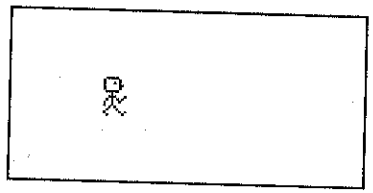
Example: Send the small person out for a walk.

(VAR) WALK



Press (CANCEL) when you think the walker's tired.

3

Command Reference

This chapter contains an alphabetical listing of the programmable commands and functions available on the HP 48. The listings include the following information:

- a brief definition of what the command or function does
- a stack diagram showing the arguments it requires (if any)
- the keys to press to gain access to it
- any flags that may affect how it works
- additional information about how it works and how to use it
- an example of its use
- related commands or functions

The next few pages explain how to read the stack diagrams in the command reference, how commands are alphabetized, 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 reference 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:

ACOS

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

{ }

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

This diagram indicates that the *analytic function* ACOS (*Arc Cosine*) takes a single argument from level 1 and returns one result (to level 1). ACOS can take either a real or complex number or an algebraic object as its argument. In the first case, it returns the numeric arccosine; in the second, it returns the symbolic arccosine expression of the argument.

Some commands affect a calculator state—a mode, a reserved variable, a flag, or a display— without taking any arguments from the stack or returning any results to the stack. No stack diagrams are shown for these commands.

Parallel Processing with Lists

Commands that can use the parallel list processing feature are denoted by the "{ }" symbol located above the stack diagram. This feature is discussed in greater detail in Appendix G.

As a rule-of-thumb, a command can use parallel list processing if all the following are true:

- The command checks for valid argument types. Commands that apply to all object types, such as DUP, SWAP, ROT, and so forth, do not use parallel list processing.
- The command takes exactly one, two, three, four, or five arguments, none of which may itself be a list. Commands, such as →LIST, that have an indefinite number of arguments do not use parallel list processing.
- The command isn't a programming branch command (IF, FOR, CASE, NEXT, and so forth).

The HP 48 also has a few commands (PURGE and DELKEYS are examples) that have list processing capability built into their definitions, and so do not also use the parallel list processing feature.

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. There are also four non-programmmable *operations* (DBUG, NEXT, SST, and SST↓) that are included with the programmable commands as a convenience because they are used interactively while programming.

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

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 noninteger).
$:n_{\text{port}}:name_{\text{backup}}$	Backup identifier.
$:n_{\text{port}}:n_{\text{library}}$	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 object.
(*x*,*y*)	Complex number in rectangular form, or user-unit coordinate.
z	Real or complex number.

ABS

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

{}

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

Keyboard Access:

MTH REAL NXT ABS

MTH MATR NORM ABS

MTH NXT CMPL ABS

MTH VECTR ABS

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

ACK

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

Keyboard Access: ◀ TIME ALRM ACK

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: Acknowledges all past-due alarms.

Keyboard Access: ◀ TIME ALRM ACKA

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

ACOS

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

{}

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

Keyboard Access: ◀ ACOS

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*ni'
```

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.

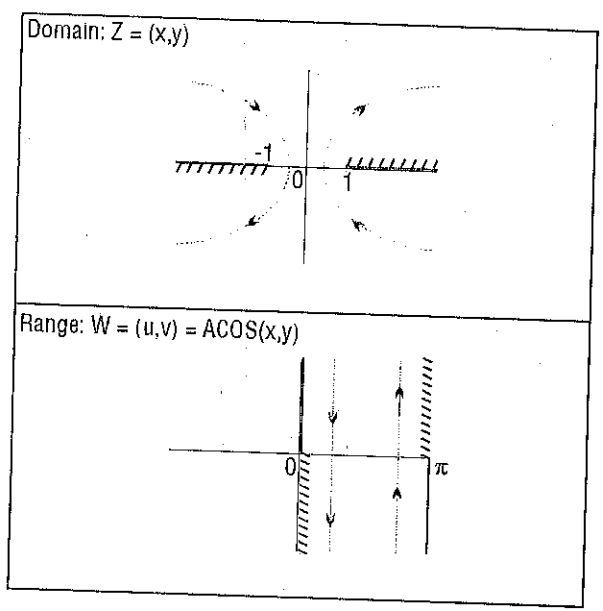
The principal branch used by the HP 48 for ACOS was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cut for the complex-valued arc cosine function occurs where the corresponding real-valued function is undefined. The principal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

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

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

Related Commands: ASIN, ATAN, COS, ISOL



Branch Cuts for ACOS(Z)

ACOSH

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

{}

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

Keyboard Access: (MTH) HYP ACOSH

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.

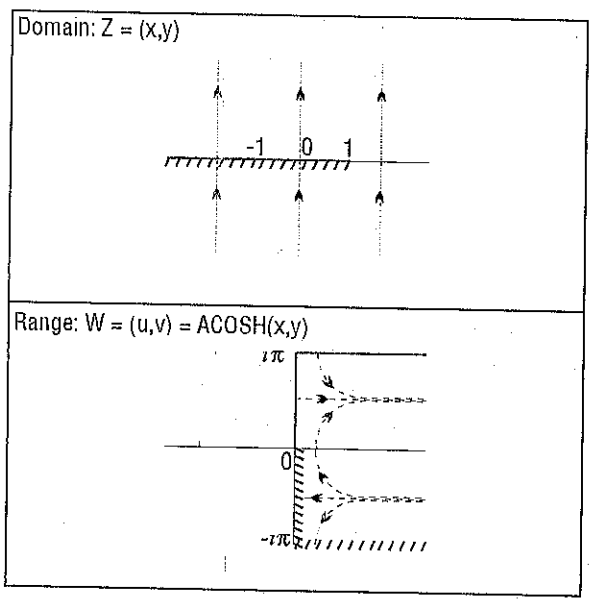
The graphs below show the domain and range of ACOSH. The graph of the domain shows where the branch cut occurs: the heavy solid line

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*ni' for the case *s1*=1 and *n1*=0. For other values of *s1* and *n1*, the horizontal half-band in the lower graph is rotated to the left and translated up and down. Taken together, the bands cover the whole complex plane, which is the domain of COSH.

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

Related Commands: ASINH, ATANH, COSH, ISOL



Branch Cut for ACOSH(Z)

ADD

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

{ }

Level 2	Level 1	→	Level 1
{ $list_1$ }	{ $list_2$ }	→	{ $list_{result}$ }
{ *list* }	$obj_{non-list}$	→	{ $list_{result}$ }
$obj_{non-list}$	{ *list* }	→	{ $list_{result}$ }

Keyboard Access: (MTH) LIST ADD

Affected by Flags: None

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

Related Commands: ΔLIST, ΠLIST, ΣLIST

ALOG

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

{}

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

Keyboard Access: ↰ 10^x

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

AMORT

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

{}

Level 1	→	Level 3	Level 2	Level 1
n	→	*principal*	*interest*	*balance*

Keyboard Access: ↰ SOLVE TVM AMORT

Affected by Flags: Financial Payment Mode (−14)

Remarks: Values must be stored in the TVM variables (*I%YR*, *PV*, *PMT*, and *PYR*). The number of payments *n* is taken from level 1 and flag −14.

Related Commands: TVM, TVMBEG, TVMEND, TVMROOT

AND

And Function: Returns the logical AND of two arguments.

{}

Level 2	Level 1	→	Level 1
$\#n_1$	$\#n_2$	→	$\#n_3$
"$string_1$"	"$string_2$"	→	"$string_3$"
T/F_1	T/F_2	→	0/1
T/F	*'symb'*	→	*'T/F AND symb'*
'symb'	*T/F*	→	*'symb AND T/F'*
'$symb_1$'	'$symb_2$'	→	'$symb_1$ AND $symb_2$'

Keyboard Access:

MTH BASE NXT LOGIC AND

PRG TEST NXT AND

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:

AND

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

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

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

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

Related Commands: NOT, OR, XOR

ANIMATE

Animate Command: Displays graphic objects in sequence.

Level n+1...Level 2	Level 1	→	Level 1
$grob_n \ldots grob_1$	n_{grobs}	→	*same stack*
$grob_n \ldots grob_1$	{ *n* { #X #Y } *delay rep* }	→	*same stack*

Keyboard Access:

PRG EQUIT NXT ANIM

Affected by Flags: None

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

If you use a list on level 1, all parameters must be present.

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

Example: The following program draws half a cylinder and rotates it:

```
« PARSURFACE ( 'COS(X)' 'SIN(X)' Y )
  STEQ
    « I 180 I + XXRNG ERASE DRAW PICT RCL
    »
  I 0 359 8 SEQ OBJ→ ANIMATE DROPN
»
```

This program also illustrates the use of SEQ and PARSURFACE. You can adjust the increment value used with SEQ (8 is used here) to change the number of images drawn by the program, or to use less memory.

APPLY

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

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

Keyboard Access: ↰ SYMBOLIC NXT APPLY

APPLY

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 this:

`'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 object.

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==i' THEN 'π/2' END
   'argument==-i' THEN '-π/2' END
   argument ASIN
  END
 »
»
```

ENTER ' Asin STO

Related Commands: QUOTE, |

ARC

Draw Arc Command: Draws an arc in *PICT* counterclockwise from x_{θ_1} to x_{θ_2}, with its center at the coordinate specified in level 4 and its radius specified in level 3.

Level 4	Level 3	Level 2	Level 1	→	Level 1
(x, y)	x_{radius}	x_{θ_1}	x_{θ_2}	→	
$\{\#n\ \#m\}$	$\#n_{\text{radius}}$	x_{θ_1}	x_{θ_2}	→	

Keyboard Access: PRG PICT ARC

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

The setting of flags −17 and −18 determine the interpretation of x_{θ_1} and x_{θ_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 $(x_{\text{radius}}, x_{\theta_1})$. 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_{θ_2}).

If $x_{\theta_1} = x_{\theta_2}$, ARC plots one point. If $|x_{\theta_1} - x_{\theta_2}| > 360$ degrees, 2π radians, or 400 grads, ARC draws a complete circle.

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

Related Commands: BOX, LINE, TLINE

ARCHIVE

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

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

Keyboard Access: ↰ MEMORY NXT ARCHIVE

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

Remarks: The specified port number can be 0 through 33. The port used (except 0) 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 in binary via Kermit protocol through the current I/O port to the specified filename.

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

Related Commands: RESTORE

ARG

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

{}

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

Keyboard Access: MTH NXT CMPL ARG

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

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} }

ARRY→

Keyboard Access: None. Must be typed in.

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, DTAG, EQ→, LIST→, OBJ→, STR→

→ARRY

Stack to Array Command: Returns a vector of n real or complex elements or a matrix of $n \times m$ real or complex elements.

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*]]

Keyboard Access: PRG TYPE →ARR

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, OBJ→, STR→, →TAG, →UNIT

ASIN

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

{}

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

Keyboard Access: ↶ ASIN

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)+(-i)^n1*π+n1*i'
```

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

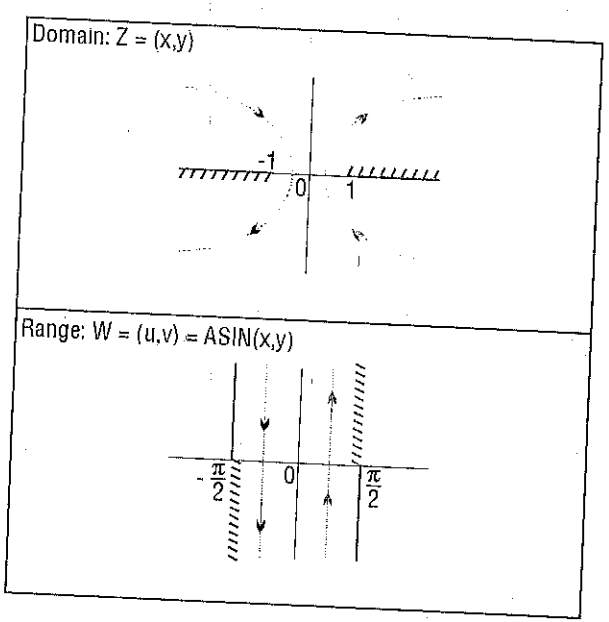
is undefined. The principal branch also preserves most of the important symmetries.

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)*(-i)^n1+π*ni' for the case *n1*=0. For other values of *n1*, the vertical band in the lower graph is translated to the right (for *n1* positive) or to the left (for *n1* negative). Taken together, the bands cover the whole complex plane, which is the domain of SIN.

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

Related Commands: ACOS, ATAN, ISOL, SIN



Branch Cuts for ASIN(Z)

ASINH

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

{}

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

Keyboard Access: (MTH) HYP ASINH

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)*(-i)^n1+π*i*ni'

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.

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

Related Commands: ACOSH, ATANH, ISOL, SINH

ASN

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

{}

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

Keyboard Access: ◀ MODES KEYS ASN

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 row number r, column number c, and plane (shift) p. The legal values for p are as follows:

Plane, p	Shift
0 or 1	unshifted
2	◀ left-shifted
3	▶ right-shifted
4	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).

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

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

ASR

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

{}

Level 1	→	Level 1
#n_1	→	#n_2

Keyboard Access: MTH BASE NXT BIT ASR

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.

ASR

An arithmetic shift is useful for preserving the sign bit of a binary integer that will be shifted. Although the HP 48 makes no special provision for signed binary integers, you can still *interpret* a number as a signed quantity.

Related Commands: SL, SLB, SR, SRB

ATAN

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

{}

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

Keyboard Access: ↩ ATAN

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.

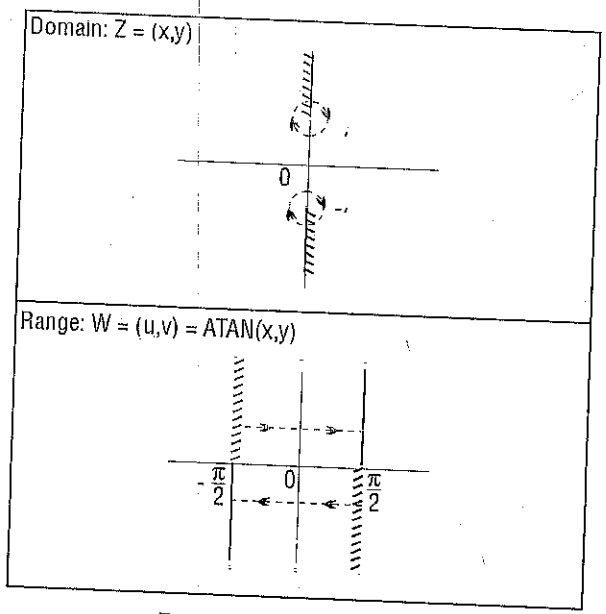
The principal branch used by the HP 48 for ATAN was chosen because it is analytic in the regions where the arguments of the *real-valued* inverse function are defined. The branch cuts for the complex-valued arc tangent function occur where the corresponding real-valued function is undefined. The principal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

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 $W = (u, v) = \mathrm{TAN}(x, y)$ in the upper graph.

Related Commands: ACOS, ASIN, ISOL, TAN



Branch Cuts for ATAN(Z)

ATANH

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

{ }

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

Keyboard Access: (MTH) HYP ATAN

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*ni'
```

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.

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

Related Commands: ACOSH, ASINH, ISOL, TANH

ATICK

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

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

Keyboard Access: ◀ PLOT PPAR NXT ATICK

Affected by Flags: None

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

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

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

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

Related Commands: AXES, DRAX

ATTACH

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

{ }

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

Keyboard Access: ◀ LIBRARY NXT ATTACH

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 (1-33), 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 an application card is installed. 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` .)

The number of libraries that can be attached to the HOME directory is limited only by the available memory. However, only one library at a time can be attached to any other directory. If you attempt to attach a second library to a non-*HOME* directory, the new library will overwrite the old one.

ATTACH

Related Commands: DETACH, LIBS

AUTO

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

Keyboard Access: ◀ PLOT NXT AUTO

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 $\pm\infty$, 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 $\pm\infty$, 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 ΣDAT, plus 1. Sets the y-range to the minimum and maximum of the elements. The x-axis is always included.
HISTOGRAM	Sets the x-axis display range to the minimum and maximum of the elements in ΣDAT. Sets the y-axis display range from 0 to the number of rows in ΣDAT.
SCATTER	Sets the x-axis display range to the minimum and maximum of the independent variable column (XCOL) in ΣDAT. Sets the y-axis display range to the minimum and maximum of the dependent variable column (YCOL).

AUTO does not affect 3D plots.

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

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

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, *W, XRNG, YRNG

AXES

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

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

XES

Keyboard Access: ◀ PLOT PPAR NXT 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*. How the argument is used depends on the type of object it is:

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

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

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

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

Example: The command sequence

```
{ (0,0) 2 "t" "y" } AXES LABEL
```

specifies an axes intersection at (0,0), tick-mark annotation every 2 units, and puts the labels t and y in *PICT*. The labels are positioned to identify the horizontal and vertical axes respectively.

Related Commands: ATICK, DRAW, DRAX, LABEL

BAR

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

Keyboard Access: ◀ PLOT NXT STAT PTYPE 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 to be plotted is specified by the XCOL command, and is stored in the first parameter of the reserved variable ΣPAR. The plotting parameters are specified in the reserved variable *PPAR*, which has the following form:

BAR

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

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

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

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

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

BARPLOT

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

Keyboard Access: ← STAT PLOT BARPLOT

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 ΣPAR. The default column is 1. Data can be positive or negative, resulting in bars above or below the axis. The y-axis is autoscaled, and the plot type is set to BAR.

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

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

BAUD

Baud Rate Command: Specifies bit-transfer rate.

{}

Level 1	→	Level 1
$n_{baudrate}$	→	

Keyboard Access: ← I/O IOPAR BAUD

Affected by Flags: None

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

For more information, refer also to the reserved variable *IOPAR* (*I/O parameters*) in appendix D, "Reserved Variables."

Related Commands: CKSM, PARITY, TRANSIO

BEEP

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

{}

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

Keyboard Access: PRG NXT I/O NXT BEEP

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

BESTFIT

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

Keyboard Access: ← STAT ΣPAR MODL BESTF

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. For a description of ΣPAR, see appendix D, "Reserved Variables."

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

BIN

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

Keyboard Access: (MTH) BASE BIN

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 b. If the current base is not binary, binary numbers can still be entered by using the suffix b (the numbers are displayed in the current base, however).

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

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

BINS

Sort Into Frequency Bins Command: 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}.

{}

Level 3	Level 2	Level 1	→	Level 2	Level 1
x_{min}	x_{width}	n_{bins}	→	[[n_{bin1} … n_{binn}]]	[n_{binL} n_{binR}]

Keyboard Access: ← (STAT) STAT VAR BINS

Affected by Flags: None

Remarks: BINS returns a matrix containing the frequency of occurrences in each bin, and a 2-element array containing the frequency of occurrences falling below or above the defined range of x-values. The array can be stored into the reserved variable ΣDAT

and used to plot a bar histogram of the bin data (for example, by executing BARPLOT).

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

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

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

Example: If the independent column of ΣDAT contains the following data:

7 2 3 i 4 6 9 0 i i 3 5 13 2 6 9 5 8 5

1 2 5 BINS returns [[5] [3] [5] [2] [2]] and [i i].

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 shows that 5 x-values (2 i i i 2) fell in bin 1, where bin 1 ranges from x-value 1 through 2.99999999999. The vector shows that one x-value was less than x_{min} (0), and one was greater than x_{max} (13).

Related Commands: BARPLOT, XCOL

BLANK

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

{}

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

Keyboard Access: (PRG) GROB BLANK

Affected by Flags: None

Related Commands: →GROB, LCD→

BOX

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

{}

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

Keyboard Access: PRG PICT BOX

Affected by Flags: None

Related Commands: ARC, LINE, TLINE

BUFLEN

Buffer Length Command: Returns the number of characters in the HP 48's serial input buffer and a single digit indicating whether an error occurred during data reception.

Level 1	→	Level 2	Level 1
	→	n_{chars}	0/1

Keyboard Access: ← I/O NXT SERIN BUFLEN

Affected by Flags: None

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

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

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

BYTES

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

{}

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

Keyboard Access: ← MEMORY BYTES

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. The size of the name alone is $(3.5 + 2 \times n)$, where n is the number of characters in the name.

Example: Objects that decompile identically can have different byte sizes and checksums. For instance,

```
« 1 »
```

and

```
1 'A' STO A « » +
```

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

Related Commands: MEM

B→R

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

{}

Level 1	→	Level 1
#*n*	→	*n*

Keyboard Access: (MTH) BASE B→R

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

Remarks: If # $n \geq$ # 1000000000000 (base 10), only the 12 most significant decimal digits are preserved in the resulting mantissa.

Related Commands: R→B

CASE

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

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

Keyboard Access: (PRG) BRCH CASE CASE

Affected by Flags: None

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

The CASE ... END structure has this syntax:

CASE
test-clause$_1$ THEN *true-clause*$_1$ END
test-clause$_2$ THEN *true-clause*$_2$ END

test-clause$_n$ THEN *true-clause*$_n$ END
default-clause (optional)
END

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

Example: The following program takes a numeric argument from the stack:

- if the argument is negative, it is added to itself
- if the argument is positive, it is negated
- if the argument is zero, the program aborts

```
« → X
  « CASE
    'X>0'
    THEN X NEG END
    'X<0'
    THEN X DUP + END
    'X==0'
    THEN 0 DOERR END
  END
```

CASE

»

»

Related Commands: END, IF, IFERR, THEN

CEIL

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

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

Keyboard Access: MTH REAL NXT NXT CEIL

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: 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.

CF

{}

Level 1	→	Level 1
(*x*, *y*)	→	
x	→	

Keyboard Access: ↩ PLOT PPAR NXT CENTR

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

CF

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

{}

Level 1	→	Level 1
$n_{flagnumber}$	→	

Keyboard Access:

↩ MODES FLAG CF

PRG TEST NXT NXT CF

Affected by Flags: None

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

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

CHOOSE

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

Level 3	Level 2	Level 1	→	Level 2	Level 1
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→	*obj* or *result*	"*1*"
"*prompt*"	{ c_1 ... c_n }	n_{pos}	→		"*0*"

Keyboard Access: PRG NXT TH CHOOS

Affected by Flags: None

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

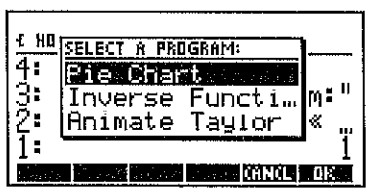
Variable	Function
"prompt"	A message that appears at the top of choose box. If *"prompt"* is an empty string (""), no message is displayed.
$\{c_1 \ \ldots \ c_n\}$	Definitions that appear within the choose box. A choice definition (c_x) can have two formats. ■ *obj*, any object. ■ { $obj_{display}$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

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

Example: CHOOSE with the following three lines on the display.

```
"Select a Program:"
{ {"Pie Chart" «PIE»} {"Inverse Function" «ROOTR»}
{"Animate Taylor" «TSA»} }
i
```

would produce the following display.



Related Commands: INFORM, NOVAL

%CH

Percent Change Function: Returns the percent change from x (level 2) to y (level 1) as a percentage of x.

{ }

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*)'

%CH

Keyboard Access: (MTH) REAL %CH

Affected by Flags: Numerical Results (−3)

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

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

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

`100_K 150_K %CH` returns `50`.

Related Commands: %, %T

CHR

Character Command: Returns a string representing the HP 48 character corresponding to the character code *n*.

{}

Level 1	→	Level 1
n	→	*"string"*

Keyboard Access:

(←)(CHARS) CHR

(PRG) TYPE (NXT) CHR

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. See the entry for NUM for a complete list of characters and character codes.

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)`.

You can use the CHARS application to find the character code for any character used by the HP 48. See "Keying in Special Characters" in chapter 2 of the *HP 48 User's Guide*.

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

CKSM

Checksum Command: Specifies the error-detection scheme.

{}

Level 1	→	Level 1
$n_{checksum}$	→	

Keyboard Access: (←)(I/O) TRANS CKSM

Affected by Flags: None

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

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

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

IR transmission should use checksum type 3.

Related Commands: BAUD, PARITY, TRANSIO

CLEAR

Clear Command: Removes all objects from the stack.

Level n ... Level 1	→	Level n ... Level 1
obj_n ... obj_1	→	

Keyboard Access: ◀ CLEAR

Affected by Flags: None

Remarks: To recover a cleared stack, press ▶ UNDO before executing any other operation. There is no programmable command to recover the stack.

Related Commands: CLVAR, PURGE

CLKADJ

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

{}

Level 1	→	Level 1
x	→	

Keyboard Access: ◀ TIME NXT NXT CLKA

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

CLLCD

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

Keyboard Access: PRG NXT CLLCD

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), then freezes the entire display.

Related Commands: DISP, FREEZE

CLOSEIO

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

Keyboard Access: ◀ I/O NXT CLOSE

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, but can conserve power without turning off the calculator.

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

CLOSEIO

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

Related Commands: BUFLEN, OPENIO

CLΣ

Clear Sigma Command: Purges the current statistics matrix (reserved variable ΣDAT).

Keyboard Access: ◀ STAT MTH CLΣ

Affected by Flags: None

Related Commands: RCLΣ, STOΣ, Σ+, Σ−

CLTEACH

Clear Teaching Examples Command: Removes the EXAMPLES subdirectory and its contents from the HOME directory.

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Related Commands: TEACH

CLUSR

Clear Variables Command: Provided for compatibility with the HP 28. CLUSR is the same as CLVAR. See CLVAR.

CLVAR

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

Keyboard Access: None. Must be typed in.

Affected by Flags: None

Related Commands: CLUSR, PGDIR, PURGE

CNRM

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

[ ]

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

Keyboard Access: MTH MATR NORM CNRM

Affected by Flags: None

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

Related Commands: CROSS, DET, DOT, RNRM

→COL

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

{}

Level 1	→	Level n+1 ...	Level 2	Level 1
[[*matrix*]]	→	[*vector*]$_{col1}$	[*vector*]$_{coln}$	$n_{colcount}$
[*vector*]	→	$element_1$	$element_n$	$n_{elementcount}$

Keyboard Access: (MTH) MATR COL →COL

Affected by Flags: None

Remarks: →COL introduces no rounding error.

Related Commands: COL→, →ROW, ROW→

COL+

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

{}

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

Keyboard Access: (MTH) MATR COL COL+

Affected by Flags: None

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

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

Related Commands: COL−, CSWP, ROW+, ROW−

COL−

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

{}

Level 2	Level 1	→	Level 2	Level 1
[[*matrix*]]$_1$	n_{column}	→	[[*matrix*]]$_2$	[*vector*]$_{column}$
[*vector*]$_1$	$n_{element}$	→	[*vector*]$_2$	$element_n$

Keyboard Access: (MTH) MATR COL COL−

Affected by Flags: None

Remarks: n is rounded to the nearest integer.

Related Commands: COL+, CSWP, ROW+, ROW−

COL→

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

Level n+1 ...	Level 2	Level 1	→	Level 1
[*vector*]$_{column1}$	[*vector*]$_{columnn}$	$n_{columncount}$	→	[[*matrix*]]
$element_1$	$element_n$	$n_{elementcount}$	→	[*vector*]

Keyboard Access: MTH MATR COL COL→

Affected by Flags: None

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

Related Commands: →COL, →ROW, ROW→

COLCT

Collect Like Terms Command: Simplifies an algebraic expression or equation by "collecting" like terms.

{}

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

Keyboard Access: ↶ SYMBOLIC COLCT

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

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

Related Commands: EXPAN, ISOL, QUAD, SHOW

COLΣ

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

{}

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

Keyboard Access: None. Must be typed in.

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.

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

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

COLΣ

Example: 2 5 COLΣ sets column 2 in ΣDAT 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 ΣPAR.

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

COMB

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

{}

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($symb_n$, $symb_m$)'

Keyboard Access: MTH NXT PROB COMB

Affected by Flags: Numerical Results (−3)

Remarks: The following formula is used to calculate $C_{n,m}$

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

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

Related Commands: PERM, !

CON

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

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*'	$z_{constant}$	→	

Keyboard Access: MTH MATR MAKE CON

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, depending on the object in level 2.

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

  If 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

COND

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

{}

Level 1	→	Level 1
[[*matrix*]]$_{nxn}$	→	$x_{conditionnumber}$

Keyboard Access: (MTH) MATR NORM COND

Affected by Flags: None

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

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

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

Example: The following program computes the above rule of thumb for the number of accurate digits:

```
«
 DUP SIZE 1 GET LOG SWAP COND LOG + 11 SWAP -
»
```


Related Commands: SNRM, SRAD, TRACE

CONIC

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

Keyboard Access: ← (PLOT) PTYPE 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 this form:

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

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

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

- *ptype* is a command name specifying the plot type. Executing the command CONIC places the command name CONIC in *PPAR*.
- *depend* is a name specifying the dependent variable. The default value is *Y*

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

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

CONJ

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

{}

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

Keyboard Access: MTH NXT CMPL NXT CONJ

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) ]`.

A square matrix *A* containing complex elements is said to be *Hermitian* if $A^H = A$, where A^H is the same as a normal transpose except that the complex conjugate of each element is used. The following program returns 1 if the input matrix is Hermitian, and a 0 if it is not.

```
« DUP TRN CONJ SAME »
```

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

CONLIB

Open Constants Library Command: Opens the Constants Library catalog.

Keyboard Access: ← EQ LIB CNST CONL

Affected by Flags: None

Related Commands: CONST

CONST

Constant Value Command: Returns the value of a constant.

{}

Level 1	→	Level 1
'*name*'	→	*x*

Keyboard Access: ← EQ LIB CNST CONS

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

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

See "Using the Constants Library" in chapter 25 of the *HP 48 User's Guide* for a list of the constants available in the HP 48's Constants Library.

Related Commands: CONLIB

CONT

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

Keyboard Access: ◀ CONT

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 )" 1 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: Converts a source unit object to the dimensions of a target unit.

{ }

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

Keyboard Access: ◀ UNITS CONV

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

CORR

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

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

Keyboard Access: ◀ STAT FIT CORR

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).

CORR

The correlation is computed with 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: COLΣ, COV, PREDX, PREDY, XCOL, YCOL

COS

Cosine Analytic Function: Returns the cosine of the argument.

{}

Level 1	→	Level 1
z	→	cos *z*
'*symb*'	→	'COS(*symb*)'
x_unit$_{angular}$	→	cos (*x_unit*$_{angular}$)

Keyboard Access: (COS)

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 Function: Returns the hyperbolic cosine of the argument.

{}

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

Keyboard Access: (MTH) HYP COSH

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

COV

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

Level 1	→	Level 1
	→	$x_{covariance}$

Keyboard Access: (↰)(STAT) FIT COV

COV

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 with the following formula:

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

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

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

CR

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

Keyboard Access: ◀ I/O PRINT CR

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

When 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*.

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

CRDIR

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

{}

Level 1	→	Level 1
'global'	→	

Keyboard Access: ◀ MEMORY DIR CRDIR

Affected by Flags: None

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

Related Commands: HOME, PATH, PGDIR, UPDIR

CROSS

Cross Product Command: CROSS returns the cross product $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ of vectors A and B.

{}

Level 2	Level 1	→	Level 1
[*vector*]$_A$	[*vector*]$_B$	→	[*vector*]$_{A\times B}$

Keyboard Access: MTH VECTR CROSS

Affected by Flags: None

Remarks: The arguments must be vectors having two or three elements, and do not both need the same number of elements. (The HP 48 automatically converts a two-element argument `[ d1 d2 ]` to a three-element argument `[ d1 d2 0 ]`.)

CROSS

Related Commands: CNRM, DET, DOT, RNRM

CSWP

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

{}

Level 3	Level 2	Level 1	→	Level 1
$[[\ matrix\ ]]_1$	$n_{column\,i}$	$n_{column\,j}$	→	$[[\ matrix\ ]]_2$
$[\ vector\ ]_1$	$n_{element\,i}$	$n_{element\,j}$	→	$\lceil\ vector\ \rfloor_2$

Keyboard Access: (MTH) MATR COL CSWP

Affected by Flags: None

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

Related Commands: COL+, COL−, RSWP

CYLIN

Cylindrical Mode Command: Sets Cylindrical coordinate mode.

Keyboard Access:

(←) (MODES) ANGL CYLIN

(MTH) VECTR (NXT) CYLIN

Affected by Flags: None

Remarks: CYLIN clears flag −15 and sets flag −16, and displays the RΔZ annunciator.

In Cylindrical mode, vectors are displayed as polar components. Therefore, a 3D vector would appear as $[R\ \angle\theta\ Z]$.

Related Commands: RECT, SPHERE

C→PX

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

{}

Level 1	→	Level 1
(x, y)	→	$\{\ \#n\ \#m\ \}$

Keyboard Access: (PRG) PLOT (NXT) C→PX

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

C→R

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

{}

Level 1	→	Level 2	Level 1
(x, y)	→	x	y
$[\ C\text{-}array\ ]$	→	$\{\ R\text{-}array\ \}_1$	$\lceil\ R\text{-}array\ \rfloor_2$

C→R

Keyboard Access:

MTH NXT CMPL C→R

PRG TYPE NXT C→E

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

DARCY

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

{}

Level 2	Level 1	→	Level 1
$x_{e/D}$	y_{Re}	→	x_{Darcy}

Keyboard Access: ↶ EQ LIB UTILS DARCY

Affected by Flags: None

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

Related Commands: FANNING

DATE

Date Command: Returns the system date to level 1.

Level 1	→	Level 1
	→	*date*

Keyboard Access: ↶ TIME DATE

Affected by Flags: Date Format (−42)

Example: If the current date is May 12, 1990, if flag −42 is clear, and if the display mode is Standard, DATE returns 5.12199. (The trailing zeros are dropped.)

Related Commands: DATE+, DDAYS, TIME, TSTR

→DATE

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

{}

Level 1	→	Level 1
date	→	

Keyboard Access: ↶ TIME →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, 1991 to December 31, 2090.

→DATE

Example: If flag −42 is set and the current system year is 1995, then 28.07 →DATE sets the system date as July 28, 1995.

Related Commands: →TIME

DATE+

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

{}

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

Keyboard Access: ◀ TIME NXT DATE+

Affected by Flags: Date Format (−42)

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

Related Commands: DATE, DDAYS

DBUG

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

Level 1	→	Level 1
≪ *program* ≫ or '*program name*'	→	

Keyboard Access: PRG NXT PRG DBUG

Affected by Flags: None

Remarks: DBUG is not programmable.

Related Commands: HALT, NEXT, SST, SST↓

DDAYS

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

{}

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

Keyboard Access: ◀ TIME NXT DDAYS

Affected by Flags: Date Format (−42)

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

Related Commands: DATE, DATE+

DEC

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

Keyboard Access: MTH BASE DEC

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, RCWS, STWS

DECR

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

{}

Level 1	→	Level 1
'name'	→	x_{new}

Keyboard Access: ◀ MEMORY MATH DECR

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

The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
«
 100 'A' STO
 WHILE A REPEAT 'A' DECR END
 'A' PURGE
»
```

Related Commands: INCR

DEFINE

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

{}

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

Keyboard Access: ◀ DEF

Affected by Flags: Numerical Results (−3)

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

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

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: Sets Degrees angle mode.

Keyboard Access: ◀ MODES ANGL DEG

Affected by Flags: None

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

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

Related Commands: GRAD, RAD

DELALARM

Delete Alarm Command: Deletes the alarm specified in level 1.

{}

Level 1	→	Level 1
n_{index}	→	

Keyboard Access: ◀ TIME ALRM DELAL

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

{}

Level 1	→	Level 1
x_{delay}	→	

Keyboard Access: ◀ I/O PRINT PRTPA DELAY

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 (nonzero) in reserved variable *IOPAR*, then XON/XOFF handshaking controls data transmission and the delay setting has no effect.

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*.

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: Clears user-defined key assignments.

Level 1	→	Level 1
x_{key}	→	
{ x_{key1} ⋯ x_{keyn} }	→	
0	→	
'S'	→	

Keyboard Access: ↰ MODES KEYS DELK

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 are stuck in User mode—probably with a "locked" keyboard—because you have reassigned or suppressed the keys necessary to cancel User mode, do a system halt ("warm start"): press and hold ON and 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

DEPND

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

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

Keyboard Access: ↰ PLOT PPAR DEPN

Affected by Flags: None

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

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

The default entry is *Y*

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

is tested, and for plot type DIFFEQ, where it specifies the initial solution value and absolute error tolerance.

Related Commands: INDEP

DEPTH

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

Level 1	→	Level 1
	→	*n*

Keyboard Access: ◀ STACK DEPTH

Affected by Flags: None

DET

Determinant Function: Returns the determinant of a square matrix.

{ }

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

Keyboard Access: MTH MATR NORM NXT DET

Affected by Flags: Tiny Element (−54)

Remarks: The argument matrix must be square. DET computes the determinant of 1×1 and 2×2 matrices directly from the defining expression for the determinant. DET computes the determinant of a larger matrix by computing the Crout LU decomposition of the

matrix and accumulating the product of the decomposition's diagonal elements.

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

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

Example: For a square matrix A, the *minor* of element a_{ij} is the determinant of the submatrix that remains after deleting row i and column j from the original matrix. Given a square matrix in level 3, i in level 2, and j in level 1, the following program, *MINOR* determines the minor of the submatrix:

```
« → M row col
  « M row ROW- DROP col COL- DROP DET
  »
»
```

For example, entering `[[ 1 2 3 ][ 4 5 6 ][ 7 8 9 ]] 2 3 MINOR` returns −6.

Related Commands: CNRM, CROSS, DOT, RNRM

DETACH

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

{}

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

Keyboard Access: ◀ LIBRARY DETACH

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

DIAG→

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

Level 2	Level 1	→	Level 1
[*array*]$_{diagonals}$	{ *dim* }	→	[[*matrix*]]

Keyboard Access: MTH MATR NXT DIAG→

Affected by Flags: None

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

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

Related Commands: →DIAG

→DIAG

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

{}

Level 1	→	Level 1
[[*matrix*]]	→	[*vector*]$_{diagonals}$

Keyboard Access: MTH MATR NXT →DIAG

Affected by Flags: None

Remarks: The input matrix does not have to be square.

Related Commands: DIAG→

DIFFEQ

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

Keyboard Access: ◀ PLOT PTYPE DIFFE

Affected by Flags: None

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

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

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

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

  If *axes* contains any strings other than "0", "1", or "n", the DIFFEQ-plotter uses the default strings "0" and "1", and plots the independent variable on the horizontal axis and the dependent variable on the vertical.
- *ptype* is a command name specifying the plot type. Executing the command DIFFEQ places the command name DIFFEQ in *PPAR*.
- *depend* is a list, { 'Y' y_0 x_{ErrTol} }, containing a name that specifies the dependent variable (the solution), and two numbers that specify the initial value of Y and the global absolute error tolerance in the solution Y. The default values for these elements are { 'Y' 0 .0001 }.

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

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

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

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

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

Related Commands: AXES, CONIC, FUNCTION, PARAMETRIC, POLAR, RKFSTEP, RRKSTEP, TRUTH

DISP

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

{}

Level 2	Level 1	→	Level 1
obj	*n*	→	

Keyboard Access: PRG NXT OUT DISP

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 multiline display format. If the object display requires more than one display line, the display starts in line *n*, and continues down the display either to the end of the object or the bottom of the display.

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

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

DO

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

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

Keyboard Access: PRG BRCH DO DO

Affected by Flags: None

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

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

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

Example: The following program counts down from 100 to 0 and leaves the integers 100 to 0 on the stack:

```
«
 100 'A' STO A
  DO 'A' DECR
  UNTIL 'A==0'
  END
 'A' PURGE
»
```

Related Commands: END, UNTIL, WHILE

DOERR

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

{}

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

Keyboard Access: PRG NXT ERROR DOERR

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.
- "*error*" displays the contents of the string. (A subsequent execution of ERRM returns "*error*". ERRN returns # 70000h.)
- 0 abandons program execution without displaying a message—0 DOERR is equivalent to pressing CANCEL.

Example: The following program takes a number from the stack and returns an error if the number is greater than 10:

```
« → X
 CASE
 'X>10'
 THEN "X IS TOO BIG" DOERR END
»
```


Related Commands: ERRM, ERRN, ERR0

DOLIST

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

Level n...Level 3	Level 2	Level 1	→	Level 1
{ *list* }$_1$ ... { *list* }$_n$	*n*	≪ *program* ≫	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*command*	→	{ *results* }
{ *list* }$_1$ ... { *list* }$_n$	*n*	*name*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	≪ *program* ≫	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	*command*	→	{ *results* }
{ *list* }$_1$ ...	{ *list* }$_n$	*name*	→	{ *results* }

Keyboard Access: PRG LIST PROG DOLIS

Affected by Flags: None

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

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

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

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

Example: ⟨ 1 2 3 ⟩ ⟨ 4 5 6 ⟩ ⟨ 7 8 9 ⟩ 3 « + + » DOLIST returns ⟨ 11 26 45 ⟩.

Related Commands: DOSUBS, ENDSUB, NSUB, STREAM

DOSUBS

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

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

Keyboard Access: PRG LIST PROC DOSUB

Affected by Flags: None

Remarks: The real number *n* can be omitted when the level 1 argument is one of the following:

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

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

During an iteration, the position of the first element used in that iteration is available to the user using the command NSUB, and the number of groups of elements is available using the command

ENDSUB. Both of these commands return an Undefined Local Name error if executed when DOSUBS is not active.

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

Examples: ⟨ A B C D E ⟩ « − » DOSUBS returns
⟨ 'A−B' 'B−C' 'C−D' 'D−E' ⟩.

⟨ A B C ⟩ 2 « DUP * * » DOSUBS returns
⟨ 'A*(B*B)' 'B*(C*C)' ⟩.

Entering

```
⟨ 1 2 3 4 5 ⟩
« → a b

  « CASE 'NSUB==1' THEN a END

    'NSUB==ENDSUB' THEN b END

    'a+b' EVAL

    END

  »

»

DOSUBS
```

returns
⟨ 1 5 7 5 ⟩.

Related Commands: DOLIST, ENDSUB, NSUB, STREAM

DOT

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

{}

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

Keyboard Access: (MTH) MATRX DOT

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 $1 \times 4 + 2 \times 5 + 3 \times 6$).

Related Commands: CNRM, CROSS, DET, RNRM

DRAW

Draw Plot Command: 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.

Keyboard Access: (←)(PLOT) DRAW

Affected by Flags: Simultaneous or Sequential Plot (−28), Curve Filling (−31)

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. DRAW does not draw axes—execute DRAX to do so.

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

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

DRAX

Draw Axes Command: Draws axes in *PICT*.

Keyboard Access: (←)(PLOT) DRAX

Affected by Flags: None

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

Related Commands: AXES, DRAW, LABEL

DROP

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

Level 1	→	Level 1
obj	→	

Keyboard Access: (←)(DROP)

Affected by Flags: None

Related Commands: CLEAR, DROPN, DROP2

DROPN

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

Level n+1 ... Level 2	Level 1	→	Level 1
obj_1 ... obj_n	n	→	

Keyboard Access: ◀ STACK NXT DROPN

Affected by Flags: None

Related Commands: CLEAR, DROP, DROP2

DROP2

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

Level 2	Level 1	→	Level 1
obj_1	obj_2	→	

Keyboard Access: ◀ STACK NXT DROP2

Affected by Flags: None

Related Commands: CLEAR, DROP, DROPN

DTAG

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

{}

Level 1	→	Level 1
:tag:obj	→	*obj*

Keyboard Access: PRG TYPE NXT DTAG

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

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

Keyboard Access:

Pressing ENTER duplicates the item on level 1.

◀ STACK NXT DUP

Affected by Flags: None

Related Commands: DUPN, DUP2, PICK

DUPN

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

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

Keyboard Access: ◀ STACK NXT DUPN

Affected by Flags: None

Related Commands: DUP, DUP2, PICK

DUP2

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

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

Keyboard Access: ◀ STACK NXT DUP2

Affected by Flags: None

Related Commands: DUP, DUPN, PICK

D→R

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

{}

Level 1	→	Level 1
x	→	$(\pi/180)\ x$
'$symb$'	→	'D→R($symb$)'

Keyboard Access: MTH REAL NXT NXT D→R

Affected by Flags: Numerical Results (−3)

Remarks: This function operates independently of the angle mode.

Related Commands R→D

e

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

Level 1	→	Level 1
	→	'e'
	→	2.71828182846

Keyboard Access:

α ◀ E ENTER

MTH NXT CONS e

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