HP-28S Quick Reference

General

List of contents	See at the end of this document.
HP-28S	Famous calculator made by Hewlett-Packard from 1988 – 1992.
Memory	32768 bytes for stack, programs and data. Approximately 31.6 kBytes available to the user.
Contrast adjustment	Press and hold ON then press + or – to change the contrast.
Number resolution	• 56 bit for real numbers, 12 decimal digits of precision, exponent range ±499 • 64 bit for binary numbers
The Stack	The HP-28S is a stack-based calculator. For details on using the stack see **STACK Menu**.
Commands and Menus	• Commands can be entered by typing their name explcitly. • Most commands and functions are organized in menus, some are directly available on the keyboard. See **Direct Key Commands**. • SYSEVAL is the only command which neither accessible thru the keyboard not thru a menu. See **System Operations**. • Some commands are present in multiple menus.
Endless-loop interruption	See **System Operations**. Note that a program or other lengthy operations can usually be interrupted by pressing "ON".
Audible feedback	Can be turned off by clearing flag 51, see **Flags**.
HP-28C	The main difference is the smaller memory of only 2 kByte. And whereas the HP-28S comes with a software version "2BB" it is "1BB" for the HP-28C.
Manuals	At least four editions of the Reference Manual exist: Version 1 dated October 1987 to version 4 dated November 1988. This Quick Reference is based on my experience with a HP-28S version "2BB". It is best described by edition 4 of the Reference Manual.
Author	A. Thimet, V1.0, © 2003, all right reserved.

Direct-Key Commands

General	The following section gives a description of useful commands that are directly accessible from the keyboard. • Commands are listed in the order as they show up on the calculator keyboard, from top left ("A") to bottom right ("+"). • On the calculator menus are generally indicated by a white label backround. • In the following text menus are indicated by bold-faced print in the left column. • References to other sections of this Quick Reference are also printed bold-faced. • When a menu key is pressed it displays six "soft labels" on the bottom of the LCD screen which are associated with the white top row keys below the display. Pressing one of these keys will activate the command written on the soft label. • Pressing "<>" to the right of the red SHIFT key will remove the soft labels from the LCD display and the white keys beneath will resume their cursor-control meanings printed in white above the buttons. See **CURSOR Menu**.
Command line editing	Pressing a command key performs different actions depending on the current input mode: • If no command line is currently being edited and Alpha Mode (see below) is not active the command is immediately applied to the stack contents. • If command line editing is in progress some commands will evaluate the entire command line and produce immediate results. These commands (ie. STO) perform an implicit ENTER. • If Alpha-Mode is active or if a command is typed in explicitly or if a command key is pressed which doesn't perform an implicit ENTER (ie. "+") then the command word or symbol is appended to the command line. To abort editing without executing any commands press ON.
ARRAY	Vector and matrix creation, manipulation and operations. See **ARRAY Menu**.
BINARY	Binary number bases (bin, dec, oct, hex) and operations including rotation. See **BINARY Menu**.
COMPLX	Complex number creation and operations. See **COMPLX Menu**.
STRING	String functions and conversions. See **STRING Menu**.
LIST	List creation and manipulation. See **LIST Menu**.
e	When 'e' (lower case E) is converted into a number using →NUM it evaluates to 2.71828… If flag 35 is clear then 'e' immediately evaluates to its numeric value. See **Flags** and **Evaluation Rules**.

REAL	Functions for real number arguments. See **REAL Menu**.
STACK	Stack manipulation. See **STACK Menu**.
STORE	Storage arithmetic. See **STORE Menu**.
MEMORY	Memory display, MENU management, paths and directories. See **MEMORY Menu**.
i	When 'i' (lower case I) is converted into a number using →NUM it evaluates to the complex number (0,1). If flag 35 is clear then 'i' immediately evaluates to its numeric value. See **Flags** and **Evaluation Rules**.
ALGEBRA	Symbolic formulae manipulation and Taylor series expansion. See **ALGEBRA Menu**.
STAT	Summation and statistics. See **STAT Menu**.
PRINT	Printing and printer control. See **PRINT Menu**.
CONTRL	Programming: Control functions, text display, sound. See **CONTRL Menu**.
BRANCH	Programming: Branch and loop instructions. See **BRANCH Menu**.
TEST	Programming: Flag manipulation and various tests. See **TEST Menu**.
CATALOG	Displays a list of all built-in functions. See **CATALOG Menu**.
UNITS	Displays a list of all built-in units. See **UNITS Menu**.
≤ ≥ < > == ≠	Comparisn operators. See **TEST Menu**.
→	The right arrow key above the character U is used to store local variables in a program, see **Programs**.
# " { } [ ] ( ) << >>	Delimiters required to enter certain kinds of data types. Note that trailing delimiters are automatically added. See **Data Types**.
NEWLINE	Used to insert a line break in a program (ENTER cannot be used because it would push the progam onto the stack)
LC	Toggle between lower and upper case characters
α	Alpha-Mode: While editing the command line many command keys will immediately evaluate the command line and produce a result (implicit execution of ENTER). However, in Alpha-Mode the command key symbols will be added to the command line und evaluation occurs only when pressing the ENTER key explicitly.
MENUS	Menu-lock. • When active, pressing a character key with an associated menu activates the menu (ie. pressing "F" activates the REAL menu). To select the character, press shift-key. • When inactive pressing a character will return the character and shift-key activates the menu. Note that the status of Menu-lock is not indicated in the LCD display!
INS	Toggle command line editing mode between insert and overwrite. Default is overwrite. See **CURSOR Menu**.
DEL	Delete character under cursor in editing mode. See **CURSOR Menu**.
← → ↑ ↓	Cursor movement in editing mode. See **CURSOR Menu**.
<>	Activates the **CURSOR Menu**, see there.
MODE	Display and angle modes and various general settings. See **MODE Menu**.
TRIG	Trigonometric functions. See **TRIG Menu**.
LOGS	Logarithms and exponential functions. See **LOGS Menu**.

SOLV	Numerical solver and root finder. Symbolic solver for quadratic formulas. See **SOLV Menu**.
PLOT	Plotting curves on the LCD display. See **PLOT Menu**.
USER	Display user variables and progams in the current directory. See **USER Menu**.
CUSTOM	Custom menu. See **CUSTOM Menu**.
NEXT, PREV	Display next or previous set of menu soft-labels if a menu is active. These soft-labels are displayed on the bottom of the LCD screen. When a menu is active the cursor keys printed above of the top row of white keys beneath the LCD display are not available. Rather, the command indicated by the soft-label will be executed when the corresponding white key is pressed.
ENTER	• When a command line is present its contents are evaluated and commands executed. • Without a command line this performs a DUP and pushes the stack. See **STACK Menu**. • Note that ENTER is not a command! It cannot be used in a program. It is only a command to the calculator to evaluate the command line.
EDIT	Edit the contents of stack level 1. After pressing ENTER the modified data overwrites the old contents.
CHS	Change sign of number. In a program enter NEG to negate stack level 1.
VIEW↑	Change the visible section of the stack so that higher stack contents are visible. Useful when the stack contains many elements. Cannot be used in programs.
EEX	Enter exponent for number. If not in edit mode "1E" is put into the command line.
VIEW↓	Change the visible section of the stack so the lower stack contents are visible. Cannot be used in programs.
DROP	Drop the stack and discard contents of stack level 1. See **STACK Menu**.
ROLL	Move a specified stack object to level 1. See **STACK Menu**.
⇐	Delete character to the left in editing mode. This does never delete the element in stack level 1.
SWAP	Exchange stack level 1 and 2. See **STACK Menu**.
'	Name or equation delimiter. See **Data Types**. Note that trailing delimiters are automatically added.
VISIT	• Put the *contents* of a variable for editing into the command line. Ie. `'A' VISIT` puts the contents of variable A into the command line for editing. • If the argument is a number the contents of the corresponding stack level are retrieved for editing. Ie. `3 "A" 2 1 3 VISIT` puts the contents of stack level 3 ("A") into the command line for editing. To abort the VISIT operation press ON. To keep modifications press ENTER. This will store the modified data in the variable or the earlier specified stack level.
COMMAND	ENTER (or any other command that involves execution of ENTER) stores a copy of the current command line provided MODE CMD has been activated (see **MODE Menu**). COMMAND can be used to retrieve this stored command line for editing.

UNDO	ENTER (or any other command that involves execution of ENTER) stores a copy of the current stack before it executes provided MODE UNDO has been activated (see **MODE Menu**). UNDO recalls the previousely stored stack layout. Note that this feature potentially requires a lot of memory!
LAST	When a command takes arguments from the stack those arguments will be saved provided MODE LAST has been specified (see **MODE Menu**). LAST retrieves these saved arguments and pushes them back onto the stack. Note that the number of saved stack arguments depends on the command. If MODE LAST is active and a command produces an error then the stack is automatically restored. If MODE LAST is not active these arguments are lost.
1/x	Reciprocal of numbers or matrices. Displayed in equations and programs using the `INV()` function notation.
STO	Stores the object in stack level 2 in the variable who's quoted name is given in stack level 1. Ie. `5 'A' STO` stores 5 in variable A and drops both objects from the stack. See **STORE Menu**.
RCL	Recall variable and push it onto the stack. This does not evaluate the contents of the variable or execute a program. The quoted variable name is replaced by the recalled object. See **STORE Menu**.
PURGE	Delete variable or program who's name is given in stack level 1. This command can operate on lists of names! To erase all variables of the current directory use `MEMORY VARS PURGE` Warning: VARS also returns subdirectory names so in the above example all subdirectories will be returned as well!! See **STORE Menu**.
∫	Numeric or symbolic integration. See **Integration**.
d/dx	Symbolic differentiation. See **Differentiation**.
^	Exponential function. Ie. `-2 3 ^` returns –8 in stack level 1. Accepts real and complex numbers.
EVAL	Evaluate quoted name or program in stack level 1. See **Evaluation Rules**.
→NUM	Same as EVAL but also converts a symbolic name into a number. Ie. `3 π *` results in `'3*π'` and →NUM converts this into 9.424…
CONT	Continue an interrupted program. See **Programs**.
%	Percentage. Note that different from other HP calculators this does drop the stack.
%CH	Percentual difference from contents in stack level 2 to contents in stack level 1.
√X	Square root. Displayed in equations and programs using the square root symbol "√".
ON	Turns calculator on, clears errors displays, aborts command line editing, interrupts programs. ON never discards data from the stack.
OFF	Turns calculator off. It automatically turns itself off after a few minutes of inactivity.
CLEAR	Clear the stack. See **STACK Menu**.
CONVERT	Convert between different units. See **UNITS menu**.

Data Types

General	• All of the data types described below can be stored on the stack and in variables. • Special delimiters are used to denote different kinds of data types. • Each data type has a type-identifier, see **TEST Menu**.
Real numbers	`3.4567E12` See **REAL Menu**.
Complex numbers	`(2.3,4.5)` where 4.5 is the imaginary part. See **COMPLX Menu**. • When using a comma as decimal separator this must be entered as `(2,3.4,5)` • Instead of the separator symbol a SPACE can be used! • Note that it is not possible to refer to variables when constructing a complex number: `(X Y)` will cause an error.
Binary numbers	`#123456` See **BINARY Menu**.
Strings	`"This is a string!"` See **STRING Menu**.
Real fields	`[1,2,3,4]` or `[[1,2] [3,4]]` • Can be a vector or matrix. See **ARRAY Menu**. • Note that it is not possible to refer to variables when constructing a field: `[X Y]` will cause an error.
Complex field	`[(1,2) (3,4) (4,5)]` or `[[(1,2) (2,3)] [(4,5) (5,6)]]` Can be a vector or matrix. See **ARRAY Menu**.
List	`{ 1 A B "String" }` A list of objects. See **LIST Menu**. Note that almost everything can be put in a list: `{* = ^}` This is important when doing symbolic manipulations on equations, see **ALGEBRA Menu**. Object delimiters cannot be put in a list.
Names	`'X2'` Used to reference stored variables of the above data types. When a number is put in single quotes the plain number is used. Other data types cannot be put in quotes, ie. `'[1]' ENTER` will cause an error but `'1.5E3'` will not.
Expressions	`'A+B'` or `'C=A+B'` Note that like other data objects expressions can be stored in variables!
Program	`<< 3 * >>` A series of program instructions. See **Programs**.

Programs

General	• A program is a series of commands surrounded by << and >> brackets. These symbols are located next to the SPACE key. • The programming language is called RPL: Reverse Polish Lisp. It is stack based with a support for many data types. The HP-28C/S was the first calculator to use RPL. Later models like the HP-48 and HP-49 used it as well. • Control instructions are described in the **CONTRL Menu** • Branching instructions are descibed in the **BRANCH Menu** • Flag manipulation and other program commands are described in **MENU Test** • Programs can be stored in variables like other objects. See **Data Types**. • There is no GOTO available. Use structured programming instead. • Programs can be interrupted by pressing "ON".
Program example	`<<ROT * SWAP 2 / CHS DUP SQ ROT - √ >> 'QE' STO` • Program QE takes 3 input values from the stack which represent coefficients a, b and c of a quadratic equation. • The program returns two values r1 and r2 on the stack. The two solutions of the quadratic equation can be calculated as r1+r2 and r1-r2.
Local Variables	A program can have local variables. Using local variables avoids conflicts with global variable names. Example: `<<→ x y <<x y + LN>> >> 'P' STO` This creates the program and stores it in a variable called P. **Important**: The SPACE after the "→" is required! The program takes two arguments from the stack and puts them into the local variables x and y. The return value is ln(x+y). Example: `1 2 P` returns 1.0986… This program could also be entered in the form of an expression: `<< → x y 'LN(x+y)' >> 'P' STO` Both the program and expression form allows to invoke the program in functional notation. Example: `'P(1,2)' EVAL` also returns 1.0986… **Important**: For some reason the sequence `P(1,2) ENTER` will not work but rather issue and error.
Editing a program	• Use `'P' VISIT` to bring back the program into the command line for editing. • Use NEWLINE to enter line breaks to make the program code more readable.
Comparisn operators	> ≥ < ≤ == ≠ and flag checking commands return either 0 or 1 onto the stack and can be used to steer branch instructions. Note that the values to be compared must be present on the stack. Example: `4 5 >` returns 0. For more details see **TEST Menu**.

Subroutines	Simply specify the name of the program to execute. Example:
	`<< SQ LN 1 + >> 'P1' STO`
	`<< P1 SWAP P1 + >> 'P2' STO`
	When P2 is invoked it calls P1 with the values in stack level 1 and 2 and adds the results that P1 produced (which is ln(x²)+1).

ARRAY Menu

General	• Arrays (or fields) are either vectors or matrices. • Arrays can be real or complex. • If an array contains a single complex value the entire array is automatically complex. • Lengthy array operations can be interrupted by pressing ON • Arrays are entered by using square brackets [and]. • Example 2x2 matrix: `[[1 2] [3 4]]`
+ −	Add/substract vectors or matrices of matching dimensions. This also works on mixed real/complex arguments but if a complex argument is involved the result will always be complex.
*	Multiplication. Either operand may be real or complex: • Multiply vector by number or number by vector → vector • Multiply matrix by number or number by matrix → matrix • Multiply matrix by vector → vector • Multiply matrix by matrix → matrix
÷	• Calculate matrix X so that M1*X=M2 where M1 and M2 are matrices in stack level 1 and 2. Or: `B A ÷` calculates X=B/A so that AX=B. • Calculate vector X so that M1*X=V2 where M1 and V2 are the matrix and vector in level 1 and 2. Or: `V M ÷` calculates X=V/M so the M*X=V. These operations produce more accurate results then using the INV command on matrices. The matrices must be square. Can often be used even if the matrix A or M is singular and thus solves systems where the number of variables does not match the number of equations.
INV (1/x)	Returns inverse of square matrix.
SQ (x²)	Returns the square of a square matrix.
→ARRY	Convert stack values into a matrix or vector: • `X1 X2 … Xn n →ARRY` results in vector [X1 X2 … Xn] • `X11 X12 … Xnm {n m} →ARRY` results in matrix [[X11…X1m] … [Xn1…Xnm]] • Note that combining a number of vectors into a matrix is not possible! • An error occurs if the stack doesn't hold enough values for the matrix or if they are not of numerical type. • If any one value on the stack is complex the resulting array will be complex.
ARRY→	The inverse operation of →ARRY. Vector and matrix dimensions are returned as a number or length-2 list in stack level 1.

PUT	Replace value of a matrix or vector: • `V {idx} X PUT` puts the number X into vector V at position idx and returns the modified vector in level 1. • `M {row col} X PUT` puts the number X into matrix M at position (row,col) and returns the modified matrix in level 1. • `'Nam' {idx} X PUT` puts the number X into vector named Nam at position idx and returns nothing. • `'Nam' {row col} X PUT` puts the number X into the matrix named Nam at position (row,col) and returns nothing. Note that you cannot put a complex number into a real matrix! Vector and matrix indices count from 1.
GET	Inverse operation of PUT: • `V {idx} GET` pushes the number at position idx in vector X onto the stack. • `M {row col} GET` pushes the number at position (row,col) in matrix M onto the stack. • `'Nam' {idx} GET` pushes the number at position idx in vector named Nam onto the stack. • `'Nam' {row col} GET` pushes the number at position (row,col) in matrix named Nam onto the stack. Vector and matrix indices count from 1.
PUTI	This is similar to PUT but it does not discard the index value but rather increments it (including row wrap) and returns it on stack level 1. Example: `V {idx} X PUTI` puts the number X into vector V at position idx and returns the modified vector (or its name) in level 2 and {idx+1} in level 1. This greatly simplyfies the input or modification of vectors and matrices.
GETI	Reverse operation of PUTI. Example: `V {idx} GETI` returns V (or its name) on stack level 3, {idx+1} on stack level 2 and the retrieved number on stack level 1.
SIZE	Returns the size of the specified vector as a length-1 list or the size of a matrix as a length-2 list in {rows columns} format.
RDM	Redimensions a matrix or vector. Added elements are set to 0. If the new dimension is smaller than the original one then elements are discarded. It is possible to redimension a matrix into a vector and a vector into a matrix. Examples: • `[1] {2 2} RDM` redimensions the vector V into a 2x3 matrix and returns the resulting matrix on stack level 1: `'[[1 0][0 0]]'` • `'Nam' {4} RDM` redimensions the vector or matrix named Nam into a length-4 vector and returns nothing.
TRN	Transpose a nxm matrix into a mxn matrix. When operating on a variable name the name is dropped from the stack. For complex matrices the elements are also conjugated (imaginary part is negated).

CON	Create a "constant" matrix or vector where all elements have a specified value. • `{3} 5 CON` creates a length-3 vector with all elements set to 5. • `{2 3} 0 CON` creates a 2x3 matrix with all elements set to 0. • `[1 2] 7 CON` replaces all elements of the vector with value 7. • `'Nam' 2 CON` replaces all elements of the matrix or vector named Nam with value 7 and returns nothing.
IDN	Create an identity matrix (all elements 0 except for 1s in the diagonal). • `5 IDN` creates a 5x5 identity matrix • `[[1 2][3 4]] IDN` sets elements of the square matrix to identity matrix values. • `'Nam' IDN` sets elements of the square matrix named Nam to identity matrix values and returns nothing.
RSD	Returns the residual: `'B' 'A' 'X' RSD` returns B – A*X in stack level 1. A must be a matrix; B and X must be of the same type, either matrix or vector.
CROSS	Cross product of two length-3 vectors A and B returned as a length-3 vector: [A2*B3-A3*B2, A3*B1-A1*B3, A1*B2-A2*B1]
DOT	Scalar product of two equally-dimensioned vectors or matrices: • `[1 2 3] [4 5 6] DOT` returns 1*4+2*5+3*6 = 32 • `[[1 2] [3 4]] [[5 6] [7 8]] DOT` returns 1*5+2*6+3*7+4*8 = 70
DET	Returns the determinant of a square matrix.
ABS	Returns the norm of a matrix or vector. This is the square root of the sum of squares of all elements.
RNRM	Row norm of a matrix or vector. • For a vector this is the largest absolute value of all elements. • For a nxm matrix: For each row sum up the absolute values of all n row elements. Then take the largest value from these m sums. This returns a single number.
CNRM	Column norm. Same as RNRM but column-oriented. For a vector this is the sum of the absolute values of all vector elements.
R→C	Combine two real matrices or vectors into a complex matrix or vector where the field in stack level 1 will be the imaginary part.
C→R	Split a complex matrix or vector into real and imaginary part. Stack level 1 will receive the imaginary part.
RE	Return the real part of a real or complex matrix or vector.
IM	Return the complex part of a real or complex matrix or vector. For a real matrix/vector this will return a matrix/vector filled with zeros.
CONJ	Conjugate a real or complex matrix or vector. This will negate all imaginary parts. Will do nothing on a real matrix or vector.
NEG (CHS)	Negate each matrix or vector element.

BINARY Menu

General	• "Binary" numbers are unsigned integer numbers with a maximum length of 64 bit. • Binary numbers can be entered and displayed in binary, octal, decimal or hex format (don't confuse binary display mode with the binary number type!). • Binary numbers are entered using the pound sign: `#3A75C`. The digits must be valid for the selected number base. • To enter a number in a number base other than the current one use a trailing specifier: d (decimal), o (octal), h (hex), b (binary). Ie. `#3Ah`. The number will automatically be converted to the current number base. • Negative binary numbers are not supported.
+ - x ÷	These can be used on binary or mixed binary/real numbers. The result will be a binary number with the fractional part cut off. Binary and complex numbers cannot be combined.
DEC	Select decimal entry format and display all binary numbers in the stack in decimal notation with a trailing 'd'.
HEX	Select hexadecimal entry format and display all binary numbers in the stack in decimal notation with a trailing 'h'.
OCT	Select octal entry format and display all binary numbers in the stack in decimal notation with a trailing 'o'.
BIN	Select binary entry format and display all binary numbers in the stack in decimal notation with a trailing 'b'.
STWS	Use the real number N in stack level 1 to specify a new word size of N=1…64 bits. N<1 is the same as N=1 and N>64 is the same as N=64. A binary number cannot be passed to STWS!
RCWS	Return the current word size.
RL	Rotate the binary number in stack level 1 one bit left. The topmost bit becomes bit0. For this and the following commands the topmost bit is determined by the current word size.
RR	Rotate one bit right. bit0 will be the topmost bit.
RLB	Rotate ony byte left. The topmost 8 bits will become bits0-7.
RRB	Rotate ony byte right. bits0-7 will become the topmost 8 bits.
R→B	Convert real number X into binary number. If X<0 the result will be 0. If X>0xFFFFFFFFFFFFFFFF the result will be 0xFFFFFFFFFFFFFFF reduced to the currently selected word size. Note that reduction is carried out after conversion to a 64-bit integer. So if the word size is 4 and 17d is entered the result will be #1d.
B→R	Convert binary number to real. Some significant digits may be lost!
SL	Shift one bit left. Inserts zero in bit0.
SR	Shift one bit right. Inserts in the topmost bit.
SLB	Shift one byte left. Inserts zero in bit0-7.
SRB	Shift one byte right. Inserts zero in the topmost 8 bits.
ASR	Shift one bit right. Duplicates the topmost bit and discards bit 0.
AND	AND operation
OR	OR operation
XOR	XOR operation
NOT	Invert all bits

COMPLX Menu

General	• Complex numbers are entered using brackets: `(1.72 378) (2,4.5)` The left number is the real part and the right one the imaginary part. Note that either a space or the delimiter symbol (either dot or comma, depends on the current RDX setting) can be used to separate the real and imaginary part. • Among others the following operations can be performed on complex numbers: • `+ − x ÷ INV` Simple arithmetics, inverse (1/x) • `SQ √ ^` Square (x^2), square root and exponential • `SIN COS TAN` Trigonometric functions and their inverse • `SINH COSH TANH` Hyperbolic functions and their inverse • `EXP LN LOG ALOG` Logarithms and their inverse
R→C	Combine two real numbers in stack level 1 and 2 to a complex number where the number in stack level 1 will be the complex part.
C→R	Split complex number into real part (stack level 2) and imaginary part (stack level 1)
RE	Return real part of complex or real number.
IM	Return complex part of complex or real number. For a real argument this will always be 0.
CONJ	Conjugate a complex or real number (negate the imaginary part). Performs no action on a real number.
SIGN	Return unary vector in the direction of the complex number: `(x/sqrt(x*x+y*y), y/sqrt(x*x+y*y))`
R→P	Convert from rectangular to polar coordinates. See **TRIG Menu**.
P→R	Convert from polar to rectangular coordinates. See **TRIG Menu**.
ABS	Absolute value of complex or real number. For a complex number this is `sqrt(re*re+im*im)`
NEG	Negative value of complex or real number.
ARG	Returns the angle θ of the complex number (x,y) vector with the x-axis. • `X≥0: θ=atan x/y` • `X<0: θ=atan x/y + π*sign(y)`

STRING Menu

General	• Strings are entered in double quotes: `"This is a string."` • The length of a string is only limited by the available memory. • Strings are based on all 255 ASCII characters.
+	Concatenate strings in stack level 1 and 2.
→STR	Convert any object type in stack level 1 into a string. The conversion preserves the current display format including multi-line mode. NEWLINE symbols inside the string are displayed as ▪. If the object in level 1 is a string no additional quotes are added.
STR→	Convert a string back into objects and evaluates them. `"3 4 + 10 *" STR→` evaluates the commands in the string and produces 70. This is essentially what ENTER does with the command line.
CHR	Convert ASCII character code in stack level 1 into a string. Note that the command does not accept binary numbers!
NUM	Return ASCII code of the first character of the string in stack level 1 as a real number.
→LCD	Writes the data of the given string into the LCD pixel memory. Each characters inside the string represents 8 pixel. Bit0 of the first character represents the pixel in the very top left corner. Bit1 of the first character the pixel below etc. Bit0 of the 2nd character represents the 2nd pixel from left at the very top of the display. If the string does not contain enough characters to fill the entire LCD screen then the remaining pixels are unchanged.
LCD→	Returns a 548 byte string representing the pixel data of the LCD screen. Each character receives the data of an 8 pixel-column, starting with the column in the top left corner of the LCD screen. The screen itself is 137x32 pixel in size.
POS	Seach for the string given in level 1 within the string in level 2 and return the position where the string was found or 0 if not found. `"This is a string" "str" POS` returns 11.
SUB	Returns a substring of the string in level 3. The numbers in level 2 and level 1 specify the start and end position of the substring (counting from 1). `"This is a string" 3 7 SUB` returns "is is". Start and end positions cannot be specified in a length-2 list.
SIZE	Returns the length of the string.
DISP	Display the string in level 2 on the LCD display line given in level 1 (1…4). See **CONTROL Menu**.

LIST Menu

General	• A list is a sequence of arbitrary objects which need not be of the same type. • Lists are entered by using curly brackets: `{1 (2,3) 5 "A" Q [7 8] {a b c}}.` • A lists may be put inside another list. • In order to avoid evaluation of names when they are typed on the command line to be put in a list they can be entered in single quotes or with Alpha-Mode turned on. The quotes are removed when the list is created.
+	Used to append objects of any type to a list: • `"Hi!" {1 2 3} +` returns `{"Hi!" 1 2 3}` • `{3 4 5} [7 8 9] +` returns `{4 5 6 [7 8 9]}` • `{1 2 3} {a b c} +` returns `{1 2 3 a b c}` rather than `{{1 2 3} a b c}`
→LIST	Combine n elements on stack levels 2…n+1 into a list. n must be given on stack level 1. Some of the list components my be lists themselves: `{1 2 3} {a b c} 2 →LIST` returns `{{1 2 3} {a b c}}`
LIST→	Splits a list into individual elements on the stack. The length of the list is returned on stack level 1.
PUT	Put an element into a list at the given position. Similar to PUT for matrices. • `{1 2 3} 2 'X' PUT` puts the element X into the list at position 2 and returns the modified list `{1 X 3}` • `'Nam' 2 'X' PUT` puts the element X into the list named Nam at position idx and returns nothing. List indices count from 1 and must be within range. Note that this is an overwrite operation, not an insert!
GET	Inverse operation of PUT: `{A B C} 2 GET` pushes the element B at position 2 onto the stack. B is not evaluated but rather returned as the name `'B'`.
PUTI	Put an element into a list at the given position and increase the position index. Similar to PUTI for matrices. Example: `{1 2 3} 1 'X' PUTI` puts the element X into the list at position 1 and returns the modified list (or its name) in level 2 and 2 (the new index) in level 1: `{X 2 3} 2`. This greatly simplyfies the input or modification of a list. The index automatically wraps around.
GETI	Inverse operation of PUTI. Example: `{1 2 3} 3 GETI` returns the list (or its name) on stack level 3, 1 (the incremented and wrapped index) on stack level 2 and the retrieved element on stack level 1. The index automatically wraps around.
POS	Searches for an element within a list. Example: `{1 (2,3) 5 "A"} 5 POS` returns 3 because the real number 5 can be found at position 3.
SUB	Return a sub-list from a given start index up to a given end index. Example: `{1 (2,3) 5 "A" 'Q'} 2 3 SUB` returns `{(2,3) 5}`. The start and end index cannot be specified in a list.

SIZE	Returns the size (number of elements) of the list. A list within a list counts as one list element.

REAL Menu

General	• Note that various REAL functions are directly accessible on they keyboard. See **Direct-Key Commands**. • Real numbers are entered without special delimiter: `3.5721E10`
NEG	Negates object. This can be a real or complex number or a real or complex matrix or vector.
FACT	Calculates n! for integer n or $\Gamma(x+1)$ for fractional x. Works for non-integer negative numbers but does not work for complex numbers. Use the following program IFAC to find $\Gamma^{-1}(x+1)$: `<<→ x <<'FACT(Y)-x' 'Y' 5 ROOT>> >> 'IFAC' STO` You can verify that Γ`(120.56417111)=1E200`
RAND	Return the next random number in the range $0 \le x < 1$.
RDZ	Takes a real number as the initializer for the random number generator. When 0 is specified the elapsed time since power-on is used.
MAXR	Largest positive real number: 9.9999999999E499
MINR	Smallest positive real number: 1.00000000000E-499
ABS	Absolute value of a real or complex number or a real or complex matrix or vector. See also **COMPLX Menu** and **ARRAY Menu**.
SIGN	Sign of a real or complex number. For the sign of a complex number see **COMPLX Menu**.
MANT	Mantissa of a real number.
XPON	Exponent of a real number.
IP	Integer part of a real number.
FP	Fractional part of a real number.
FLOOR	Return largest number $\le$ x.
CEIL	Return smallest number $\ge$ x.
RND	Perform rounding of a real or complex number or a real or complex matrix or vector according to the number of significant digits specified in the current FIX, SCI or ENG display mode. In STD mode no rounding occurs.
MAX	Return larger of the two real numbers in stack level 1 and 2.
MIN	Return smaller of the two real numbers in stack level 1 and 2.
MOD	Returns reminder of division of real numbers in levels 1 and 2. This is defined as: `x - y*floor(x/y)`
%T	Calculates percentage of total: `100*y/x`

STACK Menu

General	The stack of the HP-28S behaves similar to the stack of many other RPN (Reverse Polish Notation) calculators like the HP-41. However, there are important differences:
	• In theory the stack can hold an arbitrary number of elements. (Practically, the number is limited by the available memory.)
	  • In particular, the stack can be empty. In this case commands that take arguments from the stack will cause an error. This is different from the 4-level stack used in other HP calculators: There the stack registers always contain numbers.
	  • When dropping data from the stack the content of the highest stack level is not duplicated. Thus, it is not possible to perform "calculations with a constant" as usual.
	  • To avoid a rapidly growing stack virtually all comands remove all of their arguments from the stack before the results are pushed onto the stack.
	• Different from normal RPN calculators there is a command line. It supports advanced editing features but also introduces slight differences in behaviour as compared to normal RPN, see example below.
	Periodically erase unneeded stack objects (use CLEAR located on the "0" key) because a large number will slow down execution speed.

Examples	All examples assume an initially empty stack. `1 2 +` results in 3. Except for the result the stack is empty. `1 2 + 'X' STO` stores the result (3) in variable X. The stack is empty because like all other commands STO removes its arguments from the stack (the value and the variable name). `1 ENTER 2 ENTER +` results in 3 and an otherwise empty stack. On a normal RPN calculator the result would be 4 in stack level 1 and 1 in level 2. This is because the 2nd ENTER moves the input value (2) from the command line to stack level 1 only. `1 ENTER 2 ENTER ENTER +` returns 1 and 4 because the 3rd ENTER acts as a DUP which duplicates the element in stack level 1. `1 ENTER 2 DUP +` also returns 1 and 4 because DUP explicitly duplicates the 2. Note that ENTER is not a command! It merely tells the calculator to evaluate the command line. If the command line is empty it executes the DUP command as a convenience.
DROP	Above the "9" key: Discard the object in stack level 1 and shift all other values one stack level down.
SWAP	Shifted "⇐" key: Exchange the object in level 1 and 2 without evaluating them.
ROLL	Shifted DROP key: Move a specified stack object onto the top of the stack. Example: `10 20 30 40 50 3 ROLL` Moves the 3rd stack object (30) to the top of the stack. After the operation the stack looks like this: `10 20 40 50 30`
DUP	Same as ENTER with an empty command line: Shift up objects in the stack by 1 level. The object in level 1 is duplicated into level 2.
OVER	Pushes a copy of the element in stack level 2 onto the stack. Example: `10 20 30 40 OVER` produces `10 20 30 40 30`
DUP2	Pushes a copy of the elements in stack level 1 and 2 onto the stack. Example: `10 20 30 40 DUP2` produces `10 20 30 40 30 40`

DROP2	Discards stack elements in level 1 and 2 and rolls down the stack.
ROT	Rotates the elements in the first three stack levels up. This is equivalent to "3 ROLL". Example: `10 20 30 40 ROT` produces `10 30 40 20`
LIST→	See **LIST Menu**.
ROLLD	Moves the element on top of the stack to a higher stack position. This is the inverse operation of ROLL. Example: `10 20 30 40 50 3 ROLLD` produces `10 20 50 30 40`
PICK	Push a copy of the given stack level onto the stack. Note that "1 PICK" is equivalent to DUP and "2 PICK" is equivalent to "OVER". Example: `10 20 30 40 3 PICK` produces `10 20 30 40 20`
DUPN	Duplicate the given number of stack elements onto the top of the stack. "1 DUPN" is equivalent to DUP and "2 DUPN" is equivalent to "DUP2". Example: `10 20 30 40 3 DUP` produces `10 20 30 40 20 30 40`
DROPN	Drop a given number of objects from the stack. "1 DROPN" is equivalent to DROP and "2 DROPN" is equivalent to DROP2. Example: `10 20 30 40 3 DROP` produces `10`
DEPTH	Returns the number of elements in the stack. Example (beginning with an empty stack): Example: `10 20 30 40 DEPTH` returns 4.
→LIST	Create a list from stack elements. See **LIST Menu**.

STORE Menu

General	• All data types (see **Data Types**) can be stored in named variables. The number of variables is only limited by the available memory. • The USER key displays the USER menu with all the variables (and – since a program can be stored in a variable – the programs) of the current directory. See **USER Menu**. • Note that the USER menu's soft labels on the bottom of the LCD screen only show the first few characters of a variable in *upper case*. • Variable names are case sensitive! • Variable names can be up to 127 characters long. • Use PURGE to erase variables. • See **MEMORY Menu** for directory issues. • Unfortunately, storage arithmetic commands (STO+, STO* etc.) cannot operate on local variables! So their "shortcut effect" is really lost. See **Progams**.
STO	Stores the object in stack level 2 in the variable who's quoted name is given in stack level 1. Ie. `5 'A' STO` stores 5 in variable A and drops both objects from the stack.
RCL	Recall variable and push it onto the stack. This does not evaluate the contents of the variable or execute a program. The quoted variable name is replaced by the recalled object.
PURGE	Delete variable(s) or program(s) as specified in stack level 1. This command can operate on lists of names. Ie. to erase all variables of the current directory use `MEMORY VARS PURGE` To erase variable PROG: `'PROG' PURGE` To erase variables X and Y: `{X Y} PURGE`

STO+	A quoted name must be present in stack level 1 or 2. The 2nd argument (real or complex number, real or complex vector or matrix) will be added to the variable: `'A' 6 STO+` and `6 'A' STO+` calculates A+6 and stores the result in A. Note that even though "+" can be used with lists this is not supported for the STO+ command. **Note**: This command cannot operate on local variables!
STO-	A quoted name must be present in stack level 1 or 2. The 2nd argument (real or complex number, real or complex vector or matrix) will be substracted from the variable (or vice versa depending on the order of arguments): `'A' 6 STO-` Calculates A-6 and stores the result in A. `6 'A' STO-` Calculates 6-A and stores the result in A. **Note**: This command cannot operate on local variables!
STO*	A quoted name must be present in stack level 1 or 2. The 2nd argument (real or complex number, real or complex vector or matrix) will be multiplied to the variable: `'A' 6 STO*` and `6 'A' STO*` calculates A*6 and stores the result in A. **Note**: This command cannot operate on local variables!
STO/	A quoted name must be present in stack level 1 or 2. The 2nd argument (real or complex number, real or complex vector or matrix) will be divided by the variable (or vice versa depending on the order of arguments, see STO-): `'A' 6 STO/` Calculates A/6 and stores the result in A. `6 'A' STO/` Calculates 6/A and stores the result in A. **Note**: This command cannot operate on local variables!
SNEG	Negate the contents of a variable (real or complex number, real or complex vector or matrix) **Note**: This command cannot operate on local variables!
SINV	Negate the contents of a variable (real or complex number, real or complex square matrix) **Note**: This command cannot operate on local variables!
SCONJ	Conjugate the contents of a variable (real or complex number, real or complex vector or matrix). This negates the imaginary part of the value. **Note**: This command cannot operate on local variables!

MEMORY Menu

MEM	Return the amount of free memory in bytes.
MENU	Creates a customized user menu. See **CUSTOM Menu**.
ORDER	Takes a list of variable names and moves these names in the given order to the beginning of the current user menu.
PATH	Returns the current path as a list of directory names. This always starts with HOME which is the root directory. Note that HOME is a reserved name that cannot be used for a variable. Apparently, it cannot even put into a quoted name (`'HOME'`).

HOME	Return to the HOME directory.
	Note that there are no commands to step up one level in the directory hierarchy (no "CD .."). To get this functionality a user program must be written which uses PATH to get access to the directory names:
	`<< PATH DUP SIZE 1 - → P N`
	`  << 1 N FOR I P I GET EVAL NEXT >>`
	`>> 'UP' STO`
	This creates a command UP which steps up one directory.
	Note that GET retrieves the next directory name from the list and puts it onto the stack. This does not evaluate the name, hence EVAL is needed to actually change the directory. Unfortunately, there doesn't seem to be a command which activates the USER menu (or any other menu).

More about paths	• If a name (variable, program etc.) is not found in the current directory it is searched in the partent directory and so on until it is found. This also applies to directories so evaluating a directory name can not only change to a subdirectory of the *current* directory but also to a subdirectory of *any upper-level* directory. As a consequence of this path searching, the above UP program should be stored in the HOME directory so that it is accessible from all other directories! • The same applies to directory names. • Furthermore, the HOME directory should contain utility programs; regular work (which usually involves all sorts of temporary variables) should be carried out in a subdirectory. • It is possible to change to a subdirectory by specifying its name on the command line. But be aware that the subdirectory's associated menu label may not display the entire variable name or the true name may have lower case characters! • Path names need no be unique throughout the directory hierarchy.
CRDIR	Create a directory with given name underneath the current directory.
VARS	Return a list containing the names of all variables and subdirectories in the current directory.
CLUSR	Erases all variables in all directories.

ALGEBRA Menu

General	This menu contains commands for symbolic manipulation of expressions and equations. There's also an interactive equation editor available (FORM). **Important**: If flag 36 is clear the "numerical" evaluation mode is used. It evaluates expressions until a numerical result has been found. If an undefined variable is encountered an error is issued. Thus, in order to to see the symbolic results in this section flag 36 must be set.	

COLCT	Takes an equation or expression and collects similar expressions:
	`'1+2+LOG(10)'` → `'4'`
	`'1+X+2'` → `'3+X'`
	`'X^Z*Y*X^T*Y'` → `'X^(T+Z)*Y^2'`
	`'X+X+Y+3*X'` → `'5*X+Y'`
	COLCT operates independently on each side of an equation:
	`'1+2*X=3+4*X'` is not simplified to `'-2=2*X'` or even `'-1=X'`	

EXPAN	Expands an equation or expression:
	`'A*(B+C)'` → `'A*B+A*C'`
	`'(B+C)/A'` → `'B/A+C/A'`
	`'A^(B+C)'` → `'A^B*A^C'`
	`'X^5'` → `'X*X^4'`
	`'(X+Y)^2'` → `'X^2+2*X*Y+Y^2'`
	EXPAN doesn't perform all possible expansions in a single step. It may be necessary to apply the command repeatedly.	

SIZE	Returns the number of objects in an expression or equation. Example: `'XX*LN(Y)=CCC' SIZE` returns 6 because there are these objects: XX, *, LN, Y, =, CCC	
FORM	Allows to perform identity manipulations interactively on equations and expressions. • When invoked, the expression in stack level 0 is displayed in line 2 (and line 3 if it doesn't fit in one line) of the LCD display and a number of soft-key menus are displayed. • The commands available in the menus depend on the type of sub-expression that is currently selected by the cursor. • When done press ON to quit the interactive mode. The expression in level 1 will be replaced by the modified version. • The FORM editor can be invoked by a program.	
	COLCT	Collect similar expressions of the selected sub-expression. Similar to COLCT above.
	EXPAN	Expand products and exponentials of the selected sub-expression. Similar to EXPAN above.
	LEVEL	While this button is held down the level of the currently selected object or sub-expression is displayed.
	EXGET	Quit the FORM editor and returns: • In stack level 3: The edited expression. • In stack level 2: A copy of the currently selected sub-expression. • In stack level 1: The position index of the sub-expression.
	[←]	Move cursor left.
	[→]	Move cursor right.

	The presence of the following commands depends on the type of the current subexpression:	
	E()	Replace exponentials of an exponent by a product of exponentials: EXP(A)^B → EXP(A*B)
	E^	Inverse of E(): EXP(A*B) → EXP(A)^B
	←D	Distribute left. A*(B+C) → (A*B)+(A*C)
	D→	Distribute right.
	←A	Associate left. This moves the grouping brackets to the left.
	A→	Associate right. This moves the grouping brackets to the right.
	←M	Collect similar right-hand-side factors of surrounding expressions.
	M→	Collect similar left-hand-side factors of surrounding expressions.
	DNEG	Insert a double-negation.
	DINV	Insert a double inversion.
	*1	Insert multiplication by 1.
	/1	Insert division by 1.
	^1	Insert exponentiation by 1.
	+1-1	Insert addition of +1-1.
	→()	Distribute a prefix-operator (ie. minus sign, INV()) into the following sub-expression.
	-()	Combination of DNEG and a →() of the inner negation.
	1/()	Combination of DINV and a →() of the inner inversion.
	←→	Swap left and right side of operator. Inserts a factor –1 or 1/x when executed on substraction or division.
	L*	Replace logarithm of an exponential by a product of a logarithm and the exponent: LN(A^B) → (LN(A)*B)
	L()	Inverse of L*: (LN(A)*B) → LN(A^B)
	AF	Add fractions by expanding to a common denominator.
OBSUB	Replaces the n-th object with a new one. See also OBGET below: `'XX*LN(Y)=CCC' 4 {Q} OBSUB` returns `'XX*LN(Q)=CCC'`. `'XX*LN(Y)=CCC' 5 {Q} OBSUB` returns `'Q(XX*LN(Q),CCC)'`. `'XX*LN(Y)=CCC' 1 {-} OBSUB` returns an error.	
EXSUB	Replaces the n-th expression with a new one. See also EXGET below: `'XX*LN(Y)=CCC' 3 '2*K' EXSUB` returns `'XX*(2*K)=CCC'`.	

TAYLR	Calculates a Taylor series (polynomial) for an arbitrary function. Example: `'X/(X^2+1)' 'X' 3 TAYLR` returns `'X-X^3'`. `'SIN(X)' 'X' 5 TAYLR` returns `'X-0.1666*X^3+8.3333E-3*X^5'`. The series is developed around X=0 which may not always be desirable. To shift the point of expansion, ie. from X=0 to X=2: • Store `'Y+2'` in variable X. Make sure variable Y does not exist. • Evaluate the function f(X) to convert it to a function f(Y). • Perform the Taylor series expansion for Y. • Evaluate the resulting function around Y=0 or: • Store 'X-2' in variable Y. Make sure variable X does not exist. • Evaluate the Taylor series to convert it into a function of X. • Evaluate the result for values of X around 2. Example: Develop ln(x) around x=2: `'LN(X)' 'Y+2' 'X' STO EVAL` returns f(y)=LN(Y+2). ...`'Y' 3 TAYLR` returns the expansion around Y=0: 0.693+0.5*Y—0.125*Y^2+4.166E-2*Y^3 Now convert back to a function of X: ...`'X' PURGE 'X-2' 'Y' STO EVAL` returns: 0.693+0.5*(X-2)—0.125*(X-2)^2+4.166E-2*(X-2)^3 For a test evalue this function for X=2.5: ...`2.5 'X' STO EVAL` returns 0.9171055… The true value would be ln(2.5)=0.916290…
ISOL	Isolates the leftmost occurence of a specified variable. Example with flag 34 (principal value) set: `'A=3^(X+5)' 'X' ISOL` returns `'LN(A)/1.0986…-5'`. `'x^2=3^(X+5)' 'X' ISOL` returns `'√(3^(X+5))'`. `'3^(X+5)=x^2' 'X' ISOL` returns `'LN(X^2)/ 1.0986…-5'`. Example with flag 34 (principal value) clear: `'A=3^(X+5)' 'X' ISOL` returns `'(LN(A) +2*π*i*n1)/1.0986…-5'` where n1 is a placeholder for an arbitrary integer number. `'x^2=3^(X+5)' 'X' ISOL` returns `'s1*√(3^(X+5))'`. `'3^(X+5)=x^2' 'X' ISOL` returns `'(LN(X^2)+2*π*i*n1)/ 1.0986…-5'`.
QUAD	See **SOLV Menu**.
SHOW	Makes implicit references to a variable visible. Example: Assume variable A contains the expression `'X+Y'` and B contains a plain number. Then `'A*B' 'X' SHOW` returns `'(X+Y)*B'`. The implicit reference of A*B to variable X is resolved in the result of SHOW. Note that you could also use EVAL on 'A*B' but if A, X or Y contained numerical values these would replace their variable names!
OBGET	Returns the n-th object from an equation or expression: `'XX*LN(Y)=CCC' I OBGET` returns for I=1…6: `{XX}, {*}, {LN}, {Y}, {=}, {CCC}` See also SIZE above.
EXGET	Returns the n-th partial expression from an equation or expression: `'XX*LN(Y)=CCC' I EXGET` returns for I=1…6: `'XX', 'XX*LN(Y)', 'LN(Y)', 'Y', 'XX*LN(Y)=CCC', 'CCC'`

STAT Menu

∑DAT	A variable containing a matrix or vector. Statistics commands operate on the real nxm matrix stored in variable ∑DAT. Statistics functions cannot operate on complex data. See **Reserved Variables**.
∑PAR	A variable containing a list. It contains four parameters for statistics operations, see COL∑. See **Reserved Variables**.
∑+	Appends another row-vector of m real numbers to the ∑DAT matrix. ∑+ can also append multiple length-m vectors of data elements which are stored in a kxm matrix. A plain number can also be appended in case m=1. The number of data points must match the number of colums in ∑DAT. The number of rows n is the number of "data points". The first ∑+ operation defines the number of columns m in ∑DAT.
∑-	Removes the last line n from ∑DAT and returns it in stack level 1. In case m=1 only a real number is returned.
N∑	Return the number of data points in ∑DAT which is the number of rows of the matrix.
CL∑	Clears all statistics data by erasing ∑DAT. ∑PAR is not erased. After this the next ∑+ operation defines the size of a new ∑DAT matrix.
STO∑	Takes a matrix from the stack and stores it in ∑DAT. The number of columns in the stored matrix need not match the number of columns in – the potentially existing – ∑DAT.
RCL∑	Returns the ∑DAT matrix to the stack. This is equivalent to `'∑DAT' RCL`.
TOT	Adds up the values in each column of ∑DAT separately and returns a size-m vector.
MEAN	Calculates the mean value of the values in each column and returns a size-m vector.
SDEV	Calculates the standard deviation of the values in each column and returns a size-m vector.
VAR	Returns the variance of the values in each column and returns a size-m vector. VAR is the square of SDEV.
MAX∑	Finds the maximum value in each of the columns and returns a size-m vector.
MIN∑	Finds the minimum value in each of the columns and returns a size-m vector.
COL∑	Take two numbers from the stack and store them in ∑PAR: The 1st number (from stack level 2) defines the independent variable for LR or the horizontal coordinate for DRW∑ and SCL∑. For SCL∑ see **PLOT Menu**. The 2nd number (from stack level 1) defines the dependent variable for LR or the vertical coordinate for DRW∑ and SCL∑. By default the independent-variable column is column 1 and the dependent-variable column is column 2.
CORR	Returns a real correlation value between two columns of the data matrix. The column numbers are taken from ∑PAR and can be specified using COL∑.
COV	Calculates the covariance between two columns of the data matrix. The column numbers are taken from ∑PAR and can be specified using COL∑.

LR	Calculates the linear regression thru a set of x/y-points taken from two columns. The column numbers for the dependent (y(x)) and independent (x) set of values are specified in ∑PAR and can be modified using COL∑. The return value is the y-offset of the best-fit line in stack level 1 and the slope in level 2. The returned values are also stored in the ∑PAR list at positions 3 and 4 for later reference by PREDV. Other types of curve fits (exponential, logarithmic, etc.) can easily be implemented by user programs operating on ∑DAT.
PREDEV	Takes the x value from stack level 1 and calculates a prediction y(x) according to the linear coefficients that were calculated by LR.
UTPC	Upper-tail Probability Chi-square Distribution UTPC(n,x). This function and the following ones do not operate on statistics data.
UTPF	Upper-tail Probability F Distribution UTPF(n1,n2,x)
UTPN	Upper-tail Probability Normal Distribution UTPN(m,v,x)
UTPT	Upper-tail Probability t-Distribution utp(n,x)
COMB	Combinations `C(x,y) = Y! / [X! • (Y-X)!]` Number of possibilities to select X elements from a group of Y different elements where different sequences *do not* count separately: `10 2 COMB` returns 45. There are 45 possibilities to draw two elements from a group of 10 different elements if the order of the two drawn elements does not matter.
PERM	Permutations `P(x,y) = Y! / (Y-X)!` Number of possibilities to select X elements from a group of Y different elements where different sequences count separately. `10 2 COMB` returns 90. There are 90 possibilities to draw two elements from a group of 10 different elements if the order of the drawn elements does matter.

PRINT Menu

General	• The HP-28S works in conjunction with the thermal printer HP-82240A. • To print the current contents of the LCD: Hold down "ON", then press "L" (it has the PRINT menu associated with it), release "ON". This produces the same output as PRLCD.
Print flags	• Flag 33: Usually, every print command sends a trailing CR which triggers the immediate printout of the data. If flag 33 is set, data is collected in the printer's input buffer (max. 200 characters) and printed only after sending a CR (4dec) or LF (10dec). Note that the printer needs about 1.8 seconds to print one line so do not send data too fast. • Flag 47: When set a blank line is printed after every printout. • Flag 52: Activates the "fast print mode". It should only be set when the printer is operated with an external power supply!
Escape chars	Escape characters can be used for special print effects. The escape code is 27 or 0x1B. • 27 001…166 Print graphics characters. • 27 250 Underline off. • 27 251 Underline on. • 27 252 Normal print width (24 characters). • 27 253 Double print width (12 characters). • 27 254 Printer self test. Prints the entire character set until the printer is turned off. • 27 255 Printer reset.
PR1	Print object in stack level 1. The exact formatting of the printout depends on the number format (STD, FIX etc.), see **MODE Menu**. Independently of the multi-line setting (ML) the entire object is printed over multiple lines. The object is not dropped from the stack.
PRST	Print all objects on the stack. Objects are printed over multiple lines.
PRVAR	Print the contents of a given variable. Objects are printed over multiple lines.
PRLCD	Print the contents of the LCD screen.
CR	Advance the printer paper by one line.
TRAC	Print a running record of all activity. A printout of the command line and the contents in stack level 1 occurs whenever ENTER is executed or a command that implicitly executes ENTER. Different from other print commands all objects are only printed in one line. Not all of the data may be visible.
PRSTC	Prints the stack in compact form where one stack level occupies only one line in the printout.
PRUSR	Print the names of the USER variables in the current directory in the form of a list (similar to the VARS command in the **MEMORY Menu**). If it is empty "No User Variables" is printed.
PRMD	Displays and prints the current MODE settings. See **MODE Menu**.

CONTRL Menu

General	Special commands that are mostly used in conjunction with programs.
SST	Executes the next instruction of a suspended program. The instruction is briefly displayed. If during the execution of the next step an error occurs the program counter is not advanced. The stack can be manipulated between SST commands.
HALT	Suspend program execution. Use CONT (above the "1" key) or SST to continue the program. Multiple programs can be put in the suspended state. In this case CONT continues the most recently suspended program.
ABORT	Abort the program execution. The program cannot be continued.
KILL	Abort the current program and all other suspended programs. Usually not used within a program.
WAIT	`n WAIT` suspends program execution for n seconds. n may be fractional.
KEY	Queries the keyboard for key-presses. • If no key is waiting this command returns 0 • If one or more keys are waiting this returns a string in stack level 2 and "1" in stack level 1. The string contains the name of a single key. Example: `<< DO KEY IF 0 ≠ THEN 1 DISP END UNTIL 0 END >>` waits for keys and displays their string representation in the top row of the LCD screen (press ON to quit the program). Usually the returned string represents the text that is printed ontop of the keys, except: SPACE : " " LC : "1" INS : "INS" DEL : "DEL" ↑ : "UP" ↓ : "DOWN" ← : "LEFT" → : "RIGHT" <> : "CURSOR" ⇐ : "BACK" Shift : "SHIFT"

BEEP	Usage: `frequency duration BEEP`
	Issues a tone of given frequency and duration (in seconds).
	`<< 2 12 INV ^ 444 → F T`
	`  << 1 12 START T DUP .5 BEEP F * 'T' STO NEXT>>`
	`>>`
	This small program playes the tone ladder based on A (444 Hz). T is the frequency, F is the factor between subsequent tones which is 2^(1/12).

CLLCD	Clears the entire LCD screen and sets the message flag, see CLMF.
DISP	Usage: `object n DISP` Displays the given object in line n (1…4) of the LCD screen and sets the message flag. This does not change any values in the stack! Objects are displayed in their normal format. Except strings are not displayed with surrounding quotation marks. Lengthy objects are split over multiple lines.
CLMF	Clear Message Flag and return to normal stack view. See **PLOT Menu**.

ERRN	Return a binary number representing the code of the most recent error.
ERRM	Returns a string representing a description of the most recent error.

BRANCH Menu

General	Special commands that are mostly used in conjunction with programs:
IF	Usage: • IF test-instruction THEN true-instructions END • IF test-instruction THEN true-instructons ELSE false-instructions END The "test-instruction" must return a value on the stack. Non-0 values are interpreted as true, 0 as false. Example: `IF 0 < THEN -1 ELSE 1 END` implements the SIGN function.
IFERR	Usage: • IFERR test-instruction THEN error-instructions END • IFERR test-instruction THEN error-instructons ELSE ok-instructions END "test-instruction" is executed and if an error occurs the remaining test-instructions are skipped and the "error-instructions" are executed. Ie. this can be used to process all values on the stack without needing to know how many there are. Or type errors can be caught.
THEN	Used with IF
ELSE	Used with IF
END	Used with various branch instructions
START	Usage: • start end START instructions NEXT • start end START instructions step-size STEP "start" end "end" denote the start and end values of the loop counter. With NEXT "instructions" are executed end-start+1 times. With STEP the loop counter is incremented by "step-size" and the loop stops when the loop counter exceeds "end". Note that the value of the loop counter is not accessible to the program!
FOR	Usage: • start end FOR name instructions NEXT • start end FOR name instructions step-size STEP "start" and "end" denote the start and end values of the loop counter. The current value of the loop counter is stored in the local variable "name". With NEXT "instructions" are executed end-start+1 times. With STEP the loop counter is incremented by "step-size" and the loop stops when the loop counter exceeds "end". Example: `<<→ n << 1 n START x x DUP * NEXT n →LIST >>` returns a list of squares from 1 to n. The "instructions" are never executed if initially start>end.
NEXT	Used with START and FOR
STEP	Used with START and FOR
IFT	Similar to `IF-THEN-END`: • test-instruction true-instruction IFT If "test-instructions" evaluates to a non-0 value then the true-instruction is executed (evaluated). Otherwise no action occurs. `1 2 3 IFT` results in `1 3` because 2 evaluates to true and 3 is executed.

IFTE	Similar to `IF-THEN-ELSE-END`: • test-instruction true-instruction false-instruction IFT If "test-instructions" evaluates to a non-0 value then the true-instruction is executed (evaluated). Otherwise the false-instruction is executed. `1 2 3 IFTE` results in `2` because 1 evaluates to true and 2 is executed but 3 is not.
DO	Usage: • DO loop-instructions UNTIL test-instruction END The "loop-instructions" are evaluated until the "test-instructions" evaluates to a non-0 value. The "loop-instructions" are evaluated at least once.
UNTI	UNTIL. Used with DO
END	Used with various branch instructions
WHIL	WHILE. Usage: • WHILE test-instruction REPEAT loop-instructions END While the "test-instruction" evaluates to a non-0 value the "loop-instructions" are executed. The "loop-instructions" may never be executed.
REPEA	REPEAT. Used with WHILE.
END	Used with various branch instructions

TEST Menu

General	Special commands that are mostly used in conjunction with programs:
≠	Return true if the objects in level 1 and 2 are of different type or have the same type but a different value. Lists and programs are assumed to be identical when they contain the same elements.
> ≥ < ≤	Can be used to compare: • Real numbers (but not complex numbers) • Binary numbers • Strings (in alphabetical order that is based on ASCII codes)
SF	Set specified flag. The real flag number must be in the range 1…64. Binary numbers cannot be used as arguments to SF and the following functions. See **Flags**.
CF	Clear specified flag.
FS?	Test specified flag and return 1 when it is set, otherwise 0.
FC?	Test specified flag and return 0 when it is set, otherwise 1.
FS?C	Test specified flag and return 1 when it is set, otherwise 0. Clear the flag after the test.
FC?C	Test specified flag and return 0 when it is set, otherwise 1. Clear the flag after the test.
AND	Treats the values in stack level 1 and 2 as flags (true if non-0, false if 0). It performs the AND operation and returns 0 or 1.
OR	Performs the OR operation and returns 0 or 1.
XOR	Performs the XOR operation and returns 0 or 1.
NOT	NOT operation: A non-0 value results in 0; a 0-value results in 1.
SAME	Very similar to ==. However, SAME returns true when names are involved and the names are identical. SAME never returns an expression. Example: `'A' 5 SAME` returns 0. A quoted name is not evaluated.

==	Compare objects in stack level 1 and 2 and return 1 if they are of the same type and have the same value. Returns an expression if names are involved. Example: `'A' 5 ==` returns 'A==5'. A quoted name is not evaluated.
STOF	Takes a binary number from the stack and replaces all 64 flags with the bits of the binary number. bit0 of the binary number replaces flag 1 and bit63 replaces flag 64. See **Flags**.
RCLF	Return the settings of the 64 flags in a binary number. STOF/RCLF can be used to save and restore the machine settings in case a program needs to change flag-based operation modes.
TYPE	Returns the type of the object in stack level 1: Real number.....: 0 Complex number.....: 1 String.....: 2 Real vector or matrix.....: 3 Complex vector or matrix..: 4 List.....: 5 Name.....: 6 Local name.....: 7 Program.....: 8 Algebraic expression.....: 9 Binary number.....: 10

CATALOG Menu

General	Displays a list of all available commands in alphabetical order. Press a character to jump to the next command that starts with this character.
NEXT, PREV	Go to next/previous command
SCAN	Automatically display all entries one after the other. SCAN will be replaced by STOP which can be used to stop the scan. *Not available with some HP-28S versions!*
USE	Display the arguments that the command expects on the stack. This also brings up its own NEXT/PREV menu to scroll thru different argument sets.
FETCH	Bring the command name into the edit line.
QUIT	Quit the catalog

UNITS Menu

<table>
<tr><td>General</td><td>UNITS is not a menu but a catalog of all available built-in units.
• Use PREV/NEXT to step thru the list.
• Use SCAN to let the HP-28S step thru the units catalog. Use STOP to halt the scan. Not available on all HP-28S versions.
• Press a character to jump to the next unit which starts with this character.
• Press "1" to jump to the last entry.
• For every unit the value and the constituting SI units are given.
• Use FETCH to bring the unit name into the command line.
• Use quit to close the UNITS Menu.</td></tr>
<tr><td>SI base units</td><td><table><tr><u><th>Quantity</th><th>Unit</th><th>Abbreviation</th></u></tr><tr><td>Length</td><td>Meter</td><td>m</td></tr><tr><td>Mass</td><td>Kilograms</td><td>kg</td></tr><tr><td>Time</td><td>Seconds</td><td>s</td></tr><tr><td>Electric current</td><td>Ampere</td><td>A</td></tr><tr><td>Temperature</td><td>Kelvin</td><td>°K</td></tr><tr><td>Luminous intensity</td><td>Candela</td><td>cd</td></tr><tr><td>Amount of substance</td><td>Mol</td><td>mol</td></tr></table></td></tr>
<tr><td>CONVERT</td><td>Use this command to convert between units:
<code>100 "m" "ft" CONVERT</code> results in <code>328.08 "ft"</code>
Note that the converted number sits in stack level 2 and the new unit string in stack level 1.
Provided no variables m and ft exist this would also work:
<code>100 m ft CONVERT</code></td></tr>
<tr><td>Unit string</td><td>• The string defining the units may consist of a product of multiple elementary units:
<code>100 "m*m" "ft^2" CONVERT</code> results in 1076.39 "ft^2"
<code>100 "m/s" "ft/s" CONVERT</code> results in 328.08 "ft/s"
• The unit string may only contain a single divide sign "/". To the left positive powers and to the right of it negative powers of units are assumed.
• Powers may only be in the range 1-9. In particular, they must not be negative.
• Brackets for grouping are not allowed!
• Also, the unit may be preceeded by a magnitude prefix (see below)
<code>100 "km/s" "mi/h" CONVERT</code> results in 223693.63 "mi/h"
Warning: Some combinations of magnitude prefix and unit result in another unit. Example: "min" is minutes but not milli-inches; "ft" is feet but not femto-tons etc.
• Single quotes inside the unit string are ignored.</td></tr>
</table>

Magnitude prefixes

Symbol	Name	Factor 10E..	Symbol	Name	Factor 10E..
E	Exa	18	d	Dezi	-1
P	Peta	15	c	Centi	-2
T	Tera	12	m	Milli	-3
G	Giga	9	µ	Micro	-6
M	Mega	6	n	Nano	-9
k or K	Kilo	3	p	Pico	-12
h or H	Hecto	2	F	Femto	-15
D	Deca	1	a	Atto	-18

Temperature conversions

These are special because additive constants are involved.

- If there is only a plain temperature unit without magnitude prefix or exponent or any other unit then absolute temperature conversion is performed:
  `100 "°F" "°C" CONVERT` results in 37.78 "°C"
- Otherwise, a relative temperature conversion is performed which ignores the additive constant:
  `50 "°F^2" "°C^2" CONVERT` results in 30.86 "°C^2"

User defined units

- A user-defined unit is a length-2 list stored in a variable. The 1st element in the list is the factor to convert the unit into a built-in unit and the 2nd element is the built-in unit.
  `{7 "d"} 'wk' STO` creates a unit wk (week) which consists of 7 days (d). It can be used like any other unit:
  `7 "wk" "h" CONVERT` results in 168 "h"
- Note that user-defined units cannot be preceeded by a magnitude prefix. However, it is always possible to define a new user-unit that has the prefix built into its name.
- A dimension-less unit uses the unit string "1".
- For non-SI units use the unit string "?". This can be used for example to convert currency exchange rates.

List of built-in units. The actual values can be retrieved from the calculator!

Unit	**Name**	**Type**	**SI units**
a	Ar	Area	m^2
A	Ampere	Electric current	A
acre	Acre	Area	m^2
arcmin	Arcus Minute	Plane angle	1
arcs	Arcus Second	Plane angle	1
atm	Atmosphere	Pressure	kg/m*s^2
au	Astronomical Unit	Length	m
A°	Angström	Length	m
b	Barn	Area	m^2
bar	Bar	Pressure	kg/m*s^2
bbl	Barrel Oil	Volume	m^3
Bq	Bequerel	Radioactive activity	1/s
Btu	British Thermal Unit	Energy	kg*m^2/s^2
bu	Bushel	Volume	m^3
c	Speed of Light	Speed	m/s
C	Coulomb	Electric charge	A*s
cal	Calorie	Energy	kg*m^2/s^2
cd	Candela	Luminous intensity (Lichtstärke)	cd

chain	Chain	Length	m
Ci	Curie	Radioactive activity	1/s
ct	Carat	Mass	kg
cu	US Cup	Volume	m^3
d	Day	Time	s
dyn	Dyne	Force	kg*m/s^2
erg	Erg	Energy	kg*m^2/s^2
eV	Electronvolt	Energy	kg*m^2/s^2
F	Farad	Electric capacitance	A^2*s^4/kg*m^2
fath	Fathom	Length	m
fbm	Board Foot	Volume	m^3
fc	Footcandle	Luminance (Leuchtdichte)	cd/m^2
Fdy	Faraday	Electric charge	A*s
fermi	Fermi	Length	m
flam	Footlambert	Luminance (Leuchtdichte)	cd/m^2
ft	Internatl. Foot	Length	m
ftUS	Survey Foot	Length	m
g	Gram	Mass	kg
ga	Gravitational Acceleration	Acceleration	m/s^2
gal	US Gallon	Volume	m^3
galC	Canadian Gallon	Volume	m^3
galUK	British Gallon	Volume	m^3
gf	Gram-Force	Force	kg*m/s^2
grad	Degrees	Plane angle	1
grain	Grain	Mass	kg
Gy	Gray	Absorbed radioactive dose	m^2/s^2
h	Hour	Time	s
H	Henry	Inductance	kg*m^2/A^2*s^2
hp	Horsepower	Power	kg*m^2/s^3
Hz	Hertz	Frequency	1/s
in	Inch	Length	m
inHg	Inches Quicksilver	Pressure	kg/m*s^2
inH20	Inches Water	Pressure	kg/m*s^2
J	Joule	Energy	kg*m^2/s^2
kip	Kilopound-Force	Force	kg*m/s^2
knot	Knot	Speed	m/s
kph	Kilometers per Hour	Speed	m/s
l	Liter	Volume	m^3
lam	Lambert	Luminance (Leuchtdichte)	cd/m^2
lb	Avoirdupois Pound	Mass	kg
lbf	Pound-Force	Force	kg*m/s^2
lm	Lumen	Luminous flux (Lichtstrom)	cd
lx	Lux	Illuminance (Leuchtdichte)	cd/m^2
lyr	Lightyear	Length	m
m	Meter	Length	m
mho	Mho	Electric conductance	A^2*s^3/kg*m^2
mi	Internatl. Mile	Length	m
mil	Mil	Length	mil
min	Minute	Time	s
miUS	US statute Mile	Length	m

mmHg	Millimeter Quicksilver	Pressure	kg/m*s^2
mol	Mol	Amount of substance	mol
mph	Miles per Hour	Speed	m/s
N	Newton	Force	kg*m/s^2
nmi	Nautic mile	Length	m
ohm	Ohm	Electric resistance	kg*m^2/A^2*s^3
oz	Ounce	Mass	kg
ozfl	US Fluid Ounce	Volume	m^3
ozt	Troy Ounce	Mass	kg
ozUK	UK Fluid Ounce	Volume	m^3
P	Poise	Dynamic viscosity	kg/m*s
Pa	Pascal	Pressure	kg/m*s^2
pc	Parsec	Length	m
pdl	Poundal	Force	kg*m/s^2
ph	Phot	Luminance (Leuchtdichte)	cd/m^2
pk	Peck	Volume	m^3
psi	Pound per Square Inch	Pressure	kg/m*s^2
pt	Pint	Volume	m^3
qt	Quart	Volume	m^3
r	Radians	Plane angle	1
R	Röntgen	Radiation exposure (Strahlendosis)	A*s/kg
rad	Rad	Absorbed radioactive dose	m^2/s^2
rd	Rod	Length	m
rem	Rem	Dose equivalent	m^2/s^2
s	Second	Time	s
S	Siemens	Electric conductance	A^2*s^3/kg*m^2
sb	Stilb	Luminance (Leuchtdichte)	cd/m^2
slug	Slug	Mass	kg
sr	Steradian	Solid angle	1
st	Stere	Volume	m^3
St	Stokes	Kinematic viscosity	m^2/s
Sv	Sievert	Dose equivalent	m^2/s^2
t	Metric Ton	Mass	kg
T	Tesla	Magnetic induction	kg/A*s^2
tbsp	Tablespoon	Volume	m^3
therm	EEC Therm	Energy	kg*m^2/s^2
ton	Shot Ton	Mass	kg
tonUK	Long Ton	Mass	kg
torr	Torr	Pressure	kg/m*s^2
tsp	Teaspoon	Volume	m^3
u	Atomic Mass Unit	Mass	kg
V	Volt	Electric voltage (potential)	kg*m^2/A*s^3
W	Watt	Power	kg*m^3/s^3
Wb	Weber	Magnetix flux	kg*m^2/A*s^2
yd	Intl. Yard	Length	m
yr	Year	Time	s
°	Degree	Angle	1
°C	Degree Celsius	Temperature	°K
°F	Degree Fahrenheit	Temperature	°K

°K	Degree Kelvin	Temperature	°K
°R	Degree Rankine	Temperature	°K
µ	Micron	Length	m
?	User Unit		?
1	Dimension-less Unit		1

CURSOR Menu

General	When the cursor menu is active, the top row of white keys beneath the LCD screen assume their indicated operations (INS, DEL, …) and no menu items (soft labels) are displayed in the bottom row of the LCD screen.
INS	Toggle command line editing mode between insert and overwrite. Shift-INS deletes all characters to the left of the cursor up to the beginning of the current line. By default overwrite mode is active.
DEL	Delete character under cursor in editing mode. Shift-DEL deletes the character under the cursor and all characetrs up to the end of the current line.
← → ↑ ↓	Cursor movement in editing mode. Shifted operations move to the leftmost, rightmost, topmost or bottommost position of the edited text.

MODE Menu

STD	Select standard display format. It displays all non-0 fractional digits of a number. Note that a change of the number format affects the display of *all* values in the stack. This includes plain numbers that occur within programs or lists.
FIX	Select fixed point notation with given number of significant fractional digits.
SCI	Select scientific (exponential) notation with given number of significant digits.
ENG	Select engineering (exponential) notation with given number of significant digits where the exponent is a multiple of 3.
DEG	Use degrees for trigonometric functions (360).
RAD	Use radians for trigonometric functions (2π). Grad (400) are not supported.
CMD	Enable/disable the command line auto-save feature. In programs use `+CMD` and `-CMD`. See **Direct Key Commands**. When running a program it does not alter the auto-saved command line.
UNDO	Enable/disable the stack auto-save feature. In programs use `+UND` and `-UND`. See **Direct Key Commands**. When running a program it does not alter the auto-saved stack contents.
LAST	Enable/disable the argument auto-save feature. In programs use `+LAST` and `-LAST`. See **Direct Key Commands**. When running a program the individual program commands will alter the auto-saved arguments: `<<1 + 2 *>> 'A' STO 3 A LAST` returns `8 4 2`: 8 is the result of the program and 4 and 2 are the arguments of the last program instruction (the multiplication).

ML	Enable/disable the multi-line display feature. • Enabled: Matrices, vectors, complex numbers, lists and programs are displayed in up to 4 lines as to display them entirely. "…" is appended on the right side if necessary to indicate that not all of the object is visible (ie. for large matrices). • Disabled: All stack objects are displayed in one line only. "…" is appended on the right side if necessary. In programs use `+ML` and `-ML`.
RDX,	Toggle between using a comma and a period for the decimal point. The respective other symbol is used as a delimiter. Note that a SPACE can also be used as a delimiter! Be aware that this can be confusing: `{1.2,3.4,5.6}` is evaluated as: • `{1.2 3.4 5.6}` if the decimal sign is a comma • `{1 2,3 4,5 6}` if the decimal sign is a period In programs use `RDX,` and `RDX.`
PRMD	Display and print the current MODE settings. This includes: Display format and number of valid digits, angle mode, UNDO, LAST and COMMAND settings, multiline setting, binary number base.

TRIG Menu

SIN	Sine of a real or complex value.
ASIN	Arcus sine of a real or complex value.
COS	Cosine of a real or complex value.
ACOS	Arcus cosine of a real or complex value.
TAN	Tangent of a real or complex value.
ATAN	Arcus tangent of a real or complex value.
P→R	Converts real or complex number or a vector in stack level 1 from polar to rectangular coordinates: • real X → (x, 0) • (r, θ) → (x, y) • [r θ …] → [x y …] If the vector contains more than 2 elements only the first two are converted and the other elements are left unchanged. This is useful when handling 3-dimensional [r θ z] vectors. The current angle mode (degrees or radians) determines the units of θ.
R→P	Converts a complex number or a vector in stack level 1 from rectangular to polar coordinates. See P→R for details.
R→C	Combine two real numbers into a complex number. The value in level 1 will be the imaginary part.
C→R	Split a complex number into two real numbers in stack level 1 and 2. Level 1 will receive the imaginary part.
ARG	Return the angle θ of the complex number (x,y) vector with the x-axis. • `X≥0: θ=atan x/y` • `X<0: θ=atan x/y + π*sign(y)`

→HMS	Convert a number from fractional hours format to HMS format.
	HMS numbers are displayed in the format **H.MMSSs** where H are hours, MM are minutes (60 per hour), SS are seconds (60 per minute) and s are fractional seconds.
	Note: This format may not match the coodinate format used in GPS devices where SS is not seconds but fractional minutes!! Use the following two programs to convert to and from HM format:
	`<<DUP IP SWAP FP 0.6 * +>> '→HM' STO`
	`<<DUP IP SWAP FP 0.6 / +>> 'HM→' STO`

HMS→	Convert a number form HMS format to fractional hours.
HMS+	Add two numbers given in HMS format.
HMS-	Substract numbers given in HMS format.
D→R	Convert degress (360) to radians (2π)
R→D	Convert radians (2π) to degress (360)

LOGS Menu

General	All of the following functions operate on real as well as complex numbers but not on matrices or vectors.
LOG	Logarithm base 10.
ALOG	10^x.
LN	Logarithm base e.
EXP	e^x.
LNP1	Returns ln(1+x) which is useful when x is close to 0.
EXPM	Returns exp(x)-1 which is useful when x is close to 0.
SINH	Hyperbolic sine.
ASINH	Inverse hyperbolic sine.
COSH	Hyperbolic cosine.
ACOSH	Inverse hyperbolic cosine.
TANH	Hyperbolic tangent.
ATANH	Inverse hyperbolic tangent.

SOLV Menu

General	The commands in this menu allow to find solutions for user-defined functions. A solution is the value x where f(x)=0. This is also called a root of the function. • Note that only real but not complex roots can be found! • SOLVR is the interactive version of the solver. This mode can be invoked by a program. It offers a versatile user menu for finding iterative solutions for (usually complicated) functions. The "user interface" is the same as for example on the HP-12C when solving for n, i, PM, PV, PMT and FV. • ROOT is a non-interactive version which is mainly used in programs. • Furthermore, ISOL can isolate (unique) variables from equations and QUAD can calculate symbolic solutions for functions. • A very powerful tool is the combination of the interactive plot command DRAW (see **PLOT Menu**) and the solver: Visually interesting points in the plot (ie. approximate roots) can be digitized and passed as initial guesses to the solver.
Finding a numerical root	Follow these steps to interactively find a numerical root: 1. Store the function to be solved using STEQ, see below. 2. Press SOLVR to display a menu that shows all the variables used inside the function. 3. Store the desired values in these variables. Also store an initial guess in the variable that you want to solve for. 4. Press SHIFT and the menu button for the variable you want to solve for. This will invoke the root finding process.
STEQ	This stores the function that is to be solved in the global variable called **EQ**. The function can be: • An expression, ie. `'3-x^2'`. In this case the solver finds the value for x where the expression is 0. • An equation, ie. `'y=3*x^3 - 2*x'`. The solver finds a solution for x (or y) where the left side equals the right side. • A program, ie. `<< X DUP * 7 - Y + >>`. The solver finds the value for one of the variables used in the program (X or Y) for which the program returns 0 in stack level 1. The program must not take anything from the stack. Note that the solver and plot commands (see **PLOT Menu**) use the same equation EQ.
RCEQ	Recall the function stored in variable EQ.

SOLVR: Variables	Display the interactive solver menu with all the independent variables used in the function. The menu looks different than normal command menus in that the names are printed black on a white background. While the interactive menu is active the stack can be used normally. • An independent variable is either a formal variable (with no associated data) or a variable containing data (ie. a number). Variables containing procedures do not show up in this menu. Instead the independent variables of these procedures are listed. If any one of these nested procedures is modified the solver menu is updated automatically. Note that if a variable contains an equation the "=" sign is replaced by "-" (minus) in order to convert it to an expression which returns a result. By pressing the menu button associated with a variable a value can be stored for the variable: • The value is taken from stack level 1 as usual. • The stored value is also temporarily displayed in the top line of the LCD display.
continued SOLVR: Initial Guess	For the variable that is solved for an initial guess must be specified: • This can be a single number or list with a single value. • Or a list with two values which indicate the initial interval of the search algorithm. If the function has a different sign at these two points the solver will quickly find a root. • Or a list with three values where the first is a best guess and the other two values should be below and above the best guess. • Note: After using DRAW on the function in EQ and looking at the function curve it is usually simple to see initial guesses for the solver. DRAW offers a cross-hair pointer that can be moved over the plot and complex (x,y) coordinates can be selected and stored on the stack. These complex values can be used as initial guesses (the imaginary part will be ignored). See **PLOT Menu**.
continued SOLVR: Finding the root	Pressing SHIFT and one of the menu buttons invokes the solver for the specified variable. • The solver can be interrupted by pressing ON. In this case a best guess in the form of a length-3 list is returned. • Pressing any other key displays the current best guess in lines 2 and 3 of the LCD display and the root finding process continues.
continued SOLVR: Results	When done the solver will return the result in stack level 1 and also display it in the display line 1. In display line 2 one of these messages appears: • `Zero`: A root has been found. • `Sign reversal`: Two adjacent points have been found where the function changes it sign. Possibly, the function has a discontinuity at this point. • `Extremum`: The solver found an extremum (local maximum or minimum) or hit ±MAXR. • `Bad Guess(es)`: When evaluating the function at the initial guess points it causes an error. • `Constant?`: The function always returns the same value.

continued SOLVR: Verification	Also, there are the following buttons present in the solver menu. They can be used to verify a solution: • `EXPR=` Only present if the function is an expression or program. When pressed the expression/program is evaluated using the current values of the variables. The result should be 0. • `LEFT=` Only present if the function is an equation. When pressed it evaluates the left side of the equation. • `RT=` Only present if the function is an equation. When pressed it evaluates the right side of the equation. Left and right side should be equal. The quit the interactive menu press any other menu key are activate the CURSOR menu.
ISOL	Isolate a unique variable in a formula. See **ALGEBRA Menu**.
QUAD	Symbolically solves a quadratic equation. Example (flag 34 clear): `'X^2-5*X+6' 'X' QUAD` returns `'(5+s1)/2'`. s1 indicates an arbitrary sign so the two solutions are (5+1)/2=3 and (5-1)/2=2. This result is only returned when the "principal value" flag 34 has been turned off. If it is set the solution for s1=1 is returned which is 3. It is possible to use formal names: `'X^2-3*X-A*X+3*A' 'X' QUAD` returns `'(-(-3-A)+s1*√((-3-A)^2-4*(3*A)))/2'`. COLCT converts this to: `'.5*(3+√((-3-A)^2-12*A)*s1+A)'`. It is easy to prove that this is equivalent to the correct solution `'.5*(3+A+s1*(A-3))'` but the calculator cannot perform this simplification. • If A has an associated value then the result is evaluated using this value. • Variable s1 can be set to +1 and –1 to get the two results. • If the input expression is not a polynomial of degree 2 then a Taylor series expansion of the expression is performed up to degree 2 and then the roots of this expansion are calculated.
SHOW	Show implicit references to a variable. See **ALGEBRA Menu**.
ROOT	Root is a version of the solver that can be used in a progam. Example: `<<X DUP * 7 - >> 'X' 0 ROOT` returns 2.645… • Stack level 3 contains the function which can be an expression, an equation or a program, see STEQ above. • Level 2 contains the name of the variable that the solver solves for. • Level 1 contains an initial guess or multiple initial guesses in a list as described in SOLVR above. • The return value is a single value for X. • Unfortunately, X cannot be a local variable so it is not possible to write a program that implements a function of a local variable and where the function is solved for the local variable. See the example program for the factorial function FACT in the **REAL Menu**. When the root finding process is interrupted by pressing ON the function and variable name are returned as well as the current best solution. Thus it is possible to continue the search by simply pressing ROOT again.

PLOT Menu

General	• The PLOT menu commands allow to display function curves or statistical data on the graphics LCD display. • The HP-28S display has a resolution of 137x32 pixel. • The function to be plotted is stored in the global variable EQ. This variables is also used by the solver, see **SOLV Menu**. • Statistical data to be plotted is stored in the global matrix ∑DAT, see **STAT Menu**. • Drawing commands automatically set the "message flag" which suppresses the normal stack display until either a key is pressed or CLMF is called. • Graphical data points are represented by complex numbers where the real part specifies the horizontal x-coordinate (from left to right) and the imaginary part the vertical y-coordinate (from bottom to top). • Note that it is not possible to "overlay" multiple curves in interactive mode. As a maximum two curves can be plotted by using an equation, see STEQ. However, the non-interactive mode (see DRAW) can be used to overlay an arbitrary number of plots. • To draw graphics other than functions, ie. parameterized curves in the form (x(t),y(t)) or histograms etc. the PIXEL command can be used, see further down and *Parameterized Curves* at the end of this section.
	Follow these general steps to create a function plot: 1. Store the desired function in variable EQ using STEQ. 2. Choose the independent variable using INDEP. 3. Optionally use PMIN, PMAX, and CENTR to specify the limits of the plot area. 4. Optionally use RES to select the resolution of the plot. 5. Press DRAW.
PPAR	A global list containg data to control graphical plots. It contains in this order: • A complex number specifying the lower left corner Pmin of the plot. Set by PMIN. • A complex number specifying the upper right corner Pmax of the plot. Set by PMAX. • The independent variable for the plot. Set by INDEP. • A real number specifying the resolution of the plot. Set by RES. • A complex number specifying the coordinates of the intersection of the plot axes. Set by AXES. See **Reserved Variables**.
STEQ	Store a function in variable EQ. The form of the function determines how it is displayed: • An expression is displayed as a single curve. • An equation results in two curves, one for each side of the equal-sign. • A program is treated as expression and plots a single curve. The program must not take any arguments from the stack and it must returns a single function result to the stack. See **Reserved Variables**.

RCEQ	Recall the function from EQ.
PMIN	Takes a complex argument from the stack and stores it in PPAR as the lower left corner of the plot.
PMAX	Takes a complex argument from the stack and stores it in PPAR as the upper right corner of the plot.
INDEP	Store a name in PPAR which specifies the independent variable for a plot. The function stored in EQ is evaluated for varying values of the independent variable in order to obtain the y=f(x) values of the curve to be plotted. If no independent variable has been specified then DRAW uses the first variable in EQ instead.

DRAW	When called by pressing the DRAW menu button:
	• Clears the display
	• Calls DRAX to draw the axes and then draws the plot.
	• Dependent on the type of the function either one or two curves are plotted, see STEQ above.
	• Interactive mode is activated, see below.
	• Pressing ON aborts the plot process and interactive mode activated immediately.
	When called from a program:
	• The display is *not* cleared.
	• Axes are *not* drawn.
	• The 1 or 2 function curves are plotted.
	• Pressing ON aborts the plot process and returns to the stack view.
	• Pressing any key after the plot has been drawn returns to the stack view.
	• Interactive mode is not entered automatically but can be invoked using the DIGTZ command.
	This non-interactive mode allows to overlay multiple function plots:
	`<< 'PPAR' PURGE (0,0) PMIN (10 4) PMAX`
	`'X' INDEP CLLCD DRAX`
	`'SIN(X)+1' STEQ DRAW`
	`'(COS(X*1,5)+1)*1.5' STEQ DRAW`
	`'(X-5)^2/10 + 0.5' STEQ DRAW`
	`DIGTZ >>`
	Draws an overlay of three plots and then activates interactive mode.

Interactive Mode	While the plot is displayed: • The cursor keys can be used to move a small cross-hair-shaped ("+") cursor over the plot. • SHIFT-cursor moves the cross-hair to the top, bottom, left or right edge of the plot. • INS puts the current coordinates of the cross-hair as a complex number onto the stack (without changing to the stack view). INS can be pressed repeatedly to digitize multiple points. • DEL retrieves a copy of the display data into a string (without the cross-hair). This is identical to the `LCD→` command in the **STRING Menu**. *Note that older versions of the Reference Manual describe this command differently*. • Press ON to quit the interactive mode. The INS-feature is extremely useful. It allows for: • Finding initial guess for the solver. See **SOLV Menu**. • Moving the center of the plot to an interesting location, see CENTR. • Cutting off uninteresting areas of the plot, see PMIN and PMAX.
PPAR	Returns the PPAR list. Use menu commands or `'PPAR' STO` to store new parameters.
RES	Stores a real number in PPAR which determines the resolution of the plot. For n=1 a point is displayed in every LCD column, for n=2 in every 2md column etc. Larger values of the resolution speed up the plot process!
AXES	Takes a complex argument from the stack and stores it in PPAR as coordinate of the intersection of the plot axes. This is usually (0,0).
CENTR	Takes a complex argument and modifies the parameters Pmin and Pmax in PPAR so that the given point is displayed in the middle of the LCD screen. The height and width of the plot are not changed.
*W	Takes a real argument and multiplies the x-coordinates of Pmin and Pmax with it. Ie. `2 *W` displays a larger x-area or "zooms out". On the other hand `0.5 *W` displays a smaller area or "zooms in".
*H	Takes a real argument and multiplies the y-coordinates of Pmin and Pmax with it.
STO∑	Store a matrix in statistics variable ∑DAT. See **STAT Menu** and **Reserved Variables**.
RCL∑	Recall statistics matrix ∑DAT. See **STAT Menu**.
COL∑	Takes two real arguments that define two columns of the statistics data matrix ∑DAT which are used to plot statistics data. See DRW∑ and **STAT Menu**.
SCL∑	Modifies Pmin and Pmax in PPAR so that a DRW∑ plot will fit exactly into the LCD display.
DRW∑	Calls DRAX to draw the axes and then plots statistics data stored in matrix ∑DAT. Horizontal values are taken from the independent-variable column (usually 1), vertical values are taken from the dependent-variable column (usually 2). See COL∑ and **STAT Menu**. When invoked by pressing the DRW∑ menu key the interactive mode is activated, see above.
CLLCD	Clear the entire LCD screen.

DIGTIZ	Activate the interactive digitization cross-hair on the current display. See Interactive Mode above.
PIXEL	Takes a complex number from the stack and sets the pixel specified by the real part (horizontal position from left) and imaginary part (vertical position from bottom). The current coordinate system settings in PPAR are obeyed! If the x and y positions need to actually refer to raw LCD pixels originating at the bottom-left corner then enter: `(0 0) PMIN (137 32) PMAX`.
DRAX	Draw a horizontal and vertical coordinate axis. The location of point (0,0) is specified by the PPAR variable. Tick marks are added every 10 pixel.
CLMF	Clear Message Flag. Commands CLLCD, DISP, PIXEL, DRAX, DRAW, und DRW∑ set the "message flag" which suppresses the normal stack display. CLMF clears the message flag and thus redisplays the normal stack view.
PRLCD	Print the current contents of the LCD screen.

Parameterized Curves	The following program PPLOT expects:
	• On stack level 2 a program that takes the parameter T from the stack and returns a complex number (x(T),y(T)).
	• On stack level 1 a length-3 list containing the start and end values and the step-size of parameter T.
	`<< LIST→ DROP → F MI MA S << CLLCD DRAX MI MA FOR X X F EVAL PIXEL S STEP DIGTZ >> 'PPLOT' STO`
	Usage example (degrees selected):
	`<<DUP SIN 2 * SWAP COS R→C>> {0 360 6} PPAR`
	draws an ellipse.
	`<<DUP 2 * SIN SWAP COS R→C>> {0 360 6} PPAR`
	draws a hour glass.

USER Menu

General	Displays the names of the variables of the current directory in six "soft labels" on the bottom of the LCD screen.
NEXT PREV	Used to display the next or previous six entries or press USER again to display the first six entries.
Menu key press	When one of the white menu keys below the display is pressed the associated variable is evaluated immediately and thus the variable content is returned to the stack or the program executed or the current directory is changed. For directories see **MEMORY Menu**.
Retrieving variable names	To avoid evaluation enter a single quote and then press the menu key. This will append the variable/program/directory name to the command line rather than evaluating it. Not available for the HOME directory name.
Soft-Label names	The menu name is derived from the first few characters of the variable name. Lower case characters are displayed in upper case. Warning: Variable names are case sensitive so the menu may display two entries with the same name that actually refer to different variables! However, when retrieving the variable name (see above) the correct name is returned.

CUSTOM Menu

General	This displays the menu structure that has been created with the MENU command, see below.
MENU	Located in the **MEMORY Menu**. • Takes a list of names and creates a custom menu containing these names. • The names need not refer to existing variables. • The MENU command automatically activates the custom menu.
Custom input menu	If the first name in the list passed to MENU is STO then a *custom input menu* is created which is similar to the **SOLV menu**: Pressing a menu key stores the element from stack level 1 in the specified variable. The name STO is not included in the menu and the menu labels are displayed in outlined mode instead of solid mode.
Custom user menu	If the first name in the list passed to MENU is *not* STO then a custom user menu is created which is similar to the regular USER menu. Notably, it can be used to give access to variables, programs and directories.
CUSTOM	Use this command (located on the USER key) to activate the custom menu.
Notes	This is most useful in programs to generate a list of user choices. Unfortunately, the VARS command (see **MEMORY menu**) does not return the names in the CUSTOM menu when it is the active menu! Thus, it is not possible for a program to save the contents of the current CUSTOM menu and temporarily replace it by another customized menu. But it is possible to write a modified MENU command which not only creates a new CUSTOM menu but also stores the list of names in a global variable for later reference: `<<DUP 'GLBCST' STO CUSTOM>> 'MENUS' STO`

Integration

General	The integration symbol ∫ located on the "5" key can be used to integrate arbitrary functions numerically or polynomials (sums of powers of x) symbolically.
Symbolic integration	Example: `'x^3+2*x+5' 'x' 3 ∫` returns `'5*x + x^2 + 0.25*x^4'` • Stack level 3 contains the polynomial to integrate (or the variable where it is stored). • Level 2 contains the integration variables • Level 1 contains the degree of the polynomial to integrate
Numeric integration with explicit integration variable	Example: `'EXP(x)+5' {'x' 1 2} 1E-3 ∫` returns `9.67 0.01`: • Stack level 3 contains the function to integrate. The function result must be a real value. • Stack level 2 contains a list which specifies: The integration variable and the lower and upper limits of integration. The limits of integration must be real values. • Stack level 1 contains the desired absolute accuracy of the result. • The result after integration is 9.67… in stack level 2. • 9.669…E-3 in stack level 1 is the upper limit for the relative error. The absolute error is 9.669…E-3/9.67…=9.998…E-4 which is indeed better than the specified accuracy of 1E-3. In case the returned upper limit for the relative error is negative then the integral did not converge. The function to integrate may also be specified as a program which evaluates the integration variable and returns the function result on the stack. No value must be taken from the stack: `<<x EXP 5 +>> {'x' 1 2} 1E-3 ∫` returns `9.67 0.01`.
Numeric integration with implicit integration variable	Example: `<<EXP 5 +>> {1 2} 1E-3 ∫` returns `9.67 0.01`. • Stack level 3 must contain a program (or the name of a variable containing a program). The program implements the function to integrate and it must take one argument from the stack and return a single real result on the stack. • Stack level 2 contains a list which specifies the lower and upper limits of integration (real values). • Stack level 1 contains the desired absolute accuracy of the result.

Differentiation

General	The differentiation symbol ∂ (d/dx located on the "6" key) can perform symbolic differentiation of a very wide range of functions. Ie. many built-in functions of the HP-28S can be differentiated. In addition it is possible to specify derivatives for user-defined functions which the differentiation algorithm will use to generate complete differentials.
Complete differentiation	Invoked by issuing the ∂ command explicitly. Example: `'SIN(2*X)+X^2' 'X' ∂` results in `'COS(2*X)*2+2*X'`. If the variable X exists the result will be the differential evaluated at position X. An error occurs if X contains an improper object (ie. a list).
Partial differentiation	Invoked by using ∂ in an expression. Example: `'∂X(SIN(2*X)+X^2)' EVAL` results in `'∂X(SIN(2*X))+∂X(X^2)'`. The next EVAL will return `'COS(2*X)*∂X(2*X)+∂X(X)*2*X^(2-1)'` and then `'COS(2*X)*(2*∂X(X))+2*X'` and finally `'COS(2*X)*2+2*X'`. After this EVAL won't change the result any more. If necessary, use COLCT to simplify the resulting expression.
User functions	A user defined function or a program in functional form can be differentiated as well. Example: First create a function F(x,y) that takes two arguments: `<<→ a b 'a*b + a + b' >> 'F' STO` Then differentiate d/dx F(x,x+2): `'F(X,X+2)' 'X' ∂` . The result is: `'X+2+X+1+1'` and after COLCT `'4+2*X'`. Note that the program given above must contain an expression in single quotes. It cannot contain another program in <<>> brackets even though these kinds of program can be invoked in functional notation, see **Programs**.

User-defined differentials	Not all built-in functions can be differentiated symbolically. Example: `'%(100,3)'` is the functional notation of % and returns 3% of 100. `'%(X,3)' 'X' ∂` returns `'der%(X,3,1,0)'` because the derivative of % is not known. Here **der%** is the derivative of %. In general: `'∂X(F(X1,x2))' EVAL` returns `'derF(X1,x2,∂X(X1),∂X(X2))'` if F is not defined. `derF` is the unknow derivative of F. Each argument in the original function produces two arguments to the `derF` function. Further evaluation results in `'derF(X1,x2,0,0)'` if neither variable X1 nor X2 exists. Now, the point is that the user can specify the derivative which is then used by ∂: `<<→ x y dx dy '(x*dy+y*dx)/100'>> 'der%' STO` stores the user-defined derivative for %. After this `'%(X,3)' 'X' ∂` returns 0.03. Note: I'm not sure whether this is useful. The problem is that most interesting functions cannot be given in the required expressional notation. For example, a wide range of integrals I(x)=∫ F(k)*dk can only be calculated by numeric approximation. This can easily be done in a program and furthermore the derivative `derI` is already known: It is the function F that is being integrated. I(x) can even be executed in functional notation (see **Programs**) but ∂ requires the explicit notation as an expression in single quotes. Thus, `∂I(x)` will produce an error even though `derI` has been specified explicitly.

Evaluation Rules

Variable names	The contents of a variable replaces the variable name when: • … pressing a menu button showing the variable name • … the name is entered in the command line without quotes and ENTER is executed explicitly or implicitly. • … in a program the name is encountered without quotes. • … the **EVAL** command is executed on its quoted name. **Important**: • When the variable name contains *multiple* other names they are not evaluated when the variable is evaluated: `'A+B' 'C' STO C` recalls 'A+B' into the stack and does not evaluate A or B. Also, if C contained a list containing program or variable names the list components will not be evaluated when the variable is evaluated. Rather, the list is put back onto the stack as is. • However, when the variable contains a *single* other variable name it will be evaluated: `'A' 'C' STO C` puts the value of variable A onto the stack. • Names that do not reference an existing variable are left unchanged: `'W' PURGE W` puts 'W' onto the stack. • To retrieve the contents of a variable to the stack without evaluating it use the **RCL** command. • To edit the contents of stack level 1 use the **EDIT** command. • A shortcut for editing the contents of a variable is the **VISIT** command. It is a combination of **RCL** and **EDIT**. See **Direct Key Commands**.
Programs	Essentially the same rules apply as for variables. • "Evaluation" of a program means the execution of the program. • To edit a program use the **VISIT** shortcut. • If a variable contains the name of a program *only* then the program is executed when the variable is recalled. • If a program A contains the unquoted name of another program B, the program B will be executed as a subroutine as soon as execution of program A encounters the symbol B.
Symbolic constants	These are: **e, i, MINR, MAXR, π**. See **Direct Key Commands**. Flag 35 determines how these are evaluated: • When set these symbolic constants evaluate to their symbolic form. Use →NUM to convert them to numerical values. • When clear these symbolic constants evaluate to their numerical value.

Expressions, Equations, Functions	Flag 36 specifies how equations or functions with symbolic arguments are evaluated: • When set, the evaluation of an expression is taken only a single step further by replacing variables with their contents (which may either be another expression or a numerical value). EVAL may have to be applied repeatedly to resolve all dependencies. Consider: `'A+B' 'A' STO A` returns `'A+B'`. After EVAL it is `'A+B+B'` then `'A+B+B+B'` etc. • When clear, the variables are evaluated until a numerical result is reached. If a symbolic name is undefined an error occurs. Do not try the above excample with flag 36 clear because it will produce an endless recursion loop!! Consequences: • In symbolic mode `'Q' 1/x` returns `'INV(Q)'` even if the variable Q exists and has a numerical value. By pressing EVAL the expression is evaluated and the numerical result returned. • In numeric mode an error occurs if Q is not an existing variable. Otherwise Q is evaluated and a numerical result is returned. • Overall the symbolic evaluation mode seems to be more handy.

Flags

General	For testing and modifying flags see **TEST Menu**.
1 – 64	User flags without predefined meaning.
31 – 64	System flags:
31	LAST activated. **MODE Menu**: LAST
32	Protocol. **PRINT Menu**: TRAC
33	Printhead right. See **PRINT Menu**.
34	Principal value. **ALGEBRA Menu**: ISOL and QUAD
35	Symbolic evaluation of constants. See **Evaluation Rules**.
36	Symbolic evaluation of functions. See **Evaluation Rules**.
37 – 42	Length of binary words (1-64), default is 64 **BINARY menu**: STWS
43, 44	Binary number base. 00=decimal, 01=binary, 02=octal, 03=hex **BINARY menu**: DEC, HEX, OCT, BIN
45	Display in level 1: Single line or multi-line **MODE menu**: ML
46	(reserved)
47	Double space printing. See **PRINT Menu**.
48	Decimal sign **MODE menu**: RDX, Important: If a period is used as the decimal sign then the comma will function as number separator – and vice versa!
49, 50	Real number format **MODE Menu**: STD, ENG, FIX, SCI
51	Acoustic signal: When set a BEEP occurs for every keypress.
52	Fast print mode. See **PRINT Menu**.
53 – 56	Number of decimal digits (0-11) **MODE Menu**: STD, ENG, FIX, SCI
57	`Underflow` condition action (magnitude smaller the 1E-499): • If set an underflow will be treated as an error and abort any programs. • If clear, the program will continue, flag 61 or 62 is set and the value 0 is returned.
58	`Overflow` condition action (magnitude larger than 1E499): See "Underflow action" above. If clear, ±MAXR is returned.

59	`Infinite result` condition action (ie. ln(0), tan(90°)): See "Underflow action" above. If clear, ±MAXR is returned.
60	Angle mode (degrees or radians) **MODE menu**: DEG, RAD
61	`Underflow-` exception has occurred
62	`Underflow+` exception has occurred
63	`Overflow` exception has occurred
64	`Infinite result` exception has occurred

Reserved Variables

General	When needed the variables listed below are created automatically. Important: The variables are always created in the current directory! This means that when working in different subdirectories different sets of these variables can be kept. In other words: Different subdirectories can hold different sets of statistical data or different parameter sets for function plots and changing from one directory to another changes the entire context for SOLVR and DRAW. For directory issues see **MEMORY Menu**.
EQ	Name of the current equation used by SOLVR and DRAW, see **SOLV Menu** and **PLOT Menu**.
∑PAR	Parameter list for statistics commands, see **STAT Menu**.
PPAR	Parameter list for DRAW commands, see **PLOT Menu**.
∑DAT	Array of statistics variables, **STAT Menu**.
s1, s2…	Created by ISOL and QUAD to indicate arbitrary signs. See **SOLV Menu** and **ALGEBRA Menu**.
n1, n2…	Created by ISOL and QUAD to indicate arbitrary integer numbers. See **SOLV Menu** and **ALGEBRA Menu**.

System Operations

Contrast adjustment	• Press and hold ON • Press + or – to change the contrast • Release ON
Clear memory	• Press and hold ON • Press and and release INS and → (cursor right) • Release ON WARNING: This clears the entire memory including stack, variables, programs and flags. This is essentially a "restore to defaults".
Endless-loop interruption	• Press and hold ON • Press and release ↑ (cursor up) • Release ON This will clear the stack, go to HOME, clear UNDO, COMMAND, LAST, clear the CUSTOM menu, deselect TRACE printing. Note that a program or other lengthy operations can usually be interrupted by pressing "ON".
System test	• Press and hold ON • Press and release ↓ (cursor down) or ← (cursor left) • Release ON This will execute a system test once (↓) or repeatedly (←). Press ON/↑ to abort the test. The single test will also execute the keyboard test. *Note that the system test behaves differently with different software versions.*
Keyboard test	• Press and hold ON • Press and NEXT • Release ON This will execute a keyboard test where the user must press all keys from top left to bottom right. *Note that the system test behaves differently with different software versions.*
SYSEVAL	Execute subroutine at absolute address. For debug only. Warning: Addresses change between different versions of the HP-28S software. Calling an invalid address usually resets the machine and causes a "Memory lost" error. • `#10d SYSEVAL` return the software version number

Contents