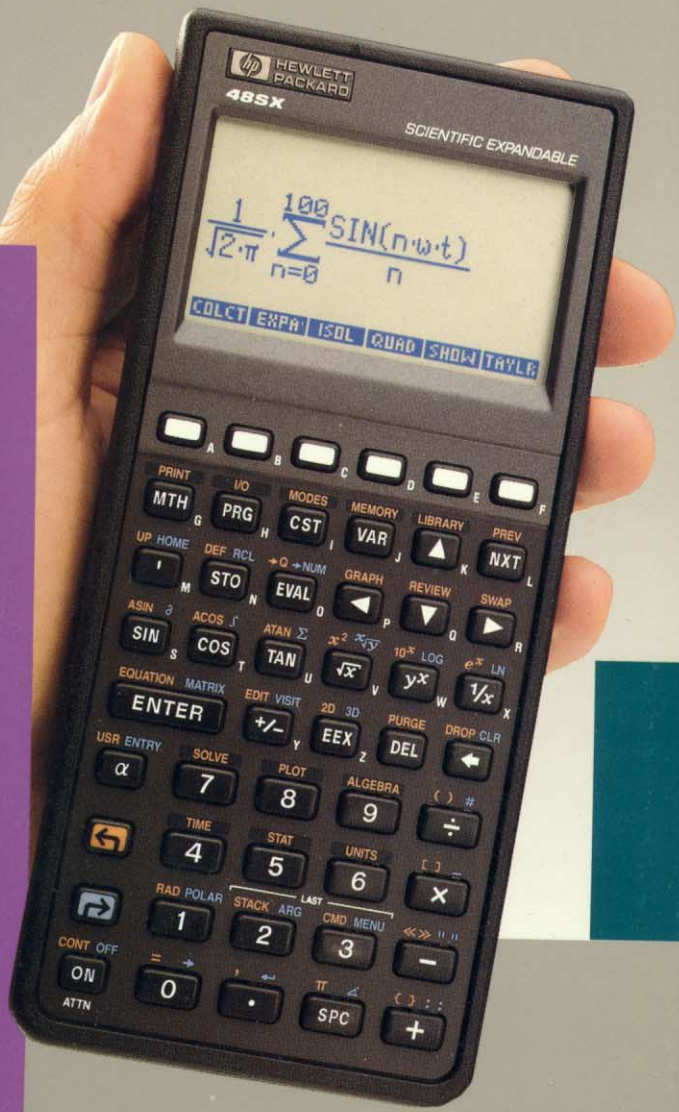
HEWLETT PACKARD

HP 48
Programmer's Reference Manual



Terms Used in Stack Diagrams

Term	Description
arg	Argument.
[*array*]	Real or complex vector or matrix.
[*C-array*]	Complex vector or matrix.
date	Date in form MM.DDYYYY or DD.MMYYYY.
{ *dim* }	List of one or two array dimensions (real numbers).
'*global*'	Global name.
grob	Graphics object.
HMS	A real-number time or angle in hours-minutes-seconds format.
{ *list* }	List of objects.
local	Local name.
[[*matrix*]]	Real or complex matrix.
n or *m*	Positive integer real number (rounded if non-integer).
$:n_{port}:name_{backup}$	Backup identifier.
$:n_{port}:n_{library}$	Library identifier.
#n	Binary integer.
{ *#n #m* }	Pixel coordinates. (Uses binary integers.)
'*name*'	Global or local name.
obj	Any object.
PICT	Current graphics object.
« *program* »	Program.
[*R-array*]	Real vector or matrix.
"*string*"	Character string.
'*symb*'	Expression, equation, or name treated as an algebraic.
T/F	Test result used as an argument: zero (false) or non-zero (true) real number.
0/1	Test result returned by a command: zero (false) or one (true).
time	Time in form HH.MMSSs.
[*vector*]	Real or complex vector.
x or *y*	Real number.
x_unit	Unit object; or, a real number treated as a dimensionless unit object.
(*x*,*y*)	Complex number in rectangular form, or user-unit coordinate.
z	Real or complex number.

Comments on the HP 48 Programmer's Reference Manual

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications.

HP 48 Programmer's Reference Manual

Please circle a response for each of the statements below. You can use the **Comments** space to provide additional opinions.

1 = Strongly Agree
2 = Agree
3 = Neutral
4 = Disagree
5 = Strongly Disagree

- I am satisfied with the product documentation. 1 2 3 4 5
- The manual is well organized. 1 2 3 4 5
- I can find the information I want. 1 2 3 4 5
- The information in the manual is accurate. 1 2 3 4 5
- I can easily understand the instructions. 1 2 3 4 5
- The manual contains enough examples. 1 2 3 4 5
- The examples are appropriate and helpful. 1 2 3 4 5
- The illustrations are clear and helpful. 1 2 3 4 5
- The amount of information is: too much appropriate too little
- The parts I refer to most frequently are:

Command Dictionary A B C D

Comments: _____

_____

_____

_____

_____

_____

Name: _____

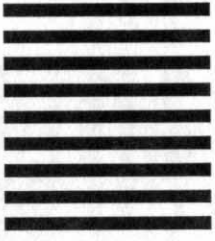
Address: _____

City/State/Zip: _____

Occupation: _____

Phone: (_____) _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 38 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
DOCUMENTATION DEPARTMENT
CORVALLIS DIVISION
1000 NE CIRCLE BLVD
CORVALLIS OR 97330-9973

HP 48

Programmer's Reference Manual

Edition 1 July 1990
Reorder Number 00048-90054

Notice

This manual and any examples contained herein are provided "**as is**" and are subject to change without notice. **Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.** Hewlett-Packard Co. shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples contained herein.

Corvallis Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Printing History

Edition 1 July 1990 Mfg. No. 00048-90053

What's in This Manual

This manual contains concise reference information for the HP 48: a brief, alphabetical listing for each programmable keyword, and tables of error messages, units, flags, and reserved variables.

The *HP 48 Owner's Manual*, on the other hand, contains conceptual, descriptive information, organized by subject area and menu.

This reference manual consists of the following:

- Command Dictionary: the main part of the manual. Organized alphabetically by command, it summarizes each programmable operation. The syntax appears in a *stack diagram* that shows what each keyword requires as arguments and returns as results.
- Table of Error and Status Messages (appendix A): an alphabetical listing of error and status messages.
- Table of Units (appendix B): definitions of the types of units available in the HP 48.
- Table of System Flags (appendix C): the predefined flags in the HP 48.
- Reserved Variables (appendix D): lists the reserved variables and their contents.

Contents

	6	**Command Dictionary**
A	464	**Table of Error and Status Messages**
B	481	**Table of Units**
C	486	**Table of System Flags**
D	494	**Reserved Variables**

Command Dictionary

The following three topics explain how to read the stack diagrams in the command dictionary, how commands are alphabetized in the dictionary, and the meaning of the command classifications at the upper right corner of each stack diagram.

How to Read Stack Diagrams. Each entry in the command dictionary includes a *stack diagram*. This is a table showing the *arguments* that the command, function, or analytic function takes from the stack and the *results* that it returns to the stack. The "→" character in the table separates the arguments from the results. The stack diagram for a command may contain more than one "argument → result" line, reflecting all possible combinations of arguments and results for that command. Consider this example:

R→C *Real-to-Complex* **Command**

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

This diagram indicates that the *command* R→C (*real-to-complex*) takes two arguments (an argument from level 2 and an argument from level 1), and returns one result (to level 1). R→C can take either real number arguments *x* and *y*, in which case it returns a complex number (*x*, *y*), or real array arguments [*R-array*$_1$] and [*R-array*$_2$], in which case it returns a complex array [*C-array*].

The definitions of the abbreviations used for argument and result objects are contained in the table, "Terms Used in Stack Diagrams," on the inside of the front cover. Often, descriptive subscripts are added to convey more information.

Some commands affect a calculator state — a mode, a reserved variable, a flag, or a display — without returning a result to the stack, and in some cases, without taking any arguments from the stack. For example, the command ERASE erases *PICT*, taking no arguments and returning no results. Its stack diagram looks like this:

ERASE *Erase PICT* **Command**

Level 1	→	Level 1
	→	

How Commands Are Alphabetized. Commands appear in alphabetical order. Command names that contain special (non-alphabetic) characters are organized as follows:

- For commands that contain *both* special and alphabetic characters:
  - A special character at the *start* of a command name is *ignored*. Therefore, the command *H follows the command GXOR and precedes the command HALT.
  - A special character *within* or at the *end* of a command name is considered to follow "Z" at the end of the alphabet. Therefore, the command R→B follows the command RSD and precedes the command R→C.
- Commands that contain *only* special characters appear at the end of the dictionary.

Classification of Operations. The command dictionary contains HP 48 *commands*, *functions*, and *analytic functions*. Commands are calculator operations that can be executed from a program. Functions are commands that can be included in algebraic objects. Analytic functions are functions for which the HP 48 provides an inverse and a derivative. The command classification is located in the upper right corner of the stack diagram.

ABS — *Absolute Value* — Function

Level 1	→	Level 1
x	→	$\|x\|$
(*x*,*y*)	→	$\sqrt{x^2+y^2}$
x_unit	→	$\|x\|$_*unit*
[*array*]	→	‖ array ‖
'*symb*'	→	'ABS(*symb*)'

Use: Returns the absolute value of its argument.

Affected by Flags: Numerical Results (−3).

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

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

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

Related Commands: NEG, SIGN

Level 1	→	**Level 1**
	→	

Use: Acknowledges the oldest past-due alarm.

Affected by Flags: Repeat Alarms Not Rescheduled (–43), Acknowledged Alarms Saved (–44).

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

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

Related Commands: ACKALL

ACKALL *Acknowledge All Alarms* **Command**

Level 1	→	**Level 1**
	→	

Use: Acknowledges all past-due alarms.

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

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

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

Related Commands: ACK

Level 1	→	Level 1
z	→	arc cos *z*
'*symb*'	→	'ACOS(*symb*)'

Use: Returns the value of the angle having the given cosine.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18).

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

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

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

'≤1*ACOS(Z)+2*π*n1'

The function ACOS is the inverse of a *part* of COS, a part defined by restricting the domain of COS such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of COS are called the *principal values* of the inverse relation. ACOS in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOS to the boundary of the restricted domain of COS form the *branch cuts* of ACOS.

The principal branch used by the HP 48 for ACOS was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

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

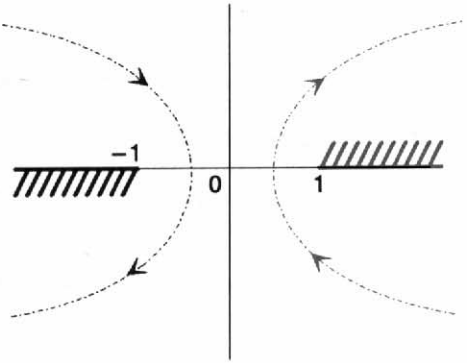
...ACOS

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

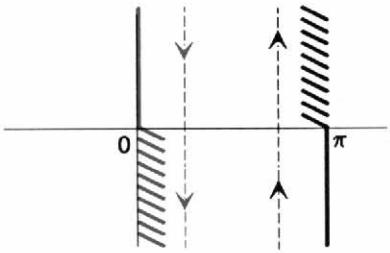
You can view these graphs with domain and range reversed to see how the domain of COS is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain `Z = (x, y)`. COS sends this domain onto the whole complex plane in the range `W = (u, v)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

Related Commands: ASIN, ATAN, COS, ISOL

Domain: Z = (x, y)



Range: W = (u, v) = ACOS(x, y)



Branch Cuts for ACOS (Z)

ACOSH *Inverse Hyperbolic Cosine* **Analytic**

Level 1	→	Level 1
z	→	acosh *z*
'*symb*'	→	'ACOSH(*symb*)'

Use: Returns the inverse hyperbolic cosine of the argument.

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

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

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

'±1*ACOSH(Z)+2*π*i*n1'

The function ACOSH is the inverse of a *part* of COSH, a part defined by restricting the domain of COSH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of COSH are called the *principal values* of the inverse relation. ACOSH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ACOSH to the boundary of the restricted domain of COSH form the *branch cuts* of ACOSH.

The principal branch used by the HP 48 for ACOSH was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued hyperbolic arc cosine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

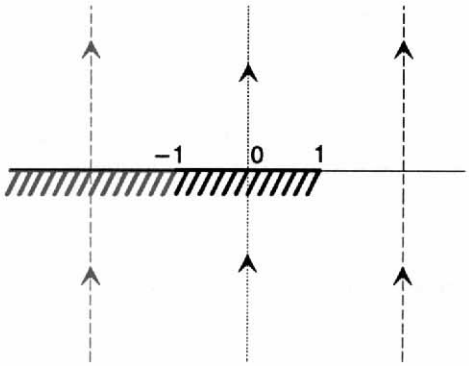
The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

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

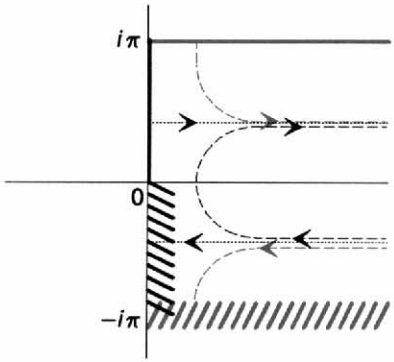
You can view these graphs with domain and range reversed to see how the domain of COSH is restricted to make an inverse *function* possible. Consider the horizontal half-band in the lower graph as the restricted domain Z = (*x*, *y*). COSH sends this domain onto the whole complex plane in the range W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

Related Commands: ASINH, ATANH, COSH, ISOL

Domain: Z = (x,y)



Range: W = (u, v) = ACOSH(x, y)



Branch Cut for ACOSH (Z)

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

Use: Returns the common antilogarithm; that is, 10 raised to the given power.

Affected by Flags: Numerical Results (−3).

Remarks: For complex arguments:

$$10^{(x,y)} = e^{cx} \cos cy + i\, e^{cx} \sin cy$$

where $c = \ln 10$.

Related Commands: EXP, LN, LOG

AND *AND* Function

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* AND *symb*'
'*symb*'	*T/F*	→	'*symb* AND *T/F*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ AND $symb_2$'

Use: Returns the logical AND of two arguments.

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

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

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

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

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

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

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

Related Commands: NOT, OR, XOR

APPLY *Apply Function to Arguments* **Function**

Level 2	Level 1	→	Level 1
{ $symb_1 \ldots symb_n$ }	*'name'*	→	*'name*($symb_1 \ldots symb_n$)'

Use: Creates an expression from the specified function name and arguments.

Affected by Flags: None.

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

The algebraic syntax for APPLY is:

`'APPLY(`*name*,$symb_1$, . . . , $symb_n$ `)'`

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

Example: The following user-defined function *Asin* is a variant of the built-in function ASIN. *Asin* checks for special numerical arguments. If the argument on the stack is symbolic (the second case in the case structure, *Asin* uses APPLY to return the expression `'Asin(`*argument*`)'`.

```
«
 → argument
 «
  CASE
   -3 FS? THEN argument ASIN END
   { 6 7 9 } argument TYPE POS
    THEN 'APPLY(Asin,argument)' EVAL END
   'argument==1' THEN 'π/2' END
   'argument==-1' THEN '-π/2' END
   argument ASIN
  END
 »
»
```

ENTER ↑ Asin STO

Related Commands: QUOTE, | (Where)

ARC *Draw Arc* **Command**

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

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

Affected by Flags: Angle Mode (−17 and −18).

The setting of flags −17 and −18 determine the interpretation of $x_{\theta 1}$ and $x_{\theta 2}$ (degrees, radians, or grads).

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

If $x_{\theta 1} = x_{\theta 2}$, ARC plots one point. If $|x_{\theta 1} - x_{\theta 2}| > 360$ degrees, 2π radians, or 200 grads, ARC draws a complete circle.

Example: In Degrees mode, with the x-axis display range (XRNG) specified as −6.5 to 6.5, the command sequence `(0,0) 1 0 90 ARC` draws an arc counterclockwise from 0 to 90 degrees with a constant radius of 10 pixels.

Related Commands: BOX, LINE, TLINE

Level 1	→	Level 1
:n_{port} :*name*	→	
:IO :*name*	→	

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

Affected by Flags: I/O Device (−33), I/O Messages (−39) *if* the argument is :IO:*name*.

Remarks: The specified port number must be 0, 1, or 2. Ports 1 and 2 must be configured as independent RAM. (See FREE.) An error will result if there is not enough independent RAM in the specified port to copy the HOME directory.

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

If you want to save flag settings, you can do so by executing RCLF and storing the resulting list in a variable.

Related Commands: RESTORE

ARG *Argument* **Function**

Level 1	→	Level 1
(x,y)	→	θ
'symb'	→	'ARG(symb)'

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

Affected by Flags: Angle mode (−17, −18).

Remarks: The polar angle θ is equal to:

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

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

ARRY→ *Array to Stack* **Command**

Level 1	→	Level nm+1 ... Level 2	Level 1
[*vector*]	→	$z_1 \ldots z_n$	{ $n_{element}$ }
[[*matrix*]]	→	$z_{11} \ldots z_{nm}$	{ n_{row} m_{col} }

Use: Takes an array and returns its elements as separate real or complex numbers. Also returns a list of the dimensions of the array.

Affected by Flags: None.

Remarks: The command OBJ→ includes this functionality. ARRY→ is included for compatibility with the HP 28S. ARRY→ is not in a menu.

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

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

Level nm+1 ... Level 2	Level 1	→	Level 1
$z_1 \ldots z_n$	$n_{element}$	→	[*vector*]
$z_{11} \ldots z_{nm}$	$\{ n_{row}\ m_{col} \}$	→	[[*matrix*]]

Use: Returns a vector of n real or complex elements or a matrix of $n \times m$ real or complex elements.

Affected by Flags: None.

Remarks: The elements of the result array should be entered into the stack in row order, with z_{11} (or z_1) in level $nm + 1$ (or $n + 1$), and z_{nm} (or z_n) in level 2. If one or more of the elements is a complex number, the result array will be complex.

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

ASIN — *Arc Sine* — Analytic

Level 1	→	Level 1
z	→	arc sin *z*
'*symb*'	→	'ASIN(*symb*)'

Use: Returns the value of the angle having the given sine.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18).

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

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

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

```
'ASIN(Z)*(-1)^n1+π*n1'
```

The function ASIN is the inverse of a *part* of SIN, a part defined by restricting the domain of SIN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SIN are called the *principal values* of the inverse relation. ASIN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASIN to the boundary of the restricted domain of SIN form the *branch cuts* of ASIN.

The principal branch used by the HP 48 for ASIN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc sine function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

...ASIN

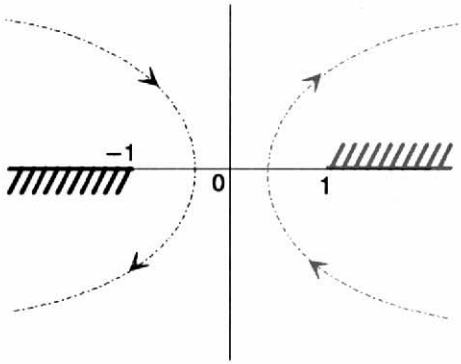
The graphs below show the domain and range of ASIN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

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

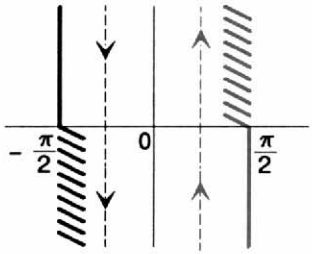
You can view these graphs with domain and range reversed to see how the domain of SIN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain `Z = (x, y)`. SIN sends this domain onto the whole complex plane in the range `W = (u, v)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

Related Commands: ACOS, ATAN, ISOL, SIN

Domain: Z = (x,y)



Range: W = (u,v) = ASIN(x,y)



Branch Cuts for ASIN (Z)

Level 1	→	Level 1
z	→	asinh *z*
'*symb*'	→	'ASINH(*symb*)'

Use: Returns the inverse hyperbolic sine of the argument.

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

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

```
'ASINH(Z)*(-1)^n1+π*i*n1'
```

The function ASINH is the inverse of a *part* of SINH, a part defined by restricting the domain of SINH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of SINH are called the *principal values* of the inverse relation. ASINH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ASINH to the boundary of the restricted domain of SINH form the *branch cuts* of ASINH.

The principal branch used by the HP 48 for ASINH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ASINH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for ASINH from the graph for ASIN (see ASIN) and the relationship asinh $z = -i \text{ asin } iz$.

Related Commands: ACOSH, ATANH, ISOL, SINH

Level 2	Level 1	→	Level 1
obj	x_{key}	→	
'SKEY'	x_{key}	→	

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

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

Remarks: The argument x_{key} is a real number *rc.p* specifying the key by its *r*ow number, *c*olumn number, and its *p*lane (shift). The values for *p* are as follows:

Plane, *p*	Shift
0 or 1	unshifted
2	↰ left-shifted
3	↱ right-shifted
4	α alpha-shifted
5	α ↰ alpha left-shifted
6	α ↱ alpha right-shifted

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

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

...ASN

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

If you find yourself stuck in User mode because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt ("warm start"): press and hold [ON] and the C key simultaneously, releasing the C key first. This cancels User mode.

Example: Executing ASN with `GETI` in level 2 and `85.3` in level 1 assigns GETI to [↱] [▪▪] on the user keyboard. ([↱] [▪▪] has a location of 85.3 because it is eight rows down, five columns across, and right-shifted.) When the calculator is in User mode, pressing [↱] [▪▪] now executes GETI (instead of executing [▪▪]).

Related Commands: DELKEYS, RCLKEYS, STOKEYS

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Shifts a binary integer one bit to the right, except for the most significant bit, which is maintained.

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

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

An arithmetic shift is useful for preserving the sign bit of a binary integer you want to shift. Although the HP 48 makes no special provision for signed binary integers, you can still *interpret* a number as a signed quantity, and in this case an arithmetic shift is meaningful.

Related Commands: SL, SLB, SR, SRB

Level 1	→	Level 1
z	→	arc tan *z*
'*symb*'	→	'ATAN(*symb*)'

Use: Returns the value of the angle having the given tangent.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Angle Mode (−17, −18).

Remarks: For a real argument, the result ranges from −90 to +90 degrees ($-\pi/2$ to $+\pi/2$ radians; −100 to +100 grads).

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

'ATAN(Z)+π*n1'

The function ATAN is the inverse of a *part* of TAN, a part defined by restricting the domain of TAN such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TAN are called the *principal values* of the inverse relation. ATAN in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATAN to the boundary of the restricted domain of TAN form the *branch cuts* of ATAN.

The principal branch used by the HP 48 for ATAN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cuts for the complex-valued arc tangent function occur where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

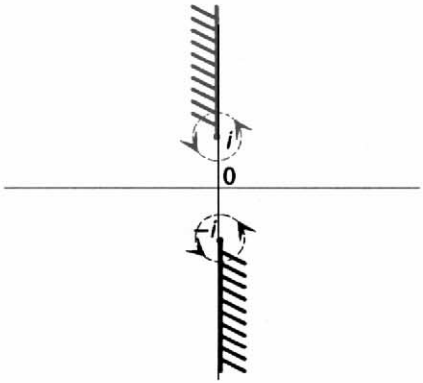
The graphs below show the domain and range of ATAN. The graph of the domain shows where the branch cuts occur: the heavy solid line marks one side of a cut, while the feathered lines mark the other side of a cut. The graph of the range shows where each side of each cut is mapped under the function.

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

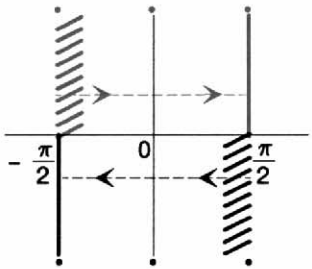
You can view these graphs with domain and range reversed to see how the domain of TAN is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*, *y*). TAN sends this domain onto the whole complex plane in the range W = (*u*,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

Related Commands: ACOS, ASIN, ISOL, TAN

Domain: Z = (x,y)



Range: W = (u,v) = ATAN(x,y)



Branch Cuts for ATAN (Z)

Level 1	→	Level 1
z	→	atanh *z*
'*symb*'	→	'ATANH(*symb*)'

Use: Returns the inverse hyperbolic tangent of the argument.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22).

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

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

```
'ATANH(Z)+π*i*n1'
```

The function ATANH is the inverse of a *part* of TANH, a part defined by restricting the domain of TANH such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of TANH are called the *principal values* of the inverse relation. ATANH in its entirety is called the *principal branch* of the inverse relation, and the points sent by ATANH to the boundary of the restricted domain of TANH form the *branch cuts* of ATANH.

The principal branch used by the HP 48 for ATANH was chosen because it is analytic in the regions where the arguments of the *real-valued* function are defined. The branch cut for the complex-valued ATANH function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

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

Related Commands: ACOSH, ASINH, ISOL, TANH

Level 1	→	Level 1
$n_{library}$	→	
$:n_{port}:n_{library}$	→	

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

Affected by Flags: None.

Remarks: To use a library object, it must be in a port and it must be attached. A library object from an application card (ROM) is automatically in a port (1 or 2), but a library object copied into RAM (such as through the PC Link) must be stored into a port using STO.

Many libraries are attached automatically when you install an application card. Others require you to ATTACH them, as do many libraries copied into RAM. (The owner's manual for the application card or library will tell you which of its library objects must be attached manually. You can also ascertain whether a library is attached to the current directory by executing LIBS.

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

There is no limit on the number of libraries that can be attached to the HOME directory, but only one library at a time can be attached to any other directory. If you attempt to attach a second library to a non-*HOME* directory, the new library will overwrite the old one.

Related Commands: DETACH, LIBS

AUTO *Autoscale* **Command**

Level 1	→	Level 1
	→	

Use: Calculates a *y*-axis display range, or an *x*- and *y*-axis display range.

Affected by Flags: None.

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

Plot Type	Scaling Action
FUNCTION	Samples the equation in *EQ* at 40 values of the independent variable, equally spaced through the *x*-axis plotting range, discards points that return ±∞, then sets the *y*-axis display range to include the maximum, minimum, and origin.
CONIC	Sets the *y*-axis scale equal to the *x*-axis scale.
POLAR	Samples the equation in *EQ* at 40 values of the independent variable, equally spaced through plotting range, discards points that return ±∞, then sets both the *x*- and *y*-axis display ranges in the same manner as for plot type FUNCTION.
PARAMETRIC	Same as POLAR.
TRUTH	No action.
BAR	Sets the *x*-axis display range from 0 to the number of elements in Σ*DAT*, plus 1. Sets the *y*-range to the minimum and maximum of the elements. The *x*-axis is always included.

(continued)

Plot Type	Scaling Action
HISTOGRAM	Sets the *x*-axis display range to the minimum and maximum of the elements in Σ*DAT*. Sets the *y*-axis display range from 0 to the number of rows in Σ*DAT*.
SCATTER	Sets the *x*-axis display range to the minimum and maximum of the independent variable column (XCOL) in Σ*DAT*. Sets the *y*-axis display range to the minimum and maximum of the dependent variable column (YCOL).

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

When executed from a program, AUTO does not draw a plot—execute DRAW to do so. (When executed from the keyboard, AUTO does draw a plot and axes.)

Example: The program `« FUNCTION AUTO DRAW DRAX »` sets the plot type to FUNCTION, autoscales the *y*-axis, plots the equation in *EQ*, and adds axes to the plot.

Related Commands: DRAW, *H, SCALE, SCLΣ, XRNG, YRNG, *W

Level 1	→	Level 1
(x, y)	→	
{ (x, y) }	→	
{ (x, y) "x-axis label" "y-axis label" }	→	
{ "x-axis label" "y-axis label" }	→	

Use: Specifies in the reserved variable *PPAR* the intersection coordinates of the *x*- and *y*-axes, and/or the labels for the *x*- and *y*-axes.

Affected by Flags: None.

Remarks: The argument for AXES (a complex number or list) is stored as the fifth parameter in the reserved variable *PPAR*. If the argument for AXES is a:

- Complex number, it replaces the current entry in *PPAR*.
- List containing a complex number, that number replaces the intersection coordinates without changing any current label specifications.
- List containing a complex number and two strings, that list replaces the current entry in *PPAR*.
- List containing two strings, the strings replace the current label strings, or are added to the entry.

The default value is `(0,0)`.

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

Example: The command sequence

```
{ (0,0) "t" "y" } AXES LABEL
```

specifies an axes intersection at `(0,0)`, and puts the labels `t` and `y` in *PICT*. The labels are positioned to identify the horizontal and vertical axes respectively.

Related Commands: DRAW, DRAX, LABEL

Level 1 → Level 1
→

Use: Sets the plot type to BAR.

Affected by Flags: None.

Remarks: When the plot type is BAR, the DRAW command plots a bar chart using data from one column of the current statistics matrix (reserved variable ΣDAT). The column is specified by the first parameter in the reserved variable ΣPAR (using the XCOL command). The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

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

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

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is `(-6.5,-3.1)`.
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is `(6.5,3.2)`.
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers, with the minimum of the two numbers specifying the horizontal location of the first bar. The default value of *indep* is *X*.
- *res* is a real number specifying the bar width, in user-unit coordinates; or a binary integer specifying the bar width in pixels. The default value is `0`, which specifies a bar width of 1 in user-unit coordinates.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.

- *ptype* is a command name specifying the plot type. Executing the command BAR places the command name BAR in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

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

Related Commands: CONIC, FUNCTION, HISTOGRAM, PARAMETRIC, POLAR, SCATTER, TRUTH

BARPLOT *Draw Bar Plot* **Command**

Level 1	→	Level 1
	→	

Use: Plots a bar chart of the specified column of the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

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

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

Related Commands: FREEZE, GRAPH, HISTPLOT, PVIEW, SCATRPLOT, XCOL

Level 1	→	Level 1
$n_{baud-rate}$	→	

Use: Specifies bit-transfer rate.

Affected by Flags: None.

Remarks: Legal *n*-values are 1200, 2400, 4800, and 9600 (default).

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D of this manual.

Related Commands: CKSM, PARITY, TRANSIO

BEEP *Beep* **Command**

Level 2	Level 1	→	Level 1
$n_{frequency}$	$x_{duration}$	→	

Use: Sounds a tone at n hertz for x seconds.

Affected by Flags: Error Beep (−56).

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

Related Commands: HALT, INPUT, PROMPT, WAIT

Level 1	→	Level 1
	→	

Use: Executes LR with each of the four curve fitting models, and selects the model yielding the largest correlation coefficient (absolute value).

Affected by Flags: None.

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

Related Commands: EXPFIT, LINFIT, LOGFIT, LR, PWRFIT

Level 1 →	Level 1
→	

Use: Selects binary base for binary integer operations. (The default base is decimal.)

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

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

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

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

BINS — *Sort Into Frequency Bins* — Command

Level 3	Level 2	Level 1	→	Level 2	Level 1
x_{min}	x_{width}	n_{bins}	→	$[[\, n_{freq\ bin\ 1} \cdots n_{freq\ bin\ n} \,]]$	$[\, n_{freq\ bin\ L}\ \ n_{freq\ bin\ R} \,]$

Use: Sorts the elements of the independent column (XCOL) of the current statistics matrix (the reserved variable ΣDAT) into $(n_{bins} + 2)$ bins, where the left edge of bin 1 starts at value x_{min} and each bin has width x_{width}.

Affected by Flags: None.

Remarks: BINS returns to level 2 a matrix containing the frequency of occurrences in each bin, and it returns to level 1 a 2-element vector containing the frequency of occurrences outside the defined range of x-values. The level-2 matrix can be stored into the reserved variable ΣDAT if it is desired to plot a histogram of the bin data as a bar plot (for example, by executing BARPLOT).

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

$$n_{freq\ bin\ n} = \mathrm{IP}\left(\frac{x - x_{min}}{x_{width}}\right) + 1$$

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

Example: Suppose the independent column of ΣDAT contains the data:

```
7 2 3 1 4 6 9 0 1 1 3 5 13 2 6 9 5 8 5
```

The command sequence `1 2 5 BINS` returns the matrix `[[ 5 ] [ 3 ] [ 5 ] [ 2 ] [ 2 ]]` to level 2 and the vector `[ 1 1 ]` to level 1.

The data has been sorted into 5 bins of width 2, starting at x-value 1 (and ending at x-value 11). The first element of the matrix indicates that there are 5 x-values (2 1 1 1 2) in bin 1, where bin 1 starts at x-value 1 and ends at x-value 2.99999999999. The vector indicates one occurrence of an x-value less than x_{min} (0), and one occurrence of an x-value greater than x_{max} (13).

Related Commands: BARPLOT, XCOL

BLANK *Blank Graphics Object* **Command**

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

Use: Creates a blank graphics object of the specified width and height.

Affected by Flags: None.

Related Commands: →GROB, LCD→

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

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

Affected by Flags: None.

Related Commands: ARC, LINE, TLINE

BUFLEN *Buffer Length* **Command**

Level 1	→	Level 2	Level 1
	→	*n*	0/1

Use: Returns the number of characters in the HP 48's serial input buffer to level 2 and an indicator of the success of the data reception to level 1.

Affected by Flags: None.

Remarks: The value returned to level 1 is `1` if no framing, overrun, or overflow errors have occurred in data reception; it is `0` if a framing error, a UART overrun error, or an input-buffer overflow has occurred. (The input buffer holds up to 255 bytes.) When a framing or overrun error occurs, data reception ceases until the error is cleared (which BUFLEN does); therefore, *n* represents the data received *before* the error.

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

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

BYTES *Byte Size* **Command**

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

Use: Returns the number of bytes and the checksum for the given object.

Affected by Flags: None.

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

If the argument is a global name, then the size represents the name *and* its contents, while the checksum represents the contents only. To figure the size of the name alone, calculate $(3.5 + 2 \times n)$, where n is the number of characters in the name..

Example: It is possible for objects that decompile identically to have different byte sizes and checksums. For instance,

```
« 1 »
```

and

```
1 'A' STO A « » +
```

both produce lists containing the number 1. However, in the first case the list contains the built-in object 1 (for a size of 7.5 bytes), while in the second case the list contains a RAM copy of 1 (for a size of 15.5 bytes).

Related Commands: MEM

Level 1	→	Level 1
#*n*	→	*n*

Use: Converts a binary integer to its floating-point equivalent.

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

Remarks: If # *n* ≥ # 1000000000000 (base 10), then only the 12 most significant decimal digits are preserved in the mantissa of the result.

Related Commands: R→B

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

Use: Starts CASE...END conditional structure.

Affected by Flags: None.

Remarks: The CASE...END structure lets you execute a series of *cases* (tests). The first test that returns a true result causes execution of the corresponding true-clause, ending the CASE...END structure. Optionally, you can include after the last test a default clause that is executed if all the tests evaluate to false.

The CASE...END structure has the syntax:

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

When CASE is executed, *test-clause*$_1$ is evaluated. If the test is true, *true-clause*$_1$ is executed, and execution skips to END. If *test-clause*$_1$ is false, execution proceeds to *test-clause*$_2$. Execution within the CASE structure continues until a true clause is executed, or until all the test clauses evaluate to false. Optionally, a default clause can be included. In this case, the default clause is executed if all the test clauses evaluate to false.

Related Commands: END, IF, IFERR, THEN

CEIL *Ceiling* **Function**

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

Use: Returns the smallest integer greater than or equal to its argument.

Affected by Flags: Numerical Results (−3).

Examples: `3.2 CEIL` returns `4`.

`-3.2 CEIL` returns `-3`.

Related Commands: FLOOR, IP, RND, TRNC

CENTR *Center* **Command**

Level 1	→	Level 1
(x, y)	→	
x	→	

Use: Adjusts the first two parameters in the reserved variable *PPAR*, (x_{min}, y_{min}) and (x_{max}, y_{max}), so that the point represented by the argument (x, y) is the plot center.

Affected by Flags: None.

Remarks: The center pixel is in row 32, column 65 when *PICT* is its default size (131 × 64).

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

Related Commands: SCALE

Level 1	→	Level 1
$n_{\text{flag number}}$	→	

Use: Clears the specified user or system flag.

Affected by Flags: None.

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

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

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

Use: Returns the percent change from *x* (level 2) to *y* (level 1) as a percentage of *x*.

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*.

If you use simple temperature units, such as `x_°C`, the calculator assumes the values represent temperatures and not differences in temperature. (For *compound* temperature units, such as `x_°C/min`, the calculator assumes temperature units represent temperature differences.) For more information on using temperature units with arithmetic functions, refer to the keyword entry for +.

Examples: Evaluating `1_m 500_cm %CH` returns `400`, because 500 cm represents an increase of 400% over 1 m.

Evaluating `100 100_r %CH` returns `-84.0845056908` (in Standard mode), because 100 radians represents a decrease of about 84% from 100.

Evaluating `100_K 150_K %CH` returns `50`. However, `100_°C 150_°C %CH` returns `13.3994372236`, the equivalent of `373.15_K 423.15_K %CH`.

Related Commands: %, %T

Level 1	→	Level 1
n	→	*"string"*

Use: CHR returns a string representing the HP 48 character corresponding to the character code *n*.

Affected by Flags: None.

Remarks: The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48.

The default character ▪ is supplied for all character codes that are *not* part of the normal HP 48 display character set.

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

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

Level 1	→	Level 1
$n_{checksum}$	→	

Use: Specifies the error-detection scheme.

Affected by Flags: None.

Remarks: Legal *n*-values are:

n-Value	Meaning
1	1-digit arithmetic checksum
2	2-digit arithmetic checksum
3	3-digit cyclic redundancy check. The default value.

IR transmission should use a checksum of 3.

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

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D of this manual.

Related Commands: BAUD, PARITY, TRANSIO

Level n ... Level 1	→	Level 1
$obj_n \ldots obj_1$	→	

Use: Removes all objects from the stack.

Affected by Flags: None.

Remarks: You can recover a CLEARed stack by pressing ◀ [LAST STACK] before executing any other operation. There is no programmable command to recover the stack.

Related Commands: CLVAR, PURGE

Level 1	→	Level 1
x	→	

Use: Adjusts the system time by *x* clock ticks, where 8192 clock ticks equals 1 second.

Affected by Flags: None.

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

Example: `-20480 CLKADJ` decrements the system time by 2.5 seconds.

Related Commands: →TIME

Level 1	→	Level 1
	→	

Use: Clears (blanks) the stack display.

Affected by Flags: None.

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

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

Example: Evaluating `« CLLCD 7 FREEZE »` blanks the display (except the menu labels) and then freezes the entire display.

Related Commands: DISP, FREEZE

CLOSEIO — *Close I/O Port* — Command

Level 1	→	Level 1
	→	

Use: Closes the serial port and the IR port, saving power. Also clears the input buffer and any error messages for KERRM.

Affected by Flags: None.

Remarks: When the HP 48 turns off, it automatically closes the serial and IR ports, but does not clear KERRM. Therefore, CLOSEIO is not needed to close the ports unless the calculator has been set to *not* time out automatically.

Executing HP 48 Kermit protocol commands automatically clears the input buffer; however, executing non-Kermit commands (such as SRECV and XMIT) does not.

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

Related Commands: BUFLEN, OPENIO

Level 1	→	Level 1
	→	

Use: Purges the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Related Commands: RCLΣ, STOΣ, Σ+, Σ−

CLUSR *Clear Variables* **Command**

Level 1	→	Level 1
	→	

Provided for compatibility with the HP 28. CLUSR is the same as CLVAR.
See CLVAR.

Level 1	→	Level 1
	→	

Use: Purges all the variables and empty subdirectories in the current directory.

Affected by Flags: None.

Related Commands: CLUSR, PGDIR, PURGE

Level 1	→	Level 1
[*array*]	→	$x_{\text{column norm}}$

Use: Returns the column norm (one-norm) of its array argument.

Affected by Flags: None.

Remarks: The column norm is the maximum value (over all columns) of the sums of the absolute values of all elements in a column. For a vector, the column norm is the sum of the absolute values of all of the elements.

For complex arrays, the absolute value of a given element (x, y) is $\sqrt{x^2 + y^2}$.

Related Commands: CROSS, DET, DOT, RNRM

Level 1	→	Level 1
'$symb_1$'	→	'$symb_2$'
x	→	*x*
(*x*, *y*)	→	(*x*, *y*)

Use: Simplifies an algebraic expression or equation by "collecting" like terms.

Affected by Flags: None.

Remarks: COLCT operates separately on the two sides of an equation, so that like terms on opposite sides of the equation are not combined.

Examples: `'6+EXP(10)' COLCT` returns `8.71828182846`.

`'5+X+9' COLCT` returns `'14+X'`.

`'X*1_m+X*9_cm' COLCT` returns `'(109_cm)*X'`.

`'X^Z*Y*X^T*Y' COLCT` returns `'X^(T+Z)*Y^2'`.

`'X+3*X+Y+Y' COLCT` returns `'4*X+2*Y'`.

Related Commands: EXPAN, ISOL, QUAD, SHOW

Level 2	Level 1	→	Level 1
x_{xcol}	x_{ycol}	→	

Use: Specifies the independent-variable and dependent-variable columns of the current statistics matrix (the reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: COLΣ combines the functionality of XCOL and YCOL. It is included in the HP 48 for compatibility with the HP 28S. It does not appear in a menu.

The independent-variable column number x_{xcol} is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1. The dependent-variable column number x_{ycol} is stored as the second parameter in Σ*PAR*. The default dependent-variable column number is 2.

COLΣ accepts non-integer real numbers, storing them in Σ*PAR*, but subsequent commands that utilize these two parameters in Σ*PAR* will cause an error.

Example: `2 5 COLΣ` sets column 2 in Σ*DAT* as the independent-variable column, sets column 5 as the dependent-variable column, and stores `2` and `5` as the first and second elements in Σ*PAR*.

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

Level 2	Level 1	→	Level 1
n	*m*	→	$C_{n,m}$
'*symb*$_n$'	*m*	→	'COMB(*symb*$_n$, *m*)'
n	'*symb*$_m$'	→	'COMB(*n*, *symb*$_m$)'
'*symb*$_n$'	'*symb*$_m$'	→	'COMB(*symb*$_n$, *symb*$_m$)'

Use: Returns the number of combinations of *n* items taken *m* at a time.

Affected by Flags: Numerical Results (−3).

Remarks: The calculation formula is:

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

The arguments *n* and *m* must each be less than 10^{12}.

Related Commands: PERM, !

CON — Constant Array — Command

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

Use: Returns a constant array—an array whose elements all have the same value.

Affected by Flags: None

Remarks: The constant value is a real or complex number taken from level 1. The resulting array is either a new array, or an existing array with its elements replaced by the constant, according to the object in level 2.

- **Creating a new array:** If level 2 contains a list of one or two integers, a new array is returned to level 1. If the list contains a single integer $n_{columns}$, a constant vector with n elements is returned to level 1. If the list contains two integers n_{rows} and $m_{columns}$, a constant matrix with n rows and m columns is returned to the stack.
- **Replacing the elements of an existing array:** If level 2 contains an array, an array of the same dimensions is returned, with each element equal to the constant. If the constant is a complex number, the original array must also be complex.

  If level 2 contains a name, the name must identify a variable that contains an array. In this case, the elements of the array are replaced by the constant. If the constant is a complex number, the original array must also be complex.

Examples: `< 2 2 > 6 CON` returns the matrix `[[ 6 6 ][ 6 6 ]]`.

`[ (2,4) (7,9) ] 3 CON` returns the complex vector `[ (3,0) (3,0) ]`.

Related Commands: IDN

CONIC *Conic Plot Type* **Command**

Level 1	→	Level 1
	→	

Use: Sets the plot type to CONIC.

Affected by Flags: None.

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

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

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

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

...CONIC

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

See "Conic Sections" in chapter 19 of the *HP 48 Owner's Manual* for examples using the CONIC plot type.

Related Commands: BAR, FUNCTION, HISTOGRAM, PARAMETRIC, POLAR, SCATTER, TRUTH

CONJ — *Conjugate* — **Analytic**

Level 1	→	Level 1
x	→	*x*
(*x*, *y*)	→	(*x*, −*y*)
[*R-array*]	→	[*R-array*]
[$C\text{-array}_1$]	→	[$C\text{-array}_2$]
'*symb*'	→	'CONJ(*symb*)'

Use: Conjugates a complex number or a complex array.

Affected by Flags: Numerical Results (−3).

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

Example: `[ (3,4) (7,2) ] CONJ` returns `[ (3,-4) (7,-2) ]`.

Related Commands: ABS, IM, RE, SCONJ, SIGN

CONT *Continue Program Execution* **Command**

Level 1 →	Level 1
→	

Use: Resumes execution of a halted program.

Affected by Flags: None.

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

Example: The program

```
« "Enter A, press ‹ CONT ›" ‹ CONT › MENU PROMPT »
```

displays a prompt message, builds a menu with the CONT command assigned to the first menu key, and halts the program for data input. After entering data, pressing CONT resumes program execution. (Note that pressing ↰ CONT is equivalent to pressing CONT.)

Related Commands: HALT, KILL, PROMPT

CONVERT *Convert Units* **Command**

Level 2	Level 1	→	Level 1
x_1 _$units_{source}$	x_2 _$units_{target}$	→	x_3 _$units_{target}$

Use: Converts a source unit object to the dimensions of a target unit.

Affected by Flags: None.

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

Related Commands: UBASE, UFACT, →UNIT, UVAL

Level 1	→	Level 1
	→	$x_{correlation}$

Use: Returns the correlation coefficient of the independent and dependent data columns in the current statistics matrix (reserved variable ΣDAT).

Affected by Flags: None.

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

The correlation is computed from the following formula:

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

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

Related Commands: COV, COLΣ, PREDX, PREDY, XCOL, YCOL

COS — *Cosine* — **Analytic**

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

Use: Returns the cosine of the argument.

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

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

For complex arguments,

$$\cos(x+iy) = \cos x \cosh y - i \sin x \sinh y$$

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

Related Commands: ACOS, SIN, TAN

COSH *Hyperbolic Cosine* **Analytic**

Level 1	→	Level 1
z	→	cosh *z*
'*symb*'	→	'COSH(*symb*)'

Use: Returns the hyperbolic cosine of the argument.

Affected by Flags: Numerical Results (−3).

Remarks: For complex arguments,

$$\cosh(x+iy) = \cosh x \cos y + i \sinh x \sin y$$

Related Commands: ACOSH, SINH, TANH

Level 1	→	Level 1
	→	$x_{covariance}$

Use: Returns the sample covariance of the independent and dependent data columns in the current statistics matrix (reserved variable ΣDAT).

Affected by Flags: None.

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

The covariance is calculated from the following formula:

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

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

Related Commands: CORR, COLΣ, PREDX, PREDY, XCOL, YCOL

Level 1 →	Level 1
→	

Use: Prints the contents, if any, of the printer buffer.

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33).

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Remarks: In the case of the HP 82240B Infrared Printer (flag −34 clear), CR leaves the printhead on the right end of the just printed line.

In the case of printing to the serial port (flag −34 set), CR sends to the printer a string that encodes the line termination method. The default termination method is carriage-return/linefeed. The string is the fourth parameter in the reserved variable *PRTPAR*. See appendix D, "Reserved Variables," for more information about *PRTPAR*.

Related Commands: DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

Level 1	→	Level 1
'global'	→	

Use: Creates an empty subdirectory with the specified name within the current directory.

Affected by Flags: None.

Remarks: Executing CRDIR doesn't change the current directory; you must evaluate the name of the new subdirectory to make it the current directory.

Related Commands: HOME, PATH, PGDIR, UPDIR

CROSS — *Cross Product* — Command

Level 2	Level 1	→	Level 1
[*vector* A]	[*vector* B]	→	[*vector* A × B]

Use: CROSS returns the cross product $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ of the vectors $[a_1\ a_2\ a_3]$ and $[b_1\ b_2\ b_3]$, where:

$$c_1 = a_2b_3 - a_3b_2$$
$$c_2 = a_3b_1 - a_1b_3$$
$$c_3 = a_1b_2 - a_2b_1$$

Affected by Flags: None.

Remarks: The arguments must be two-element or three-element *vectors*, and can be one of each. (The HP 48 automatically converts a two-element argument $[\ d_1\ d_2\ ]$ to a three-element argument $[\ d_1\ d_2\ 0\ ]$.)

Related Commands: CNRM, DET, DOT, RNRM

C→PX *Complex to Pixel* **Command**

Level 1	→	Level 1
(x, y)	→	{ #n #m }

Use: Converts the specified user-unit coordinates to pixel coordinates.

Affected by Flags: None.

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

Related Commands: PX→C

Level 1	→	Level 2	Level 1
(*x*, *y*)	→	*x*	*y*
[*C-array*]	→	[$R\text{-array}_1$]	[$R\text{-array}_2$]

Use: Separates the real and imaginary parts of a complex number or a complex array.

Affected by Flags: None.

Remarks: The result in level 2 represents the real part of the complex argument. The result in level 1 represents the imaginary part of the complex argument.

Related Commands: R→C, RE, IM

Level 1	→	Level 1
	→	*date*

Use: Returns the system date to level 1.

Affected by Flags: Date Format (−42).

Example: If the current date is May 21, 1990, if flag −42 is clear, and if the display mode is Standard, DATE returns `5.21199`. (The trailing zeros are dropped.)

Related Commands: DATE+, DDAYS, TIME, TSTR

→DATE *Set Date* **Command**

Level 1	→	Level 1
date	→	

Use: Sets the system date to *date*.

Affected by Flags: Date Format (−42).

Remarks: *date* has the form *MM.DDYYYY* or *DD.MMYYYY*, depending on the state of flag −42. *MM* is month, *DD* is day, and *YYYY* is year. If *YYYY* is not supplied, the current specification for the year is used. The range of allowable dates is January 1, 1989 to December 31, 2088.

Example: If flag −42 is set and the current system year is 1993, then `16.06 →DATE` sets the system date as June 16, 1993.

Related Commands: →TIME

DATE+ *New Date* **Command**

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

Use: Returns a past or future date, given $date_1$ in level 2 and the number of days x_{days} in level 1.

Affected by Flags: Date Format (−42).

Remarks: If x_{days} is negative, DATE+ calculates a past date.

Related Commands: DATE, DDAYS

Level 2	Level 1	→	Level 1
$date_1$	$date_2$	→	x_{days}

Use: Returns the number of days between $date_1$ and $date_2$.

Affected by Flags: Date Format (−42).

Remarks: If the level 2 date is chronologically later than the level 1 date, the result is negative.

Related Commands: DATE, DATE+

DEC — *Decimal Mode* — Command

Level 1	→	Level 1
	→	

Use: Selects decimal base for binary integer operations. (The default base is decimal.)

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

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

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

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

DECR *Decrement* **Command**

Level 1	→	Level 1
'name'	→	$x_{decrement}$

Use: Decrements by 1 the value of the real number in *name*, storing the new value $x_{decrement}$ back into *name* and returning $x_{decrement}$ to level 1.

Affected by Flags: None.

Remarks: The contents of *name* must be a real number.

Example: If `35.7` is stored in *A*, `'A' DECR` returns `34.7`.

Related Commands: INCR

DEFINE *Define Variable or Function* **Command**

Level 1	→	Level 1
'name=expression'	→	
'name(name$_1$ ... name$_n$)=expression(name$_1$ ... name$_n$)'	→	

Use: Stores the expression on the right side of the = in the variable specified on the left side, or creates a user-defined function.

Affected by Flags: Numerical Results (−3).

For arguments of the form *'name=expression'*, if flag −3 is set, *expression* will be evaluated to a number before it is stored in *name*. (If *expression* contains a formal variable, DEFINE will error if flag −3 is set.)

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

If the left side of the equation is *name* followed by parenthetical arguments *name$_1$* ... *name$_n$*, DEFINE creates a user-defined function and stores it in the variable *name*.

Examples: `'A=2*X' DEFINE` stores `'2*X'` in variable *A*.

`'A(X,Y)=2*X+3/Y' DEFINE` creates a user-defined function *A*. The contents of *A* is the program `« → X Y '2*X+3/Y' »`.

Related Commands: STO

DEG *Degrees* **Command**

Level 1	→	Level 1
	→	

Use: Sets Degrees angle mode.

Affected by Flags: None.

Remarks: DEG clears flags −17 and −18. It clears the RAD and GRAD annunciators.

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

Related Commands: GRAD, RAD

DELALARM *Delete Alarm* **Command**

Level 1	→	Level 1
n_{index}	→	

Use: Deletes the alarm specified by n_{index}.

Affected by Flags: None.

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

Related Commands: FINDALARM, RCLALARM, STOALARM

DELAY — *Delay* — Command

Level 1	→	Level 1
x_{delay}	→	

Use: Specifies how many seconds the HP 48 waits between sending lines of information to the printer.

Affected by Flags: Printing Device (−34) and I/O Device (−33).

Setting flag −34 directs printer output to the serial port. In this case, flag −33 must be clear.

If flag −34 is set and transmit pacing is enabled (non-zero) in reserved variable *IOPAR*, then XON/XOFF handshaking controls data transmission and the delay setting has no effect. (See "The IOPAR Variable" in chapter 33 of the *HP 48 Owner's Manual* for more information about the transmit pacing parameter in *IOPAR*.)

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

The delay setting is the first parameter in the reserved variable *PRTPAR*. See appendix D, "Reserved Variables," for more information about *PRTPAR*.

A shorter delay setting can be useful when the HP 48 sends multiple lines of information to your printer (for example, when printing a program). To optimize printing efficiency, set the delay just longer than the time the printhead requires to print one line of information.

If you set the delay *shorter* than the time to print one line, you may lose information. Also, as the batteries in the printer lose their charge, the printhead slows down, and, if you have previously decreased the delay, you may have to increase it to avoid losing information. (Battery discharge will not cause the printhead to slow to more than the 1.8 second default delay setting.)

Related Commands: CR, OLDPRT, PRLCD, PRST, PRSTC, PRVAR, PR1

DELKEYS *Delete Key Assignments* **Command**

Level 1	→	Level 1
x_{key}	→	
$\{ x_{key1} \cdots x_{key\,n} \}$	→	
0	→	
'S'	→	

Use: Clears the user-defined assignments of the key(s) x_{key}, which is specified as *rc.p*.

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

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

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

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

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

Related Commands: ASN, RCLKEYS, STOKEYS

Level 2	Level 1	→	Level 1
	'global'	→	
	{ *global* }	→	
	{ *global* y_{start} y_{end} }	→	
	{ y_{start} y_{end} }	→	
y_{start}	y_{end}	→	

Use: Specifies the dependent variable and/or its plotting range.

Affected by Flags: None.

Remarks: The specification for the dependent variable name and its plotting range is stored as the seventh parameter in the reserved variable *PPAR*. If the argument to DEPND is a:

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

The default entry is *Y*.

The plotting range for the dependent variable is meaningful only for plot type TRUTH, where it restricts the region for which the equation is tested.

Related Commands: INDEP

Level 1	→	Level 1
	→	*n*

Use: Returns a real number *n* representing the number of objects present on the stack (before DEPTH was executed).

Affected by Flags: None.

DET *Determinant* **Command**

Level 1	→	Level 1
[[*matrix*]]	→	$x_{\text{determinant}}$

Use: Returns the determinant of the argument matrix.

Affected by Flags: None.

Remarks: The argument matrix must be square.

Related Commands: CNRM, CROSS, DOT, RNRM

DETACH *Detach Library* **Command**

Level 1	→	Level 1
$n_{library}$	→	
$:n_{port}:n_{library}$	→	

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

Affected by Flags: None.

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

Related Commands: ATTACH, LIBS, PURGE

Level 2	Level 1	→	Level 1
obj	*n*	→	

Use: Displays *obj* in the *n*th display line.

Affected by Flags: None.

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

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

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

Example: The program

```
« "ENTER Data Now" 1 DISP 7 FREEZE HALT »
```

displays ENTER Data Now at the top of the display, "freezes" the entire display, and halts.

Related Commands: FREEZE, HALT, INPUT, PROMPT

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

Use: Starts DO...UNTIL...END indefinite loop structure.

Affected by Flags: None.

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

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

DO starts execution of the loop clause. UNTIL ends the loop clause and begins the test clause. The test clause must return a test result to the stack. END removes the test result from the stack. If its value is zero, the loop clause is executed again; otherwise, execution resumes following END.

Related Commands: END, UNTIL, WHILE

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

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

Affected by Flags: None.

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

- n_{error} or $\#n_{error}$ display the corresponding built-in error message. See appendix A, "Table of Error and Status Messages," for a complete listing of HP 48 error messages and their numbers.
- *"error"* displays the contents of the string. (A subsequent execution of ERRM returns *"error"*. ERRN returns `# 70000h`.)
- `0` abandons program execution without displaying a message—`0 DOERR` is equivalent to pressing [ATTN].

See "User-Defined Errors" in chapter 30 of the *HP 48 Owner's Manual* for a program example using DOERR.

Related Commands: ERRM, ERRN, ERR0

DOT — *Dot Product* — Command

Level 2	Level 1	→	Level 1
[*array* **A**]	[*array* **B**]	→	*x*

Use: Returns the dot product **A·B** of two arrays **A** and **B**, calculated as the sum of the products of the corresponding elements of the two arrays.

Affected by Flags: None.

Remarks: Both arrays must have the same dimensions.

Some authorities define the dot product of two complex arrays as the sum of the products of the conjugated elements of one array with their corresponding elements from the other array. The HP 48 uses the ordinary products without conjugation. If you prefer the alternate definition, apply CONJ to one or both arrays before using DOT.

Example: `[ 1 2 3 ] [ 4 5 6 ] DOT` returns `32` (by calculating $1\times4 + 2\times5 + 3\times6$).

Related Commands: CNRM, CROSS, DET, RNRM

Level 1	→	Level 1
	→	

Use: Plots the mathematical data in the reserved variable *EQ* or the statistical data in the reserved variable *ΣDAT*, using the specified *x*- and *y*-axis display ranges.

Affected by Flags: −30 (Function Plotting), −31 (Curve Filling).

Remarks: The plot type determines if the data in the reserved variable *EQ* or the data in the reserved variable *ΣDAT* is plotted. DRAW does not erase *PICT* before plotting—execute ERASE to do so. When executed from a program, DRAW does not draw axes—execute DRAX to do so.

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

Related Commands: AUTO, AXES, DRAX, ERASE, FREEZE, GRAPH, LABEL, PVIEW

Level 1	→	**Level 1**
	→	

Use: Draws axes in *PICT*.

Affected by Flags: None.

Remarks: Tick marks are placed on both axes at every tenth pixel from the axes intersection. The coordinates of the axes intersection are specified by AXES. DRAX does not draw axes labels — execute LABEL to do so.

Related Commands: AXES, DRAW, LABEL

DROP *Drop Object* Command

Level 1	→	Level 1
obj	→	

Use: Removes the level 1 object from the stack.

Affected by Flags: None.

Related Commands: CLEAR, DROPN, DROP2

DROPN *Drop n Objects* **Command**

Level n+1 ... Level 2	Level 1	→	Level 1
$obj_1 \ldots obj_n$	*n*	→	

Use: Removes the first *n* + 1 objects from the stack (the first *n* objects excluding the integer *n* itself).

Affected by Flags: None.

Related Commands: CLEAR, DROP, DROP2

DROP2 *Drop 2 Objects* **Command**

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	

Use: Removes the first two objects from the stack.

Affected by Flags: None.

Related Commands: CLEAR, DROP, DROPN

DTAG *Delete Tag* **Command**

Level 1	→	Level 1
:tag:obj	→	*obj*

Use: DTAG removes all tags (labels) from an object.

Affected by Flags: None.

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

DTAG has no effect on an untagged object.

Related Commands: LIST→, →TAG

DUP *Duplicate Object* **Command**

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

Use: DUP returns a copy to level 1 of the object in level 1.

Affected by Flags: None.

Related Commands: DUPN, DUP2, PICK

DUPN *Duplicate n Objects* **Command**

Level n+1..Level 2	Level 1	→	Level 2n..Level n+1	Level n..Level 1
$obj_n \ldots obj_1$	n	→	$obj_n \ldots obj_1$	$obj_n \ldots obj_1$

Use: Takes an integer *n* from level 1 of the stack, and returns copies of the objects in stack levels 2 through *n* + 1.

Affected by Flags: None.

Related Commands: DUP, DUP2, PICK

DUP2 *Duplicate 2 Objects* **Command**

Level 2	Level 1	→	Level 4	Level 3	Level 2	Level 1
obj_1	obj_2	→	obj_1	obj_2	obj_1	obj_2

Use: DUP2 returns copies of the objects in levels 1 and 2 of the stack.

Affected by Flags: None.

Related Commands: DUP, DUPN, PICK

D→R *Degrees to Radians* **Function**

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

Use: Converts a real number representing an angle in degrees to its equivalent in radians.

Affected by Flags: Numerical Results (−3).

Remarks: This function operates independently of the angle mode.

Related Commands R→D

Level 1	→	Level 1
	→	'e'
	→	2.71828182846

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

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

When evaluated, *e* returns its numerical representation if *either* flag −2 *or* flag −3 is *set*; otherwise, *e* returns its symbolic representation.

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

Related Commands: EXP, EXPM, i, LN, LNP1, MAXR, MINR, π

See the IF and IFERR keyword entries for syntax information.

Use: Starts false clause in conditional or error-trapping structure. See the IF and IFERR keyword entries for more information.

Related Commands: IF, IFERR, THEN, END

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

Use: Ends conditional, error-trapping, and indefinite loop structures. See the IF, CASE, IFERR, DO, and WHILE keyword entries for more information.

Related Commands: IF, CASE, IFERR, THEN, ELSE, DO, UNTIL, WHILE, REPEAT

ENG *Engineering* **Command**

Level 1	→	Level 1
n	→	

Use: Sets the number display format to Engineering mode, which displays one to three digits to the left of the radix mark and an exponent that is a multiple of three. The total number of significant digits displayed is $n + 1$.

Affected by Flags: None.

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

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

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

A number with an exponent of −499 is displayed automatically in Scientific mode.

Example: The number 103.6 in Engineering mode with five significant digits ($n = 4$) would appear as `103.60E0`. This same number with one significant digit ($n = 0$) would appear as `100.E0`.

Related Commands: FIX, SCI, STD

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

Use: EQ→ separates an equation into its left and right sides.

Affected by Flags: None.

Remarks: If the argument is an expression, then it is treated as an equation whose right side equals zero.

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

ERASE *Erase PICT* **Command**

Level 1	→	**Level 1**
	→	

Use: Erases *PICT*, leaving a blank *PICT* of the same dimensions.

Affected by Flags: None.

Related Commands: DRAW

ERRM *Error Message* **Command**

Level 1	→	Level 1
	→	*"error"*

Use: Returns a string containing the error message of the most recent calculator error.

Affected by Flags: None.

Remarks: See appendix A, "Table of Error and Status Messages," for a complete listing of HP 48 error messages and their numbers.

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

Example: The program `« IFERR + THEN ERRM END »` returns `"Bad Argument Type"` to level 1 if improper arguments (for example, a complex number and a binary integer) are in levels 1 and 2.

Also see the program example under "The IFERR...THEN...ELSE...END Structure" in chapter 30 of the *HP 48 Owner's Manual*.

Related Commands: DOERR, ERRN, ERR0

Level 1	→	Level 1
	→	$\#n_{error}$

Use: Returns the error number of the most recent calculator error.

Affected by Flags: None.

Remarks: See appendix A, "Table of Error and Status Messages," for a complete listing of HP 48 error messages and their numbers.

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

Example: The program `« IFERR + THEN ERRN END »` returns `# 202h` to level 1 if improper arguments (for, example, a complex number and a binary integer) are in levels 1 and 2.

Related Commands: DOERR, ERRM, ERR0

Level 1	→	Level 1
	→	

Use: Clears the last error number so that a subsequent execution of ERRN returns # 0h. Also clears the last error message.

Affected by Flags: None.

Related Commands: DOERR, ERRM, ERRN

Level 1	→	Level 1
obj	→	*It depends!*

Use: Evaluates the object. The effect of evaluation depends on the object type:

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

(continued)

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

Affected by Flags: Numerical Results (−3).

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

Related Commands: →NUM, SYSEVAL

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

Use: Returns the exponential, or natural antilogarithm; that is, *e* raised to the given power.

Affected by Flags: Numerical Results (−3).

Remarks: EXP uses a special algorithm to compute a more accurate result for the exponential than can be obtained by using e^.

For complex arguments,

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

Related Commands: ALOG, EXPM, LN, LOG

Level 1	→	Level 1
'$symb_1$'	→	'$symb_2$'
x	→	*x*
(*x*, *y*)	→	(*x*, *y*)

Use: Rewrites an algebraic expression or equation by expanding products and powers.

Affected by Flags: None.

Examples: `'A*(B+C)' EXPAN` returns `'A*B+A*C'`.

`'A^(B+C)' EXPAN` returns `'A^B*A^C'`.

`'X^5' EXPAN` returns `'X*X^4'`.

`'(X+Y)^2' EXPAN` returns `'X^2+2*X*Y+Y^2'`.

Related Commands: COLCT, ISOL, QUAD, SHOW

Level 1	→	Level 1
	→	

Use: Stores its name as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the exponential curve fitting model.

Affected by Flags: None.

Remarks: LINFIT is the default specification in ΣPAR.

Related Commands: LR, LINFIT, LOGFIT, PWRFIT, BESTFIT

EXPM *Exponential Minus 1* **Analytic**

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

Use: Returns $e^x - 1$.

Affected by Flags: Numerical Results (−3).

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

Related Commands: EXP, LNP1

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

Provided for compatibility with the HP 28. FACT is the same as !. See !.

FC? *Flag Clear?* **Command**

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

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

Affected by Flags: None.

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

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

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

Affected by Flags: None.

Example: If flag −44 is set, `-44 FC?C` returns 0 to level 1 and clears flag −44.

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

FINDALARM *Find Alarm* **Command**

Level 1	→	Level 1
date	→	n_{index}
{ *date time* }	→	n_{index}
0	→	n_{index}

Use: Returns the alarm index n_{index} of the first alarm due after the specified time.

Affected by Flags: None.

Remarks: If the level 1 argument is a real number *date*, FINDALARM returns the index of the first alarm due after 12:00 AM on that date. If the argument is a list { *date time* }, it returns the index of the first alarm due after that date and time. If the argument is the real number 0, FINDALARM returns the first *past-due* alarm.

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

Related Commands: DELALARM, RCLALARM, STOALARM

Level 1	→	Level 1
	→	

Use: Terminates Kermit Server mode in a device connected to an HP 48.

Affected by Flags: I/O Device flag (−33), I/O Messages (−39).

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

Related Commands: KGET, RECN, RECV, SEND, SERVER

Level 1	→	Level 1
n	→	

Use: Sets the number display format to Fix mode, which rounds the display to *n* decimal places.

Affected by Flags: None.

Remarks: Fix mode shows *n* digits to the right of the radix mark, where $0 \leq n \leq 11$. (Values for *n* outside this range are rounded up or down.) A number is displayed or printed as:

(sign) mantissa

where the mantissa can be of any form. However, the calculator automatically displays a number in Scientific mode if:

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

Example: The number 103.6 in Fix mode to four decimal places would appear as `103.6000`.

Related Commands: FIX, SCI, STD

FLOOR *Floor* Function

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

Use: Returns the greatest integer less than or equal to its argument.

Affected by Flags: Numerical Results (−3).

Examples: `3.2 FLOOR` returns 3.

`-3.2 FLOOR` returns −4.

Related Commands: CEIL, IP, RND, TRNC

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

Use: Starts FOR...NEXT and FOR...STEP definite loop structures.

Affected by Flags: None.

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

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

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

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

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

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

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

  FOR takes x_{start} and x_{finish} from the stack as the beginning and ending values for the loop counter, then creates the local variable *counter* as a loop counter. Next, the loop clause is executed; *counter* can can be referenced or have its value changed within the loop clause. STEP takes $x_{increment}$ from the stack and increments *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 the increment is positive, the loop is executed again when *counter* is less than or equal to x_{finish}. If the increment is negative, the loop is executed when *counter* is greater than or equal to x_{finish}.

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

Related Commands: NEXT, START, STEP

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

Use: Returns the fractional part of its argument.

Affected by Flags: Numerical Results (−3).

Remarks: The result has the same sign as the argument.

Examples: `-32.3 FP` returns `-.3`.

`32.3_m FP` returns `.3_m`.

Related Commands: IP

Level 2	Level 1	→	Level 1
{ }	n_{port}	→	
{$name_{backup}$ ... $n_{library}$ }	n_{port}	→	
$name_{backup}$	n_{port}	→	
$n_{library}$	n_{port}	→	

Use: Frees (makes *independent*) the previously merged RAM in the specified port (1 or 2). Any prior contents of the port are moved into user memory. If you specify any backup or library objects in level 2, then these objects are moved from port 0 to the newly freed RAM port.

Affected by Flags: None.

Remarks: The list in level 2 can be empty (in which case no objects are moved to the newly independent RAM) or it can contain any number of backup names and library numbers. Level 2 cannot be completely empty, however.

Related Commands: MERGE

FREEZE *Freeze Display* **Command**

Level 1	→	Level 1
$n_{\text{display area}}$	→	

Use: Freezes the part of the display specified by $n_{\text{display area}}$, so that it is not updated until a key press.

Affected by Flags: None.

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

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

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

For example, `2 FREEZE` freezes the stack/command-line area, `3 FREEZE` freezes the status area and the stack/command-line area, and `7 FREEZE` freezes all three areas.

Values of $n_{\text{display area}} \geq 7$ or ≤ 0 freeze the entire display (are equivalent to value 7).

Examples: The program

```
« "Ready for data" 1 DISP 1 FREEZE HALT »
```

displays the contents of the string in the top line of the display, then freezes the status area so that the string contents persist in the display after HALT is executed.

The program:

```
« ( # 0d # 0d ) PVIEW 7 FREEZE »
```

selects the graphics display and then freezes the entire display so that the graphics display persists after the program ends. (If FREEZE was not executed, the stack display would be selected after the program ends.)

Related Commands: CLLCD, DISP, HALT

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

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

Affected by Flags: None.

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

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

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

Affected by Flags: None.

Example: If flag −44 is set, `-44 FS?C` returns 1 to level 1 and clears flag −44.

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

FUNCTION *Function Plot Type* **Command**

Level 1 →	Level 1
→	

Use: Sets the plot type to FUNCTION.

Affected by Flags: None.

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

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

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

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

- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

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

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

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

If flag −30 is *set*, all equations are plotted as two separate expressions.

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

. . . FUNCTION

See "Function Plots" in chapter 19 and numerous examples in chapter 18 of the *HP 48 Owner's Manual* for uses of the FUNCTION plot type.

Related Commands: BAR, CONIC, HISTOGRAM, PARAMETRIC, POLAR, SCATTER, TRUTH

Level 2	Level 1	→	Level 1
[[*matrix*]]	$n_{position}$	→	z_{get}
[[*matrix*]]	{ n_{row} m_{col} }	→	z_{get}
'$name_{matrix}$'	$n_{position}$	→	z_{get}
'$name_{matrix}$'	{ n_{row} m_{col} }	→	z_{get}
[*vector*]	$n_{position}$	→	z_{get}
[*vector*]	{ $n_{position}$ }	→	z_{get}
'$name_{vector}$'	$n_{position}$	→	z_{get}
'$name_{vector}$'	{ $n_{position}$ }	→	z_{get}
{ *list* }	$n_{position}$	→	obj_{get}
{ *list* }	{ $n_{position}$ }	→	obj_{get}
'$name_{list}$'	$n_{position}$	→	obj_{get}
'$name_{list}$'	{ $n_{position}$ }	→	obj_{get}

Use: Returns from the level 2 array or list the real or complex number z_{get}, or, from a list, the object obj_{get}, whose position is specified in level 1.

Affected by Flags: None.

Remarks: For matrices, $n_{position}$ counts in row order.

Examples: `[[ 2 3 7 ][ 3 2 9 ][ 2 1 3 ]] < 2 3 > GET` returns 9.

`[[ 2 3 7 ][ 3 2 9 ][ 2 1 3 ]] 8 GET` returns 1.

`< A B C D E > < 1 > GET` returns `'A'`.

See also "LMED (Median of a List)" in chapter 31 of the *HP 48 Owner's Manual* for a program example using GET.

Related Commands: GETI, PUT, PUTI

Level 2	Level 1	→	Level 3	Level 2	Level 1
[[*matrix*]]	$n_{position1}$	→	[[*matrix*]]	$n_{position2}$	z_{get}
[[*matrix*]]	$\{ n_{row}\ m_{col} \}_1$	→	[[*matrix*]]	$\{ n_{row}\ m_{col} \}_2$	z_{get}
'$name_{matrix}$'	$n_{position1}$	→	'$name_{matrix}$'	$n_{position2}$	z_{get}
'$name_{matrix}$'	$\{ n_{row}\ m_{col} \}_1$	→	'$name$'$_{matrix}$	$\{ n_{row}\ m_{col} \}_2$	z_{get}
[*vector*]	$n_{position1}$	→	[*vector*]	$n_{position2}$	z_{get}
[*vector*]	$\{ n_{position1} \}$	→	[*vector*]	$\{ n_{position2} \}$	z_{get}
'$name_{vector}$'	$n_{position1}$	→	'$name_{vector}$'	$n_{position2}$	z_{get}
'$name_{vector}$'	$\{ n_{position1} \}$	→	'$name_{vector}$'	$\{ n_{position2} \}$	z_{get}
{ *list* }	$n_{position1}$	→	{ *list* }	$n_{position2}$	obj_{get}
{ *list* }	$\{ n_{position1} \}$	→	{ *list* }	$\{ n_{position2} \}$	obj_{get}
'$name_{list}$'	$n_{position1}$	→	'$name_{list}$'	$n_{position2}$	obj_{get}
'$name_{list}$'	$\{ n_{position1} \}$	→	'$name_{list}$'	$\{ n_{position2} \}$	obj_{get}

Use: From the level 2 array or list, returns the real or complex number z_{get} or, from a list, the object obj_{get}, whose position is specified in level 1. Also returns the array or list, and returns the next position in that array or list.

Affected by Flags: Index Wrap Indicator (−64).

The Index Wrap Indicator flag is cleared on each execution of GETI *until* the position (index) wraps to the first position in the array or list, at which point the flag is set. The next execution of GETI again clears the flag.

Remarks: For matrices, the position is incremented in *row* order.

See "SORT (Sort a List)" and "MNX (Finding the Minimum or Maximum Element of an Array—Technique 1)" in chapter 31 of the *HP 48 Owner's Manual* for program examples using GETI.

Related Commands: GET, PUT, PUTI

GOR — *Graphics OR* — Command

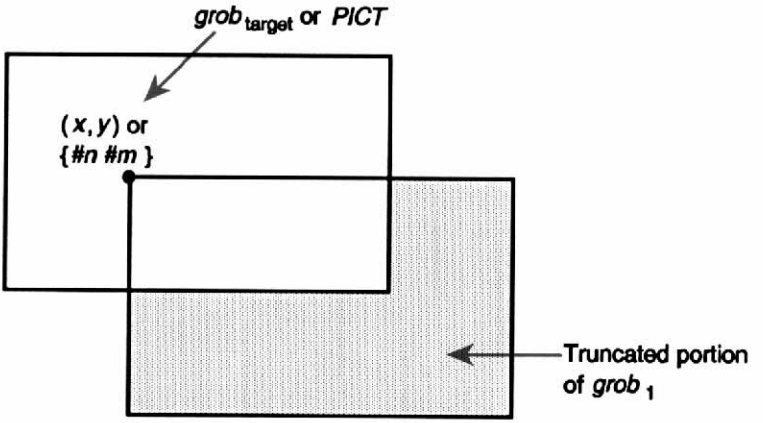
Level 3	Level 2	Level 1	→	Level 1
$grob_{target}$	{ #n #m }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Use: Superimposes $grob_1$ onto $grob_{target}$, or onto *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*. GOR uses a logical OR to determine the state (on or off) of each pixel in the overlapping portion of the argument graphics objects.

Affected by Flags: None.

Remarks: If the level 3 argument (the target graphics object) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

If $grob_1$ extends past $grob_{target}$ or *PICT* in either direction, it is truncated in that direction.



Related Commands: GXOR, REPL, SUB

GRAD *Grads Mode* **Command**

Level 1	→	Level 1
	→	

Use: Sets Grads angle mode.

Affected by Flags: None.

Remarks: GRAD clears flag −17 and sets flag −18. It displays the GRAD annunciator.

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

Related Commands: DEG, RAD

Level 1	→	Level 1
	→	

Use: Selects the Graphics environment (selects the graphics display and activates the graphics cursor and Graphics menu).

Affected by Flags: None.

Remarks: When executed from a program, GRAPH suspends program execution until [ATTN] is pressed.

Example: The program

```
« "Press ATTN to return■to stack" 1 DISP
                3 WAIT GRAPH »
```

displays an instructive message for three seconds and then selects the Graphics environment. (The ■ character in the program indicates a linefeed.)

Related Commands: PVIEW, TEXT

Level 2	Level 1	→	Level 1
obj	$n_{\text{char size}}$	→	*grob*

Use: Creates a graphics object representing the level 2 object, where the argument $n_{\text{char size}}$ specifies the character size of the representation.

Affected by Flags: None.

Remarks: $n_{\text{char size}}$ can be 0, 1 (small), 2 (medium), or 3 (large). $n_{\text{char size}}$ = 0 is the same as $n_{\text{char size}}$ = 3, except for unit objects and algebraic objects, where 0 specifies the EquationWriter application picture.

Example: The program

```
« 'Y=3*X^2' 0 →GROB PICT STO { } PVIEW »
```

returns a graphics object to the stack representing the EquationWriter application picture of `'Y=3*X^2'`, then stores the graphics object in *PICT* and shows it in the graphics display with scrolling activated.

Related Commands: →LCD, LCD→

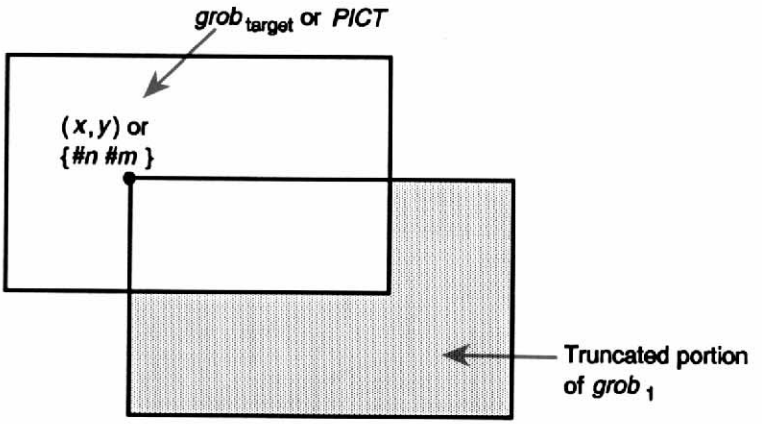
Level 3	Level 2	Level 1	→	Level 1
$grob_{target}$	{ #n #m }	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{ #n #m }	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Use: Superimposes $grob_1$ onto $grob_{target}$, or onto *PICT*, with the upper left corner pixel of $grob_1$ positioned at the specified coordinate in $grob_{target}$ or *PICT*. GXOR uses a logical exclusive OR to determine the state of the pixels (on or off) in the overlapping portion of the argument graphics objects.

Affected by Flags: None.

Remarks: GXOR is used for creating cursors, for example, where it is desirable to make the cursor image appear dark on a light background, and light on a dark background. Executing GXOR again with the same image restores the original picture.

If $grob_1$ extends past $grob_{target}$ or *PICT* in either direction, it is truncated in that direction.



If the level 3 argument (the target graphics object) is any graphics object other than *PICT*, then $grob_{result}$ is returned to the stack. If the level 3 argument is *PICT*, no result is returned to the stack.

Example: The program

```
« ERASE PICT NEG PICT { # 0d #0d }
GROB 5 × 5 11A040A011 GXOR LASTARG GXOR »
```

turns on (makes dark) every pixel in *PICT*, then superimposes a 5 × 5 graphics object on *PICT* at pixel coordinates { # 0d #0d }. Each on-pixel in the 5 by 5 graphics object turns off (makes light) the corresponding pixel in *PICT*. Then, the original picture is restored by executing GXOR again with the same arguments.

Related Commands: GOR, REPL, SUB

Level 1	→	Level 1
x_{factor}	→	

Use: Multiplies the vertical scale by x_{factor}.

Affected by Flags: None.

Remarks: Executing *H changes the *y*-axis display range — the y_{min} and y_{max} components of the first two complex numbers in the reserved variable *PPAR*. The plot center (the user-unit coordinate of the center pixel) is not changed.

Related Commands: AUTO, *W, YRNG

Level 1	→	**Level 1**
	→	

Use: Halts program execution.

Affected by Flags: None.

Remarks: Program execution is halted at the location of the HALT command in the program. The HALT annunciator is turned on. Program execution is resumed by executing CONT (usually by pressing ↰ CONT). Executing KILL (usually by pressing PRG CTRL KILL) cancels all halted programs.

Related Commands: CONT, KILL, OFF

Level 1	→	Level 1
	→	

Use: Selects hexadecimal base for binary integer operations. (The default base is decimal.)

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

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

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

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

Level 1 →	Level 1
→	

Use: Sets the plot type to HISTOGRAM.

Affected by Flags: None.

Remarks: When the plot type is HISTOGRAM, the DRAW command creates a histogram using data from one column of the current statistics matrix (reserved variable ΣDAT). The column is specified by the first parameter in the reserved variable ΣPAR (using the XCOL command). The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

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

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

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is `(-6.5,-3.1)`.
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is `(6.5,3.2)`.
- *indep* is either a name specifying a label for the horizontal axis, or a list containing such a name and two numbers that specify the minimum and maximum values of the data to be plotted. The default value of *indep* is *X*.
- *res* is a real number specifying the bin size, in user-unit coordinates; or a binary integer specifying the bin size in pixels. The default value is `0`, which specifies the bin size to be 1/13 of the difference between the specified minimum and maximum values of the data.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.

- *ptype* is a command name specifying the plot type. Executing the command HISTOGRAM places the command name HISTOGRAM in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The frequency of the data is plotted as bars, where each bar represents a collection of data points. The base of each bar spans the values of the data points, and the height indicates the number of data points. The width of each bar is specified by *res*. The overall maximum and minimum values for the data can be specified by *indep*; otherwise, the values in (x_{min}, y_{min}) and (x_{max}, y_{max}) are used.

Related Commands: BAR, CONIC, FUNCTION, PARAMETRIC, POLAR, SCATTER, TRUTH

Level 1	→	Level 1
	→	

Use: Plots a frequency histogram of the specified column in the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The data column to be plotted is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. If no data column is specified, column 1 is selected by default. The *y*-axis is autoscaled and the plot type is set to HISTOGRAM.

HISTPLOT plots *relative* frequencies, using 13 bins as the default number of partitions. The RES command lets you specify a different number of bins by specifying the bin width. To plot a frequency histogram with *numerical* frequencies, execute BINS and then BARPLOT.

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

Related Commands: BARPLOT, BINS, GRAPH, FREEZE, PVIEW, RES, SCATRPLOT, XCOL

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	HMS_1+HMS_2

Use: Returns the sum of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

Affected by Flags: None.

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS−

Level 2	Level 1	→	Level 1
HMS_1	HMS_2	→	$HMS_1 - HMS_2$

Use: Returns the difference of two real numbers, where the arguments and the result are interpreted in hours-minutes-seconds format.

Affected by Flags: None.

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, →HMS, HMS+

Level 1	→	Level 1
HMS	→	*x*

Use: Converts a real number in hours-minutes-seconds format to its decimal form (hours or degrees with a decimal fraction).

Affected by Flags: None.

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the display current mode) representing the decimal fractional part of seconds.

Related Commands: →HMS, HMS+, HMS−

Level 1	→	Level 1
x	→	*HMS*

Use: Converts a real number representing hours or degrees with a decimal fraction to hours-minutes-seconds format.

Affected by Flags: None.

Remarks: The format for HMS (a time or an angle) is *H.MMSSs*, where:

- *H* is zero or more digits representing the integer part of the number.
- *MM* are two digits representing the number of minutes.
- *SS* are two digits representing the number of seconds.
- *s* is zero or more digits (as many as allowed by the current display mode) representing the decimal fractional part of seconds.

Related Commands: HMS→, HMS+, HMS−

Level 1	→	**Level 1**
	→	

Use: Makes the *HOME* directory the current directory.

Affected by Flags: None.

Related Commands: CRDIR, PATH, PGDIR, UPDIR

Level 1	→	Level 1
	→	'i'
	→	(0,1)

Use: Returns the symbolic constant *i* or its numerical representation, (0, 1).

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

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

Related Commands: e, MAXR, MINR, π

Level 1	→	Level 1
n	→	[[*R-matrix*$_{\text{Identity}}$]]
[[*matrix*]]	→	[[*matrix*$_{\text{Identity}}$]]
'name'	→	

Use: Returns an identity matrix; that is, a square matrix with its diagonal elements equal to 1 and its off-diagonal elements equal to 0.

Affected by Flags: None.

Remarks: The result is either a new square matrix, or it is an existing square matrix with its elements replaced by the elements of the identity matrix, according to the argument in level 1.

- **Creating a new matrix:** If the argument is a real number *n*, a new real identity matrix is returned to level 1, with its number of rows and number of columns equal to *n*.
- **Replacing the elements of an existing matrix:** If the argument is a square matrix, an identity matrix of the same dimensions is returned. If the original matrix is complex, the result identity matrix will also be complex, with diagonal values `(1,0)`.

  If the argument is a name, the name must identify a variable containing a square matrix. In this case, the elements of the matrix are replaced by those of the identity matrix (complex if the original matrix is complex).

Related Commands: CON

	Level 1	→	Level 1
IF		→	
THEN	*T/F*	→	
END		→	
IF		→	
THEN	*T/F*	→	
ELSE		→	
END		→	

Use: Starts IF...THEN...END and IF...THEN...ELSE...END conditional structures.

Affected by Flags: None.

Remarks: *Conditional structures*, used in combination with program tests, enable a program to make decisions.

- IF...THEN...END executes a sequence of commands only if a test returns a non-zero (true) result. The syntax is:

  IF *test-clause* THEN *true-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is non-zero, the true clause is executed. Otherwise, program execution resumes following END.

  The test clause can be a command sequence (for example, `A B ≤`) or an algebraic (for example, `'A≤B'`). If the test clause is an algebraic, it is *automatically evaluated* to a number (→NUM or EVAL isn't necessary).

- IF...THEN...ELSE...END executes one sequence of commands if a test returns a true (non-zero) result, or another sequence of commands if that test returns a false (zero) result. The syntax is:

  IF *test-clause* THEN *true-clause* ELSE *false-clause* END

  IF begins the test clause, which must return a test result to the stack. THEN removes the test result from the stack. If the value is non-zero, the true clause is executed. Otherwise, the false clause is executed. After the appropriate clause is executed, execution resumes following END.

  If the test clause is an algebraic, it is automatically evaluated to a number (→NUM or EVAL isn't necessary).

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

Level 1	→	Level 1
IFERR	→	
THEN	→	
END	→	
IFERR	→	
THEN	→	
ELSE	→	
END	→	

Use: Starts IFERR...THEN...END and IFERR...THEN...ELSE...END error trapping structures.

Affected by Flags: Last Arguments (–55).

Remarks: *Error trapping* structures enable program execution to continue after a "trapped" error occurs.

- IFERR...THEN...END executes a sequence of commands if an error occurs. The syntax of IFERR...THEN...END is:

  IFERR *trap-clause* THEN *error-clause* END

  If an error occurs during execution of the trap clause:

  1. The error is ignored.
  2. The remainder of the trap clause is discarded.
  3. The key buffer is cleared.
  4. If any or all of the display is "frozen" (by FREEZE), that state is cancelled.
  5. If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack.
  6. Program execution jumps to the error clause.

  The commands in the error clause are executed only if an error is generated during execution of the trap clause.

- IFERR...THEN...ELSE...END executes one sequence of commands if an error occurs or another sequence of commands if an error does not occur. The syntax of IFERR...THEN...ELSE...END is:

  IFERR *trap-clause* THEN *error-clause* ELSE *normal-clause* END

  If an error occurs during execution of the trap clause:

  1. The error is ignored.
  2. The remainder of the trap clause is discarded.
  3. The key buffer is cleared.
  4. If any or all of the display is "frozen" (by FREEZE), that state is cancelled.
  5. If Last Arguments is enabled, the arguments to the command that caused the error are returned to the stack.
  6. Program execution jumps to the error clause.

  If no error occurs, execution jumps to the normal clause at the completion of the trap clause.

See "The IFERR...THEN...ELSE...END... Structure" in chapter 30 and "BDISP (Binary Display)" in chapter 31 of the *HP 48 Owner's Manual* for program examples that use error trapping structures.

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

Level 2	Level 1	→	Level 1
T/F	*obj*	→	

Use: Executes *obj* if *T/F* is non-zero. Discards *obj* if *T/F* is zero.

Affected by Flags: None.

Remarks: IFT lets you execute in stack syntax the decision-making process of the IF...THEN...END conditional structure. The "true clause" is *obj* in level 1.

Example: `X 0 > "Positive" IFT` leaves `"Positive"` in level 1 if *X* contains a positive real number.

Related Commands: IFTE

IFTE *If-Then-Else* **Function**

Level 3	Level 2	Level 1	→	Level 1
T/F	obj_{true}	obj_{false}	→	

Use: Executes obj_{true} if *T/F* is non-zero, discarding obj_{false}. Executes obj_{false} if *T/F* is zero, discarding obj_{true}.

Affected by Flags: None.

Remarks: IFTE lets you execute in stack syntax the decision-making process of the IF...THEN...ELSE...END conditional structure. The "true clause" is obj_{true} in level 2. The "false clause" is obj_{false} in level 1.

IFTE is also allowed in algebraic expressions, with the following syntax:

'IFTE(*test,true-clause,false-clause*)'

When an algebraic containing IFTE is evaluated, its first argument *test* is evaluated to a test result. If it returns a non-zero real number, *true-clause* is evaluated. If it returns zero, *false-clause* is evaluated.

Examples: The command sequence `X 0 ≥ "Positive" "Negative" IFTE` leaves `"Positive"` on the stack if *X* contains a non-negative real number, or `"Negative"` if *X* contains a negative real number.

The algebraic `'IFTE(X≠0,SIN(X)/X,1)'` returns the value of $\sin(x)/x$, even for $x = 0$, which would normally cause an `Infinite Result` error.

Related Commands: IFT

Level 1	→	Level 1
x	→	0
(*x*, *y*)	→	*y*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'IM(*symb*)'

Use: Returns the imaginary part of its (complex) argument.

Affected by Flags: Numerical Results (−3).

Remarks: If the argument is an array, IM returns a real array, the elements of which are equal to the imaginary parts of the corresponding elements of the argument array. If the argument array is real, all of the elements of the result array are zero.

Related Commands: C→R, RE, R→C

Level 1	→	Level 1
'name'	→	$x_{increment}$

Use: Increments by 1 the value of the real number in *name*, storing the new value $x_{increment}$ back into *name* and returning $x_{increment}$ to level 1.

Affected by Flags: None.

Remarks: The value in *name* must be a real number.

Example: If `35.7` is stored in *A*, `'A' INCR` returns `36.7`.

Related Commands: DECR

INDEP *Independent Variable* **Command**

Level 2	Level 1	→	Level 1
	'global'	→	
	{ *global* }	→	
	{ *global* x_{start} x_{end} }	→	
	{ x_{start} x_{end} }	→	
x_{start}	x_{end}	→	

Use: Specifies the independent variable and/or its plotting range.

Affected by Flags: None.

Remarks: The specification for the independent variable name and its plotting range is stored as the third parameter in the reserved variable *PPAR*. If the argument to INDEP is a:

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

The default entry is *X*.

Related Commands: DEPND

INPUT *Input* **Command**

Level 2	Level 1	→	Level 1
"stack prompt"	*"command-line prompt"*	→	*"result"*
"stack prompt"	{ $list_{command-line}$ }	→	*"result"*

Use: Prompts for data input to the command line and prevents the user access to stack operations.

Affected by Flags: None.

Remarks: When INPUT is executed, the stack area is blanked and program execution is suspended for data input to the command line. The contents of *"stack prompt"* are displayed at the top of the stack area. Depending on the level 1 argument, the command line may also contain the contents of a string, or it may be empty. Pressing ENTER resumes program execution and returns the contents of the command line in string form to level 1.

In its general form, the level 1 argument for INPUT is a list that specifies the content and interpretation of the command line. The list can contain *one or more* of the following parameters, *in any order*:

- *"command-line prompt"*, whose contents are placed in the command line for prompting when the program pauses.
- Either a *real number*, or a *list containing two real numbers*, that specifies the initial cursor position in the command line:
  - A real number *n* at the *n*th character from the left end of the first row (line) of the command line. A *positive n* specifies the insert cursor; a *negative n* specifies the replace cursor. 0 specifies the end of the command-line string.
  - A list that specifies the initial row and column position of the cursor: the first number in the list specifies a row in the command line (1 specifies the first row of the command line); the second number counts by characters from the left end of the specified line. 0 specifies the end of the command-line string in the specified row. A positive row number specifies the insert cursor; a negative row number specifies the replace cursor.

- One or more of the parameters ALG, α, or V, entered as unquoted names:
  - ALG activates Algebraic/Program-entry mode.
  - α ([α] [→][A]) specifies alpha lock.
  - V verifies if the characters in the result string "result", without the " delimiters, compose a valid object or objects. If the result-string characters do not compose a valid object or objects, INPUT displays the Invalid Syntax warning and prompts again for data.

You can choose to specify as few as one of the level-1 list parameters. The default states for these parameters are:

- Blank command line.
- Insert cursor placed at the end of the command-line prompt string.
- Program-entry mode.
- Result string not checked for invalid syntax.

If you specify *only* a command-line prompt string for the level 1 argument, you do not need to put it in a list.

Examples: The *HP 48 Owner's Manual* contains programming examples illustrating the use of INPUT. See "The INPUT Command" in chapter 29.

Related Commands: PROMPT, STR→

Level 1	→	Level 1
z	→	*1/z*
[[*matrix*]]	→	[[$matrix^{-1}$]]
'*symb*'	→	'INV(*symb*)'
x_unit	→	*1/x_1/unit*

Use: Returns the reciprocal or the matrix inverse.

Affected by Flags: Numerical Results (−3).

Remarks: For a *complex* argument (x, y), the inverse is the complex number

$$(x/(x^2 + y^2), -y/(x^2 + y^2))$$

Matrix arguments must be square (real or complex).

Related Commands: /, SINV

Level 1	→	Level 1
x	→	*n*
x_unit	→	*n_unit*
'*symb*'	→	'IP(*symb*)'

Use: Returns the integer part of its argument.

Affected by Flags: Numerical Results (−3).

Remarks: The result has the same sign as the argument.

Example: `32.3_m IP` returns `32_m`.

Related Commands: FP

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

Use: Returns an algebraic '$symb_2$' that rearranges '$symb_1$' to "isolate" the first occurrence of variable *global*.

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

When flag −3 is set, symbolic results are evaluated to real numbers. This means that the = sign is evaluated. If *global* or any other variable in the result equation is formal, an `Undefined Name` error results; if *global* and all other variables have values, a numerical result is returned from the calculation *global* − *expression*. This result has limited value. In general, execute ISOL with flag −3 clear.

Remarks: The result '$symb_2$' is an equation of the form '*global=expression*'. If *global* appears more than once, then '$symb_2$' is effectively the right side of an equation obtained by rearranging and solving '$symb_1$' to isolate the first occurrence of *global* on the left side of the equation.

If '$symb_1$' is an expression, it is treated as the left side of an equation '$symb_1$=0'.)

If *global* appears in the argument of a function within '$symb_1$', that function must be an *analytic* function — a function for which the HP 48 provides an inverse. Thus ISOL cannot solve 'IP(X)=0' for *X*, since IP has no inverse.

Related Commands: COLCT, EXPAN, QUAD, SHOW

Level 1	→	Level 1
	→	*"error-message"*

Use: Returns the text of the most recent Kermit error packet.

Affected by Flags: None.

Remarks: If a Kermit transfer fails due to an error packet sent from the connected Kermit device to the HP 48, then executing KERRM retrieves and displays the error message. (Kermit errors not in packets are retrieved by ERRM rather than KERRM.)

Related Commands: FINISH, KGET, PKT, RECN, RECV, SEND, SERVER

Level 1	→	Level 2	Level 1
	→	x_{nm}	0/1

Use: Returns to level 1 a test result and, if a key is pressed, returns to level 2 the row-column location x_{nm} of that key.

Affected by Flags: None.

Remarks: KEY returns a false result (0) to level 1 until a key is pressed. When a key is pressed, it returns a true result (1) to level 1 and x_{nm} to level 2. The result x_{nm} is a two-digit number that identifies the row and column location of the key just pressed. Note that, unlike WAIT, which returns a three-digit number that identifies alpha and shifted keyboard planes, KEY returns the row-column location of *any* key pressed, including [◀], [▶], and [α].

Example: The program `« DO UNTIL KEY END 71 SAME »` returns 1 to the stack if the [◀] key is pressed while the indefinite loop is running.

Related Commands: WAIT

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

Use: Used by a local Kermit to get a server Kermit to transmit the named object(s) (that is, files).

Affected by Flags: I/O Device (-33), RECV Overwrite (-36), I/O Messages (-39).

I/O Data Format (-35) affects KGET if the file is transmitted *to* an HP 48.

Remarks: To rename an object when the local device gets it, include the old and new names in an embedded list. For example, `{{ AAA BBB }} KGET` gets the variable named *AAA* but changes its name to *BBB*. `{{ AAA BBB } CCC } KGET` gets *AAA* as *BBB* and gets *CCC* under its own name. (If the original name is not legal on the HP 48, enter it as a string.)

Related Commands: FINISH, RECN, RECV, SEND, SERVER

Level 1 →	Level 1
→	

Use: Cancels all currently halted programs. (Halted programs are typically cancelled by pressing PRG CTRL KILL.) If KILL is executed within a program, that program is also cancelled.

Affected by Flags: None.

Remarks: Cancelled programs can not be resumed.

KILL cancels *only* halted programs and the program from which KILL was executed, if any. Commands that halt programs are HALT and PROMPT.

Suspended programs cannot be cancelled. Commands that suspend programs are INPUT and WAIT.

Related Commands: CONT, DOERR, HALT, PROMPT

Level 1	→	Level 1
	→	

Use: Labels axes in *PICT* with *x*- and *y*-axis variable names and with the minimum and maximum values of the display ranges.

Affected by Flags: −30 (Function Plotting).

Remarks: The horizontal axis name is chosen in the following priority order:

1. If the *axes* parameter in the reserved variable *PPAR* is a list, then the "*x-axis*" element from that list.
2. The independent variable name in *PPAR*.

The vertical axis name is chosen in the following priority order:

1. If the *axes* parameter in *PPAR* is a list, then the "*x-axis*" element from that list.
2. If flag −30 is clear, and the equation in *EQ* is of the form '*name=expression*', where *name* is not the independent variable, then *name*.
3. The dependent variable name from *PPAR*.

The independent and dependent variable names are the defaults.

Related Commands: AXES, DRAW, DRAX

Level 1	→	Level 4	Level 3	Level 2	Level 1
	→				obj_1
	→			obj_2	obj_1
	→		obj_3	obj_2	obj_1
	→	obj_4	obj_3	obj_2	obj_1

Provided for compatibility with the HP 28S. LAST is the same as LASTARG. See LASTARG.

Level 1	→	Level 4	Level 3	Level 2	Level 1
	→				obj_1
	→			obj_2	obj_1
	→		obj_3	obj_2	obj_1
	→	obj_4	obj_3	obj_2	obj_1

Use: Returns copies of the arguments of the most recently executed command.

Affected by Flags: Last Arguments (−55).

Remarks: The objects return to the same stack levels that they originally occupied. Commands that take no arguments leave the current saved arguments unchanged.

Note that when LASTARG follows a command that evaluates an algebraic or a program (as do the commands ∂, ∫, TAYLR, COLCT, DRAW, ROOT, ISOL, EVAL, and →NUM), then the last arguments saved are from the evaluated algebraic or program, not from the original command.

Related Commands: LAST

Level 1	→	Level 1
	→	*grob*

Use: Returns the current stack display to level 1 as a 131 × 64 graphics object.

Affected by Flags: None.

Example: `LCD→ PICT STO GRAPH` returns the current display to level 1 as a graphics object, stores it in *PICT*, then shows the image in the Graphics environment.

Related Commands: →GROB, →LCD

Level 1	→	Level 1
grob	→	

Use: Displays in the *stack* display the graphics object from level 1, with its upper left pixel in the upper left corner of the display.

Affected by Flags: None.

Remarks: If the graphics object is larger than 131 × 64, it is truncated.

Related Commands: BLANK, →GROB, LCD→

Level 1	→	Level 1
	→	{ *"title"* $n_{library}$ n_{port} , . . . , *"title"* $n_{library}$ n_{port} }

Use: Lists the title, number, and port of each library attached to the current directory.

Affected by Flags: None.

Remarks: The *title* of a library often takes the form *"LIBRARY-NAME :Description"*. A library without a title is listed as `""`.

Example: LIBS lists these two libraries (among others) from the HP Solve Equation Library Application Card:

```
{ "" 266 2
"UTILS :Utilities"
268 2
 :
}
```

Both libraries are in port 2; library number 266 is untitled, while library number 268 supplies *utilities* and uses the menu label `UTILS`.

Related Commands: ATTACH, DETACH

LINE *Draw Line* **Command**

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

Use: Draws a line in *PICT* between the coordinates in levels 1 and 2.

Affected by Flags: None.

Example: The program

```
« (0,0) (2,3) LINE { # 0d # 0d } PVIEW 7 FREEZE »
```

draws a line in *PICT* between two user-unit coordinates, displays *PICT* with pixel coordinate `{ # 0d # 0d }` at the upper left corner of the graphics display, and freezes the display.

Related Commands: ARC, BOX, TLINE

Level 1	→	Level 1
	→	'*symb*$_{formula}$'

Use: Returns an expression representing the best fit line according to the current statistical model, using *X* as the independent variable name, and explicit values of the slope and intercept taken from the reserved variable Σ*PAR*.

Affected by Flags: None.

Remarks: For each curve fitting model, the following table indicates the form of the expression returned by ΣLINE, where *m* is the slope, *x* is the independent variable, and *b* is the intercept.

Model	Form of Expression
LINFIT	$mx + b$
LOGFIT	$m \ln x + b$
EXPFIT	$b\, e^{mx}$
PWRFIT	bx^m

Example: If the current model is EXPFIT, and if the slope is 5 and the intercept 3, ΣLINE returns `'3*EXP(5*X)'`.

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

Level 1	→	Level 1
	→	

Use: Stores its name as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the linear curve fitting model.

Affected by Flags: None.

Remarks: LINFIT is the default specification in Σ*PAR*.

Related Commands: BESTFIT, EXPFIT, LOGFIT, LR, PWRFIT

Level 1	→	Level n+1 ...Level 2	Level 1
{ $obj_1 \ldots obj_n$ }	→	$obj_1 \ldots obj_n$	n

Use: LIST→ takes a list of n objects and returns them into separate levels, with the number of objects in level 1.

Affected by Flags: None.

Remarks: The command OBJ→ includes this functionality. LIST→ is included for compatibility with the HP 28S. LIST→ is not in a menu.

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

→LIST *Stack to List* **Command**

Level n+1 ...Level 2	Level 1	→	Level 1
$obj_1 \ldots obj_n$	n	→	$\{ obj_1 \ldots obj_n \}$

Use: →LIST takes n objects from levels above level 1 and returns a list of those n objects.

Affected by Flags: None.

Example: The program

```
« DEPTH →LIST 'A' STO »
```

combines the entire contents of the stack into a list that is stored into variable A.

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

Level 1	→	Level 1
z	→	ln *z*
'*symb*'	→	'LN(*symb*)'

Use: Returns the natural (base *e*) logarithm of the argument.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22).

Remarks: For $x=0$ or (0, 0), an `Infinite Result` exception occurs. If flag −22 is set (no error), the sign of the result (MAXR) matches that of the argument.

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

```
'LN(Z)+2*π*i*n1'
```

The function LN is the inverse of a *part* of EXP, a part defined by restricting the domain of EXP such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of EXP are called the *principal values* of the inverse relation. LN in its entirety is called the *principal branch* of the inverse relation, and the points sent by LN to the boundary of the restricted domain of EXP form the *branch cuts* of LN.

The principal branch used by the HP 48 for LN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued natural log function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

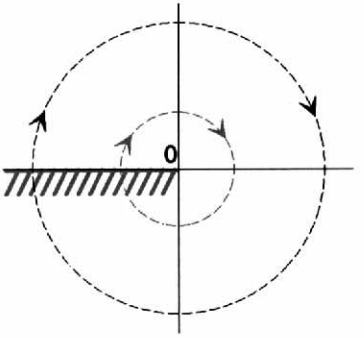
The graphs below show the domain and range of LN. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

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

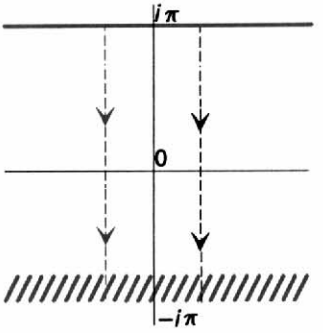
You can view these graphs with domain and range reversed to see how the domain of EXP is restricted to make an inverse *function* possible. Consider the vertical band in the lower graph as the restricted domain Z = (*x*,*y*). EXP sends this domain onto the whole complex plane in the range W = (*u*,*v*)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

Related Commands: ALOG, EXP, ISOL, LNP1, LOG

Domain: Z = (x,y)



Range: W = (u,v) = LN(x,y)



Branch Cut for LN (Z)

Level 1	→	Level 1
x	→	ln(*x*+1)
'*symb*'	→	'LNP1(*symb*)'

Use: Returns ln (*x* + 1).

Affected by Flags: Numerical Results (−3), Infinite Result Exception (−22).

Remarks: For values of *x* close to zero, `'LNP1(X)'` returns a more accurate result than does `'LN(X+1)'`. Using LNP1 allows both the argument and the result to be near zero, and it avoids an intermediate result near 1. The calculator can express numbers within 10^{-449} of zero, but within only 10^{-11} of 1.

For values of $x < -1$, an `Undefined Result` error results. For $x = -1$, an `Infinite Result` exception occurs. If flag −22 is set (no error), the sign of the result (MAXR) matches that of the argument.

Related Commands: EXPM, LN

LOG — *Common Logarithm* — **Analytic**

Level 1	→	Level 1
z	→	log *z*
'*symb*'	→	'LOG(*symb*)'

Use: Returns the common logarithm (base 10) of the argument.

Affected by Flags: Principal Solution (−1), Numerical Results (−3), Infinite Result Exception (−22).

Remarks: For $x=0$ or (0, 0), an `Infinite Result` exception occurs. If flag −22 is set (no error), the sign of the result (MAXR) matches that of the argument.

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

```
'LOG(Z)+2*π*i*n1/2.30258509299'
```

The function LOG is the inverse of a *part* of ALOG, a part defined by restricting the domain of ALOG such that 1) each argument is sent to a distinct result, and 2) each possible result is achieved. The points in this restricted domain of ALOG are called the *principal values* of the inverse relation. LOG in its entirety is called the *principal branch* of the inverse relation, and the points sent by LOG to the boundary of the restricted domain of ALOG form the *branch cuts* of LOG.

The principal branch used by the HP 48 for LOG(*z*) was chosen because it is analytic in the regions where the arguments of the real-valued function are defined. The branch cut for the complex-valued LOG function occurs where the corresponding real-valued function is undefined. The principal branch also preserves most of the important symmetries.

You can determine the graph for LOG(*z*) from the graph for LN (see LN) and the relationship $\log z = \ln z \,/\, \ln 10$.

Related Commands: ALOG, EXP, ISOL, LN

Level 1	→	Level 1
	→	

Use: Stores its name as the fifth parameter in the reserved variable ΣPAR, indicating that subsequent executions of LR are to use the logarithmic curve-fitting model.

Affected by Flags: None.

Remarks: LINFIT is the default specification in ΣPAR.

Related Commands: BESTFIT, EXPFIT, LINFIT, LR, PWRFIT

Level 1	→	Level 2	Level 1
	→	Intercept: x_1	Slope: x_2

Use: Using the currently selected statistical model, calculates the linear regression coefficients (intercept and slope) for the selected dependent and independent variables in the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The columns of independent and dependent data are specified by the first two elements in the reserved variable Σ*PAR*, set by XCOL and YCOL, respectively. (The default independent and dependent columns are 1 and 2.) The selected statistical model is the fifth element in Σ*PAR*. LR stores the intercept and slope (untagged) as the third and fourth elements, respectively, in Σ*PAR*.

The coefficients of the exponential (EXPFIT), logarithmic (LOGFIT), and power (PWRFIT) models are calculated using transformations that allow the data to be fitted by standard linear regression. The equations for these transformations appear in the table below, where b is the intercept and m is the slope. The logarithmic model requires positive x-values (XCOL), the exponential model requires positive y-values (YCOL), and the power model requires positive x- and y-values.

Transformation Equations

Model	Transformation
Logarithmic	$y = b + m \ln x$
Exponential	$\ln y = \ln b + mx$
Power	$\ln y = \ln b + m \ln x$

Related Commands: BESTFIT, COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, PREDX, PREDY, PWRFIT, XCOL, YCOL

MANT *Mantissa* Function

Level 1	→	Level 1
x	→	y_{mant}
'*symb*'	→	'MANT(*symb*)'

Use: Returns the mantissa of its argument.

Affected by Flags: Numerical Results (−3).

Example: `-1.2E34 MANT` returns `1.2`.

Related Commands: SIGN, XPON

Level 2	Level 1	→	Level 2	Level 1
'symb$_1$'	{ 'symb$_{pattern}$' 'symb$_{replacement}$' }	→	'symb$_2$'	0/1
'symb$_1$'	{ 'symb$_{pattern}$' 'symb$_{replacement}$' 'symb$_{conditional}$' }	→	'symb$_2$'	0/1

Use: Rewrites an expression.

Affected by Flags: None.

Remarks: ↑MATCH rewrites expressions or subexpressions that match a specified pattern. An optional condition can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; `1` if it did, `0` if it did not.

The pattern and replacement can be normal expressions; for example, you can replace `'SIN(π/6'` with `'1/2'`. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with `&`, such as the name `'&A'`, used in replacing `'SIN(&A+π)'` with `'-SIN(&A)'`. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↑MATCH works from bottom up; that is, it checks the lowest level (most deeply nested) subexpressions first. This approach works well for simplification. A subexpression simplified during one execution of ↑MATCH will be a simpler argument of its parent expression, so the parent expression can be simplified by another execution of ↑MATCH. Several subexpressions can be simplified by one execution of ↑MATCH provided none is a subexpression of any other.

Examples: The command sequence

```
'SIN(π/6)' { 'SIN(π/6) '1/2' } ↑MATCH
```

returns `'1/2'` to level 2 and `1` (indicating a replacement was made) to level 1.

...↑MATCH

The command sequence

```
'SIN(X+π)' ⟨ 'SIN(&A+π)' '-SIN(&A)' ⟩ ↑MATCH
```

returns `'-SIN(X)'` to level 2 and 1 to level 1.

The command sequence

```
'W+√(SQ(5))' ⟨ '√(SQ&A))' '&A' '&A≥0' ⟩ ↑MATCH
```

returns `'W+5'` to level 2 and 1 to level 1.

Related Commands: ↓MATCH

Level 2	Level 1	→	Level 2	Level 1
'symb$_1$'	{ 'symb$_{pattern}$' 'symb$_{replacement}$' }	→	'symb$_2$'	0/1
'symb$_1$'	{ 'symb$_{pattern}$' 'symb$_{replacement}$' 'symb$_{conditional}$' }	→	'symb$_2$'	0/1

Use: Rewrites an expression.

Affected by Flags: None.

Remarks: ↓MATCH rewrites expressions or subexpressions that match a specified pattern. An optional condition can further restrict whether a rewrite occurs. A test result is also returned to indicate if command execution produced a rewrite; `1` if it did, `0` if it did not.

The pattern and replacement can be normal expressions; for example, you can replace `.5` with `'SIN(π/6'`. You can also use a "wildcard" in the pattern (to match any subexpression) and in the replacement (to represent that expression). A wildcard is a name that begins with `&`, such as the name `'&A'`, used in replacing `'SIN(&A+&B)'` with `'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)'`. Multiple occurrences of a particular wildcard in a pattern must match identical subexpressions.

↓MATCH works from top down; that is, it checks the entire expression first. This approach works well for expansion. An expression expanded during one execution of ↓MATCH will contain additional subexpressions, and those subexpressions can be expanded by another execution of ↓MATCH. Several expressions can be expanded by one execution of ↓MATCH provided none is a subexpression of any other.

Examples: The command sequence

```
.5 ‹ .5 'SIN(π/6)' › ↓MATCH
```

returns `'SIN(π/6)'` to level 2 and `1` to level 1.

The command sequence

```
'SIN(U+V)' ‹ 'SIN(&A+&B)'
'SIN(&A)*COS(&B)+COS(&A)*SIN(&B)' › ↓MATCH
```

returns `'SIN(U)*COS(V)+COS(U)*SIN(V)'` to level 2 and `1` to level 1.

The command sequence

```
    'SIN(5*Z)' { 'SIN(&A+&B)'
'Σ(K=0,&A,COMB(&A,K)*SIN(K*π/2)*
    COS(&B^(&A-K)*SIN(&B)^K)'
    'ABS(IP(&A))==&A' } ↓MATCH
```

returns

'Σ(K=0,5,COMB(5,K)*SIN(K*π/2)*COS(Z)^(5-K)*SIN(Z)^K)'

to level 2 and 1 to level 1.

Related Commands: ↑MATCH

MAX — *Maximum* — Function

Level 2	Level 1	→	Level 1
x	*y*	→	max(*x*, *y*)
x	'*symb*'	→	'MAX(*x*, *symb*)'
'*symb*'	*x*	→	'MAX(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MAX($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	max(x_unit_1, y_unit_2)

Use: Returns the greater (more positive) of its two arguments.

Affected by Flags: Numerical Results (−3).

Examples: Evaluating `10 -23 MAX` returns `10`.

Evaluating `-10 -23 MAX` returns `-10`.

Evaluating `1_m 9_cm MAX` returns `1_m`.

Related Commands: MIN

MAXR *Maximum Real* Function

Level 1	→	Level 1
	→	'MAXR'
	→	9.99999999999E499

Use: Returns the symbolic constant `'MAXR'` or its numerical representation, `9.99999999999E499`.

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

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

Remarks: MAXR is the largest numerical value that can be represented by the HP 48.

Related Commands: e, i, MINR, π

Level 1	→	Level 1
	→	x_{max}
	→	$[x_{max1}\ x_{max2} \cdots x_{max\ m}]$

Use: Finds the maximum coordinate value in each of the m columns of the current statistics matrix (reserved variable ΣDAT).

Affected by Flags: None.

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

Related Commands: BINS, MEAN, MINΣ, SDEV, TOT, VAR

MEAN — *Mean* — Command

Level 1	→	Level 1
	→	x_{mean}
	→	$[\, x_{mean1}\ x_{mean2} \cdots x_{mean\,m} \,]$

Use: Computes the mean of each of the m columns of coordinate values in the current statistics matrix (reserved variable ΣDAT).

Affected by Flags: None.

Remarks: The mean is returned as a vector of m real numbers, or as a single real number if $m = 1$. The mean is computed from the formula:

$$\frac{1}{n} \sum_{i=1}^{n} x_i$$

where x_i is the ith coordinate value in a column, and n is the number of data points.

Related Commands: BINS, MAXΣ, MINΣ, SDEV, TOT, VAR

Level 1	→	**Level 1**
	→	*x*

Use: Returns the number of bytes of available memory in RAM.

Affected by Flags: None.

Remarks: The number returned is only a rough indicator of usable available memory, since recovery features (LASTARG, ↰ LAST STACK, and ↰ LAST CMD) consume or release varying amounts of memory with each operation.

Before it can assess the amount of memory available, MEM must remove objects in temporary memory that are no longer being used. This clean-up process (also called "garbage collection") also occurs automatically at other times when memory is full. Since this process can slow down calculator operation at undesired times, you can force it to occur at a desired time by executing `MEM`. In a program, execute `MEM DROP`.

Related Commands: BYTES

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

Use: Displays a built-in menu or a library menu, or defines and displays a custom menu.

Affected by Flags: None.

Remarks: A built-in menu is specified by a real number x_{menu}. The format of x_{menu} is *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu. If *pp* doesn't correspond to a page of the specified menu, the first page is displayed. The following table lists the HP 48 built-in menus and the corresponding menu numbers.

Menu #	Menu Name		Menu #	Menu Name
0	Last Menu		13	PRG DSPL
1	CST		14	PRG CTRL
2	VAR		15	PRG BRCH
3	MTH		16	PRG TEST
4	MTH PARTS		17	PRINT
5	MTH PROB		18	I/O
6	MTH HYP		19	I/O SETUP
7	MTH MATR		20	MODES
8	MTH VECTR		21	MODES Customization
9	MTH BASE		22	MEMORY
10	PRG		23	MEMORY Arithmetic
11	PRG STK		24	LIBRARY
12	PRG OBJ		25	PORT 0

(continued)

Menu #	Menu Name	Menu #	Menu Name
26	PORT 1	43	UNITS LENG
27	PORT 2	44	UNITS AREA
28	EDIT	45	UNITS VOL
29	SOLVE	46	UNITS TIME
30	SOLVE SOLVR	47	UNITS SPEED
31	PLOT	48	UNITS MASS
32	PLOT PTYPE	49	UNITS FORCE
33	PLOT PLOTR	50	UNITS ENRG
34	ALGEBRA	51	UNITS POWR
35	TIME	52	UNITS PRESS
36	TIME ADJST	53	UNITS TEMP
37	TIME ALRM	54	UNITS ELEC
38	TIME ALRM RPT	55	UNITS ANGL
39	TIME SET	56	UNITS LIGHT
40	STAT	57	UNITS RAD
41	STAT MODL	58	UNITS VISC
42	UNITS Catalog	59	UNITS Command

Library menus are specified in the same way as built-in menus, with the library number serving as the menu number.

Custom menus are specified by a list ($\langle\langle$ $list_{definition}$ $\rangle\rangle$) or a name containing a list ($name_{definition}$). Either argument is stored in reserved variable *CST*, and the custom menu is subsequently displayed. See appendix D, "Reserved Variables," for a description of the custom menu.

Note that MENU in fact takes *any* object as a valid argument and stores it in *CST*. However, the calculator can build a custom menu *only* if *CST* contains a list or a name containing a list. Thus, if an object other than a list or name containing a list is supplied to MENU, a `Bad Argument Type` error will occur when the calculator attempts to display the custom menu.

...MENU

See "Custom Menus in Programs" in chapter 29 of the *HP 48 Owner's Manual* for program examples using MENU.

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

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

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

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

`'MYMENU' MENU` displays the custom menu defined the name argument.

Related Commands: RCLMENU, TMENU

Level 1	→	Level 1
n_{port}	→	

Use: Takes the RAM from the card in the specified port (1 or 2) and merges it with the rest of main user memory. *Merged* memory is no longer *independent*.

Affected by Flags: None.

Remarks: If the RAM card contains any library or backup objects, then they are moved to port 0 before the RAM is merged. Library and backup objects can exist only in independent memory (ports 1 or 2 unmerged or port 0).

Related Commands: FREE

MIN *Minimum* Function

Level 2	Level 1	→	Level 1
x	*y*	→	min(*x*, *y*)
x	'*symb*'	→	'MIN(*x*, *symb*)'
'*symb*'	*x*	→	'MIN(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MIN($symb_1$, $symb_2$)'
x_unit_1	y_unit_2	→	min(x_unit_1, y_unit_2)

Use: Returns the lesser (more negative) of its two arguments.

Affected by Flags: Numerical Results (−3).

Example: Evaluating `10 23 MIN` returns `10`.

Evaluating `-10 -23 MIN` returns `-23`.

Evaluating `1_m 9_cm MIN` returns `9_cm`.

Related Commands: MAX

Level 1	→	Level 1
	→	'MINR'
	→	1.00000000000E-499

Use: Returns the symbolic constant `'MINR'` or its numerical representation, `1.00000000000E-499`.

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

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

Remarks: MINR is the smallest non-zero numerical value that can be represented by the HP 48.

Related Commands: e, i, MAXR, π

→	**Level 1**
→	x_{min}
→	$[\, x_{min1}\ x_{min2} \cdots x_{min\,m} \,]$

Use: Finds the minimum coordinate value in each of the *m* columns of the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

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

Related Commands: BINS, MAXΣ, MEAN, SDEV, TOT, VAR

MOD *Modulo* Function

Level 2	Level 1	→	Level 1
x	*y*	→	*x* mod *y*
x	'*symb*'	→	'MOD(*x*, *symb*)'
'*symb*'	*x*	→	'MOD(*symb*, *x*)'
'$symb_1$'	'$symb_2$'	→	'MOD($symb_1$, $symb_2$)'

Use: Returns a remainder defined by:

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

Affected by Flags: Numerical Results (−3).

Remarks: Mod (x, y) is periodic in x with period y. Mod (x, y) lies in the interval $[0, y)$ for $y > 0$ and in $(y, 0]$ for $y < 0$.

Related Commands: FLOOR, /

Level 1	→	Level 1
z	→	*−z*
$\#n_1$	→	$\#n_2$
[*array*]	→	[*−array*]
'*symb*'	→	'*−(symb)*'
x_unit	→	*−x_unit*
$grob_1$	→	$grob_2$
$PICT_1$	→	$PICT_2$

Use: Changes the sign or negates an object.

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

Remarks: Negating an array creates a new array containing the negative of each of the original elements.

Negating a binary number takes its two's complement; that is, it complements all the bits and adds 1.

Negating a graphics object "inverts" it, that is, each pixel is toggled from on (dark) to off (light) or vice-versa. If the argument is *PICT*, then the graphics object stored in *PICT* is inverted.

Related Commands: ABS, CONJ, NOT, SIGN

Level 1	→	Level 1
obj	→	*obj*

Use: Creates a new copy of the specified object, thereby removing any previous "references" (pointers) to that object or the list or backup object it came from.

Affected by Flags: Last Arguments (–55).

In order for NEWOB to free immediately the temporary memory occupied by a list, this flag must be set so that the list is not saved as a last argument.

Remarks: NEWOB has two main uses:

- NEWOB enables the purging of a library or backup object that has been recalled to the stack. (Note that it is the *contents* of a backup object that get recalled, not the backup object itself.) Recalling such an object sets a pointer to it and, as long as it is so referenced, it cannot be purged. NEWOB creates a separate copy of the object in temporary memory, thereby allowing the original copy to be purged.

  The following command sequence would recall and then purge the backup object named *FRED*:

```
:0:FRED RCL NEWOB :0:FRED PURGE
```

- NEWOB enables the purging of a list from temporary memory (a list not stored in a variable) while one or more elements extracted from the list are on the stack. Extracting (GETting) an element from such a list sets a pointer to it, and as long as the list is so referenced, it cannot be cleared from temporary memory (even though it is no longer on the stack). NEWOB creates a separate copy of the element, thereby allowing the original list to be purged during normal, internal clean-up (also called "garbage collection").

The following command sequence would get the third element out of a list that was on the stack, then enable the list to be purged during normal clean-up:

```
3 GET NEWOB
```

Related Commands: PURGE

See the FOR and START keyword entries for syntax information.

Use: Ends definite loop structures. See the FOR and START keyword entries for more information.

Related Commands FOR, START, STEP

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$
T/F	→	0/1
"*string*$_1$"	→	"*string*$_2$"
'*symb*'	→	'NOT *symb*'

Use: Returns the one's complement or logical inverse of the argument.

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

Remarks: When the argument is a binary integer or string, NOT complements each bit in the argument to produce the result.

- A binary integer is treated as a sequence of bits as long as the current wordsize.
- A string is treated as a sequence of bits, using 8 bits per character (that is, using the binary version of the character code).

When the argument is a real number or symbolic, NOT does a true/false test. The result is 1 (true) if the argument is zero; it is 0 (false) if the argument is non-zero. This test is usually done on a test result (T/F).

If the argument is an algebraic object, then the result is an algebraic of the form 'NOT *symb*'. Execute →NUM (or set flag −3 before executing NOT) to produce a numeric result from the algebraic result.

Related Commands: AND, OR, XOR

Level 1	→	Level 1
"string"	→	*n*

Use: Returns the character code *n* for the first character in the string.

Affected by Flags: None.

Remarks: The character codes are an extension of ISO 8859/1. Codes 128 through 159 are unique to the HP 48.

The following table shows the relation between character codes (results of NUM, arguments to CHR) and characters (results of CHR, arguments to NUM).

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

Character Codes (0 — 127)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
0	■	32		64	@	96	'
1	■	33	!	65	A	97	a
2	■	34	"	66	B	98	b
3	■	35	#	67	C	99	c
4	■	36	$	68	D	100	d
5	■	37	%	69	E	101	e
6	■	38	&	70	F	102	f
7	■	39	'	71	G	103	g
8	■	40	(	72	H	104	h
9	■	41	)	73	I	105	i
10	■	42	*	74	J	106	j
11	■	43	+	75	K	107	k
12	■	44	,	76	L	108	l
13	■	45	-	77	M	109	m
14	■	46	.	78	N	110	n
15	■	47	/	79	O	111	o
16	■	48	0	80	P	112	p
17	■	49	1	81	Q	113	q
18	■	50	2	82	R	114	r
19	■	51	3	83	S	115	s
20	■	52	4	84	T	116	t
21	■	53	5	85	U	117	u
22	■	54	6	86	V	118	v
23	■	55	7	87	W	119	w
24	■	56	8	88	X	120	x
25	■	57	9	89	Y	121	y
26	■	58	:	90	Z	122	z
27	■	59	;	91	[	123	{
28	■	60	<	92	\	124	|
29	■	61	=	93	]	125	}
30	■	62	>	94	^	126	~
31	...	63	?	95	_	127	■

Character Codes (128 — 255)

NUM	CHR	NUM	CHR	NUM	CHR	NUM	CHR
128	∠	160		192	À	224	à
129	⊼	161	¡	193	Á	225	á
130	▽	162	¢	194	Â	226	â
131	√	163	£	195	Ã	227	ã
132	∫	164	⊠	196	Ä	228	ä
133	Σ	165	¥	197	Å	229	å
134	▶	166	¦	198	Æ	230	æ
135	π	167	§	199	Ç	231	ç
136	∂	168	¨	200	È	232	è
137	≤	169	©	201	É	233	é
138	≥	170	ª	202	Ê	234	ê
139	≠	171	«	203	Ë	235	ë
140	∝	172	¬	204	Ì	236	ì
141	→	173	-	205	Í	237	í
142	←	174	®	206	Î	238	î
143	↓	175	¯	207	Ï	239	ï
144	↑	176	°	208	Ð	240	ð
145	γ	177	±	209	Ñ	241	ñ
146	ó	178	²	210	Ò	242	ò
147	∈	179	³	211	Ó	243	ó
148	η	180	´	212	Ô	244	ô
149	θ	181	µ	213	Õ	245	õ
150	↖	182	¶	214	Ö	246	ö
151	ρ	183	·	215	×	247	÷
152	σ	184	¸	216	Ø	248	ø
153	τ	185	¹	217	Ù	249	ù
154	ω	186	º	218	Ú	250	ú
155	Δ	187	»	219	Û	251	û
156	Π	188	¼	220	Ü	252	ü
157	Ω	189	½	221	Ý	253	ý
158	■	190	¾	222	Þ	254	þ
159	∞	191	¿	223	ß	255	ÿ

→NUM *Evaluate to Number* **Command**

Level 1	→	Level 1
obj	→	*z*

Use: Evaluates the object into a numerical result.

Affected by Flags: None.

Remarks: →NUM repeatedly evaluates a symbolic argument until a numerical result is achieved. The effect is the same as evaluating the symbolic argument in Numerical Result mode (flag −3 set).

Related Commands: EVAL, SYSEVAL

Level 1	→	Level 1
	→	n_{rows}

Use: Returns the number of rows in the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

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

OBJ→ *Object to Stack* **Command**

Level 1	→	Level n+1 ...	Level 2	Level 1
(x,y)	→		x	y
{ $obj_1 \ldots obj_n$ }	→	obj_1	obj_n	n
[$x_1 \ldots x_n$]	→	x_1	x_n	n
[[$x_{11} \ldots x_{mn}$]]	→	x_{11}	x_{mn}	{ m n }
"obj"	→			evaluated-object
'symb'	→	$arg_1 \ldots arg_n$	n	'function'
x_unit	→		x	1_unit
:tag:obj	→		obj	"tag"

Use: Separates an object into its components onto the stack. For some object types, the *number* of components is returned to level 1.

Affected by Flags: None.

Remarks: For complex numbers, lists, arrays, and strings, OBJ→ encompasses the functionality of C→R, LIST→, ARRY→, and STR→. For lists, OBJ→ also returns the number of list elements. For arrays, OBJ→ also returns the dimensions { *m n* } of the array, where *m* is the number of rows and *n* is the number of columns.

For algebraic objects, OBJ→ returns the arguments of the top-level (least-nested) function ($arg_1 \ldots arg_n$), the number of arguments of the top-level function (*n*), and the name of the top-level function (*function*).

For strings, the object sequence defined by the string is executed.

Example: The command sequence `'∫ (0,1,SIN(X),X)' OBJ→` returns:

6:	0	First argument.
5:	1	Second argument.
4:	'SIN(X)'	Third argument.
3:	'X'	Fourth argument.
2:	4	Number of arguments for ∫.
1:	∫	Function name.

Related Commands: ARRY→, C→R, DTAG, →EQ, LIST→, R→C, STR→, →TAG

OCT — *Octal Mode* — Command

Level 1	→	Level 1
	→	

Use: Selects octal base for binary integer operations. (The default base is decimal.)

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

Remarks: Binary integers require the prefix #. Binary integers entered and returned in octal base automatically show the suffix o. If the current base is not octal, then you can enter an octal number by ending it with o. It will be displayed in the current base when it is entered.

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

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

Level 1	→	**Level 1**
	→	

Use: Turns off the calculator.

Affected by Flags: None.

Remarks: When executed from a program, that program will resume execution when the calculator is turned on. This enables programming an "autostart" capability.

Related Commands: CONT, HALT, KILL

Level 1 → Level 1
→

Use: Modifies the remapping string in the reserved variable *PRTPAR* so that the extended character set of the HP 48 matches the HP 82240A Infrared Printer.

Affected by Flags: None.

Remarks: The character set in the HP 82240A Infrared Printer does not match the HP 48 character set:

- 24 characters in the HP 48 character set are not available in the HP 82240A Infrared Printer. (From the table in the keyword listing for NUM, these characters are numbers 129, 130, 143-157, 159, 166, 169, 172, 174, 184, and 185.) The HP 82240A prints a ▒ in substitution.
- Many characters in the extended character table (character codes 128 through 255) do not have the same character code. For example, the « character has code 171 in the HP 48 and code 146 in the HP 82240A Infrared Printer.

If you want to use the CHR command to print extended characters with an HP 82240A Infrared Printer, first execute OLDPRT. The remapping string modified by OLDPRT is the second parameter in *PRTPAR*. This string, empty in the default state, changes the character code of each byte to match the codes in the HP 82240A Infrared Printer character table.

If you want to print a string containing graphics data, OLDPRT must *not* be in effect.

Related Commands: CR, DELAY, PRLCD, PRST, PRSTC, PRVAR, PR1

Level 1	→	**Level 1**
	→	

Use: Opens the serial port or the IR port using the I/O parameters in the reserved variable *IOPAR*.

Affected by Flags: I/O Device (-33).

Remarks: All HP 48 Kermit-protocol commands automatically effect an OPENIO first. Therefore, you usually do not need to execute OPENIO, but you should use it if an I/O transmission does not work. OPENIO is necessary for I/O interaction with a device that interprets a closed port as a break.

OPENIO is also necessary for the automatic reception of data into the input buffer using non-Kermit commands. If the port is closed, incoming characters will be ignored. If the port is open, incoming characters will automatically be placed in the input buffer. The presence of these characters can be detected with BUFLEN and they can be read out of the input buffer using SRECV.

If the port is already open, OPENIO does not affect the data in the input buffer, but if the port is closed, then executing OPENIO clears the data in the input buffer.

For more information, refer also to the reserved variable *IOPAR* in appendix D.

Related Commands: BUFLEN, CLOSEIO

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* OR *symb*'
'*symb*'	*T/F*	→	'*symb* OR *T/F*'
'$symb_1$'	'$symb_2$'	→	'$symb_1$ OR $symb_2$'

Use: Returns the logical OR of two arguments.

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

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

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

bit_1	bit_2	bit_1 **OR** bit_2
0	0	0
0	1	1
1	0	1
1	1	1

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

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

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

Related Commands: AND, NOT, XOR

Level 1	→	Level 1
{ $global_1 \ldots global_n$ }	→	

Use: Reorders the variables in the current directory (shown in the VAR menu) to the order specified.

Affected by Flags: None.

Remarks: The names that appear first in the list will be the first to appear in the VAR menu. Variables not specified in the list are placed after the reordered variables.

If the list includes the name of a large subdirectory, there may be insufficient memory to execute ORDER. For possible remedies, refer to "Low-Memory Conditions" in chapter 5 of the *HP 48 Owner's Manual.*

Related Commands: VARS

OVER — *Over* — Command

Level 2	Level 1	→	Level 3	Level 2	Level 1
obj_1	obj_2	→	obj_1	obj_2	obj_1

Use: Returns a copy to stack level 1 of the object in level 2.

Affected by Flags: None.

Related Commands: PICK, ROLL, ROLLD, ROT, SWAP

PARAMETRIC *Parametric Plot Type* **Command**

Level 1	→	Level 1
	→	

Use: Sets the plot type to PARAMETRIC.

Affected by Flags: None.

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

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

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

- (x_{min} , y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is `(-6.5,-3.1)`.
- (x_{max} , y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is `(6.5,3.2)`.
- *indep* is a list containing a name that specifies the independent variable, and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). Note that the default value is *X*. If you don't modify *X* to include it in a list with a plotting range, the values in (x_{min} , y_{min}) and (x_{max} , y_{max}) are used as the plotting range, which generally leads to meaningless results.
- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is `0`, which specifies an interval equal to 1/130 of the difference between the maximum and minimum values in *indep* (the plotting range).
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.

- *ptype* is a command name specifying the plot type. Executing the command PARAMETRIC places the command name PARAMETRIC in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The contents of *EQ* must be an expression or program; it can't be an equation. It is evaluated for each value of the independent variable. The results, which must be a complex numbers, give the coordinates of the points to be plotted. Lines are drawn between plotted points unless flag −31 is set.

See "Parametric Plots" in chapter 19 of the *HP 48 Owner's Manual* for an example using the PARAMETRIC plot type.

Related Commands: BAR, CONIC, FUNCTION, HISTOGRAM, POLAR, SCATTER, TRUTH

Level 1	→	Level 1
n_{parity}	→	

Use: Sets the parity value in the reserved variable *IOPAR*.

Affected by Flags: None.

Remarks: Legal *n*-values are as shown below. A negative value means SRECV will not check parity, although parity will still be used during data transmission.

n-Value	Meaning
0	No parity. The default value.
1	Odd parity.
2	Even parity.
3	Mark.
4	Space.

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D of this manual.

Related Commands: BAUD, CKSM, TRANSIO

Level 1	→	Level 1
	→	{ HOME *directory-name*$_1$ ... *directory-name*$_n$ }

Use: Returns a list specifying the sequence of directory names to the current directory.

Affected by Flags: None.

Remarks: The first directory is always *HOME* and the last directory is always the current directory.

If a program needs to switch to a specific directory, it can do so by evaluating a directory list, such as one created earlier by PATH.

Related Commands: CRDIR, HOME, PGDIR, UPDIR

Level 2	Level 1	→	Level 1
(x_{min}, y_{min})	(x_{max}, y_{max})	→	
$\#n_{width}$	$\#m_{height}$	→	

Use: Replaces *PICT* with a blank *PICT* of the specified dimensions.

Affected by Flags: None.

Remarks: If the arguments are complex numbers, PDIM changes the size of *PICT* and makes the arguments the new values of (x_{min}, y_{min}) and (x_{max}, y_{max}) in the reserved variable *PPAR*. Thus, the scale of a subsequent plot is not changed. If the arguments are binary integers, *PPAR* remains unchanged, so the scale of a subsequent plot *is* changed.

PICT cannot be smaller than 131 pixels wide × 64 pixels high, nor larger than 2048 pixels wide.

Related Commands: PMAX, PMIN

Level 2	Level 1	→	Level 1
n	*m*	→	$P_{n,m}$
'*symb*$_n$'	*m*	→	'PERM(*symb*$_n$, *m*)'
n	'*symb*$_m$'	→	'PERM(*n*, *symb*$_m$)'
'*symb*$_n$'	'*symb*$_m$'	→	'PERM(*symb*$_n$, *symb*$_m$)'

Use: Returns the number of permutations of *n* items taken *m* at a time.

Affected by Flags: Numerical Results (−3).

Remarks: The calculation formula is:

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

The arguments *n* and *m* must be less than 10^{12}.

Related Commands: COMB, !

Level 1	→	Level 1
'global'	→	

Use: Purges the named directory of *all* of its variables and subdirectories (empty or not).

Affected by Flags: None.

Related Commands: CLVAR, CRDIR, HOME, PATH, PURGE, UPDIR

Level n+1...Level 2	Level 1	→	Level n+1...Level 2	Level 1
$obj_1 \ldots obj_n$	n	→	$obj_1 \ldots obj_n$	obj_1

Use: Takes an integer n from the stack and returns a copy of obj_1 (the nth remaining object).

Affected by Flags: None.

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

Level 1	→	Level 1
	→	*PICT*

Use: Puts the name `PICT` on the stack.

Affected by Flags: None.

Remarks: *PICT* is the name of a storage location in calculator memory containing the current graphics object. The command PICT enables access to the contents of that memory location as if it were a variable. Note however, that *PICT* is *not* a variable as defined in the HP 48—its name cannot be quoted and only graphics objects may be "stored" in it.

If a graphics object smaller than 131 wide × 64 pixels high is stored in *PICT*, it is enlarged to 131 × 64. A graphics object of unlimited pixel height and up to 2048 pixels wide can be stored in *PICT*.

Examples: `PICT RCL` returns the current graphics object to the stack.

`GRAPHIC 131 × 64 PICT STO` stores a graphics object in *PICT*, making it the current graphics object.

Related Commands: GOR, GRAPH, GXOR, NEG, PVIEW, RCL, REPL, SIZE, STO, SUB

Level 1	→	Level 1
(x, y)	→	
{ #n #m }	→	

Use: Turns off (makes light) the pixel at the specified coordinate in *PICT*.

Affected by Flags: None.

Related Commands: PIXON, PIX?

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

Use: Turns on (makes dark) the pixel at the specified coordinate in *PICT*.

Affected by Flags: None.

Related Commands: PIXOFF, PIX?

Level 1	→	Level 1
(x, y)	→	0/1
{ #n #m }	→	0/1

Use: Tests whether the specified pixel in *PICT* is on and returns a corresponding test result.

Affected by Flags: None.

Remarks: PIX? returns `1` (true) if the specified pixel is on (dark), and `0` (false) if the specified pixel is off (light).

Related Commands: PIXON, PIXOFF

Level 2	Level 1	→	Level 1
"data"	*"type"*	→	*"response"*

Use: Used to send command "packets" (and receive requested data) to a Kermit server. (To send HP 48 objects, use SEND.)

Affected by Flags: I/O Device (−33), I/O Messages (−39).

The I/O Data Format flag (−35) can be significant if the server sends back more than one packet.

Remarks: PKT allows you to send additional commands to a Kermit server. However, you need to understand Kermit well to take advantage of this. It is beyond the scope of the HP 48 documentation to document Kermit protocol. (Refer to *Using MS-DOS Kermit* by Christine M. Gianone, Digital Press, 1990; or *KERMIT, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987, especially chapter 11, "The Client/Server Model.")

The packet data, packet type, and the response to the packet transmission are all in string form. PKT first does an I (*initialization*) packet exchange with the Kermit server, then sends the server a packet constructed from the data and packet-type arguments supplied to PKT. The response to PKT will be either an acknowledging message (possibly blank) or an error packet (see KERRM).

For the *type* argument, only the first letter is significant.

Examples: A PKT command to send a *g*eneric *d*irectory request is `"D" "G" PKT`.

To send a *host command* packet, use a command from the server's operating system for the *data* string and `"C"` for the *type* string. For example, `"'ABC' PURGE" "C" PKT` on a local HP 48 would instruct an HP 48 server to purge variable *ABC*.

Related Commands: KERRM

Level 1	→	Level 1
(x, y)	→	

Use: Specifies (x, y) as the coordinates of the upper right corner of *PICT*.

Affected by Flags: None.

Remarks: The complex number (x, y) is stored as the second element in the reserved variable *PPAR*.

PMAX must be typed in or placed in a custom menu—it is not included in a built-in menu.

Related Commands: PDIM, PMIN, XRNG, YRNG

Level 1	→	Level 1
(x,y)	→	

Use: Specifies (x, y) as the coordinates of the lower left corner of *PICT*.

Affected by Flags: None.

Remarks: The complex number (x, y) is stored as the first element in the reserved variable *PPAR*.

PMIN must be typed in or placed in a custom menu—it is not included in a built-in menu.

Related Commands: PDIM, PMAX, XRNG, YRNG

POLAR *Polar Plot Type* Command

Level 1 →	Level 1
→	

Use: Sets the plot type to POLAR.

Affected by Flags: None.

Remarks: When the plot type is POLAR, the DRAW command plots the current equation in polar coordinates, where the independent variable is the polar angle and the dependent variable is the radius. The current equation is specified in the reserved variable *EQ*. The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

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

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

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is `(-6.5,-3.1)`.
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is `(6.5,3.2)`.
- *indep* is a name specifying the independent variable; or a list containing such a name and two numbers specifying the minimum and maximum values for the independent variable (the plotting range). The default value of *indep* is *X*.
- *res* is a real number specifying the interval, in user-unit coordinates, between values of the independent variable. The default value is `0`, which specifies an interval of 2 degrees, 2 grads, or $\pi/90$ radians.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.
- *ptype* is a command name specifying the plot type. Executing the command POLAR places the command name POLAR in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

The current equation is plotted as a function of the variable specified in *indep*. The minimum and maximum values of the independent variable (the plotting range) can be specifed in *indep*; otherwise, the default minimum value is 0 and the default maximum value corresponds to one full circle in the current angle mode (360 degrees, 400 grads, or 2π radians). Lines are drawn between plotted points unless flag −31 is set.

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

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

If flag −30 is *set*, all equations are plotted as two separate expressions.

See "Polar Plots" in chapter 19 of the *Owner's Manual* for an example using the POLAR plot type.

Related Commands: BAR, CONIC, FUNCTION, HISTOGRAM, PARAMETRIC, SCATTER, TRUTH

Level 2	Level 1	→	Level 1
"string"	*"substring"*	→	*n*
{ list }	*obj*	→	*n*

Use: Returns the position of a substring within a string or the position of an object within a list.

Affected by Flags: None.

Remarks: If there is no match for *obj* or *substring*, POS returns zero.

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

PREDV *Predicted y-value* Command

Level 1	→	Level 1
$x_{independent}$	→	$y_{dependent}$

Provided for compatibility with the HP 28. PREDV is the same as PREDY.
See PREDY.

Level 1	→	Level 1
$y_{dependent}$	→	$x_{independent}$

Use: Based on the currently selected statistical model and the current regression coefficients in the reserved variable Σ*PAR*, PREDX returns the predicted independent-variable value given a dependent-variable value.

Affected by Flags: None.

Remarks: The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable Σ*PAR*. For the linear statistical model, the equation used is:

$$y_{dependent} = (x_{independent}\ m) + b$$

where m is the slope (the third element in Σ*PAR*) and b is the intercept (the fourth element in Σ*PAR*).

For the other statistical models, the equations used by PREDX are listed in the LR keyword entry.

If you execute PREDX without having previously generated regression coefficients in Σ*PAR*, a default value of zero is used for both regression coefficients — in this case PREDX will error.

Example: Given five columns of data in Σ*DAT*, the command sequence:

```
2 XCOL 5 YCOL LOGFIT LR 23 PREDX
```

sets column 2 as the independent variable column, sets column 5 as the dependent variable column, and sets the logarithmic statistical model. It then executes LR, generating intercept and slope regression coefficients, stored in Σ*PAR*. Then, given a dependent value of 23, it returns a predicted independent value based on the regression coefficients and the statistical model.

Related Commands: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDY, PWRFIT, XCOL, YCOL

PREDY *Predicted y-value* **Command**

Level 1	→	Level 1
$x_{independent}$	→	$y_{dependent}$

Use: Based on the currently selected statistical model and the current regression coefficients in the reserved variable Σ*PAR*, PREDY returns the predicted dependent-variable value given an independent-variable value.

Affected by Flags: None.

Remarks: The value is predicted using the regression coefficients most recently computed with LR and stored in the reserved variable Σ*PAR*. For the linear statistical model, the equation used is:

$$y_{dependent} = (x_{predicted} m) + b$$

where *m* is the slope (the third element in Σ*PAR*) and *b* is the intercept (the fourth element in Σ*PAR*).

For the other statistical models, the equations used by PREDY are listed in the LR keyword entry.

If you execute PREDY without having previously generated regression coefficients in Σ*PAR*, a default value of zero is used for both regression coefficients—in this case PREDY will return 0 for statistical models LINFIT and LOGFIT, and error for statistical models EXPFIT and PWRFIT,

Example: Given four columns of data in Σ*DAT*, the command sequence:

```
2 XCOL 4 YCOL PWRFIT LR 11 PREDY
```

sets column 2 as the independent variable column, sets column 4 as the dependent variable column, and sets the power statistical model. It then executes LR, generating intercept and slope regression coefficients, stored in Σ*PAR*. Then, given an independent value of 11, it returns a predicted dependent value based on the regression coefficients and the statistical model.

Related Commands: COLΣ, CORR, COV, EXPFIT, ΣLINE, LINFIT, LOGFIT, LR, PREDX, PWRFIT, XCOL, YCOL

Level 1	→	Level 1
	→	

Use: Prints a pixel-by-pixel image of the current display (excluding the annunciators).

Affected by Flags: Printing Device (−34), I/O Device (−33)

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

Remarks: The width of the printed image of characters in the display is narrower using PRLCD than using a print command such as PR1. The difference results from the spacing between characters. On the display there is a single blank column between characters, and PRLCD prints this spacing. Print commands such as PR1 print two blank columns between adjacent characters.

Example: The command sequence `ERASE DRAW PRLCD` clears *PICT*, plots the current equation, then prints the graphics display.

Related Commands: CR, DELAY, OLDPRT, PRST, PRSTC, PRVAR, PR1

Level 1	→	Level 1
"prompt"	→	*obj*

Use: Displays the contents of "*prompt*" in the status area and halts program execution.

Affected by Flags: None.

Remarks: PROMPT is equivalent to `1 DISP 1 FREEZE HALT`.

See "The PROMPT Command" in chapter 29 of the *HP 48 Owner's Manual* for a programming example using PROMPT.

Related Commands: CONT, DISP, FREEZE, HALT, INPUT

... Level 1	→	**... Level 1**
... obj	→	*... obj*

Use: Prints all objects in the stack, starting with the object in the highest level.

Affected by Flags: Double-Spaced Printing (-37), Printing Device (-34), I/O Device (-33), Linefeed (-38).

If flag -34 is set (printer output directed to the serial port), flag -33 must be clear.

When flag -38 is set, linefeeds are *not* added at the end of each print line. Generally, flag -38 should be clear for execution of PRST.

Remarks: Objects are printed in multi-line printer format. See the PR1 keyword entry for a description of multi-line printer format.

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRSTC, PRVAR, PR1

... Level 1	→	... Level 1
... obj	→	*... obj*

Use: Prints in compact form all objects in the stack, starting with the object in the highest level.

Affected by Flags: Double-Spaced Printing (-37), Printing Device (-34), I/O Device (-33), Linefeed (-38).

If flag -34 is set (printer output directed to the serial port), flag -33 must be clear.

When flag -38 is set, linefeeds are *not* added at the end of each print line. Generally, flag -38 should be clear for execution of PRSTC.

Remarks: Compact printer format is the same as compact display format: Multiline objects are truncated and appear on one line only.

Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRVAR, PR1

Level 1	→	Level 1
'name'	→	
{ $name_1$ $name_2$ ... }	→	
$:n_{port}$:*'global'*	→	

Use: Searches the current directory path or port for the specified variable(s) and prints the name and contents of each variable.

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33), Linefeed (−38).

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

When flag −38 is set, linefeeds are *not* added at the end of each print line. Generally, flag −38 should be clear for execution of PRVAR.

Remarks: Objects are printed in multi-line printer format. See the PR1 keyword entry for a description of multi-line printer format.

Related Commands: PR1, PRST, PRSTC, PRLCD, CR, DELAY, OLDPRT

Level 1	→	Level 1
obj	→	*obj*

Use: Prints *obj* in multi-line printer format.

Affected by Flags: Double-Spaced Printing (−37), Printing Device (−34), I/O Device (−33).

If flag −34 is set (printer output directed to the serial port), flag −33 must be clear.

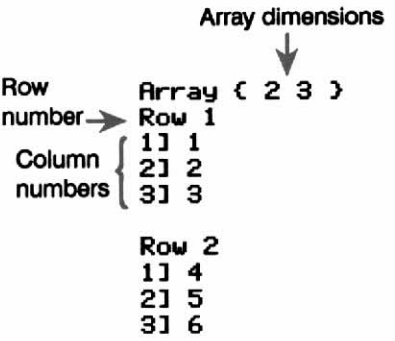
Remarks: All objects except strings are printed with their identifying delimiters. Strings are printed without the leading and trailing " delimiters.

Multiline printer format is similar to multiline display format, with the following exceptions:

- Strings and names that are more than 24 characters long are continued on the next printer line.
- The real and imaginary parts of complex numbers are printed on separate lines if they don't fit on the same line.
- Arrays are printed with a numbered heading for each row and with a column number before each element. For example, the 2 × 3 array

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

would be printed as follows:



Related Commands: CR, DELAY, OLDPRT, PRLCD, PRST, PRSTC, PRVAR

PURGE *Purge* Command

Level 1	→	Level 1
'global'	→	
{ $global_1$ ... $global_n$ }	→	
PICT	→	
:n_{port} :$name_{backup}$	→	
:n_{port} :$n_{library}$	→	

Use: Purges the named variable(s) or *empty* subdirectory(s) from the current directory.

Affected by Flags: None.

Remarks: PURGE executed in a program (or in the command line) does not save its argument for recovery by LASTARG.

To empty a named directory before purging it, use PGDIR.

To help prepare a list of variables for purging, you can use VARS.

Purging *PICT* results in replacing the current graphics object with a 0×0 graphics object.

If a list of objects (with global names, backup objects, library objects, or *PICT*) for purging contains an invalid object, then the objects preceding the invalid object are purged, and the error `Bad Argument Type` occurs.

To purge a library or backup object, tag the library number or backup name with the appropriate port number (:n_{port}), which must be 0, 1, or 2. (A library can be purged from RAM only.) For a backup object, you can replace the port number with the wildcard character `&`, in which case the HP 48 will search ports 2, 1, 0, and then main memory for the named backup object.

Library objects in RAM can be purged, while those in ROM (applicaton cards and write-protected RAM cards) cannot. A library object must be detached before it can be purged from the *HOME* directory.

Neither a library object nor a backup object can be PURGEd if it is currently "referenced" internally by stack pointers (such as an object on the stack, in a local variable, on the LAST stack, or on an internal return stack). This produces the error `Object in Use`. You can circumvent these restrictions on PURGE by using NEWOB before purging. (See NEWOB.)

Related Commands: CLEAR, CLUSR, CLVAR, NEWOB, PGDIR

PUT — *Put Element* — Command

Level 3	Level 2	Level 1	→	Level 1
[[$matrix_1$]]	$n_{position}$	z_{put}	→	[[$matrix_2$]]
[[$matrix_1$]]	{ n_{row} m_{col} }	z_{put}	→	[[$matrix_2$]]
'$name_{matrix}$'	$n_{position}$	z_{put}	→	
'$name_{matrix}$'	{ n_{row} m_{col} }	z_{put}	→	
[$vector_1$]	$n_{position}$	z_{put}	→	[$vector_2$]
[$vector_1$]	{ $n_{position}$ }	z_{put}	→	[$vector_2$]
'$name_{vector}$'	$n_{position}$	z_{put}	→	
'$name_{vector}$'	{ $n_{position}$ }	z_{put}	→	
{ $list_1$ }	$n_{position}$	obj_{put}	→	{ $list_2$ }
{ $list_1$ }	{ $n_{position}$ }	obj_{put}	→	{ $list_2$ }
'$name_{list}$'	$n_{position}$	obj_{put}	→	
'$name_{list}$'	{ $n_{position}$ }	obj_{put}	→	

Use: In the level 3 array or list, PUT replaces with z_{put} or obj_{put} the object whose position is specified in level 2, and if the array or list is unnamed, returns the new array or list.

Affected by Flags: None.

Remarks: For matrices, $n_{position}$ counts in row order.

If the argument in level 3 is a name, PUT alters the named array or list and returns nothing to the stack.

Examples: The command sequence `[[ 2 3 4 ][ 4 1 2 ]] { 1 3 } 96 PUT` returns `[[ 2 3 96 ][ 4 1 2 ]]`.

The command sequence `[[ 2 3 4 ][ 4 1 2 ]] 5 96 PUT` returns `[[ 2 3 4 ][ 4 96 2 ]]`.

The command sequence `{ A B C D E } { 3 } 'Z' PUT` returns `{ A B Z D E }`.

Related Commands: GET, GETI, PUTI

Level 3	Level 2	Level 1	→	Level 2	Level 1
[[$matrix_1$]]	$n_{position1}$	z_{put}	→	[[$matrix_2$]]	$n_{position2}$
[[$matrix_1$]]	{ n_{row} m_{col} }$_1$	z_{put}	→	[[$matrix_2$]]	{ n_{row} m_{col} }$_2$
'$name_{matrix}$'	$n_{position1}$	z_{put}	→	'$name_{matrix}$'	$n_{position2}$
'$name_{matrix}$'	{ n_{row} m_{col} }$_1$	z_{put}	→	'$name_{matrix}$'	{ n_{row} m_{col} }$_2$
[$vector_1$]	$n_{position1}$	z_{put}	→	[$vector_2$]	$n_{position2}$
[$vector_1$]	{ $n_{position1}$ }	z_{put}	→	[$vector_2$]	{ $n_{position2}$ }
'$name_{vector}$'	$n_{position}$	z_{put}	→	'$name_{vector}$'	$n_{position2}$
'$name_{vector}$'	{ $n_{position1}$ }	z_{put}	→	'$name_{vector}$'	{ $n_{position2}$ }
{ $list_1$ }	$n_{position1}$	obj_{put}	→	{ $list_2$ }	$n_{position2}$
{ $list_1$ }	{ $n_{position1}$ }	obj_{put}	→	{ $list_2$ }	{ $n_{position2}$ }
'$name_{list}$'	$n_{position1}$	obj_{put}	→	'$name_{list}$'	$n_{position2}$
'$name_{list}$'	{ $n_{position1}$ }	obj_{put}	→	'$name_{list}$'	{ $n_{position2}$ }

Use: In the level 3 array or list, replaces with z_{put} or obj_{put} the object whose position is specified in level 2, returning the new array or list *and* the next position in that array or list.

Affected by Flags: Index Wrap Indicator (−64).

The Index Wrap Indicator flag is cleared on each execution of PUTI *until* the position (index) wraps to the first position in the array or list, at which point the flag is set. The next execution of PUTI again clears the flag.

Remarks: For matrices, the position is incremented in *row* order.

Unlike PUT, PUTI returns a named array or list (to level 2). This enables a subsequent execution of PUTI at the next position of a named array or list.

Example: The following program uses PUTI and flag −64 to replace *A*, *B*, and *C* in the list with *X*.

```
« { A B C } DO 'X' PUTI UNTIL -64 FS? END »
```

Related Commands: GET, GETI, PUT

PVARS — *Port-Variables* — Command

Level 1	→	Level 2	Level 1
n_{port}	→	{ $:n_{port}:name_{backup}\cdots$ }	*memory*
n_{port}	→	{ $:n_{port}:n_{library}\cdots$ }	*memory*

Use: Returns a list of the backup objects ($:n_{port}:name$) and the library objects ($:n_{port}:n_{library}$) in the specified port. Also returns the available memory size (if RAM) or the memory type.

Affected by Flags: None.

Remarks: The port number, n_{port}, must be 0, 1, or 2.

- If n_{port} = 0, then *memory* is bytes of available main RAM.
- If the port contains independent RAM, then *memory* is bytes of available RAM in that port.
- If the port contains merged RAM, then *memory* is `"SYSRAM"`.
- If the port contains ROM, then *memory* is `"ROM"`.
- If the port is empty, then the message `Port Not Available` appears.

Related Commands: PVARS, VARS

Level 1	→	Level 1
(x, y)	→	
{ #n #m }	→	
{ }	→	

Use: Displays *PICT* with the specified coordinate at the upper left corner of the graphics display.

Affected by Flags: None.

Remarks: *PICT* must fill the entire display on execution of PVIEW. Thus, if a position other than the upper left corner of *PICT* is specified, *PICT* must be large enough to fill a rectangle that extends 131 pixels to the right and 64 pixels down.

If PVIEW is executed from a program with a coordinate argument (versus an empty list), the graphics display persists only until the keyboard is ready for input (for example, until the end of program execution). However, the FREEZE command freezes all or part of the display until a key is pressed.

If PVIEW is executed with an *empty* list argument, *PICT* is centered in the graphics display with scrolling mode activated. In this case, the graphics display persists until [ATTN] is pressed.

PVIEW does *not* activate the graphics cursor or the Graphics menu. To activate the graphics cursor and Graphics menu, execute GRAPH.

Example: The program

```
« ( # 0d # 0d ) PVIEW  7 FREEZE »
```

displays *PICT* in the graphics display with coordinates `( # 0d # 0d )` in the upper left corner of the display, then freezes the full display until a key is pressed.

Related Commands: FREEZE, GRAPH, TEXT

Level 1	→	**Level 1**
	→	

Use: Stores its name as the fifth parameter in the reserved variable Σ*PAR*, indicating that subsequent executions of LR are to use the power curve fitting model.

Affected by Flags: None.

Remarks: LINFIT is the default specification in Σ*PAR*.

Related Commands: BESTFIT, EXPFIT, LINFIT, LOGFIT, LR

Level 1	→	Level 1
{ #n #m }	→	(x, y)

Use: Converts the specified pixel coordinates to user-unit coordinates.

Affected by Flags: None.

Remarks: The user-unit coordinates are derived from the $\langle x_{min}, y_{min} \rangle$ and $\langle x_{max}, y_{max} \rangle$ parameters in the reserved variable *PPAR*. The coordinates correspond to the geometrical center of the pixel.

Related Commands: C→PX

Level 1	→	Level 1
x	→	'*a/b*'
(*x,y*)	→	'*a/b+c/d*i*'
'$symb_1$'	→	'$symb_2$'

Use: Returns a rational form of the given number.

Affected by Flags: Number Display Format (−49, −50).

Remarks: The rational result is a "best guess", since there might be more than one rational expression consistent with the given number. →Q finds a quotient of integers that agrees with the given number to the number of decimal places specified by the display format mode.

→Q also acts on numbers that are part of algebraic expressions or equations.

Example: `'Y+2.5' →Q` returns `'Y*5/2'`.

Related Commands: →Qπ, /

QUAD *Solve Quadratic Equation* **Command**

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

Use: Solves an algebraic '$symb_1$' for the variable *global*, and returns an expression '$symb_2$' representing the solution.

Affected by Flags: Principal Solution (−1).

Remarks: QUAD calculates the second-degree Taylor series approximation of '$symb_1$' to convert it to a quadratic form. The solution '$symb_2$' will be exact if '$symb_1$' is second degree or less in *global*.

Since '$symb_1$' is evaluated during execution of QUAD, any variables in '$symb_1$' other than *global* should not exist in the current directory if you want those variables to remain in the solution as formal variables.

QUAD generally does not work if the variable for which you are solving needs units to satisfy the equation.

Related Commands: COLCT, EXPAN, ISOL, SHOW

QUOTE *Quote Argument* **Function**

Level 1	→	Level 1
'symb'	→	*'symb'*
obj	→	*obj*

Use: Returns it argument unevaluated.

Affected by Flags: None.

Remarks: When an algebraic expression is evaluated, the arguments to a function in the expression are evaluated before the function. For example, when `'SIN(X)'` is evaluated, the name *X* is evaluated first, and the result is left on the stack as the argument for SIN.

This process creates a problem for functions that require symbolic arguments. For example, the function ∫ requires as one of its arguments a name specifying the variable of integration. If evaluating an integral expression caused the name to be evaluated, the result of evaluation (rather than the name itself) would be left on the stack for ∫. To avoid this problem, the HP 48 automatically (and invisibly) quotes such arguments. When the quoted argument is evaluated, the unquoted argument is returned.

If a user-defined function takes symbolic arguments, you must quote those arguments yourself with the QUOTE function, as demonstrated in the following example.

Example: The following user-defined function *ArcLen* calculates the arc length of a function:

...QUOTE

```
«
 → start end expr var
 «
  start end
  expr var ∂ SQ 1 + ∫
  var ∫
 »
»
```

ENTER ' ArcLen STO

When you use this user-defined function in an algebraic expression, you must use QUOTE to quote the symbolic arguments:

```
'ArcLen(0,π,QUOTE(SIN(X),QUOTE(X))'
```

Related Commands: APPLY, | (Where)

Level 1	→	Level 1
x	→	'a/b*π'
x	→	'a/b'
'symb$_1$'	→	'symb$_2$'
(x,y)	→	'a/b*π+c/d*π*i'
(x,y)	→	'symb'

Use: Returns a rational form of the given number *or* a rational form of the given number with π factored out, whichever yields the smaller denominator.

Affected by Flags: Number Display Format (−49, −50).

Remarks: →Qπ computes two quotients (rational expressions) and compares them: the quotient of the given number and the quotient of the given number divided by π. It returns the fraction with the smaller denominator; if the argument was divided by π, then π is a factor in the result.

The rational result is a "best guess", since there might be more than one rational expression consistent with the given number. →Qπ finds a quotient of integers that agrees with the given number to the number of decimal places specified by the display format mode.

→Qπ also acts on numbers that are part of algebraic expressions or equations.

For a complex argument, the real or imaginary part (or both) can have π as a factor.

Example: In Fix mode to four decimal places, `6.2832 →Qπ` returns `'2*π'`. In Standard mode, however, `6.2832 →Qπ` returns `3927/625`.

Related Commands: →Q, /, π

Level 1	→	Level 1
	→	

Use: Sets Radians angle mode.

Affected by Flags: None.

Remarks: RAD sets flag −17 and clears flag −18. It displays the RAD annunciator.

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

Related Commands: DEG, GRAD

Level 1	→	Level 1
	→	x_{random}

Use: Returns the next real number in a pseudo-random number sequence and updates the random number seed.

Affected by Flags: None.

Remarks: The HP 48 uses a linear congruous method and a seed value to generate a random number x, which always lies in the range $0 \leq x \leq 1$. Each succeeding execution of RAND returns a value computed from a seed based upon the previous RAND value. You can change the seed by using RDZ.

Related Commands: COMB, PERM, RDZ, !

RATIO — *Prefix Divide* — Function

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1 / z_2
[*array*]	[[*matrix*]]	→	[[*array*∗*matrix*$^{-1}$]]
[*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	→	(x/y)_unit$_x$ / unit$_y$
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'

Use: Prefix form of / (divide) generated by the EquationWriter application.

Affected by Flags: None.

Remarks: RATIO is *identical* to / (divide), except that, in *algebraic* syntax, RATIO is a *prefix* function, while /, in algebraic syntax, is an *infix* function. For example, `'RATIO(A,2)'` is equivalent to `'A/2'`.

RATIO is generated internally by the EquationWriter application when ▲ is used to start a numerator. It provides no additional functionality to / and appears externally only in the string that the EquationWriter application leaves on the stack when → ▪▪ is pressed or when the calculator runs out of memory.

Related Commands: /

Level 1	→	Level 1
	→	*obj*

Use: Returns the contents of the reserved variable *EQ* from the current directory.

Affected by Flags: None.

Remarks: To recall the contents of *EQ* from a parent directory (when *EQ* doesn't exist in the current directory) evaluate the name *EQ*.

Related Commands: STEQ

Level 1	→	Level 1
'name'	→	*obj*
PICT	→	*grob*
$:n_{port}:n_{library}$	→	*obj*
$:n_{port}:name_{backup}$	→	*obj*

Use: Returns the unevaluated contents of the specified variable or plug-in object to the stack.

Affected by Flags: None.

Remarks: RCL searches the entire current path, starting with the current directory, unless you specify a different path for your argument (*{ path name }*). The *path* is a series of names of variables specifying directories, while the *name* is the variable in the final subdirectory whose contents will be returned to the stack. The *path* subdirectory will not become the current subdirectory (unlike the case with EVAL).

To recall a library or backup object, tag the library number or backup name with the appropriate port number ($:n_{port}$), which must be 0, 1, or 2. (A library can be recalled from RAM only.) Recalling a backup object brings a copy of its *contents* to the stack, not the entire backup object.

For a backup object, you can replace the port number with the wildcard character *&*, in which case the HP 48 will search ports 2, 1, 0, and then main memory for the named backup object.

Related Commands: STO

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

Use: Recalls the alarm specified by n_{index}.

Affected by Flags: None.

Remarks: obj_{action} is the alarm execution action. If an execution action was not specified, the default entry in the list is an empty string.

x_{repeat} is the repeat interval in clock ticks, where 1 clock tick equals 1/8192 second. If a repeat interval was not specified, the default entry in the list is 0.

Related Commands: DELALARM, FINDALARM, STOALARM

	→	**Level 1**
	→	{ #n_{system} #n_{user} }

Use: Returns a list containing two 64-bit binary integers representing the states of the 64 system and user flags, respectively.

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

The current wordsize must be 64 bits (the default wordsize) to recall the states of all 64 user flags and 64 system flags. If the current wordsize is 32, for example, RCLF returns two 32-bit binary integers.

Remarks: A bit with value 1 indicates that the corresponding flag is set; a bit with value 0 indicates that the corresponding flag is clear. The rightmost (least significant) bit of #n_{system} and #n_{user} indicate the states of system flag −1 and user flag +1, respectively.

Used with STOF, RCLF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status. See "PRESERVE (Save and Restore Previous Status)" in chapter 31 of the *HP 48 Owner's Manual* for a program example using RCLF.

Related Commands: STOF

Level 1	→	Level 1
	→	{ $obj_1\ x_{key1} \ldots obj_n\ x_{key\ n}$ }
	→	{ S $obj_1\ x_{key1} \ldots obj_n\ x_{key\ n}$ }

Use: Returns the current user key assignments. This includes an `S` if the standard definitions are active (not suppressed) for those keys without user key assignments.

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

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

Related Commands: ASN, DELKEYS, STOKEYS

Level 1	→	Level 1
	→	x_{menu}

Use: Returns the menu number of the currently displayed menu.

Affected by Flags: None.

Remarks: x_{menu} has the form *mm.pp*, where *mm* is the menu number and *pp* is the page of the menu. See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers (x_{menu}).

Execution of RCLMENU when the current menu is a user-defined menu (built by TMENU) returns `0.01` (in 2 Fix mode), indicating "Last menu".

Example: If the third page of the PRG DSPL menu is currently active, RCLMENU returns `13.03` (in 2 Fix mode).

Related Commands: MENU, TMENU

Level 1	→	Level 1
	→	*obj*

Use: Returns the current statistics matrix (the contents of reserved variable Σ*DAT*) from the current directory.

Affected by Flags: None.

Remarks: To recall Σ*DAT* from a parent directory (when Σ*DAT* doesn't exist in the current directory), evaluate the name Σ*DAT*.

Related Commands: CLΣ, STOΣ, Σ+, Σ−

RCWS *Recall Wordsize* **Command**

Level 1	→	Level 1
	→	*n*

Use: Returns the current wordsize in bits (1 through 64).

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

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

Level 2	Level 1	→	Level 1
[$vector_1$]	{ $n_{elements}$ }	→	[$vector_2$]
[*vector*]	{ n_{rows} m_{cols} }	→	[[*matrix*]]
[[*matrix*]]	{ $n_{elements}$ }	→	[*vector*]
[[$matrix_1$]]	{ n_{rows} m_{cols} }	→	[[$matrix_2$]]
'global'	{ $n_{elements}$ }	→	
'global'	{ n_{rows} m_{cols} }	→	

Use: Rearranges the elements of the level 2 array according to the dimensions specified by the level 1 list.

Affected by Flags: None.

Remarks: If the list contains a single number $n_{elements}$, the result is an n-element vector. If the list contains two numbers n_{rows} and m_{cols}, the result is an $n \times m$ matrix.

Elements taken from the argument vector or matrix preserve the same row order in the result vector or matrix. If the result is dimensioned to contain fewer elements than the argument vector or matrix, excess elements from the argument vector or matrix at the end of the row order are discarded. If the result is dimensioned to contain more elements than the argument vector or matrix, the additional elements in the result at the end of the row order are filled with zeros (`(0,0)` if the argument is complex).

If the argument vector or matrix is specified by *global*, the result replaces the argument as the contents of the variable.

Examples: `[ 2 4 6 8 ] ( 2 2 ) RDM` returns
`[[ 2 4 ][ 6 8 ]].`

`[[ 2 3 4 ][ 1 6 9 ]] 8 RDM` returns
`[ 2 3 4 1 6 9 0 0 ].`

Related Commands: TRN

Level 1	→	Level 1
x_{seed}	→	

Use: Takes a real number x_{seed} as a seed for the RAND command.

Affected by Flags: None.

Remarks: If the argument is 0, a random value based on the system clock is used as the seed. After memory reset, the seed value is 0.529199358633.

Related Commands: COMB, PERM, RAND, !

Level 1	→	Level 1
x	→	*x*
(*x*, *y*)	→	*x*
[*R-array*]	→	[*R-array*]
[*C-array*]	→	[*R-array*]
'*symb*'	→	'RE(*symb*)'

Use: Returns the real part of its (complex number or array) argument.

Affected by Flags: Numerical Results (−3).

Remarks: If the argument is a vector or matrix, RE returns a real array, the elements of which are equal to the real parts of the corresponding elements of the argument array.

Related Commands: C→R, IM, R→C

Level 1	→	Level 1
'name'	→	
"name"	→	

Use: Prepares the HP 48 to receive a file from another Kermit device. The received file will be stored in a variable with the name specified.

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

The proper setting of flag −35 is automatically established if both devices are HP 48s.

Remarks: RECN is identical to RECV except that the name under which the received data will be stored is specified in the stack. A receiving device must execute RECN (or RECV) if it is in Local mode, but not if it is in Server mode.

Related Commands: FINISH, KGET, RECV, SEND, SERVER

Level 1 →	Level 1
→	

Use: Prepares the HP 48 to receive a named file from another Kermit device. The received file will be stored in a variable with the name specified by the sender.

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

The proper setting of flag -35 is automatically established if both devices are HP 48s.

Remarks: A receiving device must execute RECV (or RECN) if it is in Local mode. (If it is in Server mode, it will automatically receive any file sent to it.)

Related Commands: FINISH, KGET, RECN, SEND, SERVER

See the WHILE keyword entry for syntax information.

Use: Starts loop clause in WHILE...REPEAT...END indefinite loop structure. See the WHILE keyword entry for more information.

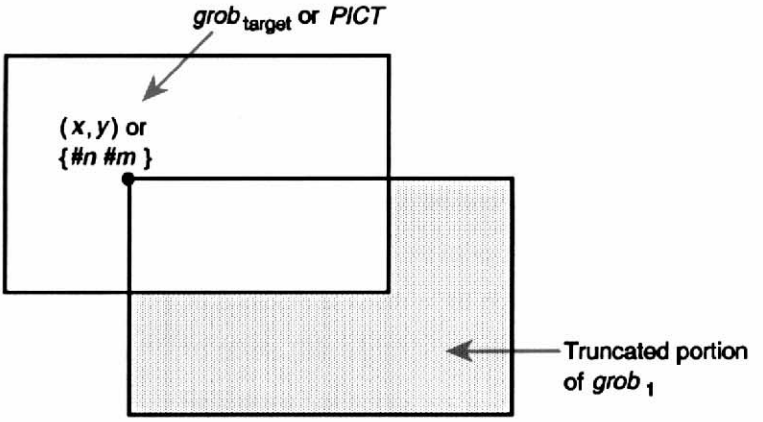
Related Commands WHILE, END

Level 3	Level 2	Level 1	→	Level 1
{ $list_{target}$ }	$n_{position}$	{ $list_1$ }	→	{ $list_{result}$ }
"$string_{target}$"	$n_{position}$	"$string_1$"	→	"$string_{result}$"
$grob_{target}$	{#n #m}	$grob_1$	→	$grob_{result}$
$grob_{target}$	(x,y)	$grob_1$	→	$grob_{result}$
PICT	{#n #m}	$grob_1$	→	
PICT	(x,y)	$grob_1$	→	

Use: Starting at the position specified in level 2, REPL replaces a portion of the target object (level 3) with the level 1 object.

Affected by Flags: None.

Remarks: For graphics objects, the upper left corner of $grob_1$ is positioned at the user-unit or pixel coordinates ‹x,y› or ‹ #n #m ›. From there, it overwrites a rectangular portion of $grob_{target}$ or *PICT*. If $grob_1$ extends past $grob_{target}$ or *PICT* in either direction, it is truncated in that direction. If the If the specified coordinate is not on the target graphics object, the target graphics object will not change.



Examples: The command sequence `{ A B C D E } 2 { F G }` REPL returns `{ A F G D E }`.

The command sequence `"ABCDE" 5 "FG"` REPL returns `"ABCDFG"`.

The command sequence

```
ERASE PICT (0,0) # 5d # 5d BLANK NEG REPL
```

replaces a portion of *PICT* with a 5 by 5 graphics object, each of whose pixels is on (dark), and whose upper left corner is positioned at `(0,0)` in *PICT*.

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

Level 1	→	Level 1
$n_{interval}$	→	
$\#n_{interval}$	→	

Use: Specifies the *resolution* of mathematical and statistical plots, where resolution is the interval between values of the independent variable used to generate the plot.

Affected by Flags: None.

Remarks: For all plot types, a real number $n_{interval}$ specifies the interval in user units. For plot types FUNCTION, CONIC, and TRUTH, a binary integer $\#n_{interval}$ specifies the interval in pixels. For plot types POLAR and PARAMETRIC, a binary integer argument does not apply. For HISTOGRAM plot type, $\#n_{interval}$ specifies the bin width. For BAR plot type, $\#n_{interval}$ specifies the bar width.

The resolution is stored as the fourth item in *PPAR* with default value 0. The interpretation of the default value is summarized in the following table.

Plot Type	Default Interval
FUNCTION, CONIC, and TRUTH	1 pixel. (A point is plotted in every column of pixels).
POLAR	2°, 2 grads, or $\pi/90$ radians.
PARAMETRIC	[independent variable range (in user units)]/130
BAR	10 pixels (Bar width = 10 pixel columns).
HISTOGRAM	10 pixels (Bin width = 10 pixel columns).
SCATTER	RES does not apply.

Related Commands BAR, CONIC, FUNCTION, HISTOGRAM, PARAMETRIC, POLAR, TRUTH

Level 1	→	Level 1
:n_{port} :$name_{backup}$	→	
backup	→	

Use: Replaces the current *HOME* directory with the specified backup copy (:n_{port}:$name_{backup}$) previously created by ARCHIVE.

Affected by Flags: None.

Remarks: The specified port number must be 0, 1, or 2. Ports 1 and 2 must be configured as independent RAM. (See FREE.)

To restore a *HOME* directory that was saved on a remote system using :IO:*name* ARCHIVE, put the backup object itself on the stack and then RESTORE it.

Example: To restore a *HOME* directory that was saved as the file *AUG1* on a remote system, first execute 'AUG1' SEND on the remote system, then execute the following on the HP 48:

```
RECV 'AUG1' RCL RESTORE
```

Related Commands: ARCHIVE

Level 1	→	Level 1
#n_1	→	#n_2

Use: Rotates a binary integer one bit to the left.

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

Remarks: The leftmost bit of #n_1 becomes the rightmost bit of #n_2.

Related Commands: RLB, RR, RRB

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Rotates a binary integer one byte to the left.

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

Remarks: The leftmost byte of $\#n_1$ becomes the rightmost byte of $\#n_2$. RLB is equivalent to doing RL eight times.

Related Commands: RL, RR, RRB

RND — *Round* — Function

Level 2	Level 1	→	Level 1
z_1	n_{round}	→	z_2
z	'$symb_{round}$'	→	'RND(z, $symb_{round}$)'
'$symb_1$'	n_{round}	→	'RND($symb_1$, n_{round})'
'$symb_1$'	'$symb_{round}$'	→	'RND($symb_1$, $symb_{round}$)'
[$array_1$]	n_{round}	→	[$array_2$]
x_unit	n_{round}	→	*y_unit*
x_unit	'$symb_{round}$'	→	'RND(*x_unit*, $symb_{round}$)'

Use: Rounds the level 2 object as specified in level 1.

Affected by Flags: Numerical Results (−3).

Remarks: If n_{round} is:

- 0 through 11, the level 2 argument is rounded to n decimal places.
- −1 through −11, the level 2 argument is rounded to n significant digits.
- 12, the level 2 argument is rounded to the current display format.

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

Examples: `(4.5792,8.1275) 2 RND` returns `(4.58,8.13)`.

`[ 2.34907 3.96351 2.73453 ] -2 RND` returns `[ 2.3 4 2.7 ]`.

Related Commands: TRNC

Level 1	→	Level 1
[*array*]	→	$x_{\text{row norm}}$

Use: Returns the row norm (infinity norm) of its argument array.

Affected by Flags: None.

Remarks: The row norm is the maximum value (over all rows) of the sums of the absolute values of all elements in a row. For a vector, the row norm is the largest absolute value of any of the elements.

Related Commands: DET, CNRM, CROSS, DOT

Level n+1...Level 2	Level 1	→	Level n...Level 2	Level 1
$obj_1 \ldots obj_n$	*n*	→	$obj_2 \ldots obj_n$	obj_1

Use: Takes an integer *n* from stack level 1 and "rolls up" (out the top and in the bottom) the first *n* objects remaining on the stack.

Affected by Flags: None.

Remarks: `3 ROLL` is equivalent to ROT.

Related Commands: OVER, PICK, ROLLD, ROT, SWAP

ROLLD *Roll Down* **Command**

Level n+1...Level 2	Level 1	→	Level n	Level n-1...Level 1
$obj_1 \ldots obj_n$	n	→	obj_n	$obj_1 \ldots obj_{n-1}$

Use: Takes an integer *n* from the stack and "rolls down" (out the bottom and in the top) the first *n* objects remaining on the stack.

Affected by Flags: None.

Related Commands: OVER, PICK, ROLL, ROT, SWAP

ROOT *Root-Finder* **Command**

Level 3	Level 2	Level 1	→	Level 1
« *program* »	'*global*'	*guess*	→	x_{root}
« *program* »	'*global*'	{ *guesses* }	→	x_{root}
'*symb*'	'*global*'	*guess*	→	x_{root}
'*symb*'	'*global*'	{ *guesses* }	→	x_{root}

Use: Returns a real number x_{root} that is a value of the level 2 variable for which the program or algebraic most nearly evaluates to zero or a local extremum.

Affected by Flags: None.

Remarks: ROOT is the programmable form of the HP Solve application.

ROOT produces an error if it cannot find a solution, returning the message `Bad Guess(es)` if one or more of the guesses lie outside the domain of the equation, or returns the message `Constant?` if the equation returns the same value at every sample point. ROOT does *not* return interpretive messages when a root is found.

ROT *Rotate Objects* **Command**

Level 3	Level 2	Level 1	→	Level 3	Level 2	Level 1
obj_1	obj_2	obj_3	→	obj_2	obj_3	obj_1

Use: Rotates the first three objects on the stack, bringing the object in stack level 3 to level 1.

Affected by Flags: None.

Remarks: ROT is equivalent to `3 ROLL`.

Related Commands: OVER, PICK, ROLL, ROLLD, SWAP

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Rotates a binary integer one bit to the right.

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

Remarks: The rightmost bit of $\#n_1$ becomes the leftmost bit of $\#n_2$.

Related Commands: RL, RLB, RRB

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Rotates a binary integer one byte to the right.

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

Remarks: The rightmost byte of $\#n_1$ becomes the leftmost byte of $\#n_2$. RRB is equivalent to doing RR eight times.

Related Commands: RL, RLB, RR

Level 3	Level 2	Level 1	→	Level 1
[*vector* **B**]	[[*matrix* **A**]]	[*vector* **Z**]	→	[*vector* **B** − **AZ**]
[[*matrix* **B**]]	[[*matrix* **A**]]	[*matrix* **Z**]]	→	[[*matrix* **B** − **AZ**]]

Use: Computes the *residual* **B** − **AZ** of three arrays **B**, **A**, and **Z**.

Affected by Flags: None.

Remarks: **A**, **B**, and **Z** are restricted as follows:

- **A** must be a matrix.
- The number of columns of **A** must equal the number of elements of **Z** if **Z** is a vector, or the number of rows of **Z** if **Z** is a matrix.
- The number of rows of **A** must equal the number of elements of **B** if **B** is a vector, or the number of rows of **B** if **B** is a matrix.
- **B** and **Z** must both be vectors or both be matrices.
- **B** and **Z** must have the same number of columns, if they are matrices.

RSD is typically used for computing a correction to **Z**, where **Z** has been obtained as an approximation to the solution **X** to the system of equations **AX** = **B**. Refer to "Improving the Accuracy of System Solutions (the RSD Command)" in chapter 20 of the *HP 48 Owner's Manual* for additional information on the use of RSD with systems of equations.

Level 1	→	Level 1
n	→	#*n*

Use: Converts a positive real integer to its binary integer equivalent.

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

Remarks: For any value of $n \leq$ `0`, the result is `# 0`. For any value of $n \geq$ `1.84467440738E19` (base 10), the result is `# FFFFFFFFFFFFFFFF` (base 16).

Related Commands: B→R

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

Use: Combines two real numbers (or two real arrays) into a single complex number (or complex array).

Affected by Flags: None.

Remarks: The level-2 argument represents the real element(s) of the complex result. The level-1 argument represents the imaginary element(s) of the complex result.

Array arguments must have the same dimensions.

Related Commands: C→R, RE, IM

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

Use: Converts a real number expressed in radians to its equivalent in degrees.

Affected by Flags: Numerical Results (−3).

Remarks: This function operates independently of the angle mode.

Related Commands: D→R

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	0/1

Use: Compares obj_1 and obj_2, returning a true result (1) if they are identical, and a false result (0) if they are not.

Affected by Flags: None.

Remarks: SAME is identical in effect to == for all object types except algebraics and names. (For algebraics and names, == returns an expression that can be evaluated to produce a test result based on numerical values.)

Examples: `« A B » (4,5) SAME` returns `0`.

`« A B » « B A » SAME` returns `0`.

`"CATS" "CATS" SAME` returns `1`.

Related Commands: TYPE, ==

Level 1	→	Level 1
	→	

Use: Causes an interruption in serial transmission or reception.

Affected by Flags: I/O Device (-33).

Remarks: SBRK is typically used when there is a problem in a serial data transmission.

Related Commands: BUFLEN, SRECV, STIME, XMIT

Level 2	Level 1	→	Level 1
x_{scale}	y_{scale}	→	

Use: Adjusts the first two parameters in *PPAR*, $\langle x_{min}, y_{min} \rangle$ and $\langle x_{max}, y_{max} \rangle$, so that x_{scale} and y_{scale} are the new plot horizontal and vertical scales.

Affected by Flags: None.

Remarks: The scale in either direction is the number of user units per tic mark. The default scale in both directions is 1 user unit per tic mark.

Related Commands: AUTO, CENTR, *H, *W

SCATRPLOT *Draw Scatter Plot* **Command**

Level 1	→	Level 1
	→	

Use: Draws a scatterplot of (x, y) data points from the specified columns of the current statistics matrix (reserved variable ΣDAT).

Affected by Flags: None.

Remarks: The data columns to be plotted are specified by XCOL and YCOL and are stored as the first two parameters in the reserved variable ΣPAR. If no data columns are specified, columns 1 (independent) and 2 (dependent) are selected by default. The y-axis is autoscaled and the plot type is set to SCATTER.

When SCATRPLOT is executed from a program, the graphics display, which shows the resultant plot, does not persist unless GRAPH or PVIEW is subsequently executed.

If GRAPH *is* subsequently executed, pressing FCN in the Graphics environment draws a line to fit the data using the currently specified statistical model.

Example: Alternatively, the following program could be used to plot a scatter plot of the data in columns 3 and 4 of ΣDAT, draw a best fit line, and display the plot:

```
« 3 XCOL 4 YCOL SCATRPLOT BESTFIT ΣLINE STEQ FUNCTION
        DRAW ( # 0d # 0d ) PVIEW 7 FREEZE »
```

Related Commands: BARPLOT, GRAPH, HISTPLOT, PVIEW, SCLΣ, XCOL, YCOL

Level 1	→	Level 1
	→	

Use: Sets the plot type to SCATTER.

Affected by Flags: None.

Remarks: When the plot type is SCATTER, the DRAW command plots points by obtaining x and y coordinates from two columns of the current statistics matrix (reserved variable ΣDAT). The columns are specified by the first and second parameters in the reserved variable ΣPAR (using the XCOL and YCOL commands). The plotting parameters are specified in the reserved variable *PPAR*, which has the form:

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

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

- (x_{min}, y_{min}) is a complex number specifying the lower left corner of *PICT* (the lower left corner of the display range). The default value is `(-6.5,-3.1)`.
- (x_{max}, y_{max}) is a complex number specifying the upper right corner of *PICT* (the upper right corner of the display range). The default value is `(6.5,3.2)`.
- *indep* is a name specifying a label for the horizontal axis. The default value of *indep* is *X*.
- *res* is not used when the plot type is SCATTER.
- *axes* is a complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.
- *ptype* is a command name specifying the plot type. Executing the command SCATTER places the command name SCATTER in *PPAR*.
- *depend* is a name specifying a label for the vertical axis. The default value is *Y*.

...SCATTER

Related Commands: BAR, CONIC, FUNCTION, HISTOGRAM, PARAMETRIC, POLAR, TRUTH

Level 1	→	Level 1
n	→	

Use: Sets the number display format to Scientific mode, which displays one digit to the left of the radix mark and *n* significant digits to the right.

Affected by Flags: None.

Remarks: Scientific mode is equivalent to scientific notation using $n + 1$ significant digits, where $0 \leq n \leq 11$. (Values for *n* outside this range are rounded up or down.) A number is displayed or printed as:

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

where the mantissa is of the form *n*.(*n*...) (with from zero to 11 decimal places) and the exponent has one to three digits.

Example: The number 103.6 in Scientific mode to four decimal places appears as `1.0360E2`.

Related Commands: ENG, FIX, STD

Level 1	→	Level 1
	→	

Use: Adjusts (x_{min}, y_{min}) and (x_{max}, y_{max}) in *PPAR* so that a subsequent scatter plot exactly fills *PICT*.

Affected by Flags: None.

Remarks: When the plot type is SCATTER, the command AUTO incorporates the functionality of SCLΣ. In addition, the command SCATRPLOT automatically executes AUTO to achieve the same result. SCLΣ is included in the HP 48 for compatibility with the HP 28. SCLΣ is not included in a menu—it must be typed in.

Related Commands: AUTO, SCATRPLOT

SCONJ *Store Conjugate* **Command**

Level 1	→	Level 1
'name'	→	

Use: Conjugates the contents of the named variable.

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, SNEG, SINV

Level 1	→	Level 1
	→	x_{sdev}
	→	$[\, x_{sdev1}\ x_{sdev2} \cdots x_{sdev\ m} \,]$

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

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 from the formula:

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

where x_i is the *i*th 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: BINS, MAXΣ, MEAN, MINΣ, TOT, VAR

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

Use: Sends a copy of the named object(s) to another Kermit device.

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

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

To rename an object when you send 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: FINISH, KGET, RECN, RECV, SERVER

SERVER *Server Mode* **Command**

Level 1	→	Level 1
	→	

Use: Selects Kermit Server mode for the HP 48.

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) is passive, merely processing requests sent to it by the local Kermit. It receives data in response to SEND, it transmits data in response to KGET, and it terminates Server mode in response to FINISH.

Server mode supports Kermit Host Command packets. This allows you, for instance, to use a personal computer's keyboard and display to type into the HP 48's command line. (This is especially convenient while testing programs.) To do so, first set up the HP 48 for data transfer, as described in "Transferring Data" in chapter 33 of the *HP 48 Owner's Manual*, under "Before beginning the transfer". Then:

1. Set the HP 48 to Server mode.
2. On your PC, type `REMOTE HOST` followed by whatever you'd like to type into the HP 48 command line (up to 89 characters). Press [Return] to transmit and execute the commands.
3. The HP 48 executes the transmitted commands and then sends back to the PC's display the resulting contents of the stack as the HP 48 would normally display them.

Note that it is helpful if a program written on a PC for the HP 48 has certain header information that the HP 48 would automatically include if the program originated there. See the discussion of ASCII mode under "ASCII and Binary Transmission Modes" in chapter 33 of the *HP 48 Owner's Manual.*

Related Commands: FINISH, KGET, RECN, RECV, SEND

Level 1	→	Level 1
$n_{\text{flag number}}$	→	

Use: Sets the user or system flag specified by $n_{\text{flag number}}$.

Affected by Flags: None.

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

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

Level 2	Level 1	→	Level 1
'*symb*$_1$'	'*name*'	→	'*symb*$_2$'
'*symb*$_1$'	{ *name*$_1$ *name*$_2$ ... }	→	'*symb*$_2$'

Use: Returns '*symb*$_2$', which is equivalent to '*symb*$_1$' except that all implicit references to a variable *name* are made explicit.

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

SIGN *Sign* Function

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

Use: For real numbers, returns the sign of its argument. For unit objects, returns the sign of the number part of the unit object. For complex numbers, returns the unit vector in the direction of its argument.

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

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

Related Commands: ABS, ARG, MANT, XPON

Level 1	→	Level 1
z	→	sin *z*
'*symb*'	→	'SIN(*symb*)'
x_unit$_{angular}$	→	sin (*x_unit*$_{angular}$)

Use: Returns the sine of the argument.

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

Remarks: For real arguments, the current angle mode determines the number's interpretation as an angle, unless the 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**

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

Use: Returns the hyperbolic sine of the argument.

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

Level 1	→	Level 1
'name'	→	

Use: Replaces the contents of the named variable with its inverse.

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, SNEG, SCONJ

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*

Use: 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 an algebraic object, or the dimensions of a graphics object.

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 other object type *besides those listed above* returns a value of 1 to level 1.

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

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Shifts a binary integer one bit to the left.

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

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Shifts a binary integer one byte to the left.

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 (SL eight times), truncated to the current wordsize.

Related Commands: ASR, SL, SR, SRB

Level 1	→	Level 1
'name'	→	

Use: Replaces the contents of the named variable with its negative.

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, SINV, SCONJ

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

Use: Returns the square of the argument.

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 (real or complex).

Related Commands: √, ^

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Shifts a binary integer one bit to the right.

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

Level 1	→	Level 1
$\#n_1$	→	$\#n_2$

Use: Shifts a binary integer one byte to the right.

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 (SR eight times).

Related Commands: ASR, SL, SLB, SR

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

Use: Reads up to *n* characters from the serial input buffer and returns them as a string to level 2. Level 1 contains an indicator of the success of 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), then 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, then SRECV will not have to wait for more characters to be received—it simply returns all the characters already in the input buffer.

The value returned to level 1 is `1` if no error was detected when reading the input buffer; it is `0` if an error was detected. The only errors that SRECV can detect are: framing errors and UART overruns (both causing `"Receive Error"` in ERRM), input-buffer overflows (causing `"Receive Buffer Overflow"` in ERRM), and parity errors (causing `"Parity Error"` in ERRM).

Parity errors do no stop data flow into the input buffer. However, if a parity error occurs, SRECV reads only the good data and the first "bad" character; that is, it 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 three 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, the point at which SRECV actually detects and clears these errors is 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, an input-buffer overflow cannot be detected by an SRECV after a BUFLEN unless more characters were received after BUFLEN was executed (causing the input buffer to overflow again). Framing and UART overrun errors cleared by BUFLEN cannot be detected at all by SRECV. If you need to know where the data error occurred, then save the number of characters returned by BUFLEN (which tells you the number of "good" characters received), because as soon as the error is cleared, new characters can enter the input buffer.

Example: If ten good bytes were received followed by a framing error, then an SRECV command told to read ten 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, SBRK, STIME, XMIT

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

Use: Begins START...NEXT and START...STEP definite loop structures.

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:

  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 lets you specify an increment value other than 1. The syntax is:

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

Related Commands: FOR, NEXT, STEP

Level 1	→	Level 1
	→	

Use: Sets the number display format to Standard mode.

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 in the fractional part 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, integer numbers $<10^3$ 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}	.000000000001	Yes
1.2×10^{-11}	1.23E-11	No
12.345	12.345	Yes

Related Commands: FIX, SCI, ENG

See the FOR and START keyword entries for syntax information.

Use: Defines increment value and ends definite loop structure. See the FOR and START keyword entries for more information.

Related Commands FOR, NEXT, START

Level 1	→	Level 1
obj	→	

Use: Stores an object from the stack in the reserved variable *EQ* in the current directory.

Affected by Flags: None.

Related Commands: RCEQ

Level 1	→	Level 1
$x_{seconds}$	→	
0	→	

Use: Specifies the period that SRECV (serial reception) and XMIT (serial transmission) wait before timing out.

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, SBRK, SRECV, XMIT

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

Use: Stores the level-2 object into the level-1 variable or object.

Affected by Flags: None.

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

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

To store backup objects and library objects, you must specify a port number. A port number must be 0, 1, or 2. Ports 1 and 2 must be configured as independent RAM, since backup and library objects can be stored in independent RAM only. (See FREE.)

To use a library object, it must be in a port and it must be attached. A library object from an application card (ROM) is automatically in a port (1 or 2), 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", you must turn the calculator off and then on. (See ATTACH in this dictionary.) This action (STOing a library object, then turning the calculator off and on) also causes the calculator to perform a *system halt*, which clears the stack, clears the LAST stack, clears local variables, and returns the MATH menu to the display.

You can also use STO to replace just 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(3)' identifies the third element in a list or vector *A*. You can store a value of 5 in *A* by executing

```
5 'A(3)' STO
```

Similarly, 'A(3,5)' would identify the element in the third row and fifth column of matrix *A*.

Related Commands: DEFINE, RCL, →

STOALARM *Store Alarm* **Command**

Level 2	Level 1	→	Level 1
	time	→	n_{index}
	{ *date time* }	→	n_{index}
	{ *date time* obj_{action} }	→	n_{index}
	{ *date time* obj_{action} x_{repeat} }	→	n_{index}

Use: Stores the alarm from level 1 in the system alarm list and returns its alarm index to level 1.

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

Remarks: If the argument for STOALARM is a real number *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,

```
« 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

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

Use: Sets the states of the system flags, or the states of the system and user flags.

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 $\#n_{system}$, 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 $\#n_{system}$ and $\#n_{user}$ correspond to the states of system flag −1 and user flag +1, respectively. If $\#n_{system}$ or $\#n_{user}$ contain fewer than 64 bits, the unspecified most significant bits are taken to have value 0.

Used with RCLF, STOF lets a program that alters the state of a flag or flags during program execution preserve the pre-program-execution flag status. See "PRESERVE (Save and Restore Previous Status)" in chapter 31 of the *HP 48 Owner's Manual* for a program example using STOF.

Related Commands: RCLF

Level 1	→	Level 1
{ $obj_1\ x_{key} \ldots obj_n\ x_{key\ n}$ }	→	
{ S $obj_1\ x_{key} \ldots obj_n\ x_{key\ n}$ }	→	
'S'	→	

Use: Defines multiple keys on the user keyboard by assigning the given objects to the specified keys (specified as *rc.p*).

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

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

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'`, then the specified key is restored to its *standard key* assignment on the user keyboard.

Related Commands: ASN, DELKEYS, RCLKEYS

STO+ *Store Plus* **Command**

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

Use: Adds a number or other object to the contents of the named variable.

Affected by Flags: None.

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

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/, +

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

Use: Calculates the difference between a number (or other object) and the contents of the named variable, returning the new value to the named variable.

Affected by Flags: None.

Remarks: The new object of the named variable is the difference between the level-2 object and the level-1 object.

The object on the stack and the object in the variable must be suitable for subtraction with each other. You can subtract any combination of objects suitable for stack subtraction (see −).

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/, −

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

Use: Multiplies the contents of the named variable by a number or other object.

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/, *

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

Use: Calculates the quotient of a number (or other object) and the contents of the named variable, returning the new value to the named variable.

Affected by Flags: None.

Remarks: The new object of the named variable is the quotient of 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. In particular, if both objects are arrays, the divisor (level 1) must be a square matrix, and the dividend (level 2) must have the same number of columns as the divisor.

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

Related Commands: STO+, STO−, STO*, /

STOΣ *Store Sigma* **Command**

Level 1	→	Level 1
obj	→	

Use: Stores *obj* in the reserved variable Σ*DAT*.

Affected by Flags: None.

Remarks: STOΣ accepts any object and stores it in Σ*DAT*, but, *unless* the object is a matrix or the name of a variable containing a matrix, an `Invalid Σ Data` error will occur upon subsequent execution of a statistics command.

Related Commands: CLΣ, RCLΣ, Σ+, Σ−

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

Use: Evaluates the text of a string as if the text were entered from the command line.

Affected by Flags: None.

Remarks: The command OBJ→ includes this functionality. STR→ is included for compatibility with the HP 28S. STR→ is not in a menu.

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

Level 1	→	Level 1
obj	→	*"obj"*

Use: Converts any object to string form.

Affected by Flags: Binary Integer Wordsize (−5 through −10), Binary Integer Base (−11, −12: HEX, DEC, OCT, BIN modes), Number Display Format (−49, −50: STD, FIX, SCI, ENG modes).

The full-precision internal form of the number is not necessarily represented in the result string. You can insure that →STR preserves the full precision of a number by selecting Standard number-display format or a wordsize of 64 bits, or both, prior to executing →STR.

Remarks: 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 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 object is already a string, →STR returns the string.

Example: You can use →STR to 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.

There are more examples under "Labeling Program Output" in chapter 29 of the *HP 48 Owner's Manual*.

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

Level 1	→	Level 1
n	→	
#*n*	→	

Use: Sets the current binary integer wordsize to *n* bits, where *n* is a value from 1 through 64. The default wordsize is 64.

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

Remarks: Values of *n* beyond the limits (1 and 64) are interpreted as 1 or 64.

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

Level 3	Level 2	Level 1	→	Level 1
"*string*$_{target}$"	$n_{start\ position}$	$n_{end\ position}$	→	"*string*$_{result}$"
{ *list*$_{target}$ }	$n_{start\ position}$	$n_{end\ position}$	→	{ *list*$_{result}$ }
grob$_{target}$	{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	*grob*$_{result}$
grob$_{target}$	(x_1 , y_2)	(x_2 , y_2)	→	*grob*$_{result}$
PICT	{ #n_1 #m_1 }	{ #n_2 #m_2 }	→	*grob*$_{result}$
PICT	(x_1 , y_2)	(x_2 , y_2)	→	*grob*$_{result}$

Use: Returns the portion of a string or list defined by the positions specified in levels 1 and 2, or, returns the rectangular portion of a graphics object or *PICT* defined by two corner pixel coordinates.

Affected by Flags: None.

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

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

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

SWAP *Swap Objects* Command

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

Use: Interchanges the first two objects on the stack.

Affected by Flags: None.

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

Level 1	→	Level 1
$\#n_{address}$	→	

Use: Evaluates unnamed operating-system objects by their memory addresses.

Affected by Flags: None.

Remarks: Using SYSEVAL with random addresses can corrupt memory.

Example: You can display the version letter of your HP 48 by executing `#30794h SYSEVAL`. For version A, for example, the display would be `"HPHP48-A"`.

Related Commands: EVAL

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	*y_unit*	→	*100y_unit/x_unit*
x_unit	*'symb'*	→	*'%T(x_unit, symb)'*
'symb'	*x_unit*	→	*'%T(symb, x_unit)'*

Use: Returns the percent (fraction) of the total, *x* (level 2), represented by *y* (level 1).

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*.

If you use simple temperature units, such as `x_°C`, the calculator assumes the values represent temperatures and not differences in temperature. (For *compound* temperature units, such as `x_°C/min`, the calculator assumes temperature units represent temperature differences.) For more information on using temperature units with arithmetic functions, refer to the keyword entry for +.

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

`100 100_r %T` returns `15.9154943092` (in Standard mode), because 100 radians represents about 16% of 100.

`100_K 50_K %T` returns `50`. However, `100_°C 50_°C %T` returns `86.6005627764`, the equivalent of `373.15_K 323.15_K %T`.

Related Commands: %, %CH

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

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

Affected by Flags: None.

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

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

TAN — *Tangent* — **Analytic**

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

Use: Returns the tangent of the argument.

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**

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

Use: Returns the hyperbolic tangent of the argument.

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

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

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

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

Level 1 →	Level 1
→	

Use: Displays the stack display.

Affected by Flags: None.

Remarks: TEXT lets you switch 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 Σ*DAT*). It subsequently waits for five seconds, and then switches back from the graphics display to the stack display.

Related Commands: GRAPH, PVIEW

See the IF *and* IFERR *keyword entries for syntax information.*

Use: Starts the true-clause in conditional or error-trapping structure. See the IF and IFERR keyword entries for more information.

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

TICKS *Ticks* Command

Level 1	→	Level 1
	→	$\#n_{time}$

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

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

Level 1	→	Level 1
	→	*time*

Use: Returns the system time to level 1 in the form HH.MMSSs

Affected by Flags: None.

Remarks: *time* is always returned in 24-hour format, regardless of the state of the Clock Format flag (−41).

Related Commands: DATE, TICKS, TSTR

Level 1	→	Level 1
time	→	

Use: Sets the system time to *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* must be supplied in 24-hour format, independent of the state of the Clock Format flag (−41).

Example: `13.3341 →TIME` sets the system time to 1:33:41 PM, independent of the state of flag −41.

Related Commands: CLKADJ, →DATE

TLINE *Toggle Line* **Command**

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

Use: For each pixel along the line in *PICT* defined by the specified coordinates, TLINE turns off (makes light) every on-pixel, and turns on (makes dark) every off-pixel.

Affected by Flags: None.

Example: The following program toggles on and off ten 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 YNRG
 ( # 0d # 0d ) PVIEW
 «
  1 10 START
   (1,1) (9,9) TLINE
   .25 WAIT
  NEXT
 »
»
```

Related Commands: ARC, BOX, LINE

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

Use: Displays a built-in menu, a library menu, or a user-defined menu.

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 "CST" in appendix D, "Reserved Variables," for more information about custom menus.

See the MENU entry for a list of the HP 48 built-in menus and the corresponding menu numbers (x_{menu}).

For an example using TMENU in a program, see the example emulating a built-in program under "Custom Menus in Programs" in chapter 29 of the *HP 48 Owner's Manual*.

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 the list argument.

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

Related Commands: MENU, RCLMENU

Level 1	→	Level 1
	→	x_{sum}
	→	$[\, x_{sum1}\ x_{sum2} \cdots x_{sum\ m} \,]$

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

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: BINS, MAXΣ, MINΣ, MEAN, SDEV, VAR

TRANSIO *I/O Translation* **Command**

Level 1	→	Level 1
n_{option}	→	

Use: Specifies the character-translation option.

Affected by Flags: None.

Remarks: Legal *n*-values are:

n-Value	Meaning
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).

There is a table of I/O Character Translations under "Character Translations (TRANSIO)" in chapter 33 of the *HP 48 Owner's Manual.*

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D of this manual.

Related Commands: BAUD, CKSM, PARITY

Level 1	→	Level 1
[[$matrix_1$]]	→	[[$matrix_{transpose}$]]
'name'	→	

Use: Returns the (conjugate) transpose of its argument.

Affected by Flags: None.

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

$$\mathbf{A}^T_{ij} = \begin{cases} \mathbf{A}_{ji} \text{ for real matrices.} \\ \text{CONJ } (\mathbf{A}_{ji}) \text{ for complex matrices.} \end{cases}$$

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

Level 2	Level 1	→	Level 1
z_1	$n_{truncate}$	→	z_2
z_1	'$symb_{truncate}$'	→	'TRNC(z_1, $symb_{truncate}$)'
'$symb_1$'	$n_{truncate}$	→	'TRNC($symb_1$, n_{round}
'$symb_1$'	'$symb_{truncate}$'	→	'TRNC($symb_1$, $symb_{truncate}$)'
[$array_1$]	$n_{truncate}$	→	[$array_2$]
x_unit	$n_{truncate}$	→	*y_unit*
x_unit	'$symb_{truncate}$'	→	'TRNC(*x_unit*, $symb_{truncate}$)'

Use: Truncates the level-2 object as specified in level 1.

Affected by Flags: Numerical Results (−3).

Remarks: If $n_{truncate}$ is:

- 0 through 11, the level-2 argument is truncated to *n* decimal places.
- −1 through −11, the level-2 argument is truncated to *n* significant digits.
- 12, the level 2 argument is 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: Evaluating `(4.5792,8.1275) 2 TRNC` returns `(4.57,8.12)`.

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

Related Commands: RND

Level 1	→	Level 1
	→	

Use: Sets the plot type to 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 the 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 complex number specifying the user-unit coordinates of the intersection of the horizontal and vertical axes; or a list containing such a number and two strings specifying labels for the horizontal and vertical axes. The default value is `(0,0)`.
- *ptype* is a command name specifying the plot type. Executing the command TRUTH places the command name TRUTH in *PPAR*.

- *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; it can't 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 $\langle x_{min}, y_{min} \rangle$ and $\langle x_{max}, y_{max} \rangle$ (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 non-zero, the pixel is turned on (made dark).

The example under "Truth (Relational) Plots" in chapter 19 of the *HP 48 Owner's Manual* uses the TRUTH plot type.

Related Commands: BAR, CONIC, FUNCTION, HISTOGRAM, PARAMETRIC, POLAR, SCATTER

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

Use: Returns a string derived from the level-2 date and the level-1 time.

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

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

Use: Lists all global variables in the current directory than contain objects of the specified type(s).

Affected by Flags: None.

Remarks: If there are no variables of the specified type(s) in the current directory, then TVARS returns an empty list.

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

Related Commands: PVARS, TYPE, VARS

Level 1	→	Level 1
obj	→	n_{type}

Use: Returns the type number of the given object.

Affected by Flags: None.

Remarks: The object types and their type numbers are shown in the following table:

Object-Type Numbers

Object Type	Number
User Objects:	
Real number	0
Complex number	1
Character string	2
Real array	3
Complex array	4
List	5
Global name	6
Local name	7
Program	8
Algebraic object	9
Binary integer	10
Graphics object	11
Tagged object	12
Unit object	13
XLIB name	14
Directory	15
Library	16
Backup object	17

Object-Type Numbers (Continued)

Object Type	Number
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	26-31

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: TVARS, VTYPE, SAME, = =

UBASE — *Convert to SI Base Units* — **Function**

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

Use: Converts a unit object to SI base units.

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**

Level 2	Level 1	→	Level 1
x_1 _$unit_1$	x_2 _$unit_2$	→	x_3 _$unit_2$*$unit_3$

Use: Factors the level 1 unit from the unit expression of the level 2 unit object.

Affected by Flags: None.

Remarks: UFACT is equivalent to the sequence `OBJ→ 3 ROLLD / OVER / UBASE *`.

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

Related Commands: CONVERT, UBASE, →UNIT, UVAL

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

Use: Creates a unit object from a real number and the unit part of a unit object.

Affected by Flags: None.

Remarks: →UNIT is the reverse of OBJ→ applied to a unit object. It allows you to add units to a real number, combining a number and the unit part of a unit object. The number part of the latter is ignored.

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

See the DO keyword entry for syntax information.

Use: Starts test-clause in DO...UNTIL...END indefinite loop structure. See the DO keyword entry for more information.

Related Commands DO, END

Level 1	→	Level 1
	→	

Use: Makes the parent of the current directory the new current directory.

Affected by Flags: None.

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

Related Commands: CRDIR, HOME, PATH, PGDIR

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

Use: 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.

Affected by Flags: None.

Remarks: The defining equations are:

$$\text{utpc}(n,x) = \left(\frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)}\right)\int_x^\infty t^{\frac{n}{2}-1}e^{-\frac{t}{2}}dt, \quad \text{for } x \geq 0$$

$$= 1, \quad \text{for } x < 0$$

For any value z, $\Gamma(z/2) = (z/2 - 1)!$, where ! is the HP 48 factorial command.

The value n must be a positive integer.

Related Commands: UTPF, UTPN, UTPT

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

Use: 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.

Affected by Flags: None.

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

$$\left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}} \left(\frac{\Gamma\left(\frac{n_1 + n_2}{2}\right)}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)} \right) \int_x^{\infty} t^{\frac{(n_1-2)}{2}} \left(1 + \left(\frac{n_1}{n_2}\right)t\right)^{-\frac{(n_1+n_2)}{2}} dt, \quad \text{for } x \geq 0$$

$$\text{utpf}(n_1, n_2, x) = 1, \quad \text{for } x < 0$$

For any value z, $\Gamma(z/2) = (z/2 - 1)!$, where ! is the HP 48 factorial command.

The values n_1 and n_2 must be positive integers.

Related Commands: UTPC, UTPN, UTPT

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

Use: 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.

Affected by Flags: None.

Remarks: The defining equation is:

$$\text{utpn}(m,v,x) = \left(\frac{1}{\sqrt{2\pi v}}\right)\int_x^{\infty} e^{-\frac{(t-m)^2}{2v}}\,dt, \quad \text{for all } x \text{ and } m, \text{ and for } v > 0$$

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

Related Commands: UTPC, UTPF, UTPT

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

Use: 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.

Affected by Flags: None.

Remarks: The defining equation is:

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

For any value z, $\Gamma(z/2) = (z/2 - 1)!$, where ! is the HP 48 factorial command.

The value n must be a positive integer.

Related Commands: UTPC, UTPF, UTPN

UVAL *Unit Value* **Function**

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

Use: Returns the number part of a unit object.

Affected by Flags: Numerical Results (−3).

Related Commands: CONVERT, UBASE, UFACT, →UNIT

VAR — *Variance* — Command

Level 1	→	Level 1
	→	$x_{variance}$
	→	$[\, x_{variance\,1} \cdots x_{variance\,m} \,]$

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

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 from the formula

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

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.

VAR is not included in a menu—it must be typed in.

Related Commands: TOT, MEAN, MAXΣ, MINΣ, BINS

VARS *Variables* Command

Level 1	→	Level 1
	→	{ $global_1$ ... $global_n$ }

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

Affected by Flags: None.

Related Commands: ORDER, PVARS, TVARS

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

Use: Returns the type number of the object contained in the named variable.

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

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

Use: Converts two numbers from the stack into a 2-element vector or a complex number.

Affected by Flags: Complex Mode (−19), Coordinate System (−15 and −16).

Remarks: When flag −19 is clear, *x* and *y* are converted into a 2-element *vector* according to the setting of flags −15 and −16. When flag −19 is set, *x* and *y* are converted into a *complex number* according to the setting of flags −15 and −16.

In Rectangular mode (flags −15 and −16 clear), *x* and *y* are taken as the *rectangular* components of the vector [*x y*] or complex number (*x*, *y*). In Polar mode, *x* and *y* are taken as the radius and polar angle; that is, [*x* ∠*y*] or (*x*, ∠*y*).

Examples: With flag −19 clear, and flags −15 and −16 clear, 2 3 →V2 returns [2 3].

With flag −19 set and flags −15 and −16 set (Polar/Spherical mode), 2 3 →V2 returns (2,∠3).

Related Commands: V→, →V3

Level 3	Level 2	Level 1	→	Level 1
x_1	x_2	x_3	→	$[x_1\ x_2\ x_3]$
x_1	x_θ	x_z	→	$[x_1\ \angle x_\theta\ x_z]$
x_1	x_θ	x_ϕ	→	$[x_1\ \angle x_\theta\ \angle x_\phi]$

Use: Converts three numbers from the stack into a 3-element vector.

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

Remarks: In Rectangular mode (flags −15 and −16 clear), x_1, x_2, and x_3 are taken as the *rectangular* components of the vector `[ x1 x2 x3 ]`. In Polar/Cylindrical mode, x_1, x_θ, and x_z are taken as the radius in the *xy*-plane, the angle θ in the *xy*-plane, and the distance along the *z*-axis from the *xy*-plane, respectively. In Polar/Spherical mode, x_1, x_θ, and x_ϕ are taken as the distance from the origin, the angle θ in the *xy*-plane, and the angle from the *z*-axis, respectively.

Examples: With flags −15 and −16 clear (Rectangular mode), `1 2 3 →V3` returns `[ 1 2 3 ]`.

With flag −15 clear and −16 set (Polar/Cylindrical mode), `1 2 3 →V3` returns `[ 1 ∠2 3 ]`.

With flags −15 and −16 set (Polar/Spherical mode), `1 2 3 →V3` returns `[ 1 ∠2 ∠3 ]`.

Related Commands: V→, →V2

V→ *Vector/Complex Number to Stack* **Command**

Level 1	→	Level n ... Level 3	Level 2	Level 1
$[x\,y]$	→		x	y
$[x_r \angle y_\theta]$	→		x_r	y_θ
$[x_1\,x_2\,x_3]$	→	x_1	x_2	x_3
$[x_1 \angle x_\theta\ x_z]$	→	x_1	x_θ	x_z
$[x_1 \angle x_\theta \angle x_\phi]$	→	x_1	x_θ	x_ϕ
$[x_1\,x_2 \ldots x_n]$	→	$x_1 \ldots x_{n-2}$	x_{n-1}	x_n
(x, y)	→		x	y
$(x_r, \angle y_\theta)$	→		x_r	y_θ

Use: Separates a vector or complex number into its component elements.

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 greater elements, V→ executes *independently* of the coordinate system mode—it 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 greater 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

Level 1	→	Level 1
x_{factor}	→	

Use: Multiplies the horizontal scale by x_{factor}.

Affected by Flags: None.

Remarks: Executing *W changes the x-axis display range — the x_{min} and x_{max} components of the first two complex numbers in the reserved variable *PPAR*. The plot center (the user-unit coordinate of the center pixel) is not changed.

Related Commands: AUTO, *H, XRNG

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

Use: Interrupts program execution for *x* seconds.

With argument 0, WAIT suspends program execution until a keystroke is executed, and then returns x_{key}, a real number that defines where the key is on the keyboard. Program execution is then resumed.

With argument −1, WAIT works just like with argument 0, except that the currently specified menu is also displayed.

Affected by Flags: None.

Remarks: x_{key} is a three-digit number that identifies a key's location on the keyboard. See the ASN keyword entry for a description of the format of x_{key}.

[◀], [▶], [α], [α][◀], and [α][▶] do not by themselves constitute a valid keystroke.

WAIT with argument 0 or −1 does not affect the display, so that messages persist even though the keyboard is ready for input (FREEZE is not required).

Normally, the MENU command would 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: The 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.

The 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. Execution of -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

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

Use: Starts WHILE...REPEAT...END indefinite loop structure.

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:

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 non-zero, execution continues with the loop clause; otherwise, execution resumes following END.

Related Commands: DO, END, REPEAT

Level 1	→	Level n ... Level 1
	→	"log_1" ... "log_n"

Use: Returns a string recording the date, time, and cause of each warmstart event.

Affected by Flags: Date Format (−42).

Remarks: Each string "log_n" has the form "*code–date time*". The meaning of each code is summarized in the following table.

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*). When [ON] is pressed, the log is cleared and the system warmstarts.
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 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.

(continued)

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	An anomaly was detected involving device configuration: **1.** The interrupt system detected that one of the five devices was not configured. **2.** During a warmstart, an unexpected device ID chain was encountered while attempting to configure 3 (Port1, Port2, Xtra) of the 5 devices. **3.** Same as 2), but detected during Deep Sleep wakeup.
9	Corrupt alarm list.
A	*Unused.*
B	Card module pulled (or card bounce).
C	Hardware reset (for example, an electrostatic-discharge or user reset).
D	An expected System (RPL) error handler not found in runstream.
E	Corrupt configuration table (bad checksum for table data).
F	System RAM card pulled.

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

- The system time was corrupt at the time that the stamp was recorded.
- The date and time stamp itself is corrupt (bad checksum).

Level 1	→	Level 1
	→	x_{sum}

Use: Sums the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The calculation formula is Σx_i.

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

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

Level 1	→	Level 1
	→	$x_{\text{sum of squares}}$

Use: Sums the squares of the values in the independent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The calculation formula is Σx_i^2.

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

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

Level 1	→	Level 1
x_{col}	→	

Use: Specifies the independent-variable column of the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

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

XCOL accepts a non-integer real number, storing it in Σ*PAR*, but subsequent commands that utilize the XCOL specification in Σ*PAR* will error.

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

Level 1	→	Level 2	Level 1
"string"	→		1
"string"	→	*"substring*$_{unsent}$*"*	0

Use: Sends serially the given string *without* using Kermit protocol.

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, a `1` is returned to the stack. If the transmission is not successful, XMIT returns the unsent portion of the string to level 2 and a `0` to level 1 to indicate the failure. ERRM will return 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 with a `0` in level 1 and `"Timeout"` in ERRM.

Related Commands: BUFLEN, SBRK, SRECV, STIME

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 $symb_2$'

Use: Returns the logical exclusive OR of two arguments.

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.

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

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

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

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

...XOR

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**

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

Use: Returns the exponent of its argument.

Affected by Flags: Numerical Results (−3).

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

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

Related Commands: MANT, SIGN

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

Use: Specifies the x-axis display range.

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* **Function**

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*)'

Use: Computes the *x*th root of a real number.

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.

For negative radicands ($y < 0$), the root (x) must be an integer.

Related Commands: ^

Level 1	→	Level 1
	→	x_{sum}

Use: Sums the products of each of the corresponding values in the independent- and dependent-variable columns of the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The calculation formula is $\Sigma x_i\, y_i$.

The independent-variable column is specified by XCOL and is stored as the first parameter in the reserved variable Σ*PAR*. The default independent-variable column number is 1. The dependent-variable column is specified by YCOL and is stored as the second parameter in reserved variable Σ*PAR*. The default dependent-variable column number is 2.

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

Level 1	→	Level 1
	→	x_{sum}

Use: Sums the values in the dependent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The calculation formula is Σy_i.

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

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

Level 1	→	Level 1
	→	$x_{\text{sum of squares}}$

Use: Sums the squares of the values in the dependent-variable column of the current statistical matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

Remarks: The calculation formula is Σy_i^2.

The dependent-variable column is specified by YCOL. By the default, the dependent-variable column number is 2.

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

Level 1	→	Level 1
y_{col}	→	

Use: Specifies the dependent-variable column of the current statistics matrix (reserved variable Σ*DAT*).

Affected by Flags: None.

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

YCOL accepts a non-integer real number, storing it in Σ*PAR*, but subsequent commands that utilize the YCOL specification in Σ*PAR* will error.

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

YRNG *y-Axis Display Range* **Command**

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

Use: Specifies the *y*-axis display range.

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

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1+z_2
[$array_1$]	[$array_2$]	→	[$array_1+array_2$]
z	'symb'	→	'z+(symb)'
'symb'	z	→	'symb+z'
'symb$_1$'	'symb$_2$'	→	'symb$_1$+symb$_2$'
{$list_1$}	{$list_2$}	→	{$list_1$ $list_2$}
{list}	obj	→	{list obj}
obj	{list}	→	{obj list}
"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$_3$

Use: Returns the sum of the arguments.

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 $(x, 0)$. To add two arrays, they 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 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 +, −, =, %, %CH, and %T. 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.)

To express a temperature difference as an argument, use absolute units (K or °R) or adjust the numerical value by subtracting the additive constant. For example, you could express the difference between 30 °C and 10 °C as 20 K, or you could express it by computing `20_°C 0_°C −`, which returns `-253.15_°C`.

When computing a temperature difference, you might want to convert the result to absolute units (K or °R). As shown in the paragraph above, the difference between 30 °C and 10 °C (or between 20 °C and 0 °C) can be expressed as either 20 K (absolute units) or −253.15 °C (not absolute units).

Examples: To add an increase of 13 °C to the temperature 153 °C, execute `13_K 153_°C +`, which returns `166_°C`. (Note that the result is converted to the units of the level-1 argument. The `13_K` in level 2 represents 13 degrees of *difference*.)

The sum `32_°F 0_°C +` returns `273.15 °C`, *not* 0 °C. This is because the values are added on an absolute temperature scale (273.15 K + 273.15 K = 546.3 K) and then converted back to the level-1 units. If one of these values actually represents a temperature difference, then you should add it as shown in the first example.

Related Commands: −, *, /, =

Level 2	Level 1	→	Level 1
z_1	z_2	→	z_1-z_2
[$array_1$]	[$array_2$]	→	[$array_1-array_2$]
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'

Use: Returns the difference of the arguments. The object in level 1 is subtracted from the object in level 2.

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 addition.)

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 argument. The units of the two arguments must be consistent.

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

The calculator assumes that the simple temperature units `x_°C` and `x_°F` represent thermometer temperatures when used as arguments to the functions +, −, =, %, %CH, and %T. 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.)

To express a temperature difference as an argument, use absolute units (K or °R) or adjust the numerical value by subtracting the additive constant. For example, you could express the difference between 30 °C and 10 °C as 20 K, or you could express it by computing `20_°C 0_°C −`, which returns `-253.15_°C`.

When computing a temperature difference, you might want to convert the result to absolute units (K or °R). As shown in the paragraph above, the difference between 30 °C and 10 °C (or between 20 °C and 0 °C) can be expressed as either 20 K (absolute units) or −253.15 °C (not absolute units).

Example: To subtract a difference of 13 °C from the temperature 153 °C, evaluate `-13_K 153_°C +`, which returns `140_°C`. (Note that the result is converted to the units of the level-1 argument. The `13_K` in level 2 represents 13 degrees of *difference*.)

To subtract the temperature 13 °C from the temperature 153 °C to find the difference between them, convert the result to kelvins in order to change the expression of the result from a temperature to a temperature *difference*. That is, evaluate `153_°C 13_°C −` and then press ↰ K (or execute `1_°K CONVERT`) to return `140_K`.

Related Commands: +, *, /, =

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'

Use: Returns the product of the arguments.

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 $(x_1x_2 - y_1y_2, x_1y_2 + x_2y_1)$.

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 for the purposes of matrix multiplication and the computation of 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 addition.)

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: +, −, /, =

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'

Use: Returns the quotient of the arguments (the level-2 object divided by the level-1 object).

Affected by Flags: Numerical Results (− 3).

Remarks: A real number *a* divided by a complex number (x, y) returns $(ax/(x^2 + y^2), -ay/(x^2 + y^2))$.

A complex number (x, y) in level 2 divided by a real number *a* in level 1 returns the complex number $(x/a, y/a)$.

A complex number (x_1, y_1) in level 2 divided by another complex number (x_2, y_2) in level 1 returns the complex quotient $((x_1 x_2 + y_1 y_2) / (x_2^2 + y_2^2), (y_1 x_2 - x_1 y_2) / (x_2^2 + y_2^2))$.

An array, **B**, divided by a matrix, **A**, solves the system of equations $\mathbf{AX} = \mathbf{B}$ for **X**; that is, $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$. This operation uses 16-digit internal precision, providing more precision than the calculation `INV(A)*B`. The matrix must be square and it 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 another binary number returns a binary integer that is the integer 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: `+`, `-`, `*`, `=`

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}
'*symb*'	*x_unit*	→	'(*symb*)^(*x_unit*)'
x_unit	'*symb*'	→	'(*x_unit*)^(*symb*)'

Use: Returns the value of the level-2 object raised to the power of the level-1 object.

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 the relationship

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

Related Commands: EXP, ISOL, LN, XROOT

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'

Use: Tests whether one object is less than another object.

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. Note that this means 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.

Related Commands: ≤, >, ≥, ==, ≠

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*'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \leq symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'*symb*'	→	'*x_unit* ≤ *symb*'
'*symb*'	*x_unit*	→	'*symb* ≤ *x_unit*'

Use: Tests whether one object is less than or equal to another object.

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 the same or numerically smaller (1 is less than 2). For real numbers, "less than or equal" also means the same or more negative (−2 is less than −1).

For strings, "less than or equal" means alphabetically previous or the same ("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. Note that this means 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.

Related Commands: <, >, ≥, ==, ≠

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*'

Use: Tests whether one object is greater than another object.

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. Note that this means 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.

Related Commands: <, ≤, ≥, ==, ≠

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'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \geq symb_2$'
x_unit_1	y_unit_2	→	0/1
x_unit	'symb'	→	'x_unit ≥ symb'
'symb'	x_unit	→	'symb ≥ x_unit'

Use: Tests whether one object is greater than or equal to another object.

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 greater or the same (2 is greater than or equal to 1). For real numbers, "greater than or equal to" also means less negative or the same (−1 is greater than or equal to −2).

For strings, "greater than or equal to" means alphabetically subsequent or the same ("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. Note that this means 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.

Related Commands: <, ≤, >, ==, ≠

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	→	'z=y_unit'
y_unit	x	→	'y_unit=z'
y_unit	x_unit	→	'y_unit=x_unit'
'symb'	x_unit	→	'symb=x_unit'
x_unit	'symb'	→	'x_unit=symb'

Use: Returns an equation formed of the two arguments.

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 keyword 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 calculator assumes that the simple temperature units `x_°C` and `x_°F` represent thermometer temperatures when used as arguments to the functions `+`, `-`, `=`, `%`, `%CH`, and `%T`. 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.)

To express a temperature difference as an argument, use absolute units (K or °R) or adjust the numerical value by subtracting the additive constant. For example, you could express the difference between 30 °C and 10 °C as 20 K, or you could express it by computing `20_°C 0_°C -`, which returns `-253.15_°C`.

Related Commands: DEFINE, EVAL, -

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'
'$symb_1$'	'$symb_2$'	→	'$symb_1$==$symb_2$'

Use: Tests if two objects are equal.

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 real number, so `6 (6,0) ==` returns 1.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison.

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

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'
'$symb_1$'	'$symb_2$'	→	'$symb_1 \neq symb_2$'

Use: Tests if two objects are not equal.

Affected by Flags: Numerical Results (−3).

Remarks: The function ≠ returns a false result (0) if the two objects are the same type and have the same value, or a true result (1) 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 `6 (6,0) ≠` returns `0`.

For unit objects, the two objects must be dimensionally consistent and are converted to common units for comparison.

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

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

Use: Returns the factorial *n*! of a positive integer argument *n*, or returns the gamma function Γ(*x*+1) of a non-integer argument *x*.

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

Remarks: For $x \geq 253.1190554375$ or *x* a negative integer, ! causes an Overflow exception—if flag −21 is set, the exception is treated as an error. For $x \leq -254.1082426465$, ! causes an Underflow exception—if flag −20 is set, the exception is treated as an error.

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

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

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

and defined for other values of *x* by analytic continuation.

Related Commands: COMB, PERM

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

Use: Integrates *integrand* from *lower limit* to *upper limit* with respect to the specified variable of integration.

Affected by Flags: Numerical Results (−3).

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:

- All the built-in functions whose antiderivatives are expressible in terms of other built-in functions — for example, SIN is integrable 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 are expressible in terms of other built-in functions — for example, `'SIN(X)-COS(X)'`.
- Derivatives of all the built-in functions — for example, `'INV(1+X^2)'` is integrable 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'` is integrable since `X` is a linear term. `'(X^2-6)^3+(X^2-6)^2'` is not integrable since `X^2-6` is not linear.
- Selected patterns composed of functions whose antiderivatives are expressible in terms of other built-in functions — for example, `'1/(COS(X)*SIN(X))'` returns `'LN(TAN(X))'`.

If the result:

- Is an expression with no integral sign in the result, a symbolic integration was successful.
- Still contains an integral sign, you can try rearranging the expression and evaluating again, or you can estimate the answer with numerical integration.

A successful result of symbolic integration has the form:

'*result*|*name=upper limit*−*result*|*name=lower limit*'

See the | (where) keyword 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 to level 1. In addition, the error of integration is stored in variable *IERR*.

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

```
1 2 '10*X' 'X' ∫
```

returns

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

A subsequent evaluation substitutes the limits of integration, returning 15.

The command sequence `'∫(1,2,10*X,X)' →NUM` returns the numerical approximation 15. Variable *IERR* is created and contains the error of integration `.00000000015`.

Related Commands: TAYLR, ∂, Σ

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

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

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 will have 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. For more information, see "Advanced Topic: User-Defined Derivatives" in chapter 23 of the *HP 48 Owner's Manual*.

Example: 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, ∫, Σ

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

Use: Returns *x* (level 2) percent of *y* (level 1).

Affected by Flags: Numerical Results (−3).

Remarks: If you use simple temperature units, such as *x*_°C, the calculator assumes the values represent temperatures and not differences in temperature. (For *compound* temperature units, such as *x*_°C/min, the calculator assumes temperature units represent temperature differences.) For more information on using temperature units with arithmetic functions, refer to the keyword entry for +.

Example: Keep in mind the properties of the temperature scale you use with %. Evaluating the absolute temperature `100_K 50 %` returns `50_K`. However, evaluating a Celsius temperature involves an implicit conversion by the calculator into and out of kelvins: `100_°C 50 %` returns `-86.575_°C`, the equivalent of `373.15_K 50 % 1_°C CONVERT`.

Related Commands: %CH, %T

Level 1	→	Level 1
	→	'π'
	→	3.14159265359

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

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

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

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

Level 4	Level 3	Level 2	Level 1	→	Level 1
'index'	$x_{initial}$	x_{final}	summand	→	x_{sum}
'index'	'initial'	x_{final}	summand	→	'Σ(index = initial,x_{final} ,summand)'
'index'	$x_{initial}$	'final'	summand	→	'Σ(index = $x_{initial}$,final,summand)'
'index'	'initial'	'final'	summand	→	'Σ(index = initial,final,summand)'

Use: Calculates the value of a finite series.

Affected by Flags: Numerical Results (−3).

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

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

'Σ(*index=initial,final,summand*)'

Examples: The command sequence `'N' 1 5 'A^N' Σ` when *A* is formal 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, ∫, ∂

Level m ... Level 2	Level 1	→	Level 1
	x	→	
	$[\, x_1\, x_2 \cdots x_m \,]$	→	
	$[[\, x_{11}\, x_{12} \cdots x_{1m} \,][\, x_{n1}\, x_{n2} \cdots x_{nm} \,]]$	→	
$x_1 \cdots x_{m-1}$	x_m	→	

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

Affected by Flags: None.

Remarks: For a statistics matrix with *m* columns, you can enter the argument for Σ+ in several ways:

- If you are entering one data point with a single coordinate value, the argument for Σ+ is a real number.
- If you are entering one data point with multiple coordinate values, the argument for Σ+ is a vector of *m* real coordinate values.
- If you are entering several data points, the argument for Σ+ is a matrix of *n* rows of *m* real coordinate values.

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

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

Example: The sequence `CLΣ [ 2 3 4 ] Σ+ 3 1 7 Σ+` creates the matrix `[[ 2 3 4 ][ 3 1 7 ]]` in Σ*DAT*. (LASTARG returns the *m*-element vector in either case.)

Related Commands: CLΣ, RCLΣ, STOΣ, Σ−

	→	**Level 1**
	→	x
	→	$[x_1\ x_2 \ldots x_m]$

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

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+

Level 1	→	Level 1
z	→	√z
x_unit	→	√x_*unit*$^{1/2}$
'symb'	→	'√(*symb*)'

Use: Returns the (positive) square root.

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

Remarks: For a complex number (x_1, y_1), the square root is the complex number $(x_2, y_2) = (\sqrt{r}\ \cos\theta/2, \sqrt{r}\ \sin\theta/2)$, where r = abs (x_1, y_1), and θ = arg (x_1, y_1).

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

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

'≤1*√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.

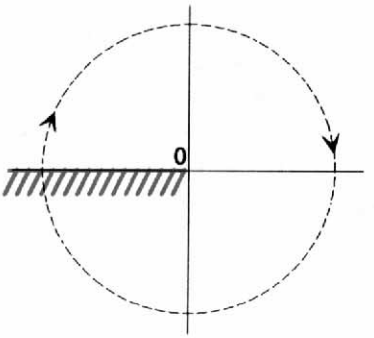
The graphs below show the domain and range of √. The graph of the domain shows where the branch cut occurs: the heavy solid line marks one side of the cut, while the feathered lines mark the other side of the cut. The graph of the range shows where each side of the cut is mapped under the function.

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

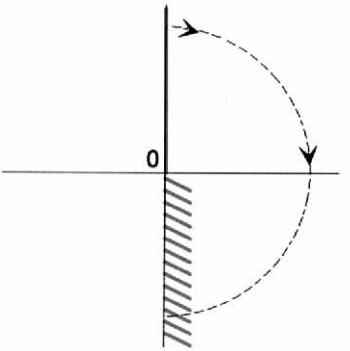
You can view these graphs with domain and range reversed to see how the domain of SQ is restricted to make an inverse *function* possible. Consider the half-plane in the lower graph as the restricted domain `Z = (x,y)`. SQ sends this domain onto the whole complex plane in the range `W = (u,v) = SQ(x,y)`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

Related Commands: SQ, ^, ISOL

Domain: Z = ⟨x, y⟩



Range: W = ⟨u, v⟩ = √⟨x, y⟩



Branch Cut for $\sqrt{z}$

	Where		Function
Level 2	**Level 1**	→	**Level 1**
$'symb_{old}'$	$\{ name_1\ 'symb_1'\ name_2\ 'symb_2' \ldots \}$	→	$'symb_{new}'$
x	$\{ name_1\ 'symb_1'\ name_2\ 'symb_2' \ldots \}$	→	x
(x, y)	$\{ name_1\ 'symb_1'\ name_2\ 'symb_2' \ldots \}$	→	(x, y)

Use: Substitutes values for names in an expression.

Affected by Flags: Numerical Results (−3).

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

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

It enables algebraics to include variable-like substitution information about names. Symbolic functions that delay name evaluation (such as $\int$ and ∂) can 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

Level n ... Level 1	→	Level 1
$obj_1 \ldots obj_n$	→	

Use: Creates local variable(s).

Affected by Flags: None.

Remarks: *Local variable structures* create one or more local variables, which exist only within the defining procedure of the local variable structure.

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 an object from the stack into each name. The resultant *local variables* exist only within the defining procedure of the local variable structure. The syntax of a local variable structure is either:

→ $name_1\ name_2 \ldots name_n$ « *program* »

or

→ $name_1\ name_2 \ldots name_n$ '*algebraic expression*'.

Example: The 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 the 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

A

Table of Error and Status Messages

In the following tables, messages are first arranged alphabetically by name and then numerically by message number.

Messages Listed Alphabetically

Message	Meaning	# (hex)
Acknowledged	Alarm acknowledged.	619
Autoscaling	Calculator is autoscaling *x*- and/or *y*- axis.	610
Awaiting Server Cmd.	Indicates Server mode active.	C0C
Bad Argument Type	One or more stack arguments were incorrect type for operation.	202
Bad Argument Value	Argument value out of operation's range.	203
Bad Guess(es)	Guess(es) supplied to HP Solve application or ROOT lie outside domain of equation.	A01

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Bad Packet Block check	Computed packet checksum doesn't match checksum in packet.	C01
Can't Edit Null Char.	Attempted to edit a string containing character "0".	102
Circular Reference	Attempted to store a variable name into itself.	129
Connecting	Indicates verifying IR or serial connection.	C0A
Constant?	HP Solve application or ROOT returned same value at every sample point of current equation.	A02
Copied to stack	→STK copied selected equation to stack.	623
Current equation:	Identifies current equation.	608
Deleting Column	MatrixWriter application is deleting a column.	504
Deleting Row	MatrixWriter application is deleting a row.	503
Directory Not Allowed	Name of existing directory variable used as argument.	12A
Directory Recursion	Attempted to store a directory into itself.	002
Empty catalog	No data in current catalog (Equation, Statistics, Alarm)	60D

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Enter alarm, press SET	Alarm entry prompt.	61A
Enter eqn, press NEW	Store new equation in *EQ*.	60A
Enter value (zoom out if >1), press ENTER	Zoom operations prompt.	622
Extremum	Result returned by HP Solve application or ROOT is an extremum rather than a root.	A06
HALT Not Allowed	A program containing HALT executed while MatrixWriter application, DRAW, or HP Solve application active.	126
I/O setup menu	Identifies I/O setup menu.	61C
Implicit () off	Implicit parentheses off.	207
Implicit () on	Implicit parentheses on.	208
Incomplete Subexpression	▶, ▼, or ENTER pressed before all function arguments supplied.	206
Inconsistent Units	Attempted unit conversion with incompatible units.	B02
Infinite Result	Math exception: Calculation such as 1/0 infinite result.	305
Inserting Column	MatrixWriter application is inserting a column.	504

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Inserting Row	MatrixWriter application is inserting a row.	503
Insufficient Memory	Not enough free memory to execute operation.	001
Insufficient Σ Data	A Statistics command was executed when Σ*DAT* did not contain enough data points for calculation.	603
Interrupted	The HP Solve application or ROOT was interrupted by [ATTN].	A03
Invalid Array Element	[ENTER] returned object of wrong type for current matrix.	502
Invalid Card Data	HP 48 does not recognize data on plug-in card.	008
Invalid Date	Date argument not real number in correct format, or was out of range.	D01
Invalid Definition	Incorrect structure of equation argument for DEFINE.	12C
Invalid Dimension	Array argument had wrong dimensions.	501

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
`Invalid EQ`	Attempted operation from GRAPHICS FCN menu when *EQ* did not contain algebraic, or, attempted DRAW with CONIC plot type when *EQ* did not contain algebraic.	607
`Invalid IOPAR`	*IOPAR* not a list, or one or more objects in list missing or invalid.	C12
`Invalid Name`	Received illegal filename, or server asked to send illegal filename.	C17
`Invalid PPAR`	*PPAR* not a list, or one or more objects in list missing or invalid.	12E
`Invalid PRTPAR`	*PRTPAR* not a list, or one or more objects in list missing or invalid.	C13
`Invalid PTYPE`	Plot type invalid for current equation.	620
`Invalid Repeat`	Alarm repeat interval out of range.	D03
`Invalid Server Cmd.`	Invalid command received while in Server mode.	C08
`Invalid Syntax`	HP 48 unable execute ENTER or STR→ due to invalid object syntax.	106

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Invalid Time	Time argument not real number in correct format, or out of range.	D02
Invalid Unit	Unit operation attempted with invalid or undefined user unit.	B01
Invalid User Function	Type or structure of object executed as user-defined function was incorrect.	103
Invalid Σ Data	Statistics command executed with invalid object stored in *ΣDAT*.	601
Invalid Σ Data LN(Neg)	Non-linear curve fit attempted when *ΣDAT* matrix contained a negative element.	605
Invalid Σ Data LN(0)	Non-linear curve fit attempted when *ΣDAT* matrix contained a 0 element.	606
Invalid ΣPAR	*ΣPAR* not list, or one or more objects in list missing or invalid.	604
LAST CMD Disabled	[LAST CMD] pressed while that recovery feature disabled.	125
LAST STACK Disabled	[LAST STACK] pressed while that recovery feature disabled.	124
LASTARG Disabled	LASTARG executed while that recovery feature disabled.	205

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Low Battery	System batteries too low to safely print or perform I/O.	C14
Memory Clear	HP 48 memory was cleared.	005
Name Conflict	Execution of | (where) attempted to assign value to variable of integration or summation index.	13C
Name the equation, press ENTER	Name equation and store it in *EQ*.	60B
Name the stat data, press ENTER	Name statistics data and store it in *ΣDAT*.	621
Negative Underflow	Math exception: Calculation returned negative, non-zero result greater than −MINR.	302
No Current Equation	SOLVR, DRAW, or RCEQ executed with nonexistent *EQ*.	104
No current equation	Plot and HP Solve application status message.	609
No Room in Port	Insufficient free memory in specified RAM port.	00B
No Room to Save Stack	Not enough free memory to save copy of the stack. LAST STACK is automatically disabled.	101

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
No Room to Show Stack	Stack objects displayed by type only due to low memory condition.	131
No stat data to plot	No data stored in Σ*DAT*.	60F
Non-Empty Directory	Attempted to purge non-empty directory.	12B
Non-Real Result	Execution of HP Solve application, ROOT, DRAW, or ∫ returned result other than real number or unit.	12F
Nonexistent Alarm	Alarm list did not contain alarm specified by alarm command.	D04
Nonexistent ΣDAT	Statistics command executed when Σ*DAT* did not exist.	602
Object Discarded	Sender sent an EOF (Z) packet with a "D" in the data field.	C0F
Object In Use	Attempted PURGE or STO into a backup object when its stored object was in use.	009
Object Not in Port	Attempted to access a nonexistent backup object or library.	00C
(OFF SCREEN)	Function value, root, extremum, or intersection was not visible in current display.	61F

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Out of Memory	One or more objects must be purged to continue calculator operation.	135
Overflow	Math exception: Calculation returned result greater in absolute value than MAXR.	303
Packet #	Indicates packet number during send or receive.	C10
Parity Error	Received bytes' parity bit doesn't match current parity setting.	C05
Port Closed	Possible I/R or serial hardware failure. Run self-test.	C09
Port Not Available	Used a port command on an empty port, or one containing ROM instead of RAM. Attempted to execute a server command that itself uses the I/O port.	00A
Positive Underflow	Math exception: Calculation returned positive, non-zero result less than MINR.	301
Power Lost	Calculator turned on following a power loss. Memory may have been corrupted.	006

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Processing Command	Indicates processing of host command packet.	C11
Protocol Error	Received a packet whose length was shorter than a null packet.	C07
	Maximum packet length parameter from other machine is illegal.	
Receive Buffer Overrun	Kermit: More than 255 bytes of retries sent before HP 48 received another packet.	C04
	SRECV: Incoming data overflowed the buffer.	
Receive Error	UART overrun or framing error.	C03
Receiving	Identifies object name while receiving.	C0E
Retry #	Indicates number of retries while retrying packet exchange.	C0B
Select a model	Select statistics curve fitting model.	614
Select plot type	Select plot type.	60C
Select repeat interval	Select alarm repeat interval.	61B

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Sending	Identifies object name while sending.	C0D
Sign Reversal	HP Solve application or ROOT unable to find point at which current equation evaluates to zero, but did find two neighboring points at which equation changed sign.	A05
Timeout	Printing to serial port: Received XOFF and timed out waiting for XON. Kermit: Timed out waiting for packet to arrive.	C02
Too Few Arguments	Command required more arguments than were available on stack.	201
Transfer Failed	10 successive attempts to receive a good packet were unsuccessful.	C06
Unable to Isolate	ISOL failed because specified name absent or contained in argument of function with no inverse.	130
Undefined Local Name	Executed or recalled local name for which corresponding local variable did not exist.	003

Messages Listed Alphabetically (continued)

Message	Meaning	# (hex)
Undefined Name	Executed or recalled global name for which corresponding variable does not exist.	204
Undefined Result	Calculation such as 0/0 generated mathematically undefined result.	304
Undefined XLIB Name	Executed an XLIB name when specified library absent.	004
Wrong Argument Count	User-defined function evaluated with an incorrect number of parenthetical arguments.	128
x and y-axis zoom.	Identifies zoom option.	627
x axis zoom.	Identifies zoom option.	625
x axis zoom w/AUTO.	Identifies zoom option.	624
y axis zoom.	Identifies zoom option.	626
ZERO	Result returned by the HP Solve application or ROOT is a root (a point at which current equation evaluates to zero).	A04
""	Identifies no execution action when EXECS pressed.	61E

Messages Listed Numerically

# (hex)	Message
	General Messages
001	Insufficient Memory
002	Directory Recursion
003	Undefined Local Name
004	Undefined XLIB Name
005	Memory Clear
006	Power Lost
008	Invalid Card Data
009	Object In use
00A	Port Not available
00B	No Room in Port
00C	Object Not in Port
101	No Room to Save Stack
102	Can't Edit Null Char.
103	Invalid User Function
104	No Current Equation
106	Invalid Syntax
124	LAST STACK Disabled
125	LAST CMD Disabled
126	HALT Not Allowed
128	Wrong Argument Count
129	Circular Reference
12A	Directory Not Allowed
12B	Non-Empty Directory
12C	Invalid Definition
12E	Invalid PPAR
12F	Non-Real Result

Messages Listed Numerically (continued)

# (hex)	Message
General Messages (continued)	
130	Unable to Isolate
131	No Room to Show Stack
Out-of-Memory Prompts	
135	Out of Memory
13C	Name Conflict
Stack Errors	
201	Too Few Arguments
202	Bad Argument Type
203	Bad Argument Value
204	Undefined Name
205	LASTARG Disabled
EquationWriter Application Messages	
206	Incomplete Subexpression
207	Implicit () off
208	Implicit () on
Floating-Point Errors	
301	Positive Underflow
302	Negative Underflow
303	Overflow
304	Undefined Result
305	Infinite Result
Array Messages	
501	Invalid Dimension
502	Invalid Array Element
503	Deleting Row
504	Deleting Column
505	Inserting Row

Messages Listed Numerically (continued)

# (hex)	Message
	Array Messages (continued)
506	Inserting Column
	Statistics Messages
601	Invalid Σ Data
602	Nonexistent ΣDAT
603	Insufficient Σ Data
604	Invalid ΣPAR
605	Invalid Σ Data LN(Neg)
606	Invalid Σ Data LN(0)
	Plot, I/O, Time and HP Solve Application Messages
607	Invalid EQ
608	Current equation:
609	No current equation.
60A	Enter eqn, press NEW
60B	Name the equation, press ENTER
60C	Select plot type
60D	Empty catalog
60F	No Statistics data to plot
610	Autoscaling
614	Select a model
619	Acknowledged
61A	Enter alarm, press SET
61B	Select repeat interval
61C	I/O setup menu
61D	Plot type:
61E	""
61F	(OFF SCREEN)
620	Invalid PTYPE
621	Name the stat data, press ENTER

Messages Listed Numerically (continued)

# (hex)	Message
	Application Messages (continued)
622	Enter value (zoom out if >1), press ENTER
623	Copied to stack
624	x axis zoom w/AUTO.
625	x axis zoom.
626	y axis zoom.
627	x and y-axis zoom.
A01	Bad Guess(es)
A02	Constant?
A03	Interrupted
A04	Zero
A05	Sign Reversal
A06	Extremum
	Unit Management
B01	Invalid Unit
B02	Inconsistent Units

Messages Listed Numerically (continued)

# (hex)	Message
	I/O and Printing
C01	Bad Packet Block check
C02	Timeout
C03	Receive Error
C04	Receive Buffer Overrun
C05	Parity Error
C06	Transfer Failed
C07	Protocol Error
C08	Invalid Server Cmd
C09	Port Closed
C0A	Connecting
C0B	Retry #
C0C	Awaiting Server Cmd.
C0D	Sending
C0E	Receiving
C0F	Object Discarded
C10	Packet #
C11	Processing Command
C12	Invalid IOPAR
C13	Invalid PRTPAR
C14	I/O: Batt Too Low
C15	Empty Stack
C17	Invalid Name
	Time Messages
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm

B

Table of Units

The following list defines the types of units available in the HP 48 for conversions and math. (Units differing only by prefix or product are not repeated. Look for m in this list, but not cm or m^2.) Combined with real numbers, units become *unit objects*.

Allowable prefixes for powers of ten are under "Unit Prefixes" in chapter 13 of the *HP 48 Owner's Manual*.

HP 48 Units

Unit (Full Name)	Value in SI Units
a (Are)	100 m^2
A (Ampere)	1 A
acre (Acre)	4046.87260987 m^2
arcmin (Minute of arc)	$4.62962962963 \times 10^{-5}$
arcs (Second of arc)	$7.71604938272 \times 10^{-7}$
atm (Atmosphere)	101325 $kg/m \cdot s^2$
au (Astronomical unit)	1495979×10^5 m
Å (Angstrom)	1×10^{-10} m
b (Barn)	1×10^{-28} m^2
bar (Bar)	100000 $kg/m \cdot s^2$
bbl (Barrel, oil)	0.158987294928 m^3
Bq (Becquerel)	1 1/s
Btu (International Table Btu)	1055.05585262 $kg \cdot m^2/s^2$

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
bu (Bushel)	0.03523907 m^3
c (Speed of light)	299792458 m/s
C (Coulomb)	1 A·s
cal (International Table calorie)	4.1868 $kg \cdot m^2/s^2$
cd (Candela)	1 cd
chain (Chain)	20.1168402337 m
Ci (Curie)	3.7×10^{10} 1/s
ct (Carat)	0.0002 kg
cu (US cup)	$2.365882365 \times 10^{-4}$ m^3
d (Day)	86400 s
dyn (Dyne)	1×10^{-5} $kg \cdot m/s^2$
erg (Erg)	1×10^{-7} $kg \cdot m^2/s^2$
eV (Electron volt)	1.60219×10^{-19} $kg \cdot m^2/s^2$
F (Farad)	1 $A^2 \cdot s^4/kg \cdot m^2$
fath (Fathom)	1.82880365761 m
fbm (Board foot)	0.002359737216 m^3
fc (Footcandle)	0.856564774909 cd/m^2
Fdy (Faraday)	96487 A·s
fermi (Fermi)	$1. \times 10^{-15}$ m
flam (Footlambert)	3.42625909964 cd/m^2
ft (International foot)	0.3048 m
ftUS (Survey foot)	0.304800609601 m
g (Gram)	0.001 kg
ga (Standard freefall)	9.80665 m/s^2
gal (US gallon)	0.003785411784 m^3
galC (Canadian gallon)	0.00454609 m^3
galUK (UK gallon)	0.004546092 m^3
gf (Gram-force)	0.00980665 $kg \cdot m/s^2$

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
grad (Grade)	0.0025
grain (Grain)	0.00006479891 kg
Gy (Gray)	1 m^2/s^2
h (Hour)	3600 s
H (Henry)	1 $kg \cdot m^2/A^2 \cdot s^2$
ha (Hectare)	10000 m^2
hp (Horsepower)	745.699871582 $kg \cdot m^2/s^3$
Hz (Hertz)	1 1/s
in (Inch)	0.0254 m
inHg (Inches of mercury)	3386.38815789 $kg/m \cdot s^2$
inH2O (Inches of water)	248.84 $kg/m \cdot s^2$
J (Joule)	1 $kg \cdot m^2/s^2$
K (Kelvin)	1 K
kip (Kilopound-force)	4448.22161526 $kg \cdot m/s^2$
knot (Knot)	0.514444444444 m/s
kph (Kilometers per hour)	0.277777777778 m/s
l (Liter)	0.001 m^3
lam (Lambert)	3183.09886184 cd/m^2
lb (Avoirdupois pound)	0.45359237 kg
lbf (Pound-force)	4.44822161526 $kg \cdot m/s^2$
lbt (Troy lb)	0.3732417 kg
lm (Lumen)	$7.9577471546 \times 10^{-2}$ cd
lx (Lux)	$7.95774715459 \times 10^{-2}$ cd/m^2
lyr (Light year)	$9.46052840488 \times 10^{15}$ m
m (Meter)	1 m
mho (Mho)	1 $A^2 \cdot s^3/kg \cdot m^2$
mi (International mile)	1609.344 m
mil (Mil)	0.0000254 m

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
min (Minute)	60 s
miUS (US statute mile)	1609.34721869 m
mmHg (Millimeter of mercury)	133.322368421 $kg/m \cdot s^2$
mol (Mole)	1 mol
mph (Miles per hour)	0.44704 m/s
m/s (Meters per second)	1 m/s
N (Newton)	1 $kg \cdot m/s^2$
nmi (Nautical mile)	1852 m
oz (Ounce)	0.028349523125 kg
ozfl (US fluid oz)	$2.95735295625 \times 10^{-5}$ m^3
ozt (Troy oz)	0.031103475 kg
ozUK (UK fluid oz)	0.000028413075 m^3
P (Poise)	0.1 $kg/m \cdot s$
Pa (Pascal)	1 $kg/m \cdot s^2$
pc (Parsec)	$3.08567818585 \times 10^{16}$ m
pdl (Poundal)	0.138254954376 $kg \cdot m/s^2$
ph (Phot)	795.774715459 cd/m^2
pk (Peck)	0.0088097675 m^3
psi (Pounds per square inch)	6894.75729317 $kg/m \cdot s^2$
pt (Pint)	0.000473176473 m^3
qt (Quart)	0.000946352946 m^3
r (Radian)	0.159154943092
R (Roentgen)	0.000258 $A \cdot s/kg$
rad (Rad)	0.01 m^2/s^2
rd (Rod)	5.02921005842 m
rem (Rem)	0.01 m^2/s^2
s (Second)	1 s
S (Siemens)	1 $A^2 \cdot s^3/kg \cdot m^2$
sb (Stilb)	10000 cd/m^2

HP 48 Units (continued)

Unit (Full Name)	Value in SI Units
slug (Slug)	14.5939029372 kg
sr (Steradian)	$7.95774715459 \times 10^{-2}$
st (Stere)	1 m^3
St (Stokes)	0.0001 m^2/s
Sv (Sievert)	1 m^2/s^2
t (Metric ton)	1000 kg
T (Tesla)	1 $kg/A \cdot s^2$
tbsp (Tablespoon)	$1.47867647813 \times 10^{-5}$ m^3
therm (EEC therm)	105506000 $kg \cdot m^2/s^2$
ton (Short ton)	907.18474 kg
tonUK (Long ton)	1016.0469088 kg
torr (Torr)	133.322368421 $kg/m \cdot s^2$
tsp (Teaspoon)	$4.92892159375 \times 10^{-6}$ m^3
u (Unified atomic mass)	1.66057×10^{-27} kg
V (Volt)	1 $kg \cdot m^2/A \cdot s^3$
W (Watt)	1 $kg \cdot m^2/s^3$
Wb (Weber)	1 $kg \cdot m^2/A \cdot s^2$
yd (International yard)	0.9144 m
yr (Year)	31556925.9747 s
° (Degree)	$2.77777777778 \times 10^{-3}$
°C (Degree Celsius)	1 K
°F (Degree Fahrenheit)	0.555555555556 K
°R (Degree Rankine)	0.555555555556 K
μ (Micron)	1×10^{-6} m
Ω (Ohm)	1 $kg \cdot m^2/A^2 \cdot s^3$

C

Table of System Flags

The following table lists the HP 48 system flags in functional groups. All flags can be set, cleared, and tested. The default state of the flags is *clear*, except for the Binary Integer Wordsize flags (flags −5 through −10).

System Flags

Flag	Description
	Symbolic Math Flags
−1	Principal Solution. *Clear*: QUAD and ISOL return a result representing all possible solutions. *Set*: QUAD and ISOL return only the principal solution.
−2	Symbolic Constants. *Clear*: Symbolic constants (e, i, π, MAXR, and MINR) retain their symbolic form when evaluated, unless the Numerical Results flag −3 is set. *Set*: Symbolic constants evaluate to numbers, regardless of the state of the Numerical Results flag −3.
−3	Numerical Results. *Clear*: Functions with symbolic arguments, including symbolic constants, evaluate to symbolic results. *Set*: Functions with symbolic arguments, including symbolic constants, evaluate to numbers.
−4	Not used.

System Flags (continued)

Flag	Description
	Binary Integer Math Flags
−5 thru −10	Binary Integer Wordsize. Combined states of flags −5 through −10 set the wordsize from 1 to 64 bits.
−11 and −12	Binary Integer Base. HEX: −11 *set*, −12 *set*. DEC: −11 *clear*, −12 *clear*. OCT: −11 *set*, −12 *clear*. BIN: −11 *clear*, −12 *set*.
−13 and −14	Not used.
	Coordinate System Flags
−15 and −16	Rectangular: −15 *clear*, −16 *clear*. Polar/Cylindrical: −15 *clear*, −16 *set*. Polar/Spherical: −15 *set*, −16 *set*.
	Angle Mode Flags
−17 and −18	Degrees: −17 *clear*, −18 *clear*. Radians: −17 *set*, −18 *clear*. Grads: −17 *clear*, −18 *set*.
	Complex Mode Flag
−19	*Clear*:→V2 and [→] [2D] create a 2-dimensional vector from 2 real numbers. *Set*:→V2 and [→] [2D] create a complex number from 2 real numbers.

System Flags (continued)

Flag	Description
	Math Exception-Handling Flags
−20	Underflow Exception. *Clear*: Underflow exception returns 0. *Set*: Underflow exception treated as an error.
−21	Overflow Exception. *Clear*: Overflow exception returns ±9.99999999999E499. *Set*: Overflow exception treated as an error.
−22	Infinite Result Exception. *Clear*: Infinite result exception treated as an error. *Set*: Infinite result exception returns ±9.99999999999E499.
−23 −24 −25 −26	Negative Underflow Indicator. Positive Underflow Indicator. Overflow Indicator. Infinite Result Indicator. When an exception occurs, corresponding flag (−23 through −26) is *set*, regardless of whether or not the exception is treated as an error.
−27 thru −29	Not used.

System Flags (continued)

Flag	Description
	Plotting and Graphics Flags
-30	Function Plotting. *Clear*: For equations of form $y = f(x)$, only $f(x)$ is drawn. *Set*: For equations of form $y = f(x)$, separate plots of y and $f(x)$ are drawn.
-31	Curve Filling. *Clear*: Curve filling between plotted points enabled. *Set*: Curve filling between plotted points suppressed.
-32	Graphics Cursor. *Clear*: Graphics cursor always dark. *Set*: Graphics cursor dark on light background and light on dark background.
	I/O and Printing Flags
-33	I/O Device. *Clear*: I/O directed to serial port. *Set*: I/O directed to IR port.
-34	Printing Device. *Clear*: Printer output directed to IR port. *Set*: Printer output directed to serial port if flag -33 is clear.
-35	I/O Data Format. *Clear*: Objects transmitted in ASCII form. *Set*: Objects transmitted in memory image form.
-36	RECV Overwrite. *Clear*: If file name received by HP 48 matches existing HP 48 variable name, new variable name with number extension is created to prevent overwrite. *Set*: If file name received by HP 48 matches existing HP 48 variable name, existing variable is overwritten.

System Flags (continued)

Flag	Description
	I/O and Printing Flags (continued)
-37	Double-Spaced Printing. *Clear*: Single-spaced printing. *Set*: Double-spaced printing.
-38	Linefeed. *Clear*: Linefeed added at end of each print line. *Set*: No linefeed added at end of each print line.
-39	I/O Messages. *Clear*: I/O messages displayed. *Set*: I/O messages suppressed.
	Time Management Flags
-40	Clock Display. *Clear*: Ticking clock displayed only when TIME menu selected. *Set*: Ticking clock displayed at all times.
-41	Clock Format. *Clear*: 12-hour clock. *Set*: 24-hour clock.
-42	Date Format. *Clear*: MM/DD/YY (month/day/year) format. *Set*: DD.MM.YY (day.month.year) format.
-43	Repeat Alarms Not Rescheduled. *Clear*: Unacknowledged repeat appointment alarms automatically rescheduled. *Set*: Unacknowledged repeat appointment alarms not rescheduled.

System Flags (continued)

Flag	Description
	Time Management Flags (continued)
–44	Acknowledged Alarms Saved. *Clear*: Acknowledged appointment alarms deleted from alarm list. *Set*: Acknowledged appointment alarms saved in alarm list.
	Display Format Flags
–45 thru –48	Number of Decimal Digits. Combined states of flags –45 through –48 sets number of decimal digits in Fix, Scientific, and Engineering modes.
–49 and –50	Number Display Format. Standard: –49 *clear*, –50 *clear*. Fix: –49 *set*, –50 *clear*. Scientific: –49 *clear*, –50 *set*. Engineering: –49 *set*, –50 *set*.
–51	Fraction Mark. *Clear*: Fraction mark is . (period). *Set*: Fraction mark is , (comma).
–52	Single-Line Display. *Clear*: Display gives preference to object in level 1, using up to four lines of stack display. *Set*: Display of object in level 1 restricted to one line.
–53	Precedence. *Clear*: Certain parentheses in algebraic expressions suppressed to improve legibility. *Set*: All parentheses in algebraic expressions displayed.
–54	Not used.

System Flags (continued)

Flag	Description
	Miscellaneous Flags
−55	Last Arguments. *Clear*: Operation arguments saved. *Set*: Operation arguments not saved.
−56	Error Beep. *Clear*: Error and BEEP-command beeps enabled. *Set*: Error and BEEP-command beeps suppressed.
−57	Alarm Beep. *Clear*: Alarm beep enabled. *Set*: Alarm beep suppressed.
−58	Verbose Messages. *Clear*: Prompt messages and data automatically displayed. *Set*: Automatic display of prompt messages and data suppressed.
−59	Fast Catalog Display. *Clear*: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation and equation name. *Set*: Equation Catalog (and messages in SOLVE, SOLVR, PLOT, and PLOTR menus) show equation name only.
−60	Alpha Lock. *Clear*: Alpha lock activated by pressing [α] twice. *Set*: Alpha lock activated by pressing [α] once.
−61	User-Mode Lock. *Clear*: 1-User mode activated by pressing [↰][USR] once. User mode activated by pressing [↰][USR] twice. *Set*: User mode activated by pressing [↰][USR] once.

System Flags (continued)

Flag	Description
	Miscellaneous Flags (continued)
-62	User Mode. *Clear*: User mode not active. *Set*: User mode active.
-63	Vectored [ENTER]. *Clear*: [ENTER] evaluates command line. *Set*: User-defined [ENTER] activated.
-64	Index Wrap Indicator. *Clear*: Last execution of GETI or PUTI did not increment index to first element. *Set*: Last execution of GETI or PUTI did increment index to first element.

D

Reserved Variables

The HP 48 contains the following *reserved variables*. They have specific purposes and their names are used as implicit arguments for certain commands. Avoid using these variables' names for other purposes: using them can affect the integrity of the commands that use these variables.

There are programmable commands to alter some values in some of these variables. Other variables or values can be altered only by storing a new list of values into the appropriate variable.

Reserved Variable	What It Contains	Used By:
ALRMDAT	Alarm parameters.	TIME ALRM commands
CST	List defining the contents of the CST (custom) menu.	MENU
"*der*"-names	User-defined derivative.	∂
EQ	Current equation.	ROOT, DRAW
IOPAR	I/O parameters.	I/O commands
n1, n2,...	Arbitrary integers.	ISOL, QUAD
PPAR	Plotting parameters.	DRAW
PRTPAR	Printing parameters.	PRINT commands

(continued)

Reserved Variable	What It Contains	Used By:
s1, s2,...	Arbitrary signs.	ISOL, QUAD
ΣDAT	Statistical data.	Statistics application, DRAW
ΣPAR	Statistical parameters.	Statistics application, DRAW

The Reserved Variables' Contents

The specific contents of most reserved variables (except *ALRMDAT*, *IOPAR*, and *PRTPAR*) can be different for each directory in memory.

ALRMDAT

ALRMDAT does not reside in a particular directory. You cannot access the variable itself, but you can access its data via RCLALARM and STOALARM (from any directory) and the Alarm Catalog.

ALRMDAT contains a list of these alarm parameters:

Parameter (Command)	Description	Default Value
Date (→DATE)	Date to go off. A real number, *MM.DDYYYY* (or *DD.MMYYYY* if flag −42 is set). Without *YYYY*, current year is used.	Current date.
Time (→TIME)	Time to go off. A real number, *HH.MMSS*.	00.0000

(continued)

Parameter (Command)	Description	Default Value
Action	Creates an appointment alarm if the parameter is a string; creates a control alarm if the parameter is any other object. An appointment alarm displays the string; a control alarm executes the non-string object.	Empty string (appointment alarm).
Repeat	Interval between automatic recurrences, given in ticks. One tick is $^1/_{8192}$ of a second.	0

Parameters without commands can be modified programmatically by storing new values in the list contained in *ALRMDAT* .

CST

CST contains a list (or a name specifying a list) of the objects that are contained in the CST (*custom*) menu. Objects in the custom menu usually have the same functionality they have in built-in menus. For example:

- Names behave like the VAR menu keys. Thus, if *ABC* is a variable name, ABC evaluates *ABC*, → ABC recalls its contents, and ← ABC stores new contents in *ABC*. Also, the menu label for the name of a directory has a bar over the left side of the label; pressing the menu key switches to that directory.
- Unit objects act like unit catalog entries. For instance, they have their left-shifted conversion capability.
- String keys echo the string.
- You can include backup objects in the list defining a custom menu by tagging the name of the backup object with its port location (0, 1, 2).

You can specify menu labels and key actions independently by embedding within the custom-menu list an inner list of the form
{ "*label-object*" *action-object* }.

See the example "Providing Different Menu Labels", under "Enhancing Custom Menus" in chapter 15 of the *HP 48 Owner's Manual.*

To provide different shifted actions for custom menu keys, specify within the inner list the three actions (objects) in yet another list. The order in this additional embedded list is the unshifted action, the left-shifted action, and then the right-shifted action. (You must specify the unshifted action in order to have the shifted actions.) See the example "Providing Shifted Functionality", under "Enhancing Custom Menus" in chapter 15 of the *HP 48 Owner's Manual.*

"der-" Names

If ∂ is applied to a function for which there is no built-in derivative, ∂ returns a new function whose name is "der" followed by the original function name. These "der"-function names are reserved variable names.

For an example, refer to "Advanced Topic: User-Defined Derivatives" in chapter 23 of the *HP 48 Owner's Manual.*

EQ

EQ contains the current equation or the name of the variable containing the current equation.

EQ supplies the equation for ROOT, as well as for the plotting command DRAW when the plot type is FUNCTION, CONIC, POLAR, PARAMETER, or TRUTH. (ΣDAT supplies the information when the plot type is HISTOGRAM, BAR, or SCATTER.)

The object in *EQ* can be an algebraic object, a number, a name, or a program. Its exact interpretation by DRAW depends on the plot type.

For graphics use, *EQ* can also be a list of equations or other objects. If *EQ* contains a list, then DRAW treats each object in turn as the current equation, plotting the objects successively. However, ROOT in the HP Solve application *cannot* solve an *EQ* containing a list.

To alter the contents of *EQ*, use the command STEQ.

IOPAR

IOPAR is a variable in the *HOME* directory. It contains a list of the I/O parameters needed for a communications link with a computer. It is created the first time you transfer data or open the serial port (OPENIO), and is automatically updated whenever you change the I/O settings.

Parameter (Command)	Description	Default Value
baud (BAUD)	Baud rate of 1200, 2400, 4800, or 9600.	9600
parity (PARITY)	0 = none, 1 = odd, 2 = even, 3 = mark, 4 = space. A positive parity is used upon both transmit and receive; a negative parity is used only upon transmit.	0
receive-pacing	Pacing is not used for Kermit I/O. For other serial I/O transfers, a non-zero real value enables pacing. Receive pacing sends an XOFF signal when the receive buffer is almost full, and sends an XON signal when it can take more data.	0 (no pacing)
transmit-pacing	Pacing is not used for Kermit I/O. For other serial I/O transfers, a non-zero real value enables pacing. Transmit pacing stops transmission upon receipt of XOFF, and resumes transmission upon receipt of XON.	0 (no pacing)
checksum (CKSM)	Error-detection scheme requested when initiating SEND. 1 = 1-digit arithmetic checksum, 2 = 2-digit, 3 = 3-digit cyclic redundancy check.	3

(continued)

Parameter (Command)	Description	Default Value
translation-code (TRANSIO)	Character-translation code: 0 = none; 1 = translate character 10 (line feed only) to/from characters 10 and 13 (line feed and carriage return); 2 = translate characters with numbers 128 through 159 (80—9F hex); 3 = translate characters with numbers 128 through 255.	1

Parameters without commands can be modified programmatically by storing new values in the list contained in *IOPAR* (use the PUT command).

n1, n2,...

The ISOL and QUAD commands return *general* solutions (as opposed to *principal* solutions) for operations. A general solution contains variable(s) for arbitrary integer(s) or arbitrary sign(s) or both.

The variable *n1* represents an arbitrary integer 0, 1, 2, etc. Additional arbitrary integers are represented by *n2*, *n3*, etc.

If flag −1 is set, then ISOL and QUAD return principal solutions, in which case the arbitrary integer is always zero.

PPAR

PPAR is a variable in the current directory. It contains a list of plotting parameters used by the command DRAW for all mathematical and statistical plots, by AUTO for autoscaling, and by the interactive (nonprogrammable) graphics operations.

Parameter (Command)	Description	Default Value
(x_{min},y_{min})	Coordinates of the lower left corner of the display range. A complex number.	(−6.50, −3.1)
(x_{max},y_{max})	Coordinates of the upper right corner of the display range. A complex number.	(6.5, 3.2)
indep (INDEP)	Independent variable: its name or a list containing its name and two real numbers (the plotting horizontal range).	*X*
res (RES)	Resolution. For plots of equations, determines the plotting interval along the *x*-axis. A binary number specifies the *pixel* resolution (how many columns of pixels between points). An integer specifies the resolution in *user* units (how many user units between points). Resolution for statistical plots is different; see below.	0
axes (AXES)	Coordinates of the axes' intersection (a complex number), or a list containing the intersection coordinates and labels (strings) for both axes.	(0, 0)
ptype (BAR, etc.)	Plot type (FUNCTION, CONIC, POLAR, PARAMETRIC, TRUTH, BAR, HISTOGRAM, or SCATTER).	FUNCTION
depend (DEPND)	Dependent variable. Its name or a list containing its name and two real numbers (the vertical plotting range).	*Y*

Parameters without commands can be modified programmatically by storing new values in the list contained in *PPAR*.

The RESET operation (→ PLOT NXT RESET) resets the *PPAR* parameters (except *ptype*) to their default values and erases *PICT*, creating a blank *PICT* of the default size (31 × 64 pixels).

The significance of the resolution parameter is different for the statistical plot types BAR and HISTOGRAM. For BAR, resolution specifies bar width. For HISTOGRAM, resolution specifies bin width. Resolution does not affect plot type SCATTER.

PRTPAR

PRTPAR is a variable in the *HOME* directory that contains a list of printing parameters. It is created automatically the first time you use a printing command.

Parameter (Command)	Description	Default Value
Delay time (DELAY)	The number of seconds the printer waits between sending lines; this should be at least as long as the time required to print the longest line. A real number in the range 0 to 6.9. If the delay is too short for the printer, you will lose data. The delay setting also affects serial printing if transmit-pacing (in *IOPAR*) is not being used.	1.8

(continued)

Parameter (Command)	Description	Default Value
Remap (OLDPRT stores the character-remapping string for the HP 82240A Infrared Printer)	The current remapping of the extended character set for printing. A string that contains as many characters as you want to remap, with the first character being the new character 128, the second being the new 129, etc. (Any character number that exceeds the string length will not be remapped.) See example below.	Empty string.
Line length	Specifies the number of characters in a line for serial printing. A real number. Does *not* affect infrared printing.	80
Line termination	A string specifying the line-termination method for serial printing. Does *not* affect infrared printing.	Control characters 13 (carriage return) and 10 (line feed).

Parameters without commands can be modified programmatically by storing new values in the list contained in *PRTPAR* (use the PUT command).

A change in a parameter is effective immediately, *except* when printing the display using the simultaneous keystrokes [ON] [PRINT] (because this does not use *PRTPAR*). This printing method is affected only by the delay parameter, a change in which will not affect [ON] [PRINT] until after the next printing command has been executed. To have a new delay time affect [ON] [PRINT] immediately, use the DELAY command, which takes a real-number argument from level 1.

Remapping Example. If the remapping string were "ABCDEFGH" and the character to be printed had value 131, then the character actually printed would be "D", since 131 − 128 = 3 and "A" has the value zero. A character code of 136 or greater would not be remapped since 136 − 128 = 8, which exceeds the length of the string.

s1, s2,...

The ISOL and QUAD commands return *general* solutions (as opposed to *principal* solutions) for operations. A general solution contains variable(s) for arbitrary integer(s) or arbitrary sign(s) or both.

The variable *s1* represents an arbitrary + or − sign. Additional arbitrary signs are represented by *s2*, *s3*, etc.

If flag −1 is set, then ISOL and QUAD return principal solutions, in which case the arbitrary sign is always +1.

ΣDAT

ΣDAT is a variable in the current directory. It contains either the current statistical matrix or the name of the variable containing this matrix. This matrix contains the data used by the Statistics applications, including the plotting of scatter, histogram, and bar plots.

Statistical Matrix for Variables 1 to m

var_1	var_2	...	var_m
x_{11}	x_{21}	...	x_{m1}
x_{12}	x_{22}	...	x_{m2}
⋮	⋮	⋮	⋮
x_{1n}	x_{2n}	...	x_{mn}

You can designate a new current statistical matrix by entering new data, editing the current data, or selecting another matrix.

The command CLΣ clears the current statistical matrix.

ΣPAR

ΣPAR is a variable in the current directory. It contains either the current statistical parameter list or the name of the variable containing this list.

Parameter (Command)	Description	Default Value
Indep. col. no. (XCOL)	Independent-variable's column number. A real number.	1
Dep. col. no. (YCOL)	Dependent-variable's column number. A real number.	2
Intercept (LR)	Coefficient of intercept as determined by the current regression. A real number.	0
Slope (LR)	Coefficient of slope as determined by the current regression. A real number.	0
Model (LINFIT, etc.)	Regression model (LINFIT, EXPFIT, PWRFIT, or LOGFIT).	LINFIT

Parameters without commands can be modified programmatically by storing new values in the list contained in *ΣPAR*.

Contacting Hewlett-Packard

For Information About Using the Calculator. If you have questions about how to use a particular command, also check all related commands. Also check the owner's manual: the table of contents, the index, and "Answers to Common Questions" in appendix A. If you can't find an answer in the manuals, you can contact the Calculator Support department:

Hewlett-Packard
Calculator Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

(503) 757-2004
8:00 a.m. to 3:00 p.m. Pacific time
Monday through Friday

HP Calculator Bulletin Board System. The Bulletin Board provides for the exchange of software and information between HP calculator users, developers, and distributors. It operates at 300/1200/2400 baud, full duplex, no parity, 8 bits, 1 stop bit. The telephone number is (503) 750-4448. The Bulletin Board is a free service—you pay for only the long-distance telephone charge.

Contents

Page	**6**	**Command Dictionary**
	464	**Appendix A: Table of Error and Status Messages**
	481	**Appendix B: Table of Units**
	486	**Appendix C: Table of System Flags**
	494	**Appendix D: Reserved Variables**

HEWLETT PACKARD

Reorder Number
00048-90054

00048-90053 English
Printed in U.S.A. 7/90