→		{ *n* }
[[*matrix*]]	→		{ *n m* }
'*symb*'	→		*n*
grob	→	$\#n_{width}$	$\#m_{height}$
PICT	→	$\#n_{width}$	$\#m_{height}$
x_unit	→		*n*

Keyboard Access:

(←) (CHARS) SIZE

(PRG) LIST ELEM SIZE

(PRG) OBJ (NXT) SIZE

Affected by Flags: None

Remarks: The size of a unit is computed as follows: the scalar (+1), the underscore (+1), each unit name (+1), operator or exponent (+1), and each prefix (+2).

Any object type not listed above returns a value of 1.

Related Commands: CHR, NUM, POS, REPL, SUB

SL

Shift Left Command: Shifts a binary integer one bit to the left.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BIT SL

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant bit is shifted out to the left and lost, while the least significant bit is regenerated as a zero. SL is equivalent to binary multiplication by 2, truncated to the current wordsize.

Related Commands: ASR, SLB, SR, SRB

SLB

Shift Left Byte Command: Shifts a binary integer one byte to the left.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BYTE SLB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The most significant byte is shifted out to the left and lost, while the least significant byte is regenerated as zero. SLB is equivalent to binary multiplication by 2^8 (or executing SL eight times), truncated to the current wordsize.

Related Commands: ASR, SL, SR, SRB

SLOPEFIELD

SLOPEFIELD Plot Type Command: Sets the plot type to SLOPEFIELD.

Keyboard Access: ↰ (PLOT) (NXT) 3D PTYPE SLOPE

Affected by Flags: None

Remarks: When plot type is set to SLOPEFIELD, the DRAW command plots a slope representation of a scalar function with two variables. SLOPEFIELD requires values in the reserved variables *EQ*, *VPAR*, and *PPAR*.

VPAR has the following form:

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

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

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

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

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

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

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

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

Related Commands: BAR, CONIC, DIFFEQ, FUNCTION, GRIDMAP, HISTOGRAM, PARAMETRIC, PARSURFACE, PCONTOUR, POLAR, SCATTER, TRUTH, WIREFRAME, YSLICE

SNEG

Store Negate Command: Replaces the contents of a variable with its negative.

{ }

Level 1	→	Level 1
'name'	→	

Keyboard Access: ↰ (MEMORY) ARITH (NXT) SNEG

Affected by Flags: None

Remarks: The named object must be a number, an array, an algebraic object, a unit object, or a graphics object. For information on negation, see NEG.

Related Commands: NEG, SCONJ, SINV

SNRM

Spectral Norm Command: Returns the spectral norm of an array.

{ }

Level 1	→	Level 1
[*array*]	→	$x_{\mathrm{spectralnorm}}$

Keyboard Access: (MTH) MATR NORM SNRM

Affected by Flags: None

Remarks: The spectral norm of a vector is its Euclidean length, and is equal to the largest singular value of a matrix.

Related Commands: ABS, CNRM, COND, RNRM, SRAD, TRACE

SOLVEQN

Start Equation Solver Command: Starts the appropriate solver for a specified set of equations.

{ }

Level 3	Level 2	Level 1	→	Level 1
n	*m*	*0/1*	→	

Keyboard Access: (↰)(EQ LIB) EQLIB SOLVE

Affected by Flags: Unit Type (60), Units Usage (61)

Remarks: SOLVEQN sets up and starts the appropriate solver for the specified set of equations, bypassing the Equation Library catalogs. It sets *EQ* (and *Mpar* if more than one equation is being solved), sets the unit options according to flags 60 and 61, and starts the appropriate solver.

SOLVEQN uses subject and title numbers (levels 3 and 2) and a "PICT" option (level 1), and returns nothing. Subject and title numbers are listed in chapter 4. If the "PICT" option is 0, *PICT* is not affected; otherwise, the equation picture is copied into *PICT*.

Related Commands: EQNLIB, MSOLVER

SORT

Ascending Order Sort Command: Sorts the elements in a list in ascending order.

Level 1	→	Level 1
{ *list* }$_1$	→	{ *list* }$_2$

Keyboard Access:

(PRG) LIST PROC (NXT) SORT

(MTH) LIST SORT

Affected by Flags: None

Remarks: The elements in the list can be real numbers, strings, lists, names, binary integers, or unit objects. However, all elements in the list must all be of the same type. Strings and names are sorted by character code number. Lists of lists are sorted by the first element in each list.

To sort in reverse order, use SORT REVLIST.

Related Commands: REVLIST

SPHERE

Spherical Mode Command: Sets Spherical coordinate mode.

Keyboard Access:

(MTH) VECTR (NXT) SPHER

(↰)(MODES) ANGL SPHER

Affected by Flags: None

Remarks: SPHERE sets flags −15 and −16, and displays the R∡∡ annunciator.

In Spherical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as [r ∡θ ∡ϕ].

Related Commands: CYLIN, RECT

SQ

Square Analytic Function: Returns the square of the argument.

{ }

Level 1	→	Level 1
z	→	z^2
x_unit	→	x^2_*unit*^{2}
[[*matrix*]]	→	[[*matrix* × *matrix*]]
'*symb*'	→	'SQ(*symb*)'

Keyboard Access: (↰)(x^2)

Affected by Flags: Numerical Results (−3)

Remarks: The square of a complex argument (x, y) is the complex number ($x^2 - y^2$, $2xy$).

Matrix arguments must be square.

Related Commands: √, ^

SR

Shift Right Command: Shifts a binary integer one bit to the right.

{ }

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: (MTH) BASE (NXT) BIT SR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant bit is shifted out to the right and lost, while the most significant bit is regenerated as a zero. SR is equivalent to binary division by 2.

Related Commands: ASR, SL, SLB, SRB

SRAD

Spectral Radius Command: Returns the spectral radius of a square matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]$_{n \times n}$	→	$x_{spectralradius}$

Keyboard Access: (MTH) MATR NORM SRAD

Affected by Flags: None

Remarks: The spectral radius of a matrix is a measure of the size of the matrix, and is equal to the absolute value of the largest eigenvalue of the matrix.

Related Commands: COND, SNRM, TRACE

SRB

Shift Right Byte Command: Shifts a binary integer one byte to the right.

{ }

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Keyboard Access: (MTH) BASE (NXT) BYTE SRB

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: The least significant byte is shifted out to the right and lost, while the most significant byte is regenerated as zero. SRB is equivalent to binary division by 2^8 (or executing SR eight times).

Related Commands: ASR, SL, SLB, SR

SRECV

Serial Receive Command: Reads up to *n* characters from the serial input buffer and returns them as a string, along with a digit indicating whether errors occurred.

{ }

Level 1	→	Level 2	Level 1
n	→	*'string'*	0/1

Keyboard Access: (↰)(I/O) (NXT) SERIA SRECV

Affected by Flags: I/O Device (−33)

Remarks: SRECV does not use Kermit protocol.

If *n* characters are not received within the time specified by STIME (default is 10 seconds), SRECV "times out", returning a 0 to level 1 and as many characters as were received to level 2.

If the level 2 output from BUFLEN is used as the input for SRECV, SRECV will not have to wait for more characters to be received. Instead, it returns the characters already in the input buffer.

If you want to accumulate bytes in the input buffer before executing SRECV, you must first open the port using OPENIO (if the port isn't already open).

SRECV can detect three types of error when reading the input buffer:

- Framing errors and UART overruns (both causing "Receive Error" in ERRM).
- Input-buffer overflows (causing "Receive Buffer Overflow" in ERRM).
- Parity errors (causing "Parity Error" in ERRM).

SRECV returns 0 if an error is detected when reading the input buffer, or 1 if no error is detected.

Parity errors do not stop data flow into the input buffer. However, if a parity error occurs, SRECV stops reading data after encountering a character with an error.

Framing, overrun, and overflow errors cause all subsequently received characters to be ignored until the error is cleared. SRECV does not detect and clear any of these types of errors until it tries to read the byte where the error occurred. Since these three errors cause the byte where the error occurred and all subsequent bytes to be ignored, the input buffer will be empty after all previously received good bytes have been read. Therefore, SRECV detects and clears these errors when it tries to read a byte from an empty input buffer.

Note that BUFLEN also clears the above-mentioned framing, overrun, and overflow errors. Therefore, SRECV cannot detect an input-buffer overflow after BUFLEN is executed, unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). SRECV also cannot detect framing and UART overrun errors cleared by BUFLEN. To find where the data error occurred, save the number of characters returned by BUFLEN (which gives the number of "good" characters received), because as soon as the error is cleared, new characters can enter the input buffer.

Example: If 10 good bytes were received followed by a framing error, then an SRECV command told to read 10 bytes would *not* indicate an error. Only when SRECV tries to read the byte that caused the framing error does it return a 0. Similarly, if the input buffer overflowed, SRECV would not indicate an error until it tried to read the first byte that was lost due to the overflow.

Related Commands: BUFLEN, CLOSEIO, OPENIO, SBRK, STIME, XMIT

SST

Execute Program Step Operation: Returns and executes the next step of a program. If the next step is a subroutine, executes the subroutine in a single step.

Keyboard Access: (PRG) (NXT) RUN SST

Affected by Flags: None

Remarks: SST is not programmable.

Related Commands: NEXT (operation), SST↓

SST↓

Execute Subroutine Step Operation: Returns and executes the next step of a program or subroutine. If the next step is a subroutine, returns and executes the first step of the subroutine.

Keyboard Access: (PRG) (NXT) RUN SST↓

Affected by Flags: None

Remarks: SST↓ is not programmable.

Related Commands: NEXT (operation), SST

START

START Definite Loop Structure Command: Begins START ... NEXT and START ... STEP definite loop structures.

	Level 2	Level 1	→	Level 1
START	x_{start}	x_{finish}	→	
NEXT			→	
STEP		$x_{\text{increment}}$	→	
STEP		$'symb_{\text{increment}}'$	→	

Keyboard Access: (PRG) BRCH START START

Affected by Flags: None

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

- START ... NEXT executes a portion of a program a specified number of times. The syntax is this:

  x_{start} x_{finish} START *loop-clause* NEXT

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values for a loop counter. Then the loop clause is executed. NEXT increments the counter by 1 and tests to see if its value is less than or equal to x_{finish}. If so, the loop clause is executed again. Notice that the loop clause is always executed at least once.

- START ... STEP works just like START ... NEXT, except that it can use an increment value other than 1. The syntax is this:

  x_{start} x_{finish} START *loop-clause* $x_{increment}$ STEP

  START takes two numbers (x_{start} and x_{finish}) from the stack and stores them as the starting and ending values of the loop counter. Then the loop clause is executed. STEP takes $x_{\text{increment}}$ from the stack and increments the counter by that value. If the argument of STEP is an algebraic or a name, it is automatically evaluated to a number.

The increment value can be positive or negative:

- □ If positive, the loop is executed again when the counter is less than or equal to x_{finish}.
- □ If negative, the loop is executed when the counter is greater than or equal to x_{finish}.

Related Commands: FOR, NEXT, STEP

STD

Standard Mode Command: Sets the number display format to Standard mode.

Keyboard Access: ⬑ (MODES) FMT STD

Affected by Flags: None

Remarks: Executing STD has the same effect as clearing flags –49 and –50.

Standard format (ANSI Minimal BASIC Standard X3J2) produces the following results when displaying or printing a number:

- Numbers that can be represented exactly as integers with 12 or fewer digits are displayed without a fraction mark or exponent. Zero is displayed as 0.
- Numbers that can be represented exactly with 12 or fewer digits, but not as integers, are displayed with a fraction mark but no exponent. Leading zeros to the left of the fraction mark and trailing zeros to the right of the fraction mark are omitted.
- All other numbers are displayed in scientific notation (see SCI) with both a fraction mark (with one number to the left) and an exponent (of one to three digits). There are no leading or trailing zeros.

In algebraic objects, integers less than 10^3 are always displayed in Standard mode.

Example: The following table provides examples of numbers displayed in Standard mode:

Number	Displayed As	Representable With 12 Digits?
10^{11}	100000000000	Yes (integer)
10^{12}	1.E12	No
10^{-11}	.00000000001	Yes
1.2×10^{-11}	1.23E-11	No
12.345	12.345	Yes

Related Commands: ENG, FIX, SCI

STEP

STEP Command: Defines the increment (step) value, and ends definite loop structure.

See the FOR and START command entries for syntax information.

Keyboard Access:

(PRG) BRCH FOR STEP

(PRG) BRCH START STEP

Remarks: See the FOR and START keyword entries for more information.

Related Commands: FOR, NEXT, START

STEQ

Store in EQ Command: Stores an object into the reserved variable *EQ* in the current directory.

{ }

Level 1	→	Level 1
obj	→	

Keyboard Access: This command must be typed in, but you can store an object in *EQ* with:

(↰)(PLOT)(↰) EQ

(↰)(PLOT)(NXT) 3D (↰) EQ

Affected by Flags: None

Related Commands: RCEQ

STIME

Serial Time-Out Command: Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

{ }

Level 1	→	Level 1
x_{seconds}	→	
0	→	

Keyboard Access: (↰)(I/O)(NXT) SERIA STIME

Affected by Flags: None

Remarks: The value for x is interpreted as a positive value from 0 to 25.4 seconds. If no value is given, the default is 10 seconds. If x is 0, there is no time-out; that is, the device waits indefinitely, which can drain the batteries.

STIME is not used for Kermit time-out.

Related Commands: BUFLEN, CLOSEIO, SBRK, SRECV, XMIT

STO

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

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
grob	*PICT*	→	
obj	:n_{port} :$name_{\text{backup}}$	→	
obj	'*name*(*index*)'	→	
backup	n_{port}	→	
library	n_{port}	→	
library	:n_{port} :n_{library}	→	

Keyboard Access: (STO)

Affected by Flags: None

Remarks: Storing a graphics object into *PICT* makes it the current graphics object.

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

To store backup objects and library objects, specify a port number (0 through 33). Ports 1 and 2 must be configured as independent RAM, since backup and library objects can be stored only in independent RAM (see the entry for FREE).

To use a library object, the object must be in a port and it must be attached. A library object from an application card (ROM) is automatically in a port (1 through 33), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

After storing a library object in a port, it must then be attached to its directory before it can be used. To make a stored library "attachable", turn the calculator off and then on. (See the entry for ATTACH.) This action (storing a library object, then turning the calculator off and on) also causes the calculator to perform a *system halt*, which clears the stack, the LAST stack, and all local variables, and returns the MATH menu to the display.

STO can also replace a single element of an array or list stored in a variable. Specify the variable in level 1 as `'name(index)'`, which is a user function with *index* as the argument. The *index* can be *n* or *n,m*, where *n* specifies the row position in a vector or list, and *n,m* specifies the row-and-column position in a matrix.

Example: `'A+B+C+D' 'SUMAD' STO` stores the expression A+B+C+D in the variable *SUMAD*.

`5 'A(3)' STO` stores the integer 5 in the third element in a list or vector *A*.

`2 'A(3,5)' STO` stores the integer 2 in the element in the third row and fifth column of matrix *A*.

Related Commands: DEFINE, RCL, →

STOALARM

Store Alarm Command: Stores an alarm in the system alarm list and returns its alarm index number.

Level 1	→	Level 1
x_{time}	→	n_{index}
{ *date time* }	→	n_{index}
{ *date time* obj_{action} }	→	n_{index}
{ *date time* obj_{action} x_{repeat} }	→	n_{index}

Keyboard Access: ⬑ (TIME) ALRM STOAL

Affected by Flags: Date Format (−42), Repeat Alarms Not Rescheduled (−43), Acknowledged Alarms Saved (−44)

Remarks: If the argument is a real number x_{time}, the alarm date will be the current system date by default.

If obj_{action} is a string, the alarm is an appointment alarm, and the string is the alarm message. If obj_{action} is any other object type, the alarm is a control alarm, and the object is executed when the alarm comes due.

x_{repeat} is the repeat interval for the alarm in clock ticks, where 8192 clock ticks equals 1 second.

n_{index} is a real integer identifying the alarm based on its chronological position in the system alarm list.

Example: With flag −42 clear, this command:

```
{ 11.06 15.2530 RUN 491520 } STOALARM
```

sets a repeating control alarm for November 6 of the currently specified year, at 3:25:30 PM. The alarm action is to execute variable *RUN*. The repeat interval is 491520 clock ticks (1 minute).

Related Commands: DELALARM, FINDALARM, RCLALARM

STOF

Store Flags Command: Sets the states of the system flags or the system and user flags.

Level 1	→	Level 1
$\#n_{\text{system}}$	→	
{ $\#n_{\text{system}}$ $\#n_{\text{user}}$ }	→	

Keyboard Access: ⬑ (MODES) FLAG (NXT) STOF

Affected by Flags: Binary Integer Wordsize (−5 through −10)

The current wordsize must be 64 bits (the default wordsize) to store all flags. For example, executing STOF with a 32-bit binary integer stores only flags −1 through −32 and *clears* the other system flags.

Remarks: With argument $\#n_{\text{system}}$, STOF sets the states of the system flags (−1 through −64) only. With argument { $\#n_{system}$ $\#n_{user}$ }, STOF sets the states of both the system and user flags.

A bit with value 1 sets the corresponding flag; a bit with value 0 clears the corresponding flag. The rightmost (least significant) bit of $\#n_{\text{system}}$ and $\#n_{\text{user}}$ correspond to the states of system flag −1 and user flag +1, respectively. If $\#n_{\text{system}}$ or $\#n_{\text{user}}$ contains fewer than 64 bits, the unspecified most significant bits are taken to have value 0.

STOF can preserve the states of flags before a program executes and changes the states. RCLF can then recall the flag's states after the program is executed.

Related Commands: RCLF

STOKEYS

Store Key Assignments Command: Defines multiple keys on the user keyboard by assigning objects to specified keys.

Level 1	→	Level 1
{ obj_1 x_{key1} … obj_n x_{keyn} }	→	
{ S obj_1 x_{key1} … obj_n x_{keyn} }	→	
'S'	→	

Keyboard Access: (↰)(MODES) KEYS STOK

Affected by Flags: User-Mode Lock (−61) and User Mode (−62) affect the status of the user keyboard.

Remarks: x_{key} is a real number of the form *rc.p* specifying the key by its row number *r*, its column number *c*, and its plane (shift) *p*. (For a definition of plane, see the entry for ASN.)

The optional initial list parameter or argument S restores all keys without user assignments to their *standard* key assignments on the user keyboard. This is meaningful only when all standard key assignments had been suppressed (for the user keyboard) by the command 'S' DELKEYS (see DELKEYS).

If the argument *obj* is the name 'SKEY', the specified key is restored to its *standard key* assignment.

Related Commands: ASN, DELKEYS, RCLKEYS

STO+

Store Plus Command: Adds a number or other object to the contents of a specified variable.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: (↰)(MEMORY) ARITH STO+

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for addition to each other. STO+ can add any combination of objects suitable for stack addition (see the entry for +).

Using STO+ to add two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to add them.

Related Commands: STO−, STO∗, STO/, +

STO–

Store Minus Command: Calculates the difference between a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↰ MEMORY ARITH STO–

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for subtraction with each other. STO– can subtract any combination of objects suitable for stack subtraction (see the entry for –).

Using STO– to subtract two arrays (where *obj* is an array and *name* is the global name of an array) requires less memory than using the stack to subtract them.

Related Commands: STO+, STO*, STO/, –

STO*

Store Times Command: Multiplies the contents of a specified variable by a number or other object.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↰ MEMORY ARITH STO*

Affected by Flags: None

Remarks: The object on the stack and the object in the variable must be suitable for multiplication with each other. When multiplying two arrays, the result depends on the order of the arguments. The new object of the named variable is the level 2 array times the level 1 array. The arrays must be conformable for multiplication.

Using STO* to multiply two arrays or to multiply a number and an array (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to multiply them.

Related Commands: STO+, STO–, STO/, *

STO/

Store Divide Command: Calculates the quotient of a number (or other object) and the contents of a specified variable, and stores the new value to the specified variable.

{ }

Level 2	Level 1	→	Level 1
obj	'*name*'	→	
'*name*'	*obj*	→	

Keyboard Access: ↰ MEMORY ARITH STO/

Affected by Flags: None

Remarks: The new object of the specified variable is the level 2 object divided by the level 1 object.

The object on the stack and the object in the variable must be suitable for division with each other. If both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.

STO/

Using STO/ to divide one array by another array or to divide an array by a number (where *obj* is an array or a number and *name* is the global name of an array) requires less memory than using the stack to divide them.

Related Commands: STO+, STO−, STO*, /

STOΣ

Store Sigma Command: Stores *obj* in the reserved variable ΣDAT.

{}

Level 1	→	Level 1
obj	→	

Keyboard Access: This command must be typed in, but you can store an object in ΣDAT with either of the following:

(↰) (PLOT) (NXT) STAT (↰) ΣDAT

(↰) (STAT) DATA (↰) ΣDAT

Affected by Flags: None

Remarks: STOΣ accepts any object and stores it in ΣDAT. However, if the object is not a matrix or the name of a variable containing a matrix, an `Invalid Σ Data` error occurs upon subsequent execution of a statistics command.

Related Commands: CLΣ, RCLΣ, Σ+, Σ−

STREAM

Stream Execution Command: Moves the first two elements from the list onto the stack, and executes *obj*. Then moves the next element (if any) onto the stack, and executes *obj* again using the previous result and the new element. Repeats this until the list is exhausted, and returns the final result.

Level 2	Level 1	→	Level 1
{ *list* }	*obj*	→	*result*

Keyboard Access: (PRG) LIST PROC STREA

Affected by Flags: None

Remarks: STREAM is nominally designed for *obj* to be a program or command that requires two arguments and returns one result.

Examples: `{ 1 2 3 4 5 } «*» STREAM` returns `120`.

`«+» STREAM` is equivalent to `ΣLIST`.

Related Commands: DOSUBS

STR→

Evaluate String Command: Evaluates the text of a string as if the text were entered from the command line.

{}

Level 1	→	Level 1
"*obj*"	→	*evaluated-object*

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: OBJ→ also includes this function. STR→ is included for compatibility with the HP 28S.

Related Commands: ARRY→, DTAG, EQ→, LIST→, OBJ→, →STR

→STR

Object to String Command: Converts any object to string form.

Level 1	→	Level 1
obj	→	"*obj*"

Keyboard Access:

(⬑) (CHARS) (NXT) →STR

(PRG) TYPE →STR

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12), Number Display Format (−49, −50)

Remarks: The full-precision internal form of a number is not necessarily represented in the result string. To ensure that →STR preserves the full precision of a number, select Standard number display format or a wordsize of 64 bits (or both) before executing →STR.

The result string includes the entire object, even if the displayed form of the object is too large to fit in the display.

If the argument object is normally displayed in two or more lines, the result string will contain newline characters (character 10) at the end of each line. The newlines are displayed as the character ■.

If the argument object is already a string, →STR returns the string.

Example: →STR can create special displays to label program output or provide prompts for input. The sequence

```
"Result = " SWAP →STR + 1 DISP 1 FREEZE
```

displays `Result` = *object* in line 1 of the display, where *object* is a string form of an object taken from level 1.

Related Commands: →ARRY, →LIST, STR→, →TAG, →UNIT

STWS

Set Wordsize Command: Sets the current binary integer wordsize to *n* bits, where *n* is a value from 1 through 64 (the default is 64).

{ }

Level 1	→	Level 1
n	→	
#*n*	→	

Keyboard Access: (MTH) BASE (NXT) STWS

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12)

Remarks: Values of *n* less than 1 or greater than 64 are interpreted as 1 or 64, respectively.

If the wordsize is smaller than an integer entered in the command line, then the *most* significant bits are not displayed upon entry. The truncated bits are still present internally (unless they exceed 64), but they are not used for calculations and they are lost when a command uses this binary integer as an argument.

Results that exceed the given wordsize are also truncated to the wordsize.

Related Commands: BIN, DEC, HEX, OCT, RCWS

SUB

Subset Command: Returns the portion of a string or list defined by specified positions, or returns the rectangular portion of a graphics object or *PICT* defined by two corner pixel coordinates.

Level 3	Level 2	Level 1	→	Level 1
[[*matrix*]]$_1$	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	{ n_{row} n_{column} }	$n_{\text{endposition}}$	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	$n_{\text{startposition}}$	{ n_{row} n_{column} }	→	[[*matrix*]]$_2$
[[*matrix*]]$_1$	{ n_{row} n_{column} }	{ n_{row} n_{column} }	→	[[*matrix*]]$_2$
"$string_{\text{target}}$"	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	"$string_{\text{result}}$"
{ $list_{\text{target}}$ }	$n_{\text{startposition}}$	$n_{\text{endposition}}$	→	{ $list_{\text{result}}$ }
$grob_{\text{target}}$	{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	$grob_{\text{result}}$
$grob_{\text{target}}$	(x_1, y_1)	(x_2, y_2)	→	$grob_{\text{result}}$
PICT	{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	$grob_{\text{result}}$
PICT	(x_1, y_1)	(x_2, y_2)	→	$grob_{\text{result}}$

Keyboard Access:

(↰) (CHARS) SUB

(PRG) LIST SUB

(PRG) GROB SUB

(MTH) MATR MAKE (NXT) SUB

Affected by Flags: None

Remarks: If $n_{\text{end position}}$ is less than $n_{\text{start position}}$, SUB returns an empty string or list. Values of n less than 1 are treated as 1; values of n exceeding the length of the string or list are treated as that length.

For graphics objects, a user-unit coordinate less than the minimum user-unit coordinate of the graphics object is treated as that minimum. A pixel or user-unit coordinate greater than the maximum pixel or user-unit coordinate of the graphics object is treated as that maximum.

Examples: `{ A B C D E } 2 4 SUB` returns `{ B C D }`.

`"ABCDE" 0 10 SUB` returns `"ABCDE"`.

`PICT { # 10d # 20d } { # 20d # 40d } SUB` returns `GRAPHIC 11 × 21`.

Related Commands: CHR, GOR, GXOR, NUM, POS, REPL, SIZE

SVD

Singular Value Decomposition Command: Returns the singular value decomposition of an $m \times n$ matrix.

{ }

Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]$_A$	→	[[*matrix*]]$_U$	[[*matrix*]]$_V$	[*vector*]$_S$

Keyboard Access: (MTH) MATR FACTR SVD

Affected by Flags: None

Remarks: SVD decomposes A into 2 matrices and a vector. U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and S is a real vector, such that $A = U \times diag(S) \times V$. S has length MIN(m,n) and contains the singular values of A in nonincreasing order. The matrix $diag(S)$ is an $m \times n$ diagonal matrix containing the singular values S.

The computed results should minimize (within computational precision):

$$\frac{|A - U \cdot \text{diag}(S) \cdot V|}{\min(m,n) \cdot |A|}$$

where diag(S) denotes the $m \times n$ diagonal matrix containing the singular values S.

Related Commands: DIAG→, MIN, SVL

SVL

Singular Values Command: Returns the singular values of an $m \times n$ matrix.

{ }

Level 1	→	Level 1
[[*matrix*]]	→	[*vector*]

Keyboard Access: (MTH) MATR FACTR (NXT) SVL

Affected by Flags: None

Remarks: SVL returns a real vector that contains the singular values of an $m \times n$ matrix in nonincreasing order. The vector has length MIN(m,n).

Related Commands: MIN, SVD

SWAP

Swap Objects Command: Interchanges the first two objects on the stack.

Level 2	Level 1	→	Level 2	Level 1
obj_1	obj_2	→	obj_2	obj_1

Keyboard Access: (↰)(SWAP)

Affected by Flags: None

Related Commands: DUP, DUPN, DUP2, OVER, PICK, ROLL, ROLLD, ROT

SYSEVAL

Evaluate System Object Command: Evaluates unnamed operating system objects specified by their memory addresses.

{ }

Level 1	→	Level 1
$\#n_{address}$	→	

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Using SYSEVAL with random addresses can corrupt memory.

Example: Display the version letter of an HP 48 by executing `#30794h SYSEVAL`. Version A, for example, would display `"HPHP48-A"`

Related Commands: EVAL, LIBEVAL

%T

Percent of Total Function: Returns the percent of the level 2 argument that is represented by the level 1 argument.

Level 2	Level 1	→	Level 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)'

Keyboard Access: (MTH) REAL %T

Affected by Flags: Numerical Results (−3)

Remarks: 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 +.

Example: `1_m 500_cm %T` returns `500`, because 500 cm represents 500% of 1 m.

`100_K 50_K %T` returns `50`.

Related Commands: %, %CH

→TAG

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

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

Keyboard Access: (PRG) TYPE →TAG

Affected by Flags: None

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

Related Commands: →ARRY, DTAG, →LIST, OBJ→, →STR, →UNIT

TAIL

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

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

Keyboard Access:

(PRG) LIST ELEM (NXT) TAIL

(↰)(CHARS) (NXT) TAIL

Affected by Flags: None

Example: "tall" TAIL returns "all".

Related Commands: HEAD

TAN

Tangent Analytic Function: Returns the tangent of the argument.

{ }

Level 1	→	Level 1
z	→	tan *z*
'*symb*'	→	'TAN(*symb*)'
x_unit$_{\text{angular}}$	→	tan (*x_unit*$_{\text{angular}}$)

Keyboard Access: (TAN)

Affected by Flags: Numerical Results (−3), Angle Mode (−17, −18), Infinite Result Exception (−22)

Remarks: 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).

Related Commands: ATAN, COS, SIN

TANH

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

{ }

Level 1	→	Level 1
z	→	tanh *z*
'*symb*'	→	'TANH(*symb*)'

Keyboard Access: (MTH) HYP TANH

Affected by Flags: Numerical Results (−3)

Remarks: For complex arguments,

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

Related Commands: ATANH, COSH, SINH

TAYLR

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

{ }

Level 3	Level 2	Level 1	→	Level 1
'*symb*'	'*global*'	*n*$_{\text{order}}$	→	'*symb*$_{\text{Taylor}}$'

Keyboard Access: (⬅)(SYMBOLIC) TAYLR

Affected by Flags: None

Remarks: The polynomial is calculated at the point *global* = 0 (called a MacLaurin series).

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

Example: The command sequence `'1+SIN(X)^2' 'X' 5 TAYLR` returns `'1+X^2-8/4!*X^4'`

Related Commands: ∂, ∫, Σ

TDELTA

Temperature Delta Function: Calculates a temperature change.

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	x_{delta}
x_unit1	*y_unit2*	→	x_unit1_{delta}
x_unit	'*symb*'	→	'TDELTA(*x_unit,symb*)'
'*symb*'	*y_unit*	→	'TDELTA(*symb,y_unit*)'
'$symb_1$'	'$symb_2$'	→	'TDELTA($symb_1$,$symb_2$)'

Keyboard Access: (↰)(EQ LIB) UTILS (NXT) TDELT

Affected by Flags: Numerical Results (−3)

Remarks: 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.

Related Commands: TINC

TEACH

Teaching Examples Function: Creates an EXAMPLES (EXAM) subdirectory in the HOME directory and loads HP 48 programming, graphing, and solver examples from ROM into it.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Remarks: Items stored in the EXAMPLES subdirectory are deleted when CLTEACH is executed.

Related Commands: CLTEACH

TEXT

Show Stack Display Command: Displays the stack display.

Keyboard Access: (PRG) (NXT) OUT TEXT

Affected by Flags: None

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

Example: The command sequence `DRAW 5 WAIT TEXT` selects the graphics display and plots the contents of the reserved variable *EQ* (or reserved variable *ΣDAT*). It subsequently waits for 5 seconds, and then switches back from the graphics display to the stack display.

Related Commands: PICTURE, PVIEW

THEN

THEN Command: Starts the true-clause in conditional or error-trapping structure.

See the IF and IFERR entries for syntax information.

Keyboard Access:

(PRG) (NXT) ERROR IFERR THEN

(PRG) BRCH CASE THEN

(PRG) BRCH IF THEN

Remarks: See the IF and IFERR entries for more information.

Related Commands: CASE, ELSE, END, IF, IFERR

TICKS

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

Level 1	→	Level 1
	→	$\#n_{time}$

Keyboard Access: (↰)(TIME) TICKS

Affected by Flags: None

Remarks: TICKS enables elapsed time computations.

Example: If the result from a previous invocation from TICKS is in level 1, then `TICKS SWAP - B→R 8192 /` returns a real number whose value is the elapsed time in seconds between the two invocations.

Related Commands: TIME

TIME

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

Level 1	→	Level 1
	→	*time*

Keyboard Access: (↰)(TIME) TIME

Affected by Flags: None

Remarks: *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).

Related Commands: DATE, TICKS, TSTR

→TIME

Set System Time Command: Sets the system time.

{}

Level 1	→	Level 1
time	→	

Keyboard Access: (↰)(TIME) →TIM

Affected by Flags: None

Remarks: *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.

→TIME

Example: `13.3341 →TIME` sets the system time to 1:33:41 PM.

Related Commands: CLKADJ, →DATE

TINC

Temperature Increment Command: Calculates a temperature increment.

{ }

Level 2	Level 1	→	Level 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($symb_1$,$symb_2$)'

Keyboard Access: (⬏)(EQ LIB) UTILS (NXT) TINC

Affected by Flags: Numerical Results (−3)

Remarks: 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.

Related Commands: TDELTA

TLINE

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.

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

Keyboard Access: (PRG) PICT TLINE

Affected by Flags: None

Example: The following program toggles on and off 10 times the pixels on the line defined by user-unit coordinates `(1,1)` and `(9,9)`. Each state is maintained for .25 seconds.

```
«
 ERASE 0 10 XRNG 0 10 YRNG
 { # 0d # 0d } PVIEW
 «
  1 10 START
   (1,1) (9,9) TLINE
   .25 WAIT
  NEXT
 »
»
```

Related Commands: ARC, BOX, LINE

TMENU

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

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

Keyboard Access: ⬅ MODES MENU TMEN

Affected by Flags: None

Remarks: 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 48 built-in menus and the corresponding menu numbers (x_{menu}).

Examples: `7 TMENU` displays the first page of the MTH MATR menu.

`48.02 TMENU` displays the second page of the UNITS MASS menu.

`768 TMENU` displays the first page of commands in library 768.

`{ A 123 "ABC" } TMENU` displays the custom menu defined by the list argument.

`'MYMENU' TMENU` displays the custom menu defined by the name argument.

Related Commands: MENU, RCLMENU

TOT

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

Level 1	→	Level 1
	→	x_{sum}
	→	[x_{sum1} x_{sum2} ... x_{summ}]

Keyboard Access: ⬅ STAT 1VAR TOT

Affected by Flags: None

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

Related Commands: MAXΣ, MINΣ, MEAN, PSDEV, PVAR, SDEV, VAR

TRACE

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

Level 1	→	Level 1
[[*matrix*]]$_{n \times n}$	→	x_{trace}

Keyboard Access: MTH MATR NORM NXT TRACE

Affected by Flags: None

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

TRANSIO

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

{ }

Level 1	→	Level 1
n_{option}	→	

Keyboard Access: ⬅ I/O IOPAR TRAN

Affected by Flags: None

Remarks: 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)

Related Commands: BAUD, CKSM, PARITY

TRN

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

{ }

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

Keyboard Access: MTH MATR MAKE TRN

Affected by Flags: None

Remarks: TRN replaces an $n \times m$ matrix **A** with an $m \times n$ matrix $\mathbf{A}^T$, where:

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

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

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

Example: `[[ 2 3 1 ][ 4 6 9 ]] TRN` returns `[[ 2 4 ][ 3 6 ][ 1 9 ]]`.

Related Commands: CONJ

TRNC

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

{ }

Level 2	Level 1	→	Level 1
z_1	n_{truncate}	→	z_2
z_1	'$symb_{\mathrm{truncate}}$'	→	'TRNC(z_1,$symb_{\mathrm{truncate}}$)'
'$symb_1$'	n_{truncate}	→	'TRNC($symb_1$,n_{truncate})'
'$symb_1$'	'$symb_{\mathrm{truncate}}$'	→	'TRNC($symb_1$,$symb_{\mathrm{truncate}}$)'
[*array*]$_1$	n_{truncate}	→	[*array*]$_2$
x_unit	n_{truncate}	→	*y_unit*
x_unit	'$symb_{\mathrm{truncate}}$'	→	'TRNC(*x_unit*,$symb_{\mathrm{truncate}}$)'

Keyboard Access: (MTH) REPL (NXT) (NXT) TRNC

Affected by Flags: Numerical Results (−3)

Remarks: n_{truncate} (or '$\mathrm{symb}_{\mathrm{truncate}}$' 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.

Examples: `(4.5792,8.1275) 2 TRNC` returns `(4.57,8.12)`.

`[ 2.34907 3.96351 2.73453 ] -2 TRNC` returns `[ 2.3 3.9 2.7 ]`.

Related Commands: RND

TRUTH

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

Keyboard Access: (↰)(PLOT) PTYPE TRUTH

Affected by Flags: None

Remarks: 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).

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

TSTR

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

{ }

Level 2	Level 1	→	Level 1
date	*time*	→	*"DOW DATE TIME"*

Keyboard Access: ↰ TIME NXT NXT TSTR

Affected by Flags: Date Format (−42), Time Format (−41)

Remarks: 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.

Example: With flags −42 and −41 clear, `2.061990 14.55 TSTR` returns `"TUE 02/06/90 02:55:00P"`

Related Commands: DATE, TICKS, TIME

TVARS

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

Level 1	→	Level 1
n_{type}	→	{ *global* ... }
{ n_{type} ... }	→	{ *global* ... }

Keyboard Access: ↰ MEMORY DIR TVARS

Affected by Flags: None

Remarks: 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.

Related Commands: PVARS, TYPE, VARS

TVM

TVM Menu Command: Displays the TVM Solver menu.

Keyboard Access: This command must be typed in, but you can also access the menu using ↰ SOLVE TVM SOLVR.

Affected by Flags: None

Related Commands: AMORT, TVMBEG, TVMEND, TVMROOT

TVMBEG

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

Keyboard Access: This command must be typed in, but you can control begin/end mode with (⬅)(SOLVE) TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMEND, TVMROOT

TVMEND

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

Keyboard Access: This command must be typed in, but you can control begin/end mode with (⬅)(SOLVE) TVM BEG.

Affected by Flags: None

Related Commands: AMORT, TVM, TVMBEG, TVMROOT

TVMROOT

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

{ }

Level 1	→	Level 1
'TVM variable'	→	$x_{TVM variable}$

Keyboard Access: (⬅)(SOLVE) TVM TVMR

Affected by Flags: None

Related Commands: AMORT, TVM, TVMBEG, TVMEND

TYPE

Type Command: Returns the type number of an object.

Level 1	→	Level 1
obj	→	n_{type}

Keyboard Access:

(PRG) TYPE (NXT) (NXT) TYPE

(PRG) TEST (NXT) TYPE

Affected by Flags: None

Remarks: 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

Object Type Numbers (continued)

Object Type	Number
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory	15
Library	16
Backup object	17
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
External object	27-31

The HP 28S TYPE command returns number 8 for built-in functions and built-in commands (HP 48 TYPE numbers 18 and 19).

Related Commands: SAME, TVARS, VTYPE, ==

UBASE

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

{}

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

Keyboard Access: ⬅ (UNITS) UBASE

Affected by Flags: Numerical Results (−3)

Example: `30_knot UBASE` returns `15.4333333333_m/s`.

Related Commands: CONVERT, UFACT, →UNIT, UVAL

UFACT

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

{}

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

Keyboard Access: ⬅ (UNITS) UFACT

Affected by Flags: None

Remarks: UFACT is equivalent to this sequence:

```
OBJ→ 3 ROLLD / OVER / UBASE *
```

Example: `1_W 1_N UFACT` returns `1_N*m/s`.

Related Commands: CONVERT, UBASE, →UNIT, UVAL

→UNIT

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

{ }

Level 2	Level 1	→	Level 1
x	*y_unit*	→	*x_unit*

Keyboard Access:

(PRG) TYPE →UNIT

(⬑)(UNITS) →UNIT

Affected by Flags: None

Remarks: →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.

Related Commands: →ARRY, →LIST, →STR, →TAG

UNTIL

UNTIL Command: Starts test-clause in a DO ... UNTIL ... END indefinite loop structure.

See the DO entry for syntax information.

Keyboard Access: (PRG) BRCH DO UNTIL

Remarks: See the DO entry for more information.

Related Commands DO, END

UPDIR

Up Directory Command: Makes the parent of the current directory the new current directory.

Keyboard Access: (⬑)(UP)

Affected by Flags: None

Remarks: UPDIR has no effect if the current directory is *HOME*.

Related Commands: CRDIR, HOME, PATH, PGDIR

UTPC

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.

{ }

Level 2	Level 1	→	Level 1
n	*x*	→	*utpc(n, x)*

Keyboard Access: (MTH) (NXT) PROB (NXT) UTPC

Affected by Flags: None

Remarks: The defining equations are these:

- For x ≥ 0:

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

- 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 HP 48 factorial command.

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

Related Commands: UTPF, UTPN, UTPT

UTPF

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.

{ }

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

Keyboard Access: (MTH) (NXT) PROB (NXT) UTPF

Affected by Flags: None

Remarks: 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 48 factorial command.

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

Related Commands: UTPC, UTPN, UTPT

UTPN

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.

{ }

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

Keyboard Access: (MTH) (NXT) PROB (NXT) UTPN

Affected by Flags: None

Remarks: 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 $x < m$.

Related Commands: UTPC, UTPF, UTPT

UTPT

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.

{ }

Level 2	Level 1	→	Level 1
n	x	→	$utpt(n, x)$

Keyboard Access: (MTH) (NXT) PROB (NXT) UTPT

Affected by Flags: None

Remarks: 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 HP 48 factorial command.

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

Related Commands: UTPC, UTPF, UTPN

UVAL

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

UVAL *Unit Value* **Function**

Level 1	→	Level 1
x_unit	→	*x*
'*symb*'	→	'UVAL(*symb*)'

Keyboard Access: (↰)(UNITS) UVAL

Affected by Flags: Numerical Results (−3)

Related Commands: CONVERT, UBASE, UFACT, →UNIT

VAR

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

Level 1	→	Level 1
	→	x_{variance}
	→	[$x_{\text{variance1}}$ … $x_{\text{variancem}}$]

Keyboard Access: (↰)(STAT) 1VAR (NXT) VAR

Affected by Flags: None

Remarks: 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:

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

where x_1 is the ith coordinate value in a column, $\overline{x}$ is the mean of the data in this column, and n is the number of data points.

Related Commands: MAXΣ, MEAN, MINΣ, PSDEV, PVAR, SDEV, TOT

VARS

Variables Command: Returns a list of all variables' names in the VAR menu (the current directory).

Level 1	→	Level 1
	→	{ *global*$_1$ … *global*$_n$ }

Keyboard Access: (⬅)(MEMORY) DIR VARS

Affected by Flags: None

Related Commands: ORDER, PVARS, TVARS

VERSION

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

Level 1	→	Level 2	Level 1
	→	*"version number"*	*"copyright message"*

Keyboard Access: None. Must be typed in.

Affected by Flags: None

VTYPE

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

{ }

Level 1	→	Level 1
'name'	→	n_{type}
$:n_{port}$: $name_{backup}$	→	n_{type}
$:n_{port}$: $n_{library}$	→	n_{type}

Keyboard Access: (PRG) TYPE (NXT) (NXT) VTYPE

Affected by Flags: None

Remarks: If the named variable does not exist, VTYPE returns −1.

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

Related Commands: TYPE

→V2

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

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	[*x y*]
x	*y*	→	[*x* ∡*y*]
x	*y*	→	(*x*, *y*)
x	*y*	→	(*x*, ∡*y*)

Keyboard Access: (MTH) VECTR →V2

Affected by Flags: Complex Mode (−19), Coordinate System (−16)

Remarks: The result returned depends on the setting of flags −16 and 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)

Examples: With flag −19 clear, and flags −16 clear, `2 3 →V2` returns `[ 2 3 ]`.

With flag −19 set and flag −16 set (Polar/Spherical mode), `2 3 →V2` returns `(2,∡3)`.

Related Commands: V→, →V3

→V3

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

{ }

Level 3	Level 2	Level 1	→	Level 1
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}]

Keyboard Access: (MTH) VECTR →V3

Affected by Flags: Coordinate System (−15 and −16)

Remarks: 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∡$_{\mathrm{theta}}$ x_z]
Polar/Spherical (flag −15 and −16 set)	[x_1 x∡$_{\mathrm{theta}}$ x∡$_{\mathrm{phi}}$]

Examples: With flag −16 clear (Rectangular mode), `i 2 3 →V3` returns `[ i 2 3 ]`.

With flag −15 clear and −16 set (Polar/Cylindrical mode), `i 2 3 →V3` returns `[ i ∡2 3 ]`.

With flags −15 and −16 set (Polar/Spherical mode), `i 2 3 →V3` returns `[ i ∡2 ∡3 ]`.

Related Commands: V→, →V2

V→

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

{ }

Level 1	→	Level n .. Level 3	Level 2	Level 1
[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}

Keyboard Access: (MTH) VECTR V→

Affected by Flags: Coordinate System (−15 and −16)

The elements of the argument complex number or vector are converted from their values in Rectangular mode (the form in which the complex number or vector is stored internally) to the current coordinate system mode before being returned to the stack. This means that the element values returned to the stack always match the *displayed* element values of the argument vector or complex number.

Remarks: 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.

Examples: With flag −16 clear (Rectangular mode), `(2,3) V→` returns `2` to level 2 and `3` to level 1.

With flag −15 clear and flag −16 set (Polar/Cylindrical mode), `[ 2 ∠7 4 ] V→` returns `2` to level 3, `7` to level 2, and `4` to level 1.

`[ 9 7 5 3 ] V→` returns `9` to level 4, `7` to level 3, `5` to level 2, and `3` to level 1, independent of the state of flags −15 and −16.

Related Commands: →V2, →V3

*W

Multiply Width Command: Multiplies a plot's horizontal scale by x_{factor}.

{ }

Level 1	→	Level 1
x_{factor}	→	

Keyboard Access: ↰ PLOT PPAR NXT *W

Affected by Flags: None

Remarks: Executing *W changes the x-axis display range (x_{min} and x_{max} in the reserved variable *PPAR*). The plot center (the user-unit coordinate of the center pixel) does not change.

Related Commands: AUTO, *H, XRNG

WAIT

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

{ }

Level 1	→	Level 1
x	→	
0	→	x_{kev}
−1	→	x_{kev}

Keyboard Access: PRG NXT IN WAIT

Affected by Flags: None

Remarks: 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_{kev}, which defines where the pressed key is on the keyboard, and resumes program execution.

  x_{kev} 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_{kev}.
- Argument −1 works as with argument 0, except that the currently specified menu is also displayed.

↰, ↱, α, α ↰, and α ↱ are not by themselves valid keys.

Arguments `0` or `-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.

Examples: This program:

```
« "Press [1] to add■Press any other key to subtract"
1 DISP 0 WAIT IF 82.1 SAME THEN + ELSE - END »
```

displays a prompting message and halts program execution until a key is pressed. If the [1] key (location 82.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

This program:

```
  « { ADD { } { } { } { } SUB } MENU "Press [ADD]
  to add■ Press [SUB] to subtract" 1 DISP -1 WAIT
  IF 11.1 SAME THEN + ELSE - END »
```

builds a custom menu with labels ADD and SUB and a prompting message. Executing `-1 WAIT` displays the custom menu (note that it's not active) and suspends execution for keyboard input. If the ADD menu key (location 11.1) is pressed, two numbers on the stack are added. If any other key is pressed, two numbers on the stack are subtracted.

Related Commands: KEY

WHILE

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

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

Keyboard Access: (PRG) BRCH WHILE WHILE

Affected by Flags: None

Remarks: 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.

Related Commands: DO, END, REPEAT

WIREFRAME

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

Keyboard Access: (↰)(PLOT) (NXT) 3D PTYPE WIREF

Affected by Flags: None

Remarks: 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_{eve} y_{eye} z_{eve} 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*

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

WSLOG

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

Level 1	→	Level 4 ... Level 1
	→	"log_4" ... "log_1"

Keyboard Access: None. Must be typed in.

Affected by Flags: Date Format (−42)

Remarks: 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 and then ON to wake the calculator up. ON SPC puts the HP 48 in "Coma mode" (very low power *with the system clock stopped*). Pressing ON then clears the log and warmstarts the system.
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 a "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 IR transmission (timeout).
3	Run through address 0.
4	System time is corrupt.
5	A Deep Sleep wakeup (for example, ON, Alarm) detected no change to port status, but some changes in data on one or both cards.

Code	Description
6	Unused.
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	One of the following anomalies involving device configuration was detected: ■ The interrupt system detected that one of the five devices was not configured. ■ During a warmstart, an unexpected device ID chain was encountered while attempting to configure 3 (Port1, Port2, Xtra) of the 5 devices. ■ Same as previous, but detected during Deep Sleep wakeup.
9	The alarm list is corrupt.
A	Unused.
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.
E	The configuration table is corrupt (bad checksum for table data).
F	The system RAM card was removed.

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.

ΣX

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

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ⬅ STAT SUMS ΣX

Affected by Flags: None

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

Related Commands: NΣ, XCOL, ΣX*Y, ΣX^2, ΣY, ΣY^2

ΣX^2

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 *ΣDAT*).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ⬅ STAT SUMS ΣX^2

Affected by Flags: None

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

Related Commands: NΣ, ΣX, XCOL, ΣX*Y, ΣY, ΣY^2

XCOL

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

{ }

Level 1	→	Level 1
n_{col}	→	

Keyboard Access: ⬅ STAT ΣPAR XCOL

Affected by Flags: None

Remarks: The independent-variable column number is stored as the first parameter in the reserved variable *ΣPAR*. The default independent-variable column number is 1.

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

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

XMIT

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

{ }

Level 1	→	Level 2	Level 1
"*string*"	→		1
"*string*"	→	"$substring_{\text{unsent}}$"	0

Keyboard Access: ⬅ I/O NXT SERIA XMIT

Affected by Flags: I/O Device (−33)

Remarks: 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.

Related Commands: BUFLEN, SBRK, SRECV, STIME

XOR

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

{ }

Level 2	Level 1	→	Level 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* XOR *symb*'
'*symb*'	*T/F*	→	'*symb* XOR *T/F*'
'*symb*$_1$'	'*symb*$_2$'	→	'*symb*$_1$ XOR *symb*$_2$'

Keyboard Access:

(MTH) BASE (NXT) LOGIC XOR

(PRG) TEST (NXT) XOR

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: 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.

Related Commands: AND, NOT, OR

XPON

Exponent Function: Returns the exponent of the argument.

{ }

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

Keyboard Access: (MTH) REAL (NXT) XPON

Affected by Flags: Numerical Results (−3)

Examples: `1.2E34 XPON` returns `34`.

`12.4E3 XPON` returns `4`.

`'A*1E34' XPON` returns `'XPON(A*1E34)'`.

Related Commands: MANT, SIGN

XRECV

XModem Receive Command: Prepares the HP 48 to receive an object via XModem. The received object is stored in the given variable name.

Level 1	→	Level 1
'*name*'	→	

Keyboard Access: ⬅ I/O NXT XRECV

Affected by Flags: I/O Device (−33), RECV Overwrite (−36)

Remarks: 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 48s, 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).

Related Commands: BAUD, RECV, RECN, SEND, XSEND

XRNG

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

{ }

Level 2	Level 1	→	Level 1
x_{min}	x_{max}	→	

Keyboard Access: ⬅ PLOT PPAR XRNG

Affected by Flags: None

Remarks: 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.

Related Commands: AUTO, PDIM, PMAX, PMIN, YRNG

XROOT

xth Root of y Command: Computes the xth root of a real number.

{ }

Level 2	Level 1	→	Level 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*)'

Keyboard Access: ➡ $\sqrt[x]{y}$

Affected by Flags: Numerical Results (−3)

Remarks: Note that while the *stack* syntax is y x `XROOT` (the root is the second argument), the *algebraic* syntax is `XROOT(`x`,`y`)` (the root is the first argument) for consistency with the EquationWriter application.

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

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

Related Commands: ^

XSEND

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

Level 1	→	Level 1
'name'	→	

Keyboard Access: ⬅ (I/O) (NXT) XSEN

Affected by Flags: I/O Device (−33)

Remarks: A receiving HP 48 must execute XRECV to receive an object via XModem.

To start the transfer more quickly, 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 48s, 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).

Related Commands: BAUD, RECN, RECV, SEND, XRECV

XVOL

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

{ }

Level 2	Level 1	→	Level 1
x_{left}	x_{right}	→	

Keyboard Access: ⬅ (PLOT) (NXT) 3D VPAR XVOL

Affected by Flags: None

Remarks: 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*. See appendix D, "Reserved Variables," for more information about *VPAR*.

Related Commands: EYEPT, XXRNG, YVOL, YYRNG, ZVOL

XXRNG

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

{ }

Level 2	Level 1	→	Level 1
x_{min}	x_{max}	→	

Keyboard Access: ⬅ (PLOT) (NXT) 3D VPAR XXRN

Affected by Flags: None

Remarks: 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*. See appendix D, "Reserved Variables," for more information about *VPAR*.

Related Commands: EYEPT, NUMX, NUMY, XVOL, YVOL, YYRNG, ZVOL

ΣX∗Y

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).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ⬅ STAT SUMS ΣX*Y

Affected by Flags: None

Remarks: The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable ΣPAR. The default independent-variable column number is 1.

The dependent-variable column is specified by YCOL and is stored as the second parameter in reserved variable ΣPAR. The default dependent-variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX^2, ΣY, ΣY^2

ΣY

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

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ⬅ STAT SUMS ΣY

Affected by Flags: None

Remarks: The dependent variable column is specified by YCOL, and is stored as the second parameter in the reserved variable ΣPAR. The default dependent variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX∗Y, ΣX^2, YCOL, ΣY^2

ΣY^2

Sum of Squares of y-Values Command: Sums the squares of the values in the dependent variable column of the current statistical matrix (reserved variable ΣDAT).

Level 1	→	Level 1
	→	x_{sum}

Keyboard Access: ⬅ STAT SUMS ΣY^2

Affected by Flags: None

Remarks: The dependent variable column is specified by YCOL. The default dependent variable column number is 2.

Related Commands: NΣ, ΣX, XCOL, ΣX∗Y, ΣX^2, YCOL, ΣY

YCOL

Dependent Column Command: Specifies the dependent variable column of the current statistics matrix (reserved variable ΣDAT).

{ }

Level 1	→	Level 1
n_{col}	→	

Keyboard Access: (⬅)(STAT) ΣPAR YCOL

Affected by Flags: None

Remarks: The dependent variable column number is stored as the second parameter in the reserved variable *ΣPAR*. The default dependent variable column number is 2.

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

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

YRNG

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

{ }

Level 2	Level 1	→	Level 1
y_{min}	y_{max}	→	

Keyboard Access: (⬅)(PLOT) PPAR YRNG

Affected by Flags: None

Remarks: 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.

Related Commands: AUTO, PDIM, PMAX, PMIN, XRNG

YSLICE

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

Keyboard Access: (⬅)(PLOT)(NXT) 3D PTYPE YSLIC

Affected by Flags: None

Remarks: 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*

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

YVOL

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

{ }

Level 2	Level 1	→	Level 1
y_{near}	y_{far}	→	

Keyboard Access: ↰ PLOT NXT 3D VPAR YVOL

Affected by Flags: None

Remarks: 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*.

Related Commands: EYEPT, XVOL, XXRNG, YYRNG, ZVOL

YYRNG

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

{ }

Level 2	Level 1	→	Level 1
y_{near}	y_{far}	→	

Keyboard Access: ↰ PLOT NXT 3D VPAR YYRN

Affected by Flags: None

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

Related Commands: EYEPT, XVOL, XXRNG, YVOL, ZVOL

ZFACTOR

Gas Compressibility Z Factor Function: Calculates the gas compressibility correction factor for nonideal behavior of a hydrocarbon gas.

{ }

Level 2	Level 1	→	Level 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($symb_1$,$symb_2$)'

Keyboard Access: ↰ EQ LIB UTILS ZFACT

ZFACTOR

Affected by Flags: Numerical Results (−3)

Remarks: 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.

ZVOL

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

{}

Level 2	Level 1	→	Level 1
x_{low}	x_{high}	→	

Keyboard Access: ⬏ (PLOT) (NXT) 3D VPAR ZVOL

Affected by Flags: None

Remarks: 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*.

Related Commands: EYEPT, XVOL, XXRNG, YVOL, YYRNG

+

Add Analytic Function: Returns the sum of the arguments.

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1+z_2
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1??+??2}$
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*"
$\#n_1$	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*'
$grob_1$	$grob_2$	→	$grob_3$

Keyboard Access: (+)

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: 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 level 1 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 $<$, $>$, $\le$, $\ge$, $==$, and $\neq$. 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.

Examples: `{ 1 2 3 } { A B C } +` returns `{ 1 2 3 A B C }`.

`5_ft 9_in +` returns `69_in`.

`[[ 0 1 ][ 1 3 ]] [[ 2 1 ][ 0 1 ]] +` returns `[[ 2 2 ][ 1 4 ]]`.

`'FIRST' 'SECOND' +` returns `'FIRST+SECOND'`

Related Commands: −, *, /, =

−

Subtract Analytic Function: Returns the difference of the arguments: the object in level 1 is subtracted from the object in level 2.

{ }

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1-z_2
[*array*]$_1$	[*array*]$_2$	→	[*array*]$_{1-2}$
z	'*symb*'	→	'*z−symb*'
'*symb*'	z	→	'*symb−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*'

Keyboard Access: ⊟

Affected by Flags: Numerical Results (−3)

Remarks: 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 level 2 number and the two's complement of the level 1 number. (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 level 2 number and the two's complement of the level 1 number.

The difference of two unit objects is a unit object with the same dimensions as the level 1 object. 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.

Example: `25_ft 8_in` − returns `292_in`.

`[[ 5 i ][ 3 3 ]] [[ 2 i ][ 0 i ]]` − returns `[[ 3 0 ][ 3 2 ]]`.

`'TOTAL' 'PART'` − returns `'TOTAL-PART'`

Related Commands: +, *, /, =

*

Multiply Analytic Function: Returns the product of the arguments.

{ }

Level 2	Level 1	→	Level 1
z_1	z_2	→	$z_1 z_2$
[[*matrix*]]	[*array*]	→	[[*matrix* × *array*]]
z	[*array*]	→	[*z* × *array*]
[*array*]	*z*	→	[*array* × *z*]
z	'*symb*'	→	'*z* * *symb*'
'*symb*'	*z*	→	'*symb* * *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 × $unit_y$
x	*y_unit*	→	*xy_unit*
x_unit	*y*	→	*xy_unit*
'*symb*'	*x_unit*	→	'*symb* * *x_unit*'
x_unit	'*symb*'	→	'*x_unit* * *symb*'

Keyboard Access: ⊠

Affected by Flags: Numerical Results (−3), Binary Integer Wordsize (−5 through −10)

Remarks: 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 (level 2) by an array (level 1) returns a matrix product. The matrix must have the same number of columns as the array in level 1 has rows (or elements, if it is a vector).

Although a vector is entered and displayed as a *row* of numbers, the HP 48 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 scalar parts and the unit parts are multiplied separately.

Related Commands: +, −, /, =

/

Divide Analytic Function: Returns the quotient of the arguments: the level 2 object divided by the level 1 object.

{ }

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1 / z_2
[*array*]	[[*matrix*]]	→	[[$matrix^{-1}$ × *array*]]
[*array*]	*z*	→	[*array/z*]
z	'*symb*'	→	'*z/symb*'
'*symb*'	*z*	→	'*symb/z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ / $symb_2$'
#n_1	n_2	→	#n_3
n_1	#n_2	→	#n_3
#n_1	#n_2	→	#n_3
x_$unit_1$	y_$unit_2$	→	(x/y)_$unit_1$ / $unit_2$
x	y_*unit*	→	(x/y)_1/*unit*
x_*unit*	*y*	→	(x/y)_*unit*
'*symb*'	x_*unit*	→	'*symb*/x_*unit*'
x_*unit*	'*symb*'	→	'x_*unit*/*symb*'

Keyboard Access:

⬑ (SOLVE) SYS [illegible]

(÷)

Affected by Flags: Numerical Results (−3)

Remarks: 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, $\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$. 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 scalar parts and the unit parts are divided separately.

Related Commands: +, −, *, =

^

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

{ }

Level 2	Level 1	→	Level 1
w	*z*	→	w^z
z	'*symb*'	→	'*z*^(*symb*)'
'*symb*'	*z*	→	'(*symb*)^*z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$^($symb_2$)'
*x*_*unit*	*y*	→	x^y_$unit^y$
*x*_*unit*	'*symb*'	→	'(*x*_*unit*)^(*symb*)'

Keyboard Access: (y^x)

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

Remarks: If either argument is complex, the result is complex.

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

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

Related Commands: EXP, ISOL, LN, XROOT

<

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

{ }

Level 2	Level 1	→	Level 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*'

Keyboard Access: (PRG) TEST [illegible]

Affected by Flags: Numerical Results (−3)

Remarks: The function < returns a true test result (1) if the level 2 argument is less than the level 1 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 +.

Related Commands: ≤, >, ≥, ==, ≠

≤

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

Level 2	Level 1	→	Level 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*'

Keyboard Access: (PRG) TEST ≤

Affected by Flags: Numerical Results (−3)

Remarks: The function ≤ returns a true test result (1) if the level 2 argument is less than or equal to the level 1 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 temperature. 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 +.

Related Commands: <, >, ≥, ==, ≠

>

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

{ }

Level 2	Level 1	→	Level 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*'

Keyboard Access: (PRG) TEST >

Affected by Flags: Numerical Results (−3)

Remarks: The function > returns a true test result (1) if the level 2 argument is greater than the level 1 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 temperature. 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 +.

Related Commands: <, ≤, ≥, ==, ≠

≥

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

{ }

Level 2	Level 1	→	Level 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*'

Keyboard Access: (PRG) TEST ≥

Affected by Flags: Numerical Results (−3)

Remarks: The function ≥ returns a true test result (1) if the level 2 argument is greater than or equal to the level 1 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 temperature. 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 +.

Related Commands: <, ≤, >, ==, ≠

=

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

Level 2	Level 1	→	Level 1
z_1	z_2	→	'$z_1 = z_2$'
z	'*symb*'	→	'*z=symb*'
'*symb*'	*z*	→	'*symb=z*'
'$symb_1$'	'$symb_2$'	→	'$symb_1 = symb_2$'
y	*x_unit*	→	'*y=x_unit*'
y_unit	*x*	→	'*y_unit=x*'
y_unit	*x_unit*	→	'*y_unit=x_unit*'
'*symb*'	*x_unit*	→	'*symb=x_unit*'
x_unit	'*symb*'	→	'*x_unit=symb*'

Keyboard Access: ↰ =

Affected by Flags: Numerical Results (−3)

Remarks: 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.

The numerical evaluation of an equation using the HP Solve application implicitly involves the subtraction of terms. See the entry for "−" for information about the effects of subtraction.

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.

Related Commands: DEFINE, EVAL, –

==

Logical Equality Function: Tests if two objects are equal.

{}

Level 2	Level 1	→	Level 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*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ ==$symb_2$'

Keyboard Access: PRG TEST ==

Affected by Flags: Numerical Results (−3)

Remarks: 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, so, for example, 6 (6,0) == returns 1.

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 temperature. 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 +.

Related Commands: SAME, TYPE, <, ≤, >, ≥, ≠

≠

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

{}

Level 2	Level 1	→	Level 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*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ ≠ $symb_2$'

Keyboard Access: PRG TEST ≠

Affected by Flags: Numerical Results (−3)

Remarks: 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 real number, so, for example, 6 (6,0) ≠ returns 0.

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 +.

Related Commands: SAME, TYPE, <, ≤, >, ≥, ==

!

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.

{ }

Level 1	→	Level 1
n	→	*n*!
x	→	Γ(*x*+1)
'*symb*'	→	'(*symb*!)'

Keyboard Access: (MTH) (NXT) PROB !

Affected by Flags: Numerical Results (−3), Underflow Exception (−20), Overflow Exception (−21)

Remarks: 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 this:

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

and defined for other values of x by analytic continuation:

$$\Gamma(x+1) = n \cdot \Gamma(x)$$

Related Commands: COMB, PERM

∫

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

{ }

Level 4	Level 3	Level 2	Level 1	→	Level 1
lower limit	*upper limit*	*integrand*	'*name*'	→	'$symb_{\text{integral}}$'

Keyboard Access: (↱) (∫)

Affected by Flags: Numerical Results (−3), Number Format (−45 to −50)

Remarks: 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 to level 1. The HP 48 does symbolic integration by *pattern matching*. The HP 48 can integrate the following:

- 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.

A successful result of symbolic integration has this form:

'*result*|(*name*=*upper limit*)-(*result*|(*name*=*lower limit*))'

See the | (where) entry for more information about its functionality. A second evaluation substitutes the limits of integration into the variable of integration, completing the procedure.

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.

Examples: In Symbolic Results mode (flag −3 clear) this command sequence:

```
1 2 '10*X' 'X' ∫
```

returns

```
'10*(X^2/2)|(X=2)-(10*(X^2/2)|(X=1))'
```

Subsequent evaluation substitutes the limits of integration, and returns 15.

In Numeric Results mode (flag −3 set) the above command sequence returns the numerical approximation 15. In addition, the variable *IERR* is created, and contains the error of integration `.00000000015`.

Related Commands: TAYLR, ∂, Σ

∂

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

{ }

Level 2	Level 1	→	Level 1
'*symb*$_1$'	'*name*'	→	'*symb*$_2$'
z	'*name*'	→	0
x_unit	'*name*'	→	0

Keyboard Access: ⮣ ∂

Affected by Flags: Numerical Results (−3)

Remarks: 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 the HP 48 does not provide a derivative, ∂ returns a new function whose name is *der* followed by the original function name.

Example: In Radians mode, the command sequence `'∂X(SIN(Y))'` `EVAL` returns `'COS(Y)*∂X(Y)'`.

When Y has the value `X^2`, the command sequence `'SIN(Y)'` `'X'` `∂` returns `'COS(X^2)*(2*X)'`. The differentiation has been executed in stack syntax, so that all of the steps of differentiation have been carried out in a single operation.

Related Commands: TAYLR, $\int$, Σ

%

Percent Function: Returns x (level 2) percent of y (level 1).

{ }

Level 2	Level 1	→	Level 1
x	*y*	→	*xy/100*
x	*'symb'*	→	*'%(x,symb)'*
'symb'	*x*	→	*'%(symb,x)'*
'symb₁'	*'symb₂'*	→	*'%(symb₁ , symb₂)'*
x	*y_unit*	→	*(xy/100)_unit*
x_unit	*y*	→	*(xy/100)_unit*
'symb'	*x_unit*	→	*'%(symb,x_unit)'*
x_unit	*'symb'*	→	*'%(x_unit,symb)'*

Keyboard Access: (MTH) REAL %

Affected by Flags: Numerical Results (−3)

Remarks: 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 +.

Example: `23.7 995 %` returns `235.815`.

`15 176_kg %` returns `26.4_kg`.

`100_°C 50 %` returns `50_°C`.

Related Commands: %CH, %T

π

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

Level 1	→	Level 1
	→	'π'
	→	3.14159265359

Keyboard Access: (⬅)(π)

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Evaluating π returns its numerical representation if flag −2 or −3 is set; otherwise, returns its symbolic representation.

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

In Radians mode with flags −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.

Related Commands: e, i, MAXR, MINR, →Qπ

Σ

Summation Function: Calculates the value of a finite series.

Level 4	Level 3	Level 2	Level 1	→	Level 1
'indx'	x_{init}	x_{final}	*smnd*	→	x_{sum}
'indx'	*'init'*	x_{final}	*smnd*	→	'Σ(*indx=init*,x_{final},*smnd*)'
'indx'	x_{init}	*'final'*	*smnd*	→	'Σ(*indx*=x_{init},*final*,*smnd*)'
'indx'	*'init'*	*'final'*	*smnd*	→	'Σ(*indx=init,final,smnd*)'

Keyboard Access: (↱)(Σ)

Affected by Flags: Symbolic Constants (−2), Numerical Results (−3)

Remarks: 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 this:

'Σ(*index=initial,final,summand*)'

Examples: In Symbolic Results mode (flags −2 and −3 clear), the command sequence 'N' 1 5 'A^N' Σ returns 'A+A^2+A^3+A^4+A^5'.

The command sequence 'N' 1 'M' 'A^N' Σ returns 'Σ(N=1,M,A^N)'.

Related Commands: TAYLR, ∫, ∂

Σ+

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

Level m ... Level 2	Level 1	→	Level 1
	x	→	
	[x_1 x_2 ... x_m]	→	
	[[x_{11} ... x_{1m}] [x_{n1} ... x_{nm}]]	→	
x_1 ... x_{m-1}	x_m	→	

Keyboard Access: (↰)(STAT) DATA Σ+

Affected by Flags: None

Remarks: 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 of *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 *ΣDAT*). If *ΣDAT* does not exist, Σ+ creates an *n* × *m* matrix and stores the matrix in *ΣDAT*. If *ΣDAT* 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 *ΣDAT* 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.

Example: The sequence CLΣ [2 3 4] Σ+ 3 1 7 Σ+ creates the matrix [[2 3 4][3 1 7]] in *ΣDAT*.

Related Commands: CLΣ, RCLΣ, STOΣ, Σ−

Σ−

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 *ΣDAT*).

	→	Level 1
	→	x
	→	[x_1 x_2 … x_m]

Keyboard Access: (↰)(STAT) DATA Σ-

Affected by Flags: None

Remarks: 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+.

Related Commands: CLΣ, RCLΣ, STOΣ, Σ+

√

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

Level 1	→	Level 1
z	→	$\sqrt{z}$
x_unit	→	$\sqrt{x}_u nit^1/2$
'$symb$'	→	'$\sqrt{(symb)}$'

Keyboard Access: ($\sqrt{x}$)

Affected by Flags: Principal Solution (−1), Numerical Results (−3)

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

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

where $r = \text{ABS}\ (x_1, y_1)$, and $\theta = \text{ARG}\ (x_1, y_1)$.

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 √.

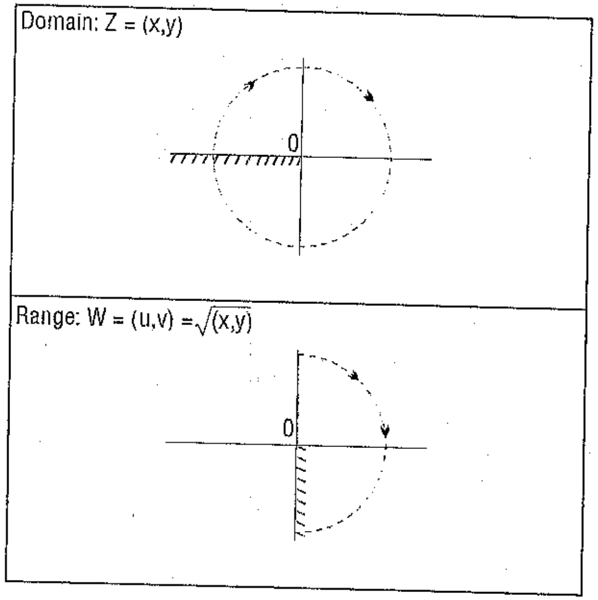
The principal branch used by the HP 48 for √ was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued square root function occurs where the corresponding real-valued function is undefined. The principal 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.

Related Commands: SQ, ^, ISOL



_

Attach Unit Function: Attaches a unit expression to a real number. Performed automatically in the Unit Catalog (⮣ UNITS).

Level 2	Level 1	→	Level 1
x	*'unit expression'*	→	*x_unit*

Keyboard Access: ⮣ _

Affected by Flags: None

Related Commands: →UNIT

| (Where)

Where Function: Substitutes values for names in an expression.

Level 2	Level 1	→	Level 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*)

Keyboard Access: ⮠ SYMBOLIC NXT ▨

Affected by Flags: Numerical Results (−3)

Remarks: | 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.

Related Commands: APPLY, QUOTE

→

Create Local Variables Command: Creates local variables.

Level n ... Level 1	→	Level 1
obj$_1$ ... *obj*$_n$	→	

Keyboard Access: ↱ →

Affected by Flags: None

Remarks: *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 from the stack 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*'

Example: This program:

```
« → x y « x y * x y - + » »
```

takes an object from level 2 and stores it in local variable x, takes an object from level 1 and stores it in local variable y, and executes calculations with x and y in the defining procedure (in this case a program). When the defining procedure ends, local variables x and y disappear.

User-Defined Functions. A user-defined function is a variable containing a program that consists solely of a local variable structure.

For example, the variable A, containing this program:

```
« → x y z 'x*y/2+z' »
```

is a user-defined function. Like a built-in function, a user-defined function can take its arguments in stack syntax or algebraic syntax, and can take symbolic arguments. In addition, a user-defined function is differentiable if its defining procedure is an algebraic expression that contains only differentiable functions.

Related Commands: DEFINE, STO

4

Equation Reference

The Equation Library consists of 15 subjects and more than 100 titles. Each subject and title has a number that you can use with SOLVEQN to specify the set of equations. These numbers are shown in parentheses after the headings.

See the end of this section for references given in each subject (Reference: x). Remember that some equations are estimates and assume certain conditions. See the references or other standard texts for assumptions and limitations of the equations.

Solutions in the examples have been rounded to four decimal places.

Columns and Beams (1)

Variable Names and Descriptions

ϵ	Eccentricity (offset) of load
σcr	Critical stress
σmax	Maximum stress
θ	Slope at x
A	Cross-sectional area
a	Distance to point load
c	Distance to edge fiber (Eccentric Columns), or Distance to applied moment (beams)
E	Modulus of elasticity
I	Moment of inertia