



WP43S REFERENCE MANUAL

This manual documents *WP 43S*, a free scientific software for the calculator *DM42* of *SwissMicros*. You can redistribute *WP 43S* and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

WP 43S is published and distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. Please see the GNU General Public License at <http://www.gnu.org/licenses/> for more details.

This manual is preliminary still; it will change while we develop *WP 43S* in course of this project. We reserve the right to do so at any time. The fundamental principles of *WP 43S* will stay constant, however. Stay informed by watching https://gitlab.com/Over_score/wp43s

Copyright © 2015 - 2020 Walter Bonin, Auf der Platte 9, 61440 Oberursel, Germany

All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without prior written permission of the author. For the time being, the locations highlighted cyan are open construction sites – information is missing there or needs further discussion and investigation to be determined. Any contributions in this matter are highly appreciated.

HP is a registered trade mark of Hewlett-Packard.

The pictures on p. 156 and bottom of p. 157 were kindly supplied by SwissMicros as well as the drawing on p. 217, the picture on p. 215 by Martin Lorang. The plots in Appendix H are based on material found in Wikipedia. The other pictures, diagrams, and graphics were created by the author.

Internet addresses are specified as found and verified at 2019-06-26. Please note such addresses may change without notice at any time.

This manual is published in English since it became the *lingua franca* of our time (after Greek, Latin, and French) – using it we can reach the maximum number of people without further translations. I apologize to the people of other languages and inserted some ‘translator’s notes’ where applicable.

Printed in the USA

ISBN-13: 978-172950106-1
ISBN-10: 172950106-0

WP 43S would not have been created without our love for *Classics*, *Woodstocks*, *Stings*, *Spices*, *Nuts*, *Voyagers*, and *Pioneers*. Thus we want to quote what was printed in Hewlett-Packard pocket calculator manuals until 1980, so it will not fade:

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sales."

*Statement of Corporate Objectives
Hewlett-Packard*

DRAFT

TABLE OF CONTENTS

Welcome!	9
Print Conventions and Common Abbreviations	9
Section 1: Index of Items (<i>IOI</i>)	12
0 - 9	16
A	17
B	20
C	23
D	28
E	32
F	34
G	37
H	39
I	39
J	42
K	44
L	45
M	49
N	55
O	56
P	57
Q	60
R	60
S	68
T	75
U	77
V	78
W	79
X	80
Y	84

Z	84
A, α	84
β	86
Γ, γ	86
Δ, δ	86
ε	86
ζ	87
Π, π	87
Σ, σ	87
X	89
(, +, -, ×, /, ^	90
→,	91
%	94
The Rest	95
 Names of System Variables and System Flags	98
Purposes of System Flags	103
Nonprogrammable Commands and Keys	107
Command Parameter Input and Closing It	107
Alphanumeric Input in X and Closing It	110
 Section 2: Menus and Catalogs	114
One to Find and Rule Them All – the CATALOG	115
Accessing Cataloged Items Rapidly	118
Further Menus and Their Contents	120
Constants	129
Unit Conversions	139
 Section 3: Calling and Executing Operations	149
Using XEQ for Executing Operations	149
Operations Requiring Trailing Parameters	150
Operations Changing Data Types	152

Appendix A: Hardware	155
Appendix B: Memory Management	160
Memory Map	160
Data Types	161
Statistical Summation Registers	164
SAVE and LOAD...	164
Range of Real Numbers	165
Limitations	170
What Happens when Changing Word Size?	172
Special Results	173
Program Step Size	178
Appendix C: Messages and Error Codes	181
Appendix D: Comparison to the Function Sets of <i>HP-42S</i>, <i>HP-16C</i>, <i>HP-21S</i>, and <i>WP 34S</i>	187
Corresponding Operations on <i>HP-42S</i>	187
Corresponding Operations on <i>HP-16C</i>	193
Corresponding Operations on <i>HP-21S</i>	195
Corresponding Operations on <i>WP 34S</i>	197
New Commands on your <i>WP 43S</i>	201
Reference Literature	206
Appendix E: Emulating a <i>WP 43S</i> on Your Computer	208
Appendix F: Flashing and Updating Your <i>WP 43S</i>	214
How to Create Your <i>WP 43S</i> by Flashing a <i>DM42</i>	214
How to Update Your <i>WP 43S</i>	216
Overlays	217
Appendix G: Troubleshooting Guide	218
Calculator Frozen	218
Fresh Battery Constantly Low	219

Keymap Trouble	219
Appendix H: Advanced Mathematical Functions and Tasks	220
Number Generating Functions	220
Statistical Distribution Functions (PMF, PDF, CDF, etc.)	222
More Statistical Formulas, also for Curve Fitting	231
Curve Fitting Models Provided	237
Error Propagation in Calculations	244
Solving Differential Equations	246
Orthogonal Polynomials	250
Even More Mathematical Functions	255
Appendix I: Information for Advanced Users	261
Recursive Programming	261
Building <i>WP 43S</i> Almost from Scratch	262
Index of Everything Provided	264
Appendix J: Release Notes	271
WP 43S Quick Reference Guide	Q-1
USING MENUS	1
MEMORY	1
DATA TYPES	2
MODES	3
DISPLAY FORMATS	3
PROGRAMMING	4
EXECUTING FUNCTIONS AND PROGRAMS	4
CLEARING AND DELETING	6
MATRIX OPERATIONS	7
PROBABILITY	8
STATISTICS	9
ADVANCED OPERATIONS	10
OPERATIONS ON SHORT INTEGERS	12
OPERATIONS ON ALPHANUMERIC STRINGS	13

Background Considerations and Facts

B-1

Accessing Items	1
Alpha Register	3
Angles	3
Backward Compatibility	4
Calculation Internals	5
Character Sets	5
Complex Notation and Storage	7
Display Limits	8
Display Segmentation	14
Echo and Fallback	16
Equations	16
Layouting	17
Menus	18
Number Range	18
Plotting?	19
Precision and Accuracy	21
Prefixes	21
Sorting in Detail	22
Stack Size	27
Stack Lift Disabling Functions	28
Structured Programming	28
UNDO	29

WELCOME!

This is the reference volume of the *WP 43S* documentation. It supplements the *WP 43S Owner's Manual* with detailed information about each and every *item* (i.e. command, *menu*, *catalog*, browser, application, constant, conversion, digit, and character) provided in your *WP 43S*. The *Index of Items* in Section 1 takes over a third of this volume.

Section 2 presents the structure and contents of all *menus* and *catalogs*. Section 3 shows further access methods to operations and lists all operations requiring at least one parameter.

The appendices cover additional special topics as listed in the *Table of Contents* above.

Enjoy!

Walter Bonin

Print Conventions and Common Abbreviations

Throughout this manual, standard text font is Arial. Emphasis is added by underlining or **bold** printing. Calculator COMMANDS, MENUS, PREDEFINED VARIABLES and SYSTEM FLAGS are generally called by their *names*, printed capitalized in running text (*menus* underlined). Quoted text is printed blue (as well as translator's footnotes). Specific terms, titles, trademarks, names or abbreviations are printed in italics, hyperlinks in blue underlined italics. The latter will beam you to its target in the .pdf file – it cannot work in a printed copy for obvious reasons; thus such a link generally refers to a page number, to the Table of Contents, or to a fully specified external address.

- Bold italic Arial letters such as *n* are used for variables; bold normal letters for constant **sample values** (e.g. specific labels, numbers, or characters).
- Courier is used for file names, binary and hexadecimal codes, and describing numeric formats.
- Times New Roman regular letters are for unit symbols and for mathematical functions. Italics are for *unit names* in running text.
- Times New Roman bold capitals are used for **REGISTER ADDRESSES**, lower case bold italics for *register contents*. So e.g. the value *y* lives in *register Y* and *r45* in **R45**. Overall stack contents are generally quoted in the order [*x, y, z, ...*]. We keep the term *register* for the space where an individual object is stored, although the actual size of such a *register* may vary widely following the size of the object stored therein.
- This **KEY** font (created by Luiz Vieira of Brasil) is taken for references to calculator keys, including **SOFTKEYS** in general. For shifted operations like **GTO** or **LBL**, the respective color is used. **Alphanumeric** and numeric calculator outputs (like 1.234×10^{-56} or $7,089 \cdot 10^{-12}$) are printed as you see them on the calculator screen.
- We will use decimal points in most parts of this manual (but you may set your WP 43S to decimal commas as well, of course). Although that point is less visible than a comma, ‘comma people’ seem to be more tolerant against points used as radix marks than vice versa (based on the number of complaints read).

All this holds unless stated otherwise locally.

The following abbreviations are used throughout this manual:

ADM = angular display mode (see Section 2 of the OM).

AIM = alpha input mode (see Section 2 of the OM).

ASL = automatic stack lift (see Section 1 of the OM).

BCD = binary coded decimal.

- CDF* = cumulated distribution function (see *Section 2* of the *OM*).
DT = data type (see *Appendix B*).
FM = flash memory (a special kind of *RAM*, see the *OM*, *Sect. 3*).
GP = general purpose.
HP = *Hewlett-Packard*.
IOI = *Index of Items* (see pp. 12ff).
LCD = liquid crystal display.
PDF = probability density function (see *Section 2* of the *OM*).
OH = Owner's Handbook.
OM = Owner's Manual.
PEM = program-entry mode (see *Section 3* of the *OM*).
PMF = probability mass function (see *Section 2* of the *OM*).
px = pixels.
RAM = random access memory, allowing read and write.
RPN = reverse Polish notation (see *Section 1* of the *OM*).
SRS = subroutine return stack (see *App. B* on pp. 160ff).
TVM = *Time Value of Money* – a preprogrammed application for dealing with investments and loans, featured by all financial *HP* calculators since 1972 (see the *OM*, *Section 5*).

Some more abbreviations may be used and explained locally.

SECTION 1: INDEX OF ITEMS (10)

All the *items* provided on your WP 43S (more than 850) are listed below with their *names* (as they are displayed and printed in routines) in column 1 and in column 2 the keystrokes necessary to call them. Most *items* shall be picked from *menus* (see pp. 114ff). For such *items*, we list the keys calling the respective *menu*, the *prefix* of the respective row (if applicable), and the label as shown therein; we are confident you will find the corresponding *softkey*. *Items* stored in CONST are listed with their *names* only since they are sorted alphabetically and will be explained in detail in a separate chapter below.

Each item provided is identified by its unique reserved name of up to 7 characters – it may be accessible under one or more different labels printed on the bezel or displayed in *menus*, featuring less or more characters than its *name* (see some unit conversions, for example). These labels are not required to be unique.¹

On your WP 43S, sorting (e.g. of names) works in the following order:²

Accented letters follow their parents, as do superscripts and subscripts.

In principle, *WP 43S* operations work as the corresponding ones did on the *WP 34S* where applicable (see *App. E*). Referring to vintage *HP* calculators, most functions and keystroke programming will work as they did on the *HP-42S*, bit and integer functions as on the *HP-16C*, unless

¹ Actually, there are two separate sets of *items*: set 1 contains commands, constants, *menus*, global program labels, and reserved symbols; set 2 is for *registers*, *system flags* and variables. The *name* of each *item* must be unique in its set.

² Characters printed on grey background are inaccessible for users for the time being. The entire sorting table covering all characters is printed in an appendix.

specified otherwise. Also for functions inspired by other vintage calculators as mentioned in the index below, their manuals may contain helpful additional information.

Some 300 functions featured in your *WP 43S* are new compared to *HP's RPN* pocket calculators. Operations carrying familiar *names* but deviating in their functionality from previous *HP RPN* calculators or the *WP 34S* are marked light red.³

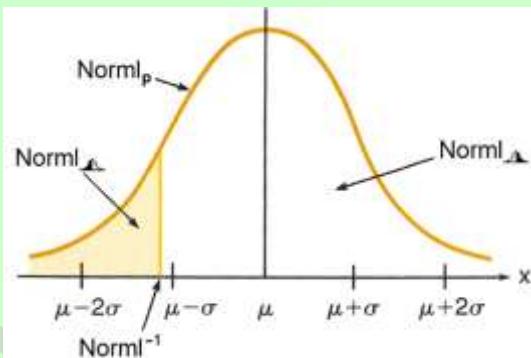
Operations working with the accumulated statistical data are marked light blue. For probability distributions, the naming rules are as pictured here for the *normal* distribution (cf. the OM).

Operations whose results are reflected in the *status bar* are printed on grey in the first column. Commands asking for your confirmation are printed red. Those printed red or orange cannot be undone or reverted by UNDO.

All operations may be also entered in *PEM* unless marked violet – on the other hand, the majority of functions contained in P.FN and TEST carry most use in *PEM*.

For the vast majority of operations, their remarks in the table start with a number:

- (0) represents functions without any effects on the *stack* (e.g. mode setting functions);
- (1) is for *monadic functions*,
- (2) for *dyadic functions*, and
- (3) for *triadic functions* as defined in Section 1 of the OM;
- (-1) stands for functions pushing one object on the *stack* and



³ We did neither compare the *RPL* calculators nor the *HP Prime*. They are exceeding the realm of shirt pocket calculators.

(-2) for functions pushing two objects on the stack.

Note some functions overwrite two stack levels instead of pushing two values on it: e.g. →POL and →REC, as you may have expected.⁴

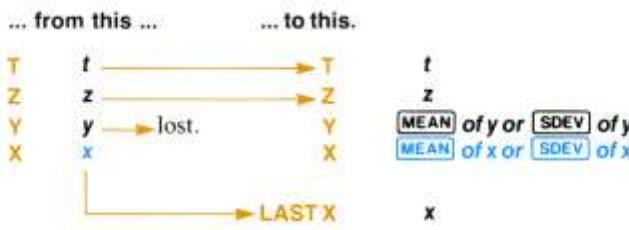
Operation or function **parameters** will be taken from the lowest *stack register(s)* unless mentioned explicitly in column 2 of the *IOI* – then they have to trail the command. Some parameters of statistical distributions shall be given in *registers I, J, and K* as specified.

Three examples of the parameter notation used throughout the *IOI* are shown below. Assume **R12** contains **15.67** generally here, i.e. **r12 = 15.67**.

1. **n** represents an arbitrary integer number which must be keyed in directly, while
 - n** represents such a number which may be specified indirectly via a *register* or variable as well (as shown in the addressing tables in *Section 1* of the *OM*); and
 - n** stands for the respective number itself;

⁴ On the *HP-42S*, however, also statistical functions returning two values do that – while the *Spices* (e.g. *HP-34C*) and *Voyagers* (e.g. *HP-15C*) push both results on the stack instead as you expect from *RPN* calculators. For your information, the picture below shows what the *HP-55*, *HP-19C/29C*, *HP-67/97*, *HP-41C*, and *HP-42S* do there:

The illustration below shows what happens in the stack when you execute **MEAN** or **SDEV**. The contents of the stack registers are changed...



Alas, *HP* does not give any reason for this deviation from simple logic until today. In our opinion this is not reasonable, so for *WP 43S* we stick to the paradigm as implemented on the *Voyagers* in this matter (as we did for *WP 34S* and *31S* before).

Example: RSD 12 rounds x to 12 significant digits, while
RSD →12 rounds x to 15 significant digits.

2. r (or s) represents an arbitrary *register address* or variable *name* which must be keyed in directly or picked from a *menu*, while
 \underline{r} (or \underline{s}) represents such an address or *name* which may be specified indirectly as well; and
 r (or s) stands for the contents of the address specified – r or s may be used as an address itself;

Example: STO 12 stores x into **R12**, while
STO →12 stores x into **R15**.

3. **label** represents an arbitrary program label which must be keyed in directly or picked from a *menu*, while
label represents such a label which may be specified indirectly (as shown in the addressing table in *Section 3* of the OM); and
label stands for the respective label itself, regardless of the way it was specified.

Example: GTO 12 goes to local label **12**, while
GTO →12 goes to local label **15**.

Note that for any command **XYZ** requiring one trailing input parameter, you can enter **XYZ → X** and it will take its parameter from **X** instead – like a good old traditional *RPN* command.

The *data types* a particular function operates on are listed in { } under “remarks” if there are any restrictions – cf. *App. B* on pp. 160ff. Many bit and integer functions make only sense operating on *short integers* (*DT 10*). Other functions typically work with more *DTs*. Functions stating *DTs 8** or *9** instead of 8 or 9 operate on each matrix element instead of the entire matrix (as explained in *Section 2* of the OM). Wherever operations return *DTs* differing from their input, the output types are listed as well.⁵

⁵ This applies for ${}^{\circ}\text{C} \rightarrow {}^{\circ}\text{F}$, for instance: For *real number* input, output will stay *real*. For *integer* input, however, output will become *real*.

ASL is enabled after each command except the following:

1. After CLX, ENTER \uparrow , $\Sigma+$, and $\Sigma-$ it is disabled (cf. Sect. 1 of the OM); thus, numeric input immediately following one of these four operations will overwrite x instead of pushing it on the *stack* as usual.⁶
2. The following (neutral) commands leave stack lift status as is: xxx

Below, the functions checked already are highlighted green, those which don't work yet (for whatever reason) are marked red. Green highlighting doesn't necessarily mean the function works correctly but its results look like in the right ballpark. What wasn't checked yet isn't highlighted at all. This applies to the respective *DTs*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	 etc.	(1) (2); {1} → {2}
$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$		Convert temperatures. See pp. 139ff.
10^x		(1) {1, 2, 3, 8*, 9*, 10} Returns 10^x , the inverse of $\lg(x)$.
1COMPL	 	(0) Sets 1's complement mode for operations on <i>short integers</i> . See Section 2 of the OM.

Some functions operating on *long integers* will return either such integers or *reals*, depending on the input value. E.g. $\sqrt[3]{x}$ will return 3 for an input of 27, i.e. for a proper cube, but will return a *real* for an input of 28 although this is a *long integer* as well. The same function operating on a *short integer* will return 3 for both cases, in whatever base applicable. See the OM, Section 2, *Integers: Summary of Functions*.

⁶ Some reasoning why ASL is disabled for these four:

- a) CLX is for clearing **X** to make room for a corrected value. This new value shall overwrite x – an extra zero on the stack makes no sense.
- b) ENTER \uparrow is a *stack lift* manually initiated by the user. An additional ASL immediately after this command makes no sense.
- c) $\Sigma+$ and $\Sigma-$ are dedicated commands for adding or subtracting data points (see the chapter about *Statistical Calculations* in Section 2 of the OM). These two commands were exclusively designed for data input since their first appearance on the HP-45 and are not really meant to be mixed with calculations.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$1/x$		(1) {2, 3, 8*, 9}; {1} → {2} Inverts the number x or all elements of the matrix x . Take $[M]^{-1}$ for inverting the matrix instead (see p. 90).
2COMPL		(0) Sets 2's complement mode for operations on <i>short integers</i> . See Section 2 of the OM.
2^x		(1) {1, 2, 3, 8*, 9*, 10} Returns 2^x , the inverse of $\text{lb}(x)$.
$\sqrt[3]{x}$		(1) {1, 2, 3, 8*, 9*, 10}; {(1)} → {(2)} Returns the cube root of x . See the DT tables in Section 2 of the OM for more.
ABS		Points to $ x $ on p. 93. Maintained for backward compatibility only.
ACOS		Points to arccos on p. 18.
$\text{ac} \rightarrow m^2$		(1) {2}; {1} → {2}
$\text{ac}_{\text{US}} \rightarrow m^2$		Convert areas. See pp. 139ff.
ADV		Menu. See p. 121.
AGM		(2) {2, 3}; {1} → {2} Returns the <i>arithmetic-geometric mean</i> of x and y . Will throw an error for x or y being negative. See p. 255 for more.
AGRAPH	 	(0) Alpha graphics. Displays a graphics image. Each character in the source s specifies an 8-dot-1-column pattern. The X- and Y-registers specify the pixel location of the bottom left point of this block (valid inputs are $1 \leq x \leq 400$ and $1 \leq y \leq 232$). So one row (8 px high) starting in column 1 needs up to 400 characters (in various sources) to specify – the more blank space is found in this row

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		the less characters are required for describing it. Cf. <i>HP-42S Owner's Manual</i> , pp. 135 – 140, and <i>HP-42S Programming Examples & Techniques</i> , pp. 195 – 223.
ALL	[g] [DISP] ALL n	(0) Sets the numeric display format to show all decimals of <i>real</i> or <i>complex numbers</i> whenever displayable (without trailing decimal zeros). ALL 0 works almost like ALL in HP-42S. For $x \geq 10^{16}$ (or earlier for <i>complex numbers</i>), display will switch to SCI or ENG (depending on ALLENG) with maximum displayable precision using the large font. The same will happen if $x < 10^{-n}$ and more than 16 digits are required to show x entirely (see examples in Section 2 of the OM). The limits differ in RBR – see p. 61.
AND	[f] [BITS] AND	(2) {10} Works bitwise as in HP-16C (see OM, Sect. 2). (2) {1, 2} → {1} Works as in HP-28S, i.e. x and y are interpreted before executing this operation. Zero is ‘false’; any other <i>real number</i> or <i>long integer</i> is ‘true’.
ANGLES	[f] [CATALOG] VARS ANGLES	Submenu of tagged angular variables defined at execution time. See pp. 115f.
arccos	[TRI] arccos	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $\text{arccos}(x)$. ⁷
arcosh	[g] [EXP] [g] arcosh [TRI] [g] arcosh	(1) {2, 3, 8*, 9*} Returns $\text{arcosh}(x)$.

⁷ Precisely, ARCCOS returns the principal value of $\text{arccos}(x)$, i.e. a *real* part $\in [0, \pi]$ in 4^r , or $\in [0, 1]$ in 4π , or $\in [0^\circ, 180^\circ]$ in 4° or $4''$, or $\in [0^\circ, 200^\circ]$ in 4° . Cf. ISO/IEC 9899.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
arcsin	TRI arcsin	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $\text{arcsin}(x)$. ⁸
arctan	TRI arctan	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $\text{arctan}(x)$. ⁹
arsinh	g EXP g arsinh	(1) {2, 3, 8*, 9*}
	TRI g arsinh	Returns $\text{arsinh}(x)$.
artanh	g EXP g artanh	(1) {2, 3, 8*, 9*}
	TRI g artanh	Returns $\text{artanh}(x)$.
ASIN	f CATALOG FCNS ASIN	Points to ARCSIN above. Maintained for backward compatibility only.
ASR	f BITS f ASR n	(1) {10}  Works like n (≤ 63) consecutive ASR commands in HP-16C, corresponding to a division of x by 2^n . ASR 0 executes as NOP, but loads L. See SR and Section 2 of the OM.
ASSIGN	f ASN item, location	(0) Assigns an <i>item</i> (like a function, menu, label, or character) to a specific location on the keyboard or in a <i>menu</i> . See the OM, Sect. 6.
ATAN	f CATALOG FCNS ATAN	Points to ARCTAN above. Maintained for backward compatibility only.

⁸ Precisely, ARCSIN returns the principal value of $\text{arcsin}(x)$, i.e. a *real* part $\in [-\pi/2, \pi/2]$ in 4^{r} , or $\in [-0.5, 0.5]$ in 4^{n} , or $\in [-90^\circ, 90^\circ]$ in 4^{d} or 4^{s} , or $\in [-100^\circ, 100^\circ]$ in 4^{g} . Cf. ISO/IEC 9899.

⁹ Precisely, ARCTAN returns the principal value of $\text{arctan}(x)$, i.e. a *real* part $\in [-\pi/2, \pi/2]$ in 4^{r} , for example (cf. ASIN), if SPCRES is set. Else the result interval for ATAN becomes $(-\pi/2, \pi/2)$ in 4^{r} , for example. Cf. ISO/IEC 9899.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
atm→Pa		(1) {2}; {1} → {2}
au→m		Convert pressures and distances. See pp. 139ff.
A:		Submenu. See p. 139.
BACK	 	(0) Jumps n steps backwards ($0 \leq n \leq 255$) in a program. E.g. BACK 1 goes to the previous program step. If BACK attempts to cross an END, an error is thrown. Reaching step 000 stops program execution and lights . See also SKIP. ATTENTION: If you edit a section of your routine crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually .
bar→Pa		(1) {2}; {1} → {2} Converts pressures. See pp. 139ff.
BATT?		(-1) {} → {2} Measures the battery voltage in the range between 1.9 V and 3.4 V and returns this value. Measurement resolution is 1mV. Voltages < 2.5 V are considered low, voltages < 2.0 V may lead to your WP 34S turning off automatically.
bbl→m ³	 	(1) {2}; {1} → {2} Converts volumes. See pp. 139ff.
BC?		(-1) {10} Tests if the specified bit in x is clear.
BEEP		(0) Sounds a sequence of four tones. See also TONE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
BeginP	 Begin	(0) Sets “Begin” mode in <i>TVM</i> : payments occur at the beginning of each period. Typical for savings plans and leasing. Compare ENDP.
BestF	 BestF <i>n</i>	<p>(0) Instructs your <i>WP 43S</i> to select the ‘best’ curve fit model for the current statistical data by picking the one with maximum <i>correlation</i> out of the models allowed (almost like <i>BEST</i> in <i>HP-42S</i>).</p> <p>Relevant for L.R., CORR, COV, s_{XY}, \hat{x}, and \hat{y}. You can accelerate computation of these functions significantly by excluding fit models making no sense for your data (e.g. for physical or technical reasons). The parameter <i>n</i> carries this information. Each fit model corresponds to a number as listed:</p> <ul style="list-style-type: none"> • LINF 1 • EXPF 2 • LOGF 4 • POWERF 8 • ROOTF 16 • HYPF 32 • PARABF 64 • CAUCHF 128 • GAUSSF 256 <p>Take the numbers of all models you can exclude and sum them up – the result is <i>n</i>.</p> <p>Example: Excluding the three 3-parameter models leads to $n = 64 + 128 + 256 = 448$. Hence, call BESTF 448 to look for the best-fitting 2-parameter model.</p> <p>Note ORTHOF is not part of the set of models under investigation. See pp. 231ff for more.</p>

Item	Keystrokes	Remarks (see pp. 12ff for general information)
BestF?	g INFO ▼ BestF?	(-1) {} → {1} Returns the 'best' curve fit model for the current statistical data by picking the one with maximum <i>correlation</i> out of the models allowed, encoded as an integer according to the list at BESTF.
Binom_p	g PROB g Binom: Binom_p etc.	(1) {2}; {1} → {2}
Binom_A		<i>Binomial</i> distribution with the number of successes g in X , the probability of a success p_o in I , and the sample size n in J . See p. 222 for more.
Binom_A		<i>Binom⁻¹</i> returns the maximum number of successes m for a given probability p in X , p_o in I and n in J .
Binom:		<i>Submenu</i> . See p. 124.
BITS	f BITS	<i>Menu</i> . See p. 118.
B_n	g X.FN B_n	(1) {1, 2}
B_n*	g X.FN B_n*	B _n and B _n * return the Bernoulli number for an integer n > 0 given in X , working with different definitions (see both formulas on p. 220).
BS?	f BITS g BS? n	(0) {10} Tests if the specified bit in x is set.
Btu→J	f U→ E: Btu→J	(1) {2}; {1} → {2} Converts energies. See pp. 139ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
cal→J	f [U] E: cal→J	(1) {2}; {1} → {2} Converts energies. See pp. 139ff.
CASE	g P.FN P.FN2 g CASE s	(0) Works like SKIP but takes the number of steps to skip from s. Example: Assume a program section: <pre> ... 100 CASE 12 101 GTO 01 102 GTO 02 103 GTO 07 104 GTO 05 105 LBL 01 ... 132 LBL 02 ... 153 LBL 05 ... 234 LBL 07 ... </pre> <p>Executing this program, $r12$ will be checked in step 100: if $r12 \leq 1$ then the program will proceed to step 101 and continue with a jump to step 105, for $r12 = 2$ the program will go to step 102, etc., resulting in a nice controlled dispatcher for $1 \leq r12 \leq 4$.</p> <p>ATTENTION: CASE might surprise you for $r12 > 4$ in the example above. Take care of the input you provide!</p> <p>If you edit a section of your routine crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually.</p>
CATALOG	f CATALOG	Catalog of everything. See pp. 115ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CauchF	f STAT ▼ f CauchF	(0) Selects the <i>Cauchy</i> (a.k.a. <i>Lorentz</i>) peak fit model. Relevant for CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
Cauch_p	g PROB f Cauch: Cauch_p etc.	(1) {2}; {1} → {2} Cauchy-Lorentz (a.k.a. <i>Lorentz</i> or <i>Breit-Wigner</i>) distribution with the location x_0 specified in I and the shape γ in J . See p. 225 for more. Cauch ⁻¹ returns x for a given probability p in X , with x_0 in I and γ in J .
Cauch_▲		
Cauch_△		
Cauch⁻¹		
Cauch:	g PROB f Cauch:	<i>Submenu</i> . See p. 124.
CB	f BITS g CB n	(1) {10} Clears the specified bit in x , i.e. sets it to 0 .
CEIL	g INTS g CEIL	(1) {8*}; {1, 2} → {1} Returns the smallest integer $\geq x$. Compare FLOOR.
CF	g FLAGS CF n g MODE CF n g CLR CF n	(0) Clears the <i>flag</i> specified, i.e. sets it to 0 .
CHARS	f CATALOG CHARS	<i>Submenu</i> of characters. See pp. 115ff.
chī→m	f U→ x: ▼ g chī→m	(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
CLALL	g CLR g CLall	(0) Clears all <i>registers</i> , numbered <i>user flags</i> , variables, and programs in <i>RAM</i> . Modes will stay as they are. See also CLCVAR, CLFALL, CLPALL, CLREGS, CLSTK, and RESET.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CLCVAR	g CLR CLCVAR	(0) Clears all variables used in the <i>current program</i> , i.e. sets all such <i>real</i> , <i>complex</i> and <i>integer</i> variables to zero, all <i>time</i> variables to 0:00:00 , all <i>date</i> variables to January 1st of year 0 , all <i>text strings</i> to zero length, and all the elements of all matrix variables used to 0 .
CLFALL	g FLAGS g CLFall	(0) Clears (after confirmation unless in <i>PEM</i>) all numbered global and local <i>user flags</i> (cf. CF). Lettered flags are not cleared since they may coincide with <i>system flags</i> .
	g CLR f CLFall	
CLK	g CLK	Menu. See p. 114.
CLLCD	g CLR f CLLCD	(0) Clears all pixels $\geq x$ and $\geq y$ on the screen.
CLMENU	g CLR CLMENU	(0) Clears all <i>menu key</i> definitions for the programmable <i>menu</i> . See MENU.
	g P.FN P.FN2 ...	
CLP	g CLR CLP	(0) Clears the <i>current program</i> in <i>RAM</i> or <i>FM</i> . Freed memory is returned to the pool of free space.
CLPALL	g CLR f CLPall	(0) Clears <u>all</u> programs in <i>RAM</i> . Cf. CLP.
CLR	g CLR	Menu. See p. 121.
CLREGS	g CLR f CLREGS	(0) Clears (after confirmation unless in <i>PEM</i>) all global and local <i>GP registers</i> allocated (see also LOCR), i.e. sets all these registers to 0 . The contents of the <i>stack</i> and L are kept.
CLSTK	g CLR f CLSTK	Clears all stack registers currently allocated (i.e. either X ... T or X ... D). All other <i>register</i> contents are kept. Cf. CLREGS.
	0 f FILL	
CLX	g CLR CLX	(1) Clears stack register X , disabling ASL. Cf. CLREGS and CLSTK.
	« (for closed x)	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CLΣ	g CLR CLΣ	(0) Clears the statistical summation <i>registers</i> and releases the memory allocated for them (see p. 164).
	f STAT g CLΣ	
CNST	g P.FN f CNST n	(-1) {} → (2) Returns the constant stored at position <i>n</i> in CONST (see below and pp. 129ff). Allows for indirectly addressing these constants, also beyond ∞ .
COMB	g PROB C_{y,x}	(2) {} Returns the number of possible <u>subsets</u> of <i>x</i> items taken out of a set of <i>y</i> items (i.e. choose <i>x</i> out of <i>y</i>). No item occurs more than once in a subset, and <u>different orders</u> of the same <i>x</i> items are <u>not counted</u> separately. Compare PERM.
		(2) {2, 3} See pp. 220f for the formula.
CONJ	g CPX conj	(1) {3, 9*} Returns the <i>complex conjugate</i> of <i>x</i> .
CONST	f CONST	Menu. Cf. CNST above and see pp. 129ff.
CONVG?	g TEST g CONVG? r	(0) {2} Checks for convergence by comparing <i>x</i> and <i>y</i> as determined by the lowest five bits of <i>r</i> . a) The very lowest 2 bits set the tolerance limit: $0 = 10^{-14}, \quad 1 = 10^{-24}, \quad 2 = 10^{-32}$. b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the numbers <i>x</i> and <i>y</i> relatively, 1 = compare them absolutely.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		c) The top bit tells how special numbers will be treated: 0 = NaN and $\pm\infty$ are considered converged, 1 = they are not considered converged. Now, $r = a + 4b + 16c$.
CORR	f STAT ▲ r	(-1) {} → {2} Returns the coefficient of correlation for the current statistical data and curve fit model. See pp. 231ff for more.
cos	TRI cos	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the cosine of the angle in X (see Section 2 of the OM for details).
cosh	g EXP g cosh TRI g cosh	(1) {2, 3, 8*, 9*} Returns the hyperbolic cosine of x.
COV	f STAT ▲ cov	(-1) {} → {2} Returns the population covariance for the two data sets {x} and {y} entered via Σ+ , depending on the curve fit model selected. See s_{xy} for the sample covariance and pp. 231ff for more.
CPX	g CPX	Menu. See p. 121.
CPXS	f CATALOG VARS CPXS	Submenu of complex variables defined at execution time. See pp. 115f.
CPX?	g TEST ▲ CPX?	(0) Checks if x is complex. Returns true if X contains data of type 3 or 9 with nonzero imaginary part.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CROSS	f [MATX] f cross	(2) {8} Requires two <i>real</i> 2D or 3D vectors in X and Y and returns their cross product. Crossing of 2D vectors works as it does for <i>complex numbers</i> .
	g [CPX] cross	(2) {3} → {2} When two <i>complex numbers</i> are crossed, your WP 43S simply returns a <i>real number</i> that is equal to the signed <i>magnitude</i> of the resulting moment vector. See an example in the OM.
ct→kg	f [U→] m: f carat → kg	(1) {2}; {1} → {2}
cün→m	f [U→] x: ▶ g cün→m	Convert masses and distances. See pp. 139ff.
cwt→kg	f [U→] m: cwt→kg	
CX→RE	CC (works in run mode only)	(-1) {3} → {2}; {9} → {8}
	g [CPX] f CX→RE	Cuts a closed <i>complex</i> object <i>x</i> , putting either <ul style="list-style-type: none">• (for L) its <i>real</i> part in Y and its <i>imaginary</i> part in X or• (for Q) its <i>magnitude</i> in Y and <i>phase</i> in X.
DATE	g [CLK] DATE	(-1) {} → {6} Recalls the date from the real-time clock and displays it in the format selected. See D.MY, M.DY, and Y.MD. Furthermore, DATE shows the day of week (see <i>Section 2</i> of the OM).
DATES	f [CATALOG] VARS f DATES	Submenu of date variables defined at execution time. See pp. 115f.
DATE→	g [CLK] DATE→	(-2) {2, 6} → {1} Assumes <i>x</i> containing a date in the format selected (or a <i>real number</i> in corresponding format) and pushes its three components as integers on the stack. Reversible by →DATE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DAY		(1) {2, 6} → {1} Assumes x containing a date in the format selected (or a real number in corresponding format) and extracts the day.
DBLR		{10} Double word length commands for remainder, multiplication and division (see the HP-16C OH, Section 4, pp. 52ff).
DBLx		DBLR and DBL/ accept a double size dividend in Y and Z (most significant bits in Y), the divisor in X as usual, and return the result in X .
DBL/		DBLx takes x and y as factors as usual but returns the product in X and Y (most significant bits in X).
dB→fr		(1) {2}; {1} → {2}
dB→pr		Convert ratios. See pp. 139ff.
DEC		(0) {1, 2, 10} Decrements r by 1. Does not load L even for target address X .
DECOMP		(-1) {1, 2} → {1} Decomposes x (after converting it to an improper fraction, if applicable), returning a stack [denominator(x), numerator(x), ...]. This works with maximum precision always, regardless of the settings of DENMAX, DENFIX, and DENANY. Example: For $x = 2.25$, DECOMP returns $x = 4$ and $y = 9$.
DEG		(0) Sets the ADM to decimal degrees.
DEG→		(1) {1, 2, 4} → {4} Converts angles as described on p. 147.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DELITM	g CLR g DELITM	(0) Deletes a user-defined <i>item</i> from memory. See the OM, Section 6.
DENMAX	g MODE f DENMAX	(1) Works like /c on HP-35S, but the maximum legal denominator is 9 999. For $x < 1$ or $x > 9\ 999$, DENMAX will be set to 9 999. For $x = 1$, the current DENMAX setting is recalled, replacing x .
DET	f CAT. FCNS DET	Points to M explained on p. 93. Maintained for backward compatibility only.
DIGITS	f CAT. f DIGITS	Submenu of digits defined. See pp. 115ff.
DISP	g DISP	Menu. See p. 121.
DOT	g CPX dot	(2) {3} → {2} Returns $Re(x)\ Re(y) + Im(x)\ Im(y)$
	f MATX f dot	(2) {8} → {2}; {9} → {3} Requires two matrices in x and y and returns their dot (scalar) product. The dot product is defined as the sum of the products of the corresponding elements in both matrices. Note both matrices must be of matching size; else DOT will throw an error. See the OM, Section 2.
DROP	g DROP↓	Drops x ... from the stack. See Section 1 of the OM for details.
DROPy	g STK DROPy	Drops y
DSE	f LOOP DSE r	(0) {1, 2, 10} Given $r = ccccc.fffii$, DSE decrements r by ii , skipping next program step if then $cccccc \leq fff$ (cf. the rear side of your WP 43S). If r features no fractional part then fff is 0. If $ii = 0$, $cccccc$ will be decremented by 1. DSE does not load L even for target address X. Note that neither fff nor ii can be negative, and DSE is only sensible with $cccccc > 0$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DSL		(0) {1, 2, 10} Works like DSE but skips if $cccc < fff$.
DSTACK		(0) Sets the maximum number of <i>stack registers</i> displayed. For an input of 1, only x will be shown directly above the <i>menu section</i> ; for 2, x and y will be displayed; maximum input is 4. Expanded views of e.g. matrices and dual- or multi-level returns like SUM will work as described in the OM regardless of the DSTACK parameter set. In any case, command input will be echoed directly below the <i>status bar</i> . This command is for old-school calculator users who feel distracted by a multitude of <i>stack registers</i> displayed changing simultaneously while only the lowest ones are really relevant.
DSZ		(0) {1, 10} Decrements r by 1 and skips the next step if $r = 0$ thereafter. Does not load L even for target address X . Cf. HP-16C.
D.MS	 (for closed input)	(0) Sets the ADM to <i>sexagesimal degrees</i> .
D.MS→		(1) {1, 2, 4} → {4}
D.MS→D		Convert angles as described on p. 147.
D.MY		(0) Sets the format dd.mm.yyyy for <i>dates</i> .
D→D.MS		(1) {1, 2, 4} → {4} Converts angles as described on p. 147.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
D→J		(1) {2, 6} → {1} Assumes x containing a <i>date</i> in the format selected (or a <i>real number</i> in corresponding format) and converts it to a <i>Julian day number</i> according to the J/G setting. Please see p. 43.
D→R		(1) {1, 2, 4} → {4} Converts angles as described on p. 147.
EIGVAL		(-1) {8, 9} Evaluates the matrix x and pushes a matrix containing its eigenvalues on the stack.
EIGVEC		(-1) {8, 9} Evaluates the matrix x and pushes a matrix containing its eigenvectors on the stack.
END		(0) Last command in a program and terminal for searching local labels as described in the OM, Section 3. Cannot be skipped by a test. Works like RTN in all other aspects.
ENDP		(0) Sets “End” mode in TVM: payments occur at the end of each period. Typical for loans and investments. Cf. BEGINP.
ENG		(0) Sets engineer’s display format (see Section 2 of the OM).
ENORM		(1) {8, 9} → {2} Calculates the Euclidean norm of the matrix in \mathbf{X} . The Euclidean norm is defined as the square root of the sum of squares of all matrix elements. Works like FNRM on HP-42S. For a vector, ENORM returns its length. Compare $ x $ on p. 93.

See the OM, Sect. 2.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ENTER↑	ENTER↑	(-1) Separates two entries in input. Copies x into Y, disabling ASL. See p. 111 and the OM, Section 1, for details.
ENTRY?	g TEST g ENTRY?	(0) Checks the (internal) entry flag. It is set if: <ul style="list-style-type: none"> any character is entered in AIM, or any command is accepted for entry (be it via ENTER↑, a function key, or R/S with a partial command line). Useful in routines, e.g. after PAUSE.
EQN	g EQN	Menu. See p. 122.
EQ.DEL	g EQN f DELETE	Deletes an equation.
EQ.EDI	g EQN EDIT	Opens the Equation Editor to edit an existing equation.
EQ.NEW	g EQN NEW	Opens the Equation Editor to enter a new equation.
erf	g X.FN erf	(1) {2}; {1} → {2}
erfc	etc.	Returns the error function or its complement. See pp. 255ff for more.
ERR	g P.FN ERR n	(0) Raises the error specified. The consequences are the same as if the corresponding error really occurred, so e.g. a running routine will be stopped and the message will be thrown. See App. C on pp. 181ff for the error codes. Compare MSG.
EVEN?	g TEST f EVEN?	(0) Checks if x is integer (see INT?) and even.
e^x	e^x	(1) {2, 3, 8*, 9*, 10}; {1} → {2} Returns e^x , the inverse of $\ln(x)$. Note $e^{i\pi} = -1$. See the DT tables in Section 2 of the OM for more.
EXITALL	g P.FN P.FN2 EXITall	(0) Exits all menus.

See Section 4 of
the OM.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
EXP	g EXP	Menu. See p. 122.
ExpF	f STAT ▼ ExpF	(0) Selects the exponential curve fit model. Relevant for CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
Expon_p	g PROB	(1) {2}; {1} → {2}
Expon_λ	f Expon: Expon_p etc.	Exponential distribution with the rate λ in J . See pp. 222ff for more.
Expon_Δ		
Expon⁻¹		Expon ⁻¹ returns the survival time t_s for a given probability p in X , with λ in J .
Expon:	g PROB f Expon:	Submenu. See p. 124.
EXPT	f PARTS EXPT	(1) {1, 2} → {1} Returns the exponent h of the number $x = m \cdot 10^h$ displayed. Compare MANT.
e^x-1	g EXP f e^x- 1	(1) {2, 8*} For $x \approx 0$, this returns a more accurate result for the fractional part than e^x does.
E:	f U→ E:	Submenu. See p. 137.
FB	f BITS g FB n	(1) {10} Inverts ('flips') the specified bit in x .
FBR	g a.FN g FBR	(0) Font browser. Shows all characters designed for your <i>WP 43S</i> .
FCNS	f CAT. FCNS	Submenu of all functions. See pp. 115ff.
FC?	g FLAGS FC? n	(0) Tests if the specified <i>flag</i> is clear.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FC? <i>C</i>	g FLAGS f FC? <i>C</i> <i>n</i> etc.	(0) Tests if the specified flag is clear. Clears, flips, or sets this flag after testing, respectively.
FC? <i>F</i>		
FC? <i>S</i>		
fēn→m	f U→ x: ▶ g fēn→m	(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
FF	g FLAGS FF <i>n</i>	(0) Flips the flag specified.
FIB	g X.FN f FIB	(1) {1} Returns the Fibonacci number (4791 is maximum input). (1) {2, 3} Returns the extended Fibonacci number. See pp. 220f.
FILL	f FILL	Fills the whole stack with <i>x</i> .
FIN	g FIN	Menu. See p. 122.
FIX	g DISP FIX <i>n</i>	(0) Sets fixed point display format (see the OM, Section 2).
FLAGS	g FLAGS	Menu. See p. 122.
FLASH	f CAT. PROGS FLASH	Submenu of global labels defined in FM at execution time. See pp. 115f.
FLASH?	g INFO f FLASH?	(-1) {} → {1} Returns the number of free bytes in FM.
FLOOR	g INTS g FLOOR	(1) {8*}; {1, 2} → {1} Returns the greatest integer ≤ <i>x</i> . Cf. CEIL.
fm.→m	f U→ x: ▲ fathom → m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 139ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FP	f PARTS FP	(1) {1, 2, 8*, 10} Returns the fractional part of x . Compare IP.
FP?	g TEST f FP?	(0) Tests x for having a fractional part $\neq 0$.
F _p (x)	g PROB F: F _p (x)	(1) {2}; {1} → {2} <i>Fisher's F distribution. F_p(x) equals Q(F) on HP-21S. The degrees of freedom shall be specified in I and J. See pp. 222ff for more.</i>
F _Δ (x)	etc.	
F _Δ (x)		
F ⁻¹ (p)		
fr→dB	f U→ ▲ field ratio → dB	(1) {2}; {1} → {2} Converts ratios. See pp. 139ff.
FS?	g FLAGS FS? n	(0) Tests if the specified flag is set.
FS?C	g FLAGS	(0) Tests if the specified flag is set. Clears, flips, or sets this flag after testing, respectively.
FS?F	f FS?C n	
FS?S	etc.	
ft.→m	f U→ x: f ft.→m	(1) {2}; {1} → {2} Convert distances and volumes. See pp. 139ff.
ft _{us} →m	f U→ x: ▲ survey foot _{us} → m	
fz _{UK} →m ³	f U→ f V: f floz _{UK} → m ³	
fz _{us} →m ³	f U→ f V: f floz _{us} → m ³	
F:	g PROB F:	Submenu. See p. 124.
f'	g EQN f'	Submenus for calculating the 1 st or 2 nd derivative of a given equation. See the OM (Section 4) for more.
f''	g EQN f''	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$f'(x)$	 $f'(x) \text{ labl}$	(1, 2} → {2} $f'(x)$ [$f'(x)$] returns the 1 st [2 nd] derivative of the function $f(x)$ at position x . This $f(x)$ must be specified in a routine starting with LBL <i>labl</i> . On return, Y , Z , and T will be cleared and the position x will be in L . See Section 4 of the OM for more.
$f''(x)$	 	ATTENTION: $f'(x)$ and $f''(x)$ fill all <i>stack registers</i> with x before calling the routine specified.
F&p:		Submenu. See p. 139.
GAP	 	(0) Defines the interval for inserting digit group separators in <i>reals</i> . For integers, the intervals are fixed to 4 digits for binary and 3 for any other base – except 4, 8, and 16 where the interval is 2. In input, gaps will always be inserted as chosen for <i>reals</i> . After GAP 2, 1, or GAP 0, no group separators will be displayed in any number at all. See Section 2 of the OM.
GaussF	 	(0) Selects the Gauss peak fit model. Relevant for CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
GCD	 	(2) {1; 10} Returns the Greatest Common Divisor of x and y . ¹⁰ This will always be positive.
g_d	 etc.	(1) {2, 3}; {1} → {2} Returns the Gudermannian function or its inverse. See p. 256 for details.
g_d^{-1}		

¹⁰ $\text{GCD}(x, y) = \left| \frac{x}{\text{LCM}(x, y)} \right|$. See also LCM.

Translator's notes for French and German readers: GCD correspond à PGCD en français. GCD entspricht ggT auf Deutsch.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Geom_p	g PROB g Geom: Geom_p etc.	(1) {2}; {1} → {2} Geometric distribution: The CDF returns the probability for a 1 st success after $m = x$ Bernoulli experiments. The probability p_0 for a success in each such experiment must be specified in J . See pp. 222ff for more.
Geom_A		
Geom_A		
Geom⁻¹		Geom ⁻¹ returns the number of failures f before 1 st success for given probabilities p in X , p_0 in J .
Geom:	g PROB f Geom:	Submenu. See p. 124.
gl_{UK}→m³	f U+ f V= gl_{UK}→m³ etc.	(1) {2}; {1} → {2}
gl_{US}→m³		Convert volumes. See pp. 139ff.
GRAD	g MODE GRAD	(0) Sets the ADM to <i>grades</i> (a.k.a. <i>gradians</i> or <i>gon</i>).
GRAD→	g L→ f GRAD→	(1) {1, 2, 4} → {4} Converts angles as described on p. 147.
GTO	f GTO labl	(0) In <i>PEM</i> , inserts an unconditional branch to <i>labl</i> . Else positions the program pointer to <i>labl</i> .
GTO.	f GTO □ nnn	to local label <i>n</i> (specify ≤ 2 digits).
	f GTO □ labl	to step <i>nnn</i> – specify ≥ 3 digits until becoming definite in <i>current pgm</i> .
	f GTO □ ▲	(0) Positions the program pointer ... to the global label specified (shall be picked from PROG). directly <u>after previous END</u> , i.e. to the top of <i>current program</i> (see Sect. 3 of the OM); if being there already, jumps to the top of previous program.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
GTO.	f GTO	(0) Positions directly after next END, i.e. to the top of next program.
	f GTO	to the end of used program memory in RAM, i.e. right to the final END.
ha\rightarrowm²	f U\rightarrow f A: ha \rightarrow m ²	(1) {2}; {1} \rightarrow {2} Converts areas. See pp. 139ff.
H_n	g X.FN Orthog H_n	(2) {2}; {1} \rightarrow {2}
H_{np}	... Orthog f H_{np}	<i>Hermite polynomials</i> for probability (H_n) and physics (H_{np}). See p. 250 for details.
hp_E\rightarrowW hp_M\rightarrowW hp_{UK}\rightarrowW	f U\rightarrow P: hp _E \rightarrow W etc.	(1) {2}; {1} \rightarrow {2} Convert powers. See pp. 139ff.
Hyper_p Hyper_△ Hyper_▲ Hyper⁻¹	g PROB g Hyper: Hyper _p etc.	(1) {2}; {1} \rightarrow {2} <i>Hypergeometric distribution</i> with the number of successes g in X , the probability of a success p_o in I , the sample size n in J , and the batch size n_o in K . See pp. 222ff for the formula and the OM, Sect. 2, for an example. Hyper ⁻¹ returns the maximum number of successes m for a given probability p in X , p_o in I , n in J , and n_o in K .
Hyper:	g PROB g Hyper:	Submenu. See p. 124.
HypF	f STAT ▼ f HypF	(0) Selects the hyperbolic fit model. Relevant for CORR, COV, L.R., s _{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
IDIV	g INTS f IDIV	(2) {1, 10}; {2} \rightarrow {1} Integer division, working like]/ + [P] . ¹¹

Item	Keystrokes	Remarks (see pp. 12ff for general information)
IDIVR	 	(1, 2, 10) Like IDIV but returns also the remainder in Y . ¹¹
iHg→Pa	 	(1) {2}; {1} → {2} Converts pressures. See pp. 139ff.
Im	 	(1) {2, 3} → {2}; {9} → {8};
	 	Returns the imaginary part of <i>x</i> . Compare RE.
INC	 	(0) {1, 2, 10} Increments <i>r</i> by 1. Does not load L even for target address X .
INDEX	 	(1) Indexes a named matrix. You can also index a matrix by editing it (see M.EDIT or M.EDIN). After exiting the <i>Matrix Editor</i> , the matrix is no longer indexed. See also <i>Matrix Utility Functions</i> in the <i>HP-42S Owner's Manual</i> , pp. 223ff.
INFO		Menu. See p. 122.
INPUT	 	Works in programs only: Recalls the content of the source specified into X , displays the name of the source along with <i>r</i> , and halts program execution, allowing you to enter or calculate a value; pressing R/S then stores <i>x</i> into said destination and continues program execution – pressing EXIT instead cancels INPUT, so R/S thereafter will continue with the source content as it was. If you use an input variable <i>name</i> undefined at execution time, INPUT automatically creates the variable with an initial value of zero.

¹¹ See the OM, Section 2, for the DTs of quotient and remainder, if applicable.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
INTS		Menu. See p. 122.
INT?		(0) Tests x for being an integer, i.e. having a fractional part equal to zero. Cf. FP?.
INVRT		Works like $[M]^{-1}$ on p. 93. Maintained for backward compatibility only.
in. \rightarrow m		(1) {2}; {1} \rightarrow {2} Converts distances. See pp. 139ff.
IP	 	(1) {1, 81}; {2, 10} \rightarrow {1} Returns the integer part of x . Cf. FP.
ISE		(0) {1, 2, 10} Given $cccccc.ffffii$ in the source s , ISE increments s by ii , skipping next program step if $cccccc \geq ffff$ then (cf. the rear side of your WP 43S). If s has no fractional part then $ffff = 0$ and $ii = 0$. If $ii = 0$, $cccccc$ will be incremented by 1. ISE does not load L even for target address X . Note that neither $ffff$ nor ii can be negative, but $cccccc$ can.
ISG		(0) {1, 2, 10} Works like ISE but skips if $cccccc > ffff$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ISM?	g [INFO] ISM?	(-1) {} → {1} Returns the <i>integer sign mode</i> set for <i>short integers</i> : true 2 for 2's complement, true 1 for 1's complement, false 0 for unsigned, or true -1 for sign & mantissa mode.
ISZ	f [LOOP] ISZ s	(0) {1, 10} Increments <i>s</i> by 1, skipping next program step if <i>s</i> = 0 thereafter. ISZ does not load L even for target address X. Known from HP-16C.
I _{xyz}	g [X.FND] f I _{xyz}	(3) {1, 2} Returns the <i>regularized Beta function</i> . See p. 256.
IΓ _p	g [X.FND] f IΓ _p	(2) {1, 2}
IΓ _q	etc.	Returns the <i>regularized Gamma function</i> (one of two kinds).
I+	f [MATX] ▲ I+	(1) Increments or decrements the row index <i>i</i> of the indexed matrix. See INDEX and also J+, J-, RCLEL, STOEL, RCLIJ, and STOIJ.
I-	f [MATX] ▲ I-	
I/O	f I/O	Menu. See p. 122.
J _y (x)	g [X.FND] g J _y (x)	(2) {2}; {1} → {2} Returns the <i>Bessel function of first kind</i> and order <i>y</i> . See p. 257 for details.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
J+	f MATX ▲ J+	(1) Increments or decrements the column index j of the indexed matrix. If GROW is set and i and j are pointing at the last element of the matrix, executing J+ creates a new row at the end of the matrix. See INDEX and also I+, I-, RCLEL, STOEL, RCLIJ, and STOIJ.
J-	f MATX ▲ J-	
J/G	g CLK ▲ f J/G	(0) {2, 6} Takes x specifying the date the <i>Gregorian</i> calendar became valid in the region you are interested in. This x shall be entered as a real number formatted according to the <i>DDM</i> set (D.MY, M.DY, or Y.MD) reflecting the crucial <i>Gregorian</i> date. If x is later than the current J/G setting (see J/G?), it may be also entered as a <i>Gregorian date</i> ; if x is earlier than the current J/G, it may be entered as a <i>Julian date</i> instead.
J/G?	g INFO ▲ J/G?	(-1) {} → {6} Returns the current setting of J/G (see there).
J→Btu	f U→ E: J→Btu etc.	(1) {2}; {1} → {2}
J→cal		Convert energies. See pp. 139ff.
J→D	g CLK f J→D	(1) {1} → {6} Takes x as a <i>Julian day number</i> ¹² and converts it to a common date according to J/G (see above) and the date format selected.
J→Wh	f U→ E: J→Wh	(1) {2}; {1} → {2} Converts energies. See pp. 139ff.

¹² Translator's note: *Julian day number* translates to «jour Julien» in French and to “Julianisches Datum” or “Julianische Tageszahl” in German. See the corresponding articles in Wikipedia for more information about these counting numbers. Note a *Julian day number* differs from a *Julian (calendar) date*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
KEY		See KEYG and KEYX below.
KEYG	g [P.FN] P.FN2 KEYG key#, <u>labl</u>	Specifies the label to be branched to (KEYG) or called (KEYX) when a particular softkey is pressed. KEYG and KEYX work in PEM only and will be translated to a program step
KEYX	g [P.FN] P.FN2 KEYX key#, <u>labl</u>	<p>KEY key# GTO labl or KEY key# XEQ labl ,</p> <p>respectively. Key numbers go from 1 to 18 with 1 corresponding to F1, 9 to f [F3], and 14 to g [F2], for example.</p>
KEY?	g [TEST] g KEY? <i>r</i>	(0) Tests if a key was pressed while a routine was running or paused. If no key was pressed in that interval then the next program step after KEY? will be executed; else it will be skipped and the code of said key will be stored in <i>r</i> . Key codes reflect the rows and columns on the keyboard (see the OM, Section 3; cf. GETKEY on HP-42S).
kg→ct	f [U→] m: f kg → carat	(1) {2}; {1} → {2} Convert masses. See pp. 139ff.
kg→cwt	f [U→] m: kg→cwt etc.	
kg→lb.		
kg→oz		
kg→scw	f [U→] m: f kg → sh.cwt	
kg→sto	f [U→] m: f kg → stone	
kg→s.t	f [U→] m: ▲ kg → short ton	
kg→ton	f [U→] m: ▲ kg→ton	
kg→trz	f [U→] m: f kg → tr.oz	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
KTYP?	KTYP?	<p>(-1) {} → {1}</p> <p>Assumes a key code in the address specified (cf. KEY?), checks it, and returns its key type:</p> <ul style="list-style-type: none"> • 0 ... 9 if it corresponds to a digit ... • 10 if it corresponds to , , or , • 11 if it corresponds to or , • 13 if it corresponds to a softkey, and • 12 if it corresponds to any other key. <p>Helps in user interaction with routines (see the OM, Section 3)..</p>
LASTx		(-1) See Section 1 of the OM. Actually, this command will be recorded as RCL L in routines.
lbf ^f →Nm	lbf ^f →Nm	(1) {2}; {1} → {2}
lbf→N	F&p: lbf→N	Convert torques and forces. See pp. 139ff.
LBL	labl	(0) Identifies programs and routines for execution and branching. Read more about labels and specifying them in Section 3 of the OM.
LBL?	LBL? labl	(0) Tests for existence of the label specified, anywhere in program memory. See LBL for more.
lb. ^f →kg	m: lb. ^f →kg	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.
LCM	LCM	(2) {1; 10} Returns the Least Common Multiple of x and y. ¹³ This will always be positive.

¹³ $\text{LCM}(x, y) = \left| \frac{xy}{\text{GCD}(x, y)} \right|$. Cf. GCD.

Translator's notes for French readers: LCM correspond à PPCM en français,
 Translator's notes for German readers: LCM entspricht kgV auf Deutsch..

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LEAP?	 	(0) {2, 6} Assumes x containing a date in the format selected (or a real number in corresponding format), extracts the year, and tests for a leap year.
LgNrm _p	 	(1) {2}; {1} → {2}
LgNrm _Δ		<i>Log-normal distribution with $\mu = \ln \bar{x}_g$ specified in I and $\sigma = \ln \varepsilon$ in J. See \bar{x}_g and ε below and pp. 222ff for more.</i>
LgNrm _Δ		<i>LgNrm⁻¹ returns x for a given probability p in X, with μ in I and σ in J.</i>
LgNrm:	 	Submenu. See p. 124.
lī→m	 	(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
LinF	 	(0) Selects the linear fit model. Relevant for CORR, COV, L.R., s _{XY} , \hat{x} , and \hat{y} . See pp. 222ff for more.
LJ	 	(10) Left justifies a bit pattern within its word size as in HP-16C. The stack will lift, placing the left-justified word in Y and the count of bit-shifts necessary to left justify the word in X. Example for word size 8: 1 0110 ₂ LJ returns $x = 3$ and $y = 1011\ 0000_2$
L _m	 	(2) {2}; {1} → {2}
L _{max}		<i>Laguerre polynomials and Laguerre's generalized polynomials. See pp. 237f for more.</i>
LN		(1) {2, 3, 8*, 9*, 10}; {1} → {2} Returns the natural logarithm of x . See the DT tables in Section 2 of the OM for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LNβ	g X.FN g lnβ	(2) {2, 3}; {1} → {2} Returns the natural logarithm of <i>Euler's Beta function</i> (see p. 86).
LNΓ	g X.FN g lnΓ	(1) {2, 3}; {1} → {2}
	g PROB ▲ lnΓ	Returns the natural logarithm of $\Gamma(x)$ (see p. 86). Allows also for calculating really great factorials (see an example in the OM).
LN(1+x)	g EXP f ln 1+x	(1) {2, 8*} For $x \approx 0$, this function returns a more accurate result for the fractional part than $\ln(x)$ does.
LOAD	f I/O LOAD	Restores the entire backup from the file in the <i>FAT system</i> in <i>FM</i> whereto it was written by SAVE . LOAD calls LOADP , LOADR , LOADV , LOADΣ , LOADSS , recalls the lettered <i>registers</i> (incl. the <i>stack!</i>), and signals Backup restored . ¹⁴
LOADP	f I/O LOADP	(0) Loads the entire program memory from backup and appends it to the programs already in <i>RAM</i> (if there is sufficient space – else an error will be thrown). ¹⁴
LOADR	f I/O LOADR	(0) Recalls the numbered <i>GP registers</i> from backup (incl. the <i>local registers</i> allocated). Lettered <i>registers</i> will not be recalled. ¹⁴ The number of registers copied is the minimum of the registers held in the backup and allocated in <i>RAM</i> at execution time.
LOADSS	f I/O LOADSS	(0) Recovers the system state from backup, ¹⁴ meaning the entire calculator <i>configuration</i> (user assignments, system variables and <i>flags</i> as covered by RESET – see p. 63) plus all global <i>user flags</i> .

¹⁴ LOAD and LOADP are not programmable. See **SAVE** on p. 73 and App. B (pp. 166ff).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LOADV	f I/O LOADV	(0) Recalls the user-defined variables from backup. ¹⁴
LOADΣ	f I/O LOADΣ	(0) Recovers the statistical summation registers from backup. Throws an error if there is none. ¹⁴
LocR	g P.FN g LocR n	(0) Allocates <i>n</i> local registers (≤ 99) and 16 local flags for the current routine. The new registers and flags are cleared. Any subsequent LOCR in the same routine, if applicable, will set the amount of local registers to the new number. Cf. the OM, Sect. 3.
LocR?	g INFO f LocR?	(-1) {} → {1} Returns the number of local registers currently allocated for the current routine.
LOG ₁₀	g lg	(1) {1, 2, 3, 8*, 9*, 10} ({1} → {2}) Returns the logarithm of <i>x</i> for base 10. ¹⁵
LOG ₂	g EXP lb x	(1) {1, 2, 3, 8*, 9*, 10} ({1} → {2}) Returns the logarithm of <i>x</i> for base 2. ¹⁵
LogF	f STAT ▼ LogF	(0) Selects the logarithmic curve fit model. Relevant for CORR, COV, L.R., s _{XY} , \hat{x} , and \hat{y} . See pp. 222ff for more.
Logis _p	g PROB	(1) {2}; {1} → {2} <i>Logistic</i> distribution with μ given in I and s in J . See pp. 222ff for details.
Logis _▲	f Logis: Logis _p	
Logis _△	etc.	
Logis ⁻¹		
Logis:	g PROB f Logis:	Submenu. See p. 124.

¹⁵ See the DT tables in Section 2 of the OM for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LOG_xy	f [EXP] log_{xy}	(2) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}) Returns the logarithm of y for the base number x. See the DT tables in Sect. 2 of the OM for more.
LOOP	f [LOOP]	Menu. See p. 123.
L.INTS	f [CATALOG] VARS L.INTS	Submenu of long integer variables defined at execution time. See pp. 115ff.
L.R.	f [STAT] ▲ L.R.	(-2) or (-3) {} → {2} Pushes the parameters a_2 (in Z), a_1 (in Y), and a_0 (in X) of the fit curve through the data points accumulated in the statistical summation registers on the stack, according to the curve fit model selected (see LINF, ORTHOF, EXPF, POWERF, LOGF, HYPF, ROOTF, PARABF, CAUCHF, GAUSSF). For a straight line, a_0 is its y-intercept and a_1 is its slope. See pp. 222ff for more.
l.y.→m	f [U→] x: l.y.→m	(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
m²→ac	f [U→] f A: m² → acre	(1) {2}; {1} → {2} Convert areas. See pp. 139ff.
m²→ac_{us}	... m² → acre_{us}	
m²→ha	... m² → ha	
m²→m²	... f m² → m²	
m³→bbl	f [U→] f V: f m³ → barrel	(1) {2}; {1} → {2} Convert volumes. See pp. 139ff.
m³→fz_{uk}	... f m³ → floz_{uk}	
m³→fz_{us}	etc.	
m³→gl_{uk}	... m³ → gl_{uk}	
m³→gl_{us}	etc.	
m³→qt.		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MANT	f PARTS MANT	(1) {2}; {1} → {2} Returns the mantissa m of the number $x = m \cdot 10^h$ displayed. Cf. EXPT.
MASKL	f BITS MASKL n	(-1) {} → {10} Work like MASKL and MASKR on HP-16C, but with the mask length (or its address) following the command instead of being taken from X. Thus, the mask is pushed on the stack. MASKL 0 and MASKR 0 return 0.
MASKR	f BITS MASKR n	Example: For WSIZE 8, MASKL 3 will return a mask word 1110 0000 ₂ . Use it e.g. for extracting the top three bits of an arbitrary byte via AND.
MATRS	f CATALOG VARS MATRS	Submenu of matrix variables defined at execution time. See pp. 115f.
MATR?	g TEST ▲ MATR? (0)	Checks if x is a <i>real</i> or <i>complex</i> matrix.
MATX	f MATX	Menu. See p. 123.
Mat_X	f MATX SIM EQ Mat X	(-1) Returns the solution vector for a system of linear equations (see the OM, Section 2).
max	g X.FN g max	(2) {1, 2, 4, 5, 6, 7, 10} Returns the maximum of x and y .
MEM?	g INFO MEM?	(-1) {} → {1} Returns the number of free bytes in RAM (i.e. the pool size), taking into account all registers currently allocated and their current contents.
MENU	g P.FN P.FN2 MENU	Displays the programmable menu. See the HP-42S OM, Part 2, Section 10, p. 146.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MENUS	f CAT. MENUS	Submenu of all <i>menus</i> defined at execution time. See pp. 115ff.
min	g X.FN g min	(2) {1, 2, 4, 5, 6, 7, 10} Returns the minimum of x and y .
MIRROR	f BITS f MIRROR	(1) {10} Reflects the bit pattern in x (e.g. $0001\ 0111_2$ would become $1110\ 1000_2$ for word size 8).
mi.→m	f U→ x: f mi.→m	(1) {2}; {1} → {2}
mmHg→Pa	f U→ F&p: ▲ mmHg → Pa	Convert distances and pressures. See pp. 139ff.
MOD	g MOD	(2) {1, 2, 10}
	g INTS f MOD	Returns $y \bmod x$ (modulo, see <i>Section 2</i> of the OM for examples). Cf. RMD.
MODE	g MODE	Menu. See p. 123.
MONTH	g CLK f MONTH	(1) {2, 6} → {1} Assumes x containing a <i>date</i> in the format selected (or a <i>real number</i> in corresponding format) and extracts the month.
MSG	g P.FN P.FN2 f MSG	(0) {1, 2} Throws the (<i>temporary</i>) error message specified by the integer part of x . Cf. ERR. See App. C on pp. 181ff for the respective error codes.
MULπ	g MODE MULπ	(0) Sets the <i>ADM</i> to <i>multiples of π</i> .
MULπ→	g L→ f MULπ→	(1) {1, 2, 4} → {4} Converts angles as described on p. 147.
MVAR	g P.FN f MVAR name	(0) Defines a <i>menu variable</i> . Such variables are required for VARMNU. Works in <i>PEM</i> only. See the OM, <i>Section 3</i> .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MyMenu	f [CAT] MENUS MyMenu	User menu. See the OM, Section 6.
Myα	f [CAT] MENUS Myα	User menu in AIM.
M.DELR	f DELR with M.EDIT displayed	(0) {8, 9} Deletes the current row of elements (where the cursor is in). Will not work if the matrix has only one row.
M.DIM	f [MATX] f DIM name	(0) {1, 2} Creates a new named matrix or re-dimensions an existing matrix to IP(y) rows and IP(x) columns. See DIM in the HP-42S Owner's Manual, p. 217.
M.DIM?	g [INFO] g DIM? f [MATX] ▲ f DIM?	{8, 9} → {1} Returns the dimensions of the matrix <i>x</i> (rows to Y, columns to X). Note the matrix is saved in L. Former y goes into Z, former z into T, etc.
M.DY	g [CLK] ▲ M.DY	(0) Selects the format mm/dd/yyyy for dates.
M.EDI	f [MATX] EDIT	(2) {8, 9} Opens <i>x</i> using the Matrix Editor (like MATRIX EDIT in HP-42S). ¹⁶ See Section 2 of the OM.
M.EDIN	f [MATX] f EDITN name	(2) Opens a named matrix using the Matrix Editor (like MATRIX EDITN in HP-42S). ¹⁶ See Section 2 of the OM.
M.EDIT		Submenu for matrix editing, called by M.EDI and M.EDIN. See p. 123.

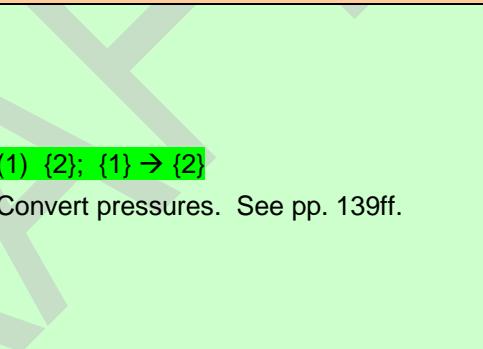
¹⁶ EDIT and EDITN disable ASL. In the HP-42S, both don't actually disable ASL; they preserve the stack lift state – you can observe this if you do ENTER vs. a stack-lift-enabling operation (e.g. x₂y) just before invoking them. This behavior is not really useful – it is dropped here since HP-42S compatibility is not required.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
M.GET	 	(0) {1, 2} → {8, 9} Gets a sub-matrix with IP(y) rows and IP(x) columns out of the indexed matrix into X (like GETM in HP-42S). Cf. M.PUT.
M.GOTO		(0) Asks for target row and column and moves to this matrix element. with M.EDIT displayed
M.GROW		(0) Allows the indexed matrix to grow automatically (see J+ above and Section 2 of the OM; see also GROW in the HP-42S Owner's Manual, p. 213.). Cf. M.WRAP.
M.INSR		(0) Inserts a new row of elements containing 0., left of the current cursor position in the matrix. with M.EDIT displayed
M.LU	 	(1) WP 34S: Takes a <i>descriptor</i> of a square matrix in X . Transforms (<i>X</i>) into its LU decomposition in-situ. The value in X is replaced by a <i>descriptor</i> that defines the pivots that were required to calculate the decomposition. The most significant digit is the pivot for the first diagonal entry, the next most significant for the second and so forth. xxx
M.NEW	NEW	(2) {1, 2} → {8} Creates a new matrix (like NEW in HP-42S). Its number of rows shall be supplied in Y and its number of columns in X . M.NEW returns a matrix <i>x</i> with all its elements set to zero.
M.PUT	 	(0) {8, 9} Puts the matrix <i>x</i> as is into the indexed matrix (like PUTM in HP-42S). Cf. M.GET.
M.R>R	 	(0) {8, 9} Swaps row <i>x</i> and row <i>y</i> of the indexed matrix (like R>>R in HP-42S).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
M.SIMQ		_submenu of MATX, called by SIM_EQ.
M.SQR?	g TEST ▲ f M.SQR?	(0) Returns true if x is a square matrix.
M.WRAP	f WRAP with M.EDIT displayed	(0) Controls the index pointers (see Section 2 of the OM). Cf. M.GROW.
m:	f U m:	submenu. See p. 125.
m→au	f U x: m→au	
m→chī	... ▼ g m→chī	
m→cūn	... ▼ g m→cūn	
m→fēn	... ▼ g m→fēn	
m→fm.	... ▲ m → fathom	
m→ft.	... f m→ft.	
m→ft _{us}	... ▲ m → survey foot _{us}	
m→in.	... g m→in.	(1) {2}; {1} → {2}
m→lī	... ▼ m→lī	Convert distances or heights. See pp. 139ff.
m→l.y.	... m→l.y.	
m→mi.	... f m→mi.	
m→nmi.	... f m→nmi.	
m→pc	... m→pc	
m→pt.	... g m → point	
m→yd.	... g m→yd.	
m→yīn	... ▼ m→yīn	
m→zhàn	... ▼ m → zhàng	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
NAND	f BITS f NAND	(2) Works in analogy to AND. See p. 18.
NaN?	g TEST ▲ f NaN?	(0) Returns true if x is Not a Number.
NBin _p	g PROB	(1) {2}; {1} → {2}
NBin _▲	g NBin: NBin _p	Negative binomial distribution with the total number of failures f in X , the probability of a success p_0 in I , and the number of draws n in J . See pp. 222ff for more information.
NBin _▲	etc.	
NBin ⁻¹		
NBin:	g PROB g NBin:	Submenu. See p. 124.
NEIGHB	g INFO f NEIGHB	(2) {1} Returns ... <ul style="list-style-type: none">• $x + 1$ for $x < y$;• x for $x = y$;• $x - 1$ for $x > y$.
		(2) {2} Returns the nearest machine-representable number to x in the direction towards y in the mode set. For <ul style="list-style-type: none">• ... $x < y$, it is the machine successor of x ;• ... $x = y$, it is y ;• ... $x > y$, it is the machine predecessor of x. NEIGHB may be useful investigating numeric stability (see NEIGHBOR in the HP-71 Math Pac).
NEXTP	g X.FN g NEXTP	(1) {1, 10}; {2} → {1} Returns the next prime number greater than $ P(x) $. See also PRIME? on p. 59.
nmi.→m	f U→ x: f nmi.→m	(1) {2}; {1} → {2}
Nm→lbft	f U→ ▽ Nm → lbft-ft	Convert distances and torques. See pp. 139ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
NOP	g P.FN P.FN2 f NOP	'Empty' step (for historical reasons only).
NOR	f BITS f NOR	(2) Works in analogy to AND. See p. 18.
Norml_p	g PROB Norml: Norml_p etc.	(1) {2}; {1} → {2}
Norml_μ		Normal distribution with an arbitrary mean μ given in I and standard deviation σ in J . See Section 2 of the OM for an application example and pp. 222ff for more.
Norml_△		
Norml⁻¹		Norml ⁻¹ returns x for a given probability p in X , with μ in I and σ in J .
Norml:	g PROB Norml:	_submenu. See p. 124.
NOT	f BITS NOT	<p>(1) {10}</p> <p>Inverts x bit-wise as on <i>HP-16C</i>.</p> <p>(1) {1, 2} → {1}</p> <p>Returns 1 for $x = 0$, and 0 for $x \neq 0$.</p>
nΣ	g Σ n	<p>(-1) {} → {1}</p> <p>Recalls the number of accumulated data points.</p>
N→lbf	f U→ F&p: N→lbf	(1) {2}; {1} → {2} Converts forces. See pp. 139ff.
ODD?	g TEST f ODD?	(0) Checks if x is integer (see INT?) and odd.
OFF	g OFF	(0) Turns your WP 43S off. In PEM, inserts a step to turn it off under program control.
OR	f BITS OR	(2) Works in analogy to AND, see p. 18.
OrthoF	f STAT g OrthoF	(0) Selects the linear orthogonal fit model. Relevant for CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
ORTHOG	g X.FN Orthog	submenu. See p. 126.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
oz→kg	f [U→] m: oz→kg	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.
ParabF	f [STAT] ▼ f ParabF	(0) Selects the parabolic fit model. Relevant for COV, CORR, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
PARTS	f [PARTS]	Menu. See p. 123.
PAUSE	g [P.FN] PAUSE n	(0) Within a routine running, refreshes the display and pauses program execution for n ticks (s. TICKS), with $0 \leq n \leq 99$. The pause will terminate early when you press a key.
Pa→atm	f [U→] F&p: ▼ Pa→atm	
Pa→bar	... Pa→bar	
Pa→iHg	... f Pa → in.Hg	
Pa→mmH	... f Pa → mmHg	
Pa→psi	... Pa→psi	
Pa→tor	... f Pa → torr	
pc→m	f [U→] x: pc→m	Converts distances. See pp. 139ff.
PERM	g [PROB] P_{yx}	(2) {1} Returns the number of possible <u>arrangements</u> (a.k.a. <i>permutations</i>) of x <i>items</i> taken out of a set of y <i>items</i> . No <i>item</i> occurs more than once in an arrangement, and <u>different orders</u> of the same x <i>items</i> are <u>counted</u> separately. Cf. COMB.
		(2) {2, 3} See pp. 220f for the formula.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PGMINT	f ADV f PGMINT <i>labl</i>	Specifies the address of the expression to be integrated or solved, respectively.
PGMSLV	f ADV f PGMSLV <i>labl</i>	See Section 4 of the OM.
PIXEL	g P.FN P.FN2 g PIXEL	(0) Turns on a single pixel (dot) on the screen. The location of the pixel is given by the numbers in the X - and Y -registers. See AGRAPH on p. 17 for more.
PLOT	f STAT g PLOT	(0) Plots the <i>n</i> data points given by the <i>nx2</i> matrix <i>x</i> . See p. B-19 for more.
P_n	g X.FN Orthog P_n	(1) {2}; {1} → {2} <i>Legendre polynomials.</i> See pp. 237f for more.
POINT	g P.FN P.FN2 g POINT	(1) {1, 2} Turns on a square point (3x3 px ■) on the screen. The position of its center is given by the integer parts of the numbers in X and Y . See AGRAPH on p. 17 for more.
Poiss _p	g PROB g Poiss: Poiss _p	(1) {2}; {1} → {2}
Poiss _λ	etc.	Poisson distribution with the number of successes <i>g</i> in X and the Poisson parameter λ in I . See pp. 222ff for details.
Poiss _λ		Poiss ⁻¹ returns the maximum number of successes <i>m</i> for a given probability <i>p</i> in X and λ in I .
Poiss:	g PROB g Poiss:	_submenu_. See p. 124.
PopLR	g P.FN g PopLR	(0) Pops the local <i>registers</i> allocated to the <i>current routine</i> (see the OM, Section 3) <u>without returning to the calling routine</u> . See LOCR and RTN.
PowerF	f STAT ▼ PowerF	(0) Selects the power curve fit model. Relevant for CORR, COV, L.R., s_{xy} , \hat{x} , and \hat{y} (see pp. 222ff for more).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PRCL	g P.FN f PRCL	(0) Copies the <i>current program</i> (from <i>FM</i> or <i>RAM</i>) and appends it to <i>RAM</i> , where it can then be edited (see the <i>OM</i>). PRCL allows for duplicating programs in <i>RAM</i> . Will only work with enough space at destination. Recall a library routine from <i>FM</i> , edit it, and PSTO – this way you can modify this part of the <i>FM</i> library (see PSTO).
PRIME?	g TEST f PRIME?	(0) {1, 2, 10} Checks if $ P(x) $ is a prime. Returns true for prime and false for composite. For $x > 3.3 \times 10^{24}$, true means 'probably prime' (see p. 170 for more).
PRINT	g PRINT	Menus. See p. 124.
PROB	g PROB	
PROG		Submenu of global labels defined when calling XEQ etc. See Section 3 of the OM.
PROGS	f CAT. PROGS	Submenu of global labels defined at execution time. See pp. 115f.
pr→dB	f U→ ▲ power ratio → dB	(1) {2}; {1} → {2}
psi→Pa	f U→ F&p: psi→Pa	Convert ratios or pressures. See pp. 139ff.
PSTO	g P.FN f PSTO	(0) Copies the <i>current program</i> (see the <i>OM</i>) from <i>RAM</i> and appends it to the <i>FM</i> library. Cf. PRCL. This program must include at least one LBL statement with a global label (preferably at its beginning). If a program with the same label already exists in the library it will be deleted first. Global labels may be browsed in CATALOG PROGS and called by XEQ .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
pt. \rightarrow m		(1) {2}; {1} \rightarrow {2} Converts print heights. See pp. 139ff.
PUTK		(0) Assumes a key code in the address specified. Stops program execution, takes said code and puts it in the keyboard buffer resulting in immediate execution of the corresponding call. R/S is required to resume program execution then. May help in user interaction with routines (see the OM, Section 3).
P.FN		Menu. See p. 124.
P.FN2		_submenu_. See p. 124.
P:		_submenu_. See p. 137.
qt. \rightarrow m ³		(1) {2}; {1} \rightarrow {2} Converts volumes. See pp. 139ff.
RAD		(0) Sets the ADM to radians.
RAD \rightarrow		(1) {1, 2, 4} \rightarrow {4} Converts angles as described on p. 147.
RAM		_submenu_ of global labels defined at execution time. See pp. 115f.
RANGE		(0) {1, 2} Limits the range of displayable <i>real numbers</i> to $\pm 10^{\pm n}$ with $n = \text{IP}(x)$, $99 \leq n \leq 6145$. For greater input, n will be set to 6145 (startup default); for less it will be set to 99. RANGE allows for saving screen space for reasonable data.
RANGE?	 	(-1) {} \rightarrow {1} Returns the current range setting.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RANI#	 	(-1) {} → {1} Returns an integer random number n with $\text{IP}[\min(x, y)] \leq n \leq \text{IP}[\max(x, y)]$. After executing RANI#, the stack looks like $[n, x, y, \dots]$, so you can drop or roll down n and call RANI# again to get another random integer with the same parameters. Use RANI# e.g. for throwing dices.
RAN#	 	(-1) {} → {2} Returns a random number between 0 and 1 like RAN does in HP-42S. See also SEED.
RBR		Calls the register browser – see the OM, Section 5. You may call RBR also in PEM but it is not programmable. Within RBR, <i>real</i> and <i>complex numbers</i> are generally displayed in their format chosen; since it uses the small font all the way, more digits can be shown here in ALL 0 than in a standard numeric row. For <i>reals</i> and ALL 0, RBR display will turn to SCI or ENG at 33 digits in worst case. Extended display precision may be observed for <i>complex numbers</i> as well.
RCL		(-1) Recalls the content of a register or variable and pushes it on the stack.
RCLCFG		(0) Recalls a calculator configuration stored by STO CFG (see the OM, Sections 2 and 6). Cf. also RESET on p. 63.
RCLEL	   with M.EDIT displayed	(-1) {} → {2, 3} Recalls a copy of the current element a_{ij} of the indexed matrix. Cf. STO EL.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RCLIJ	 	(-2) {} → {1} Recalls the current values of the matrix index pointers into X (= column number) and Y (= row number). If both pointers equal zero, then there is no indexed matrix. Cf. STOIJ.
RCLS	Stack r	Recalls 4 or 8 values from a set of registers starting at address r, and pushes them on the stack. This is the converse command of STOS.
RCL+		(1) Recalls a content of a register or variable, executes the operation specified, and puts the result on the stack like a monadic function. ¹⁷
RCL-		
RCL×		
RCL/		RCL-12 replaces x with x - r12.
RCL↑		(1) {1, 2, 4, 5, 6, 10}
		Replaces x with the maximum of r and x. ¹⁷
RCL↓		(1) {1, 2, 4, 5, 6, 10}
		Replaces x with the minimum of r and x. ¹⁷
RDP	 	(1) {2, 3, 4, 5, 8*, 9*} Rounds x to n decimal places ($0 \leq n \leq 99$, think of FIX format), taking the RM setting into account. See RM and compare RSD.
Re	 	(1) {2, 3} → {2}; {9} → {8} Returns the real part of x. Cf. IM.
REALS		Submenu of real variables defined at execution time. See pp. 115f.

¹⁷ Only legal operations according to the DT matrices in Section 2 of the OM will work. See also the examples given there.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
REAL?		(0) Checks if x is a <i>real</i> number or matrix.
RECV		(0) Prepares your WP 43S for receiving data via serial I/O. See SEND and Section 3 in the OM for more.
REGIST		Submenu of <i>named registers</i> defined. See pp. 115f.
RESET	 	Executes CLALL and resets your WP 43S to <i>startup default configuration</i> , i.e. 2COMPL, ALL 0, DEG, DENMAX 0, DSTACK 4, GAP 3, J/G 1752.0914, LinF, LocR 0, RM 0, RSD 34, TDISP 0, WSIZE 64, and Y.MD. In this course, RESET also sets RANGE to 6145, resets the random number generator, sets the <i>flags</i> ASLIFT, DECIM., DENANY, MULTx, PROPFR, TDM24, and clears all other <i>system flags</i> (see pp. 103ff). See said commands and <i>system flags</i> for more.
RE→CX	 (works in <i>run mode</i> only)	(2) {2} → {3} Composes a <i>complex number</i> out of two <i>reals</i> or integers x and y , setting C and taking either... <ul style="list-style-type: none">• (for L) the <i>real</i> part from Y and <i>imaginary</i> part from X, or• (for ⊖) <i>magnitude</i> from Y and <i>phase</i> from X.
	 	(2) {8} → {9} Works in analogy for two <i>real</i> matrices x and y .
Re↔Im		(1) {3, 9*} Swaps <i>real</i> and <i>imaginary</i> part of <i>complex objects</i> .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RJ	 RJ	<p>(10)</p> <p>Right justifies a bit pattern within its word size, in analogy to LJ (see there). The stack will lift, placing the right-justified word in Y and the count of bit-shifts necessary to justify the word in X.</p> <p>Example: 10 1100₂ RJ results in y = 1011₂ and x = 2.</p>
RL	 RL <i>n</i>	<p>(1) {10} </p> <p>Works like <i>n</i> consecutive RLs on HP-16C, similar to RLn there ($0 \leq n \leq 63$). RL 0 executes as NOP, but loads L. See the OM, Sect. 2, for more.</p>
RLC	 RLC <i>n</i>	<p>(1) {10} </p> <p>Works like <i>n</i> consecutive RLCs on HP-16C, similar to RLCn there ($0 \leq n \leq 64$). RLC 0 acts as NOP, but loads L. See the OM, Sect. 2, for more.</p>
RM	 RM <i>n</i>	<p>(0) Sets floating point rounding mode. This rounding mode is used only for RSD or when converting from the extended precision internal format (typically 39 digits) to packed reals. It will <u>not</u> alter the display nor change the behavior of ROUND. The following 7 modes are supported:</p> <ul style="list-style-type: none"> 0: round half even: $\frac{1}{2}\uparrow$ 0.5 rounds to next even number (default, used in science). 1: round half up: $\frac{1}{2}\uparrow$ 0.5 rounds up ('businessman's rounding' ¹⁸). 2: round half down: $\frac{1}{2}\downarrow$ 0.5 rounds down. 3: round up: $\leftarrow\uparrow\rightleftharpoons$ rounds away from 0. 4: round down: $\rightarrow\downarrow\leftarrow\rightleftharpoons$ rounds towards 0.

¹⁸ Translator's notes for French and German readers: Cela correspond à l'arrondi commercial. / Das entspricht kaufmännischer Rundung.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		5: ceiling: $\lceil x \rceil$ rounds towards $+\infty$. 6: floor: $\lfloor x \rfloor$ rounds towards $-\infty$. The abbreviations printed on grey background are used in STATUS for indicating the respective rounding modes (see Section 5 of the OM).
RMD	 	(2) {1, 2, 10} Returns the remainder of a division. Equals RMD on HP-16C but works for <i>reals</i> as well. See the OM, Section 2, for examples. Cf. MOD.
RM?		(-1) {} → {1} Returns the floating point rounding mode set. See RM for more.
RNORM		(1) {8, 9} Calculates the row norm of the matrix x , i.e. the maximum value (over all rows) of the sums of the absolute values of all elements in a row (like RNRM on HP-42S). For a vector, the row norm is the largest absolute value of any of its elements.
RootF		(0) Selects the root fit model; relevant for CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y} . See pp. 231ff for more.
ROUND		(1) {2, 3, 4, 5, 6, 8*, 9*} Rounds x using the current display format like RND on HP-42S.
ROUNDI		(1) {8*}; {2} → {1} Rounds x to next integer. $\frac{1}{2}$ rounds to 1. Cf. IP.
RR		(1) {10}  Works like n consecutive RRs on HP-16C, similar to RRn there ($0 \leq n \leq 63$). RR 0 executes as NOP, but loads L. See the OM, Sect. 2, for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RRC	 	(1) {10}  Works like n consecutive RRCs on HP-16C, similar to RRCn there ($0 \leq n \leq 64$). RRC 0 executes as NOP, but loads L. See the OM, Sect. 2, for more.
RSD	 	(1) {2, 3, 4, 5, 8*, 9*} Rounds x to n significant digits ($1 \leq n \leq 34$), taking the RM setting into account. Think of SCI. Cf. RM and RDP.
RSUM	 	(1) {8, 9} Calculates the row sum of the matrix x , returning an $m \times 1$ matrix filled with the row sums of the $m \times n$ input matrix.
RTN		(0) In PEM, RTN is the logically last command in a routine (see the OM, Section 3). In a routine executing, RTN pops local data (cf. POPLR) and returns to the caller, i.e. moves the program pointer one step behind the XEQ instruction that called said routine. If there is none (i.e. the routine is top level), program execution halts, the program pointer is set to step 0000, and \overline{t} is lit. If pressed in run mode with no routine executing, RTN resets the program pointer to the start of current program (see the OM, Section 3). If this program is in FM, the pointer is set to step 0000 in RAM, and \overline{t} is lit.
RTN+1	 	(0) Works like RTN but: in a routine executing, RTN+1 moves the program pointer two steps behind the XEQ instruction that called said routine.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R-CLR		<p>(0) {2}</p> <p>Interprets x in the form $sss.nn$. Clears nn registers starting with address sss.</p> <p>Example: For $x = 34.567$, R-CLR will clear R34 through R89.</p> <p>ATTENTION: For $nn = 0$, clearing will cover the maximum available:</p> <ul style="list-style-type: none"> • For $sss \in [0; 99]$, it will stop at R99. • For $sss \in [100; 111]$, it will stop at K. • For $sss \geq 112$, it will stop at the highest currently allocated local register.
R-COPY		<p>(0) {2}</p> <p>Interprets x in the form $sss.nnddd$. Takes nn registers starting with address sss and copies their contents to ddd etc.</p> <p>Example: For $x = 7.0304567$, r07, r08, r09 will be copied into R45, R46, R47, respectively.</p> <p>For $x < 0$, R-COPY will take nn registers from FM instead, starting with register number sss. Destination will be in RAM always.</p> <p>ATTENTION: For $nn = 0$, copying will cover the maximum available as explained with R-CLR. Then x must be negative.</p>
R-SORT		<p>(0) {2}</p> <p>Interprets x in the form $sss.nn$. Sorts the contents of nn registers starting with address sss.</p> <p>Example: Assume $x = 49.036\ 9$, $r49 = 1.2$, $r50 = -3.4$, and $r51 = 0$; then R-SORT will return $r49 = -3.4$, $r50 = 0$, and $r51 = 1.2$.</p> <p>ATTENTION: For $nn = 0$, sorting will cover the maximum available as explained with R-CLR.</p>

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R-SWAP	 	(0) {2} Works like R-COPY but swaps the contents of source and destination registers.
R→D	 	(1) {1, 2, 4} → {4} Converts angles as described on p. 147.
R↑		Rotates the stack contents one level up or down, respectively. See Section 1 of the OM for details.
R↓		
S	 	(-2) {} → {2} Takes the statistical sums accumulated, calculates the sample standard deviations s_y and s_x and pushes them on the stack. See Sect. 2 of the OM for the output format and pp. 231ff for the formula.
SAVE		(0) Saves user program space, registers and system state to a file in the FAT system, and returns Saved. Recall your backup using the different flavors of LOAD.
SB	 	(1) (10) Sets the specified bit in x .
SCI	 	(0) Sets scientific display format (see Section 2 of the OM).
scw→kg	 	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.
SDIGS?	 	(-1) {} → {1} Returns the number of significant digits set by SETSIG. Also returned by STATUS.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SDL	g P.FN2 f SDL <i>n</i> etc. or g DISP f SDL <i>n</i> etc.	(1) {1, 2} Shifts digits left (right) by <i>n</i> decimal positions, equivalent to multiplying (dividing) <i>x</i> by 10^n . SDR for long integers ends with zero. Compare SL and SR for binary integers.
SEED	g PROB ▲ SEED	(0) {2} Stores a seed for random number generation. For $x \leq 0$, the seed is taken from the real-time clock.
SEND	f I/O f SEND	(0) Sends all RAM data to the device connected via serial I/O. See RECV and the OM, Section 3 for more.
SETCHN	g DISP ▲ CHINA	(0) Sets regional format preferences. ¹⁹
SETDAT	g CLK ▲ SETDAT	(0) Sets the date for the real-time clock. ²⁰
SETEUR	g DISP ▲ EUROPE	(0) Set regional format preferences. ¹⁹
SETIND	g DISP ▲ INDIA	
SETJPN	g DISP ▲ JAPAN	
SETSIG	g MODE f SETSIG	(0) {1} Sets the number of significant digits (1 ... 34) for rounding after each operation. SETSIG 0 sets maximum precision.
SETTIM	g CLK ▲ SETTIM	(0) Sets the time for the real-time clock. ²⁰
SETUK	g DISP ▲ UK etc.	(0) Set regional format preferences. ¹⁹
SETUSA		

¹⁹ See Section 2 of the OM about localization of numeric output.

²⁰ Works on the calculator only – the simulator takes this information from the PC clock.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SF	g [FLAGS] SF <i>n</i>	(0) Sets the <i>flag</i> specified.
	g [MODE] SF <i>n</i>	
SHOW	f SHOW	(0) {1, 2, 3, 4, 7} Shows all digits or characters stored in X in top numeric row until next keystroke, using small font. For a <i>complex number</i> , either part of it will take one display row if one row cannot take both parts. For a <i>real</i> , <i>time</i> , or <i>date</i> up to 34, 35, or <i>xxx</i> digits will be shown. For a <i>long integer</i> , up to 296 digits can be shown using up to 7 display rows; for any greater integer, just its most significant 294 digits can be shown, trailed by ellipses. The simulator allows for extracting even longer integers (see App. E).
SIGN	f PARTS sign	(1) {8} {1, 2, 10} → {1} Returns 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to the mathematical function $\text{signum}(x)$.
		(1) {3} Returns the unit vector of the <i>complex number</i> x (cf. UNITV). Maintained for backward compatibility only.
SIGNMT	f BITS ▼ SIGNMT g INTS ▲ SIGNMT	(0) Sets sign-and-mantissa mode for operations on <i>short integers</i> . See the OM, Sect. 2.
SIM_EQ	f MATX SIM EQ <i>n</i>	(0) Solves a system of <i>n</i> linear equations $(MATA) \cdot \overrightarrow{MATX} = \overrightarrow{MATB}$. If these matrices are not defined before, they will be created automatically at execution time. See Section 2 of the OM for more.
sin	[TRI] sin	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the sine of the angle in X.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
sinc	[TRI] f sinc	(1) {2, 3, 8*, 9*} ... for $x \neq 0$ and 1 for $x = 0$. Tagged input of data type 4 will be converted in radians before computing. Untagged input is taken as radians. Returns $\sin(x)/x$...
sincπ	[TRI] f sincπ	(1) {2, 3, 8*, 9*} Returns $\sin(\pi x)/\pi x$...
sinh	[TRI] g sinh g EXP g sinh	(1) {2, 3, 8*, 9*} Returns the hyperbolic sine of x .
SKIP	g P.FN P.FN2 g SKIP n	(0) Skips n program steps forwards ($0 \leq n \leq 255$). So e.g. SKIP 2 skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. ATTENTION: If you edit a section of your routine crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to manually maintain all those statements individually .
SL	f BITS ▲ SL n	(1) {10} C ← [] ← 0 Works like n (≤ 63) consecutive SLs on HP-16C. SL 0 executes as NOP, but loads L. See Section 2 of the OM for more.
SLVQ	f ADV SLVQ	{1, 2, 3} → {1 or 2 or 3} Solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input stack [c, b, a, \dots], and tests the result. The following holds for <i>real</i> parameters: <ul style="list-style-type: none">If $r := b^2 - 4ac \geq 0$, SLVQ returns $-\frac{b \pm \sqrt{r}}{2a}$ in Y and X. In a routine, the step after SLVQ will be executed.Else, SLVQ returns the first <i>complex</i> root in X and the second in Y (the <i>complex conjugate</i> of the first). In a routine, the step after SLVQ will be skipped.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		In either case and also for <i>complex</i> parameters, SLVQ returns r in Z . Higher <i>stack registers</i> are kept unchanged. L will contain equation parameter c .
s_m	f [STAT] s_m	(-2) {} → {2} Takes the statistical data accumulated and pushes the <i>standard errors</i> (i.e. <i>std. deviations</i> of the means \bar{y} and \bar{x}) on the stack. Output format will be like the one of s (see the OM, Section 2).
s_{mw}	f [STAT] f s_{mw}	(-1) {} → {2} Returns the <i>standard error</i> for weighted data, i.e. the <i>standard deviation</i> of the mean \bar{x}_w .
SNAP	g [SNAP]	(0) Stores a snapshot of the screen (a.k.a. screenshot) in a BMP file on the calculator's <i>USB</i> flash disk in the /SCREENS directory. The file name comprises date and time of storage. The file can be dealt with on your computer.
SOLVE	f [ADV] SOLVE var	{2, 3} Solves the equation $f(var) = 0$, with f calculated by the equation specified (in PEM by PGMSLV). Two initial estimates of the root must be supplied in X and Y when calling SOLVE. It returns var_{root} in X , the second last var -value tested in Y , then $f(var_{root})$ in Z , and 0 in T . Additionally, SOLVE acts as binary test in programs, so the next program step will be skipped if SOLVE fails to find a root. See Section 4 of the OM for more. ATTENTION: SOLVE fills all <i>stack registers</i> with x before calling the routine specified.
Solver	g [EQN] Solver	Submenu for solving a given equation. See the OM, Section 4, for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SPEC?	g TEST ▲ f SPEC?	(0) True if x is ‘special’ (i.e. $\pm\infty$ or NaN).
SR	f BITS ▲ SR n	(1) {10}  Works like n (≤ 63) consecutive SRs on HP-16C. SR 0 executes as NOP, but loads L. Cf. ASR. See Section 2 of the OM for more.
SSIZE?	g INFO SSIZE?	(-1) {} → {1} Returns the number of <i>stack registers</i> currently allocated, 4 or 8.
STAT	f STAT	Menu. See p. 125.
STATUS	f STATUS g FLAGS STATUS	Displays number of free bytes in RAM, flags set, local registers allocated, and system variables. See the OM, Sect. 5 for more.
STK	g STK	Menu. See p. 125.
STO	STO r	(0) Stores x into destination. ²¹
STOCFG	STO f Config r	(0) Stores the current calculator <i>configuration</i> in a variable or <i>register</i> for later use as described in Sections 2 and 6 of the OM. RCLCFG recalls such data. Cf. also RESET on p. 63. ²¹
STOEL	f MATX g STOEL STO g ...EL	(1) {1, 2, 3} Stores a copy of x into the indexed matrix at the current element, a_{ij} . Cf. RCLEL. ²¹
STOIJ	f MATX ▲ STOIJ STO g ...IJ	(1) {1} Sets the index pointers to IP(x) (column number) and IP(y) (row number). Cf. RCLIJ. ²¹
STOP	R/S	(0) Stops program execution. May be inserted in programs to wait for input, for example.

²¹ L will not be loaded.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
STOS	STO f Stack r	(0) Stores the entire stack in a set of 4 or 8 registers, starting at the destination address specified. Cf. RCLS. ²¹
STO+	STO + r	
STO-	STO - r	(0) Executes the specified operation on r and stores the result (e.g. r - x) at the address specified. ²²
STOx	STO x r	
STO/	STO / r	
sto→kg	f U→ m: f stone → kg	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.
STO↑	STO f Max r	
	STO ▲ r	(0) {1, 2, 4, 5, 6, 10}
STO↓	STO f Min r	Stores the maximum (or minimum) of r and x in the address specified. ²²
	STO ▼ r	
STRI?	g TEST g STRI?	(0) True if x is a <i>text string</i> (cf. STR? in HP-42S).
STRING	f CATALOG VARS STRING	Submenu of <i>text string</i> variables defined at execution time. See pp. 115f.
SUM	f STAT SUM	(-2) {} → {2} Recalls the linear sums Σy and Σx . Useful in basic 2D vector algebra. Output is labeled in analogy to s.
s_w	f STAT f s_w	(-1) {} → {2} Calculates the <i>standard deviation</i> for weighted data (where the weight y of each data point x was entered via [Σ+]). See pp. 222ff for the formula.

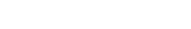
²² Only legal operations according to the *DT* matrices in Section 2 of the OM will work. See also the examples given there. **L** will not be loaded.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
s_{xy}	f [STAT] ▲ s_{xy}	(-1) { } → {2} Calculates the <i>sample covariance</i> for the two data sets {x} and {y} entered via [Σ+] , depending on the curve fit model selected. See pp. 231ff for the formula and COV for the <i>population covariance</i> .
SYSTEM	g [MODE] g SYSTEM	(0) Returns to <i>DMCP</i> . Works on the calculator only (not on the simulator). See App F.
SYS.FL	f [CATALOG] SYS.FL	Submenu of system flags. See p. 115.
s(a)	f [STAT] ▲ f s(a)	(-2) { } → {2} Pushes the standard errors s(a₁) (in Y), and s(a₀) (in X) for the parameters of the line fitted through the data points accumulated in the statistical summation <i>registers</i> on the stack. Works for LINF only. See pp. 237f for more.
S.INTS	f [CAT.] VARS S.INTS	Submenu of short integer variables defined at execution time. See pp. 115f.
s.t→kg	f [U→] m: ▲ short ton → t	(1) {2}; (1) → {2}
s→year	f [U→] f s→year	Convert masses and times. See pp. 139ff.
tan	TRI tan	(1) {2, 3, 8*, 9}; {1, 4} → {2} Returns the tangent of the angle in X. Returns "Not a Number" for $x = \pm 90^\circ$ or equivalents if SPCRES is set.
tanh	g EXP g tanh	(1) {2, 3, 8*, 9}
	TRI g tanh	Returns the hyperbolic tangent of x.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
TDISP	g CLK ▲ TDISP <i>n</i>	(0) Sets time display format. TDISP 0 allows for the full time string (pure trailing decimal zeros will not be displayed), TDISP 1 or 2 truncates it to <i>hours : minutes</i> only, TDISP 3 to <i>hours : minutes : seconds</i> , and <i>n</i> ≥ 4 displays also <i>n</i> – 3 digits for decimal fractions of <i>seconds</i> . Example: With TDISP 6, a time string might look like 157:26:38.409
TEST	g TEST	Menu. See p. 125.
TICKS	g P.FN TICKS	(–1) { } → {1} Returns the number of ticks from the real-time clock at execution time. 1 tick = 0.1 s. Counting starts when the calculator is turned on.
TIME	g CLK TIME	(–1) { } → {5} Recalls the time from the real-time clock (or the PC clock for the simulator) at execution. See Sect. 2 of the OM for the output format.
TIMER	f TIMER	Starts the timer application based on the real-time clock and following the timer of HP-55. See the OM, Section 5 for a detailed description.
TIMES	f CAT. VARS f TIMES	Submenu of time variables defined at execution time. See pp. 115f.
T_n	g X.FN Orthog T_n	(2) {2}; {1} → {2} <i>Chebyshev polynomials of first kind.</i> See pp. 237f for details.
TONE	f I/O f TONE <i>n</i>	(0) Sounds a tone according to <i>n</i> (= 1 ... 9).
ton→kg	f U→ m: ▲ ton→kg	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
TOP?	g TEST g TOP?	(0) Returns ... • false if called with the program pointer being in a subroutine; • true if called in the top routine (i.e. if the program-running flag is set and the SRS pointer is clear).
tor→Pa	f [U→] F&p: f torr → Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 139ff.
t _p (x)	g PROB	(1) {2}; {1} → {2}
t _Δ (x)	t: t_p(x)	Student's <i>t</i> distribution. The degrees of freedom are stored in I . $t_{\Delta}(x)$ equals $Q(t)$ on HP-21S. See Section 2 of the OM for an application example and pp. 222ff for more mathematical details.
t _Δ (x)	etc.	
t ⁻¹ (p)		
TRANS	f CAT FCNS TRANS	Works like $[M]^T$ on p. 93. Maintained for backward compatibility only.
TRI	TRI	Menu. See p. 125.
trz→kg	f [U→] m: f tr.oz → kg	(1) {2}; {1} → {2} Converts masses. See pp. 139ff.
TVM	g FIN TVM	Submenu for the application <i>Time Value of Money</i> . See Section 5 of the OM.
t:	g PROB t:	Submenu. See p. 124.
t \leftrightarrow	g STK t\leftrightarrow r	Swaps <i>t</i> and <i>r</i> , in analogy to <i>x\leftrightarrow</i>
ULP?	g INFO f ULP?	(1) {1, 2} Returns 1 times the smallest power of ten which can be added to <i>x</i> or subtracted from <i>x</i> to actually change the (internal) value of <i>x</i> in your WP 43S in the mode set. Thus, 1 is returned for integers. Indicated in STATUS.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
U_n	g X.FN Orthog U_n	(2) {2}; {1} → {2} <i>Chebyshev polynomials of second kind.</i> See pp. 237f for details.
UNDO	f ⌂	Recalls the stack, the summation registers, and the system flags as they were before the last operation executed.
UNITY	f MATX f UNITY	(1) {8, 9} Returns the unit vector for the matrix <i>x</i> (like UVEC in HP-42S). Each element of the matrix is adjusted so its overall Euclidean norm becomes 1 (see ENORM); for a vector, its <i>magnitude</i> will become 1.
	g CPX UNITY	(1) / (3) Returns a <i>complex number</i> with <i>magnitude</i> $ r = 1$ in direction of <i>x</i> .
UNSIGN	f BITS ▼ UNSIGN	(0) Sets unsigned mode for mode for operations on <i>short integers</i> . Cf. UNSGN on HP-16C. See Section 2 of the OM.
	g INTS ▲ UNSIGN	
U→	f U→	Menu. See p. 125.
VAR		Submenu of variables defined at execution time when calling STO, RCL, etc.
VARMNU	g P.FN f VARMNU <i>label</i>	Creates a variable <i>menu</i> using the MVAR instructions following the global label specified. See the OM, Section 3.
VARS	f CAT. VARS	Submenu of variables defined at execution time. See pp. 115f.
VERS?	g INFO g VERS?	(0) Shows firmware version and build number until next keystroke (see the OM, Section 2).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
VIEW		(0) Shows r until the next key is pressed. Example: If a variable called Test12 contains -123.45 , VIEW  Test12 ENTER↑ will display Test12 = -123.45
V:		Submenu. See p. 137.
V ₄		(2) {8} → {4} Returns the angle between two 2D or 3D vectors $\vartheta = \arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{ \vec{v}_1 \vec{v}_2 }\right)$
WDAY		(1) {2, 6} → {1} Assumes x containing a <i>date</i> in the format selected (or a <i>real number</i> in a corresponding format) and returns the name of the respective day and a corresponding integer (Monday = 1). ²³
Weibl _p	  etc.	(1) {2}; {1} → {2}
Weibl _I		Weibull distribution with its shape parameter b in I and its characteristic lifetime T in J . See pp. 222ff for details.
Weibl _A		
Weibl ⁻¹		Weibl ⁻¹ returns the survival time t_s for a given probability p in X , with b in I and T in J .
Weibl:	 	Submenu. See p. 124.
WHO?	 	(0) Displays credits to the brave men who made this project work (temporary message).
Wh→J		(1) {2}; {1} → {2} Converts energies. See pp. 139ff.

²³ Translator's note: These day numbers correspond to Chinese weekdays 1 to 6 directly. For Portuguese weekdays ('segunda-feira' etc.), add 1 to days 1 to 5.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
W_m	 etc.	(1) {2, 3}; {1} → {2}
W_p		
W^{-1}		
WSIZE		(0) Sets the word size (a.k.a. bit width) for DT 10. Works almost like on HP-16C, but with the parameter $1 \leq n \leq 64$ trailing the command instead of taken from X . WSIZE 0 sets the word size to maximum, i.e. 64 bits. WSIZE works as it does in WP 34S:
		Reducing word size truncates the short integer values in L and in the stack registers. All other memory content stays as is (see App. B on pp. 172f). Increasing the word size adds significant empty bits to short integer values in L and in the stack registers. All other memory content stays as is.
WSIZE?		(-1) {} → {1} Recalls the word size set.
$W \rightarrow hp_E$		(1) {2}; {1} → {2} Convert powers. See pp. 139ff.
$W \rightarrow hp_M$	etc.	
$W \rightarrow hp_{UK}$		
\hat{x}		(1) {2}; {1} → {2} Returns a forecast \hat{x} for a given y (in X) according to the curve fit model chosen. See L.R. for more.
\bar{x}		(-2) {} → {2} Calculates the arithmetic means of the y - and x -data accumulated and pushes them on the stack. See also s , s_m , and σ .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x^2		(1) {1, 2, 3, 8*, 9*, 10}
x^3		Return the square or cube of x , respectively.
XEQ		(0) Executes the function or routine with the label specified. – In PEM, XEQ inserts a call to the subroutine with the label specified.
\bar{x}_G		(-2) {} → {2} Calculates the <i>geometric means</i> of the y - and x -data accumulated and pushes them on the stack. See pp. 231ff for the formula. Output format will be similar to the one of \bar{x} . See also ε , ε_m , and ε_p .
\bar{x}_H		(-2) {} → {2} Calculates the <i>harmonic means</i> of the y - and x -data accumulated and pushes them on the stack.
xIN		with type = NILADIC, MONADIC, DYADIC, TRIADIC, or ..._COMPLEX defines how many stack levels are used for parameter input to the function under consideration. Furthermore it does some initialization work (e.g. set SSIZE8). xIN is the recommended way to start an XROM routine. Thereafter, SSIZE8 is clear. Note xIN cannot nest and XROM routines using xIN cannot call user code.
x_{\max}		(-2) {} → {2} Returns the maximum (or minimum) of the y - and x -data accumulated and pushes them on the stack.
x_{\min}		ATTENTION: These extrema will not be updated after Σ- (see there).
XNOR		(2) Work in analogy to AND. See p. 18.
XOR		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
xOUT	[XEQ] way	Cleans and reverts the settings of xIN, taking care of a proper return including the correct setting of <i>I</i> and the stack. Typically, way = xOUT NORMAL . Generally, xOUT shall be the last command of an XROM routine.
\bar{x}_{RMS}	f [STAT] ▲ g \bar{x}_{RMS}	(-2) {} → {2} Calculates the <i>quadratic means</i> of the <i>y-</i> and <i>x-</i> data accumulated and pushes them on the stack. Quadratic means are often used in electrical engineering for alternating currents.
\bar{x}_w	f [STAT] f \bar{x}_w	(-1) {} → {2} Returns the <i>arithmetic mean</i> for weighted data (where the weight <i>y</i> of each data point <i>x</i> was entered via [Σ+]). See pp. 231ff for the formula. See also <i>s_w</i> and <i>s_{mw}</i> .
$\sqrt[x]{y}$	g EXP $\sqrt[x]{y}$	(2) Returns the <i>xth</i> root of <i>y</i> . For <i>y</i> < 0 and CPXRES set, it may return <i>complex numbers</i> . See the DT tables in Section 2 of the OM for more.
X.FN	g X.FN	Menu. See p. 126.
x!	f x!	(1) {1, 10} Returns the <i>factorial n!</i> . Note this is only defined for positive integers. 20! is the biggest factorial < 2^{64} . 450! is maximum allowed for <i>long integers</i> .
		(1) {2, 3} Returns $\Gamma(x + 1)$. 2 123.549 956 662 463 236 31 is the maximum <i>x</i> for <i>real numbers</i> .
x:	f U→ x:	Submenu. See p. 137.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$x \rightarrow DATE$	g CLK $x \rightarrow DATE$	(1) {2} → {6} Interprets the <i>real number</i> x as a date coded in the date format selected (Y.MD, D.MY, or M.DY) and converts it to a proper <i>date</i> .
$x \rightarrow \alpha$	g a.FN $x \rightarrow \alpha$	(1) {1, 2, 10} → {7} Interprets the integer part of x as a character code and converts it to the respective character α , similar to XTOA in the HP-42S.
$x \leftrightarrow$	g STK $x \leftrightarrow r$	Swaps x and r , in analogy to $x \leftarrow y$. Will be listed like $x \leftrightarrow J$, $x \leftrightarrow .12$, $x \leftrightarrow \rightarrow 12$, etc. in programs.
$x \leftrightarrow y$	$x \leftrightarrow y$	Swaps the contents of <i>stack registers</i> X and Y .
$x = ?$	g TEST $x = ? r$	(0) etc.
$x \neq ?$		Compare x with r . See $x < ?$ for more.
$x \approx ?$	g TEST Δ $x \approx ? r$	(0) {2, 3, 4, 5, 8, 9} Will be true if the <u>rounded</u> values of x and r are equal (see ROUND). See $x < ?$ for more.
$x < ?$	g TEST $x < ? r$	(0) {1, 2, 4, 5, 6, 7, 10} etc.
$x \leq ?$		Compare x with r . <i>Text strings</i> are compared according to their sorting order.
$x \geq ?$		
$x > ?$		
$x = +0?$	g TEST Δ $x = +0?$	(0) {2, 3, 10} etc.
$x = -0?$		These two tests are for comparing <i>short integers</i> in modes 1COMPL and SIGNMT, and for <i>real or complex numbers</i> if SPCRES is set. Then e.g. $0./(-7)$ will display -0 .

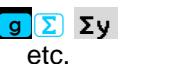
Item	Keystrokes	Remarks (see pp. 12ff for general information)
\hat{y}		(1) {2}; {1} → {2} Returns a forecast y (in X) for a given x according to the curve fit model chosen. See L.R. for more.
$yd.\rightarrow m$		(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
YEAR		(1) {2, 6} → {1} Assumes x containing a date in the format selected (or a <i>real number</i> in corresponding format) and extracts the year.
year→s		(1) {2}; {1} → {2}
$yin\rightarrow m$		Convert times and distances. See pp. 139ff.
y^x		(2) Raises y to the power of x . For $y < 0$ and CPXRES set, it may return <i>complex numbers</i> . See the DT tables in Sect. 2 of the OM for more.
Y.MD		(0) Sets the format yyyy-mm-dd for dates.
$y\leftrightarrow$		Swaps y and r , in analogy to $x\leftrightarrow$.
$zhàn\rightarrow m$		(1) {2}; {1} → {2} Converts distances. See pp. 139ff.
$z\leftrightarrow$		Swaps z and r , in analogy to $x\leftrightarrow$.
αINTL		Submenu. See pp. 115ff.
		Menu in AIM (see p. 127).
αLENG?		(-1) {} → {1}
		Returns the number of characters found in the <i>text string</i> r , similar to ALENG in HP-42S. ²⁴

Item	Keystrokes	Remarks (see pp. 12ff for general information)
αMATH		Submenu. See pp. 115ff.
		Menu in AIM. See p. 128.
$\alpha\text{POS?}$	 	(-1) {} → {1} Looks in the <i>text string r</i> for the target given in X . If a match is found, αPOS returns the position number where the target was found (counting the left-most character as position 0). If a match is not found, αPOS returns -1. ²⁴
	 	The target may be an individual character code or an <i>text string</i> . αPOS saves a copy of the target in L . It works similar to POSA in <i>HP-42S</i> .
αRL		(0) Rotates the <i>text string r</i> by x characters like AROT in <i>HP-42S</i> , but with $x \geq 0$. $\alpha\text{RL } 0$ executes as NOP, but loads L . ²⁴
αRR		(0) Works like αRL but rotates to the right.
αSL		(0) Shifts the x leftmost characters out of the <i>text string r</i> , like ASHF in <i>HP-42S</i> . This allows for deleting the first x characters in the string. $\alpha\text{SL } 0$ executes as NOP, but loads L . ²⁴
αSR		(0) Works like αSL but for the x rightmost characters out of <i>r</i> , deleting the last x characters in the string. ²⁴
αFN		Menu. See p. 128.
$\text{A...}\Omega$		Submenu of Greek letters, see pp. 115ff.
$\alpha\cdot$		Submenu. See pp. 115ff.
		Menu in AIM. See p. 128.

²⁴ This command will throw an error if there is no *text string* or an empty string in *r* at execution time.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\alpha \rightarrow x$		(-1) {} → {10 ₁₆ } Pushes the character code of the leftmost character in the <i>text string r</i> on the <i>stack</i> and removes this character from the string, similar to ATOX in HP-42S. ²⁴
$\beta(x,y)$		(2) {2, 3}; {1} → {2} Returns <i>Euler's Beta function</i> $B(x, y) = \frac{\Gamma(x) \Gamma(y)}{\Gamma(x+y)}$ with $Re(x) > 0$ and $Re(y) > 0$. Called β here to avoid ambiguity. See $\Gamma(x)$ below.
Γ_{xy}		(2) {2}; {1} → {2}
γ_{xy}		Returns the <i>upper / lower incomplete Gamma function</i> . See pp. 255ff for more.
$\Gamma(x)$		(1) {2, 3}; {1} → {2} Returns $\Gamma(x)$. Note $x!$ calls $\Gamma(x + 1)$. See also LNG.
δx		Predefined global label for $f'(x)$ and $f''(x)$ – see Section 4 of the OM.
$\Delta\%$		(1) {2}; {1} → {2} Returns $100 \frac{x-y}{y}$ leaving y unchanged, like %CH in HP-42S. Use it also for calculating mark-ups or margins as explained in the OM, Sect. 2.
ε		(-2) {} → {2} Calculates the <i>scattering factors</i> ε_y and ε_x for <i>log-normally</i> distributed sample data and pushes them on the <i>stack</i> . ε works for the <i>geometric mean</i> \bar{x}_g in analogy to the <i>standard deviation</i> s for the <i>arithmetic mean</i> \bar{x} but <u>multiplicative</u> instead of additive. See pp. 222ff for more information.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ε_m	f STAT g ε_m	(-2) { } → {2} Works like ε above but returns the <i>scattering factors</i> of the two <i>geometric means</i> (in analogy to the standard error for <i>arithmetic means</i>).
ε_p	f STAT g ε_p	(-2) { } → {2} Works like ε but returns the <i>scattering factors</i> of the two <i>populations</i> .
$\zeta(x)$	g X.FN ▲ f $\zeta(x)$	(1) {2, 3} Returns <i>Riemann's Zeta</i> . See p. 259 for more.
π	g π	(-1) { } → {2} Recalls π .
Π_n	f ADV Π_n labl	Computes a product using the routine specified. See Section 4 of the OM for more. ATTENTION: Π_n fills all <i>stack registers</i> with x before calling the routine specified.
Σ	g Σ	Menu. See p. 129.
s	f STAT s	(-2) { } → {2} Works like s but returns the <i>standard deviations</i> of the two <i>populations</i> instead. See pp. 222ff.
$\Sigma^{1/x}$	g Σ ▲ $\Sigma^{1/x}$ etc.	(-1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions beyond the linear model. Calling these sums by name significantly improves program readability. Note they are stored in dedicated <i>registers</i> of your WP 43S (see App. B on pp. 160ff.).
Σ^{1/x^2}		
$\Sigma^{1/y}$		
Σ^{1/y^2}		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\Sigma \ln^2 x$	 etc.	ATTENTION: Depending on your input data, logarithmic or inverted sums may become infinite or even non-numeric. If this happens no error will be thrown, regardless of the status of SPCRES.
$\Sigma \ln^2 y$		
$\Sigma \ln x$		
$\Sigma \ln xy$		For space reasons, two sums are abbreviated: $\Sigma \ln xy$ denotes $\sum \ln(x)\ln(y)$ and
$\Sigma \ln y/x$		$\Sigma \ln y/x$ denotes $\sum \frac{\ln(y)}{x}$.
Σ_n		Computes a sum using the routine specified. See Section 4 of the OM for more. ATTENTION: Σ fills all <i>stack registers</i> with x before calling the routine specified.
σ_w		(-1) {} → {2} Works like s_w but returns the <i>standard deviation</i> of the <i>population</i> instead. See pp. 222ff.
$\Sigma x, \Sigma x^2$		
$\Sigma x^2 y$... 	(-1) {} → {2}
$\Sigma x^2/y$... 	Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see $\Sigma 1/x$ above for more).
Σx^3	... 	
Σx^4		
$\Sigma x \ln y$... 	
Σxy	... 	
$\Sigma x/y$... 	
$\Sigma y, \Sigma y^2$		
$\Sigma y \ln x$		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\Sigma+^{25}$	 	$\{8\} \rightarrow \{2\}$ If \mathbf{X} contains an $n \times 2$ matrix then $\Sigma+$ adds n 2D data points to the statistical sums. Then the display will show the last data point added (corresponding to the last row of said matrix) and the matrix will be saved in \mathbf{L} . See the OM, Section 2.
		$\{1, 2\}$ Adds one 2D data point to the statistical sums. Note both \mathbf{X} and \mathbf{Y} must contain DT 1 or 2.
$\Sigma-^{25}$	 	$\{1, 2\}$ Works like $\Sigma+$ but subtracts one 2D data point. Updates neither x_{\max} nor x_{\min} .
$\chi^2_p(x)$	  $\chi^2_p(x)$ etc.	$(1) \{2\}; \{1\} \rightarrow \{2\}$ <i>Chi-square distribution (with its degrees of freedom given in I). $\chi^2_\Delta(x)$ equals $Q(\chi^2)$ on HP-21S. See Section 2 of the OM for an application example and pp. 222ff for more.</i>
$\chi^2_\Delta(x)$		
$\chi^2_\Delta(x)$		
$(\chi^2)^{-1}$		
$\chi^2:$	 	Submenu. See p. 124.

²⁵ $\Sigma+$ and $\Sigma-$ return *temporary information* as shown in Section 2 of the OM and disable ASL. Both commands may also be used for 2D vector adding and subtracting (see the command SUM and the corresponding example in Section 2 of the OM)..

Item	Keystrokes	Remarks (see pp. 12ff for general information)									
$(-1)^x$	 	(1) {1, 2, 3, 8*, 9*, 10} If x is non-integer, returns $\cos(\pi x)$.									
$[M]^T$	 	(1) {8, 9} Returns the transpose of the matrix x (like TRANS in HP-42S). The transpose is another matrix with rows changed by columns. If A is an $n \times m$ matrix and a_{ij} is an element of it then A^T will be an $m \times n$ matrix B with $b_{ij} = a_{ji}$. The transpose is done in-situ and does not require any additional memory.									
$[M]^{-1}$	 	(0) {8, 9} Takes the square matrix in X and inverts it in-situ (like INVRT on HP-42S).									
+		(2) Returns $y + x$ for compatible objects. ²⁶									
$+/-$	 (for closed input)	(1) 'Unary minus', returns $x \times (-1)$. ²⁶									
$\pm\infty?$	 	(0) {2} → {1} Tests x for infinity. Returns <table style="margin-left: 20px;"><tr><td>true</td><td>1</td><td>for $x = +\infty$,</td></tr><tr><td>true</td><td>-1</td><td>for $x = -\infty$, and</td></tr><tr><td>false</td><td>0</td><td>else.</td></tr></table>	true	1	for $x = +\infty$,	true	-1	for $x = -\infty$, and	false	0	else.
true	1	for $x = +\infty$,									
true	-1	for $x = -\infty$, and									
false	0	else.									
-		(2) Returns $y - x$ for compatible numeric objects. ²⁶									
\times		(2) Returns $y \times x$ for compatible numeric objects. ²⁶									
$\times\text{MOD}$	 	(3) {1, 2, 10} Returns $(z \times y) \bmod x$ for $x > 1, y > 0, z > 0$. ²⁷									

²⁶ See the combination tables in the OM, Section 2, for details and compatibility.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
/	[/]	(2) Like \times , but returns $y \times x^{-1}$. ²⁶ Returns y div x if both y and x are of DT 1 or 10; cf. IDIV.
\wedge MOD	[g] [INTS] [g] \wedge MOD	(3) {1, 2, 10} Returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. ²⁷
\rightarrow		Reserved symbol for indirect addressing.
\rightarrow DATE	[g] [CLK] \rightarrow DATE	(3) {1, 2} \rightarrow {6} Assumes the three components of a date (year, month, and day) supplied on the stack in proper order for the date format selected and converts them to a single <i>date</i> in x . Thus inverts DATE \rightarrow .
\rightarrow DEG	[g] [L \leftrightarrow] \rightarrow DEG	(1) {1, 2, 4} \rightarrow {4} Convert angles as described on pp. 147f.
\rightarrow D.MS	[g] [L \leftrightarrow] \rightarrow D.MS	
\rightarrow GRAD	[g] [L \leftrightarrow] \rightarrow GRAD	
\rightarrow HR	[f] [CATALOG] FCNS \rightarrow HR	(1) {5} \rightarrow {2} Operates on <i>times</i> like \rightarrow REAL below. Maintained for backward compatibility only.
\rightarrow H.MS	[f] [h.ms] (for closed input)	(1) {1, 2, 5} \rightarrow {5} Converts x to a sexagesimal <i>time</i> – cf. p. 110.
\rightarrow INT	[f] [#] base (for closed input)	(1) {1, 2, 10} \rightarrow {10} Converts the integer part of x to a <i>short integer</i> of the base specified. Conversion to decimal may be abbreviated by [#D], to hexadecimal by [#H]. Note [#I] converts to a <i>long integer</i> . Cf. p. 110.
\rightarrow MUL π	[g] [L \leftrightarrow] \rightarrow MUL π	(1) {1, 2, 4} \rightarrow {4} Converts angles as described on pp. 147f.

²⁷ See MOD.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→POL	g →P	<p>{2}; {1} → {2}</p> <p>Assumes X and Y containing 2D <i>Cartesian</i> coordinates of a point or components of a vector (x, y). Converts them to the respective polar coordinates or components (r, θ). Reverted by →REC, see there.</p> <p>For switching the display format of <i>complex numbers</i>, see POLAR.</p>
→RAD	g L →RAD	<p>(1) {1, 2, 4} → {4}</p> <p>Converts angles as described on pp. 147f</p>
→REAL	f .d (for closed input)	<p>(1) {1, 2, 4, 5, 6, 10} → {2}</p> <p>Converts x to a <i>real number</i>. Any object (e.g. a <i>time</i>) tagged sexagesimal will be converted in a decimal number. For <i>dates</i>, the date format chosen is taken into account. Numbers shown as fractions will be displayed as decimal numbers (cf. FRACT and PROPFR).</p> <p>To return the <i>real part</i> of a <i>complex number</i>, take RE. To cut a <i>complex number</i> into its parts, use CC or CX→RE.</p>
→REC	f R	<p>{2}; {1, 4} → {2}</p> <p>Assumes X and Y containing 2D polar coordinates of a point or components of a vector (r, θ). Converts them to the respective Cartesian coordinates or components (x, y). Inverted by →POL. – For switching the format of <i>complex numbers</i>, see POLAR.</p>

Item	Keystrokes	Remarks (see pp. 12ff for general information)
»	g [STK] »	<p>Shuffles the contents of the <i>stack registers X, Y, Z, and T</i> at execution time.</p> <p>Examples:</p> <ul style="list-style-type: none"> »xxyz works like ENTER↑ (but does <u>not</u> disable ASL!), »yxzt works like x↔y, »yztx works like R↓ in a 4-level stack, »txyz works like R↑ in a 4-level stack, <p>but also »yytt or »zzzx is possible.</p> <p>ATTENTION: This is a very powerful command although it does not look it. Note it will affect the <u>bottom four stack registers only</u>; there is no connection to A ... D, <u>regardless of stack size</u>. Playing with » you may lose some stack contents and make a mess of the stack easily.</p>
M	f [MATX] M	<p>(1) {8} → {2}; {9} → {3}</p> <p>Requires a square matrix in X and returns its determinant. The original matrix is saved in L.</p>
x	<p>f x or f [PARTS] f x </p> <p>f x or g [CPX] f x </p>	<p>(1) {1, 2, 4, 10}</p> <p>Returns the absolute (unsigned) value of x.</p> <p>(1) {8*}</p> <p>Returns a matrix with the absolute values of all input matrix elements. Cf. ENORM.</p> <p>(1) {3} → {2}</p> <p>Returns the <i>magnitude</i> $\sqrt{\operatorname{Re}(x)^2 + \operatorname{Im}(x)^2}$ in X.</p> <p>(1) {9*} → {8}</p> <p>Returns a <i>real</i>/matrix with the <i>magnitudes</i> of all input matrix elements. Cf. ENORM.</p>

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		(2) {2, 3}; {1} → {2} Returns $\left(\frac{1}{x} + \frac{1}{y}\right)^{-1}$; useful in electrical engineering especially. Returns 0. for $x \times y = 0$.
%		(1) {2}; {1} → {2} Returns $\frac{xy}{100}$, leaving y unchanged.
%MRR		(3) {2}; {1} → {2} Returns the mean rate of return in % per period, i.e. $100 \cdot \left(\sqrt[z]{x/y} - 1 \right)$ with $x = FV$ = future value after z periods, $y = PV$ = present value. For $z = 1$, Δ% returns the same result easier.
%T		(1) {2}; {1} → {2} Returns $100^x/y$, interpreted as % of total. Leaves y unchanged.
%Σ		(1) {2}; {1} → {2} Returns $100^x/\sum x$.
%+MG		(2) {2}; {1} → {2} Calculates a sales price by adding a margin of x % to the cost y , as %MU-Price in HP-17B. Formula: $P_{sale} = y / \left(1 - \frac{x}{100}\right)$ You may use %+MG for calculating net amounts as well; just enter a negative percentage in x .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
\sqrt{x}	g or g EXP	(1) {1, 2, 3, 8*, 9*, 10}; (<{1} → {2}, {1, 2} → {3}) Returns the square root of x . See the DT tables in Section 2 of the OM for more.
\int	f ∫ var (listed in programs as trailed by the integration variable)	{2} Integrates the function given in the routine specified by PGMINT over the variable specified. Lower and upper integration limits must be supplied by the corresponding variables \downarrow Lim and \uparrow Lim, accuracy by ACC. \int returns the (approximated) integral in X and an upper limit of its uncertainty in Y . ²⁸ ATTENTION: \int fills all stack registers with x before calling the routine specified in PGMINT.
	g EQN ∫	Integrates the current equation. ²⁸
$\int f$	g EQN	Submenus. See pp. 121f.
$\int f dx$	f	
π	g or g CPX π or f PARTS π	(1) {2} → {4} Returns 180° (or equivalent) for $x < 0$ and 0 else. (1) {3} → {4} Returns the phase or argument $\arg(x) = \arctan\left(\frac{\text{Im}(x)}{\text{Re}(x)}\right)$. Cf. $ x $. (1) {9*} → {8} Returns a matrix with the phases of all input matrix elements. Cf. $ x $.
$\pi \rightarrow$	g L→	Menu of angular conversions. See p. 129.

²⁸ See Section 4 of the OM for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
■ ADV	g PRINT ■ ADV	(0) Prints the current contents of the print buffer and a linefeed. ATTENTION: The printer will actually print only when a linefeed is sent to it.
■ CHAR	g PRINT f ■ CHAR n	(0) Sends a single character (with the code specified) to the printer. Character codes $n > 127$ can only be specified indirectly. ■ MODE setting will be honored. See ■ ADV.
■ DLAY	g PRINT g ■ DLAY n	(0) Sets a delay of n ticks (see TICKS) to be used with each linefeed on the printer.
■ LCD	g PRINT ■ LCD	(0) Sends the contents of the entire LCD to the printer, so you get a hardcopy of the screen.
■ MODE	g PRINT g ■ MODE n	(0) Sets print mode. Legal print modes are: 0: Use the printer font and character set wherever possible (default). All characters feature the same width (5 columns + 2 columns spacing). 1: Use the variable pitch display font, resulting in some jitter on the printout but packing more characters in a row. 2: Use the small display font, which allows for packing even more info in a row. 3: Send the output to the serial line. Works for plain ASCII only – no characters will be translated. Line setup is the same as for serial communication: 9600 baud, 8 bits, no parity.
■ PROG	g PRINT ■ PROG	(0) Prints the listing of the <i>current program</i> (see Section 3 of the OM), one row per step.
■ r	g PRINT ■ r r	(0) Prints the content of the <i>register</i> specified, right adjusted, <u>without</u> labeling the output. If you want a heading label, compose the string in X first or use ■ REGS. See ■ ADV.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
REGS		<p>(1) Interprets x in the form $sss.nn$. Prints the contents of nn registers starting with number sss. Each register takes one row starting with its label.</p> <p>ATTENTION for $nn = 0$:</p> <ul style="list-style-type: none"> • For $sss \in [0; 99]$, printing will stop at R99. • For $sss \in [100; 111]$, printing stops at K. <p>For $sss \geq 112$, printing stops at the highest allocated local register.</p>
STK		(0) Prints the entire stack contents. Each of the 4 or 8 registers prints in a separate row starting with its label.
TAB		(0) Positions the print head to print column n (0 to 165, where $n > 127$ can only be specified indirectly). Useful in formatting (in MODE 1 or 2 in particular). Allows also for printer plots. If n is less than current print head position, a linefeed will be entered to reach the new position. See ADV.
USER		(0) Prints all variable names and global program labels in alphabetic order. The variable names are printed first; if you are not interested in the program labels, press R/S to stop the listing.
WIDTH		<p>(-1) Returns the number of print columns that x would take in the print mode set. See MODE.</p> <p>Second use: in MODE 1 or 2, WIDTH returns the width of x in px (including the last column being always blank) in the specified font.</p>
x		(0) Prints x without a heading label. See ADV.
Σ		(0) Prints the summation registers. Each register prints in one row starting with its label.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
¤#	g PRINT f ¤# n	(0) Sends a single <i>byte</i> , without translation, to the printer (e.g. a control code). <i>n</i> > 127 can only be specified indirectly. ¤MODE setting will not be honored. See ¤ADV.
#	f CATALOG FCNS # n	For inserting an integer $0 \leq n \leq 255$ in a single program step. Kept for backward compatibility to WP 34S only.
#B	f BITS g #B	(1) {10} Counts the bits set in <i>x</i> (like on HP-16C).

Names of System Variables and System Flags

There is a *name* overlap between some constants and commands on one side and predefined variables and *system flags* on the other. Thus, the latter set is kept separate from the other *items*. As required for the *items* above, these *names* must be unique.

The current status of *items* with their *names* printed on grey in the table below can be seen on the screen directly (e.g. in the *status bar*), for those printed on orange it is returned by STATUS. If the second column is green, the corresponding *item* may be modified by the user, if it is yellow, the *item* is read-only for the user and is written by the system exclusively. The contents of *items* printed on light blue can be stored by STOCFG and recalled by RCLCFG.

Some system flags can be accessed via lettered user flags as well (cf. Section 1 of the OM). Keystrokes are only listed if applicable.

Name	Keystrokes	Remarks (see pp. 12ff for general information)
A		Reserved variable for <i>register A</i>
ACC	f ADV ∫fdx ACC	Reserved <i>real</i> variable for the accuracy of integration (see <i>Section 4</i> of the OM).
ADM		Reserved integer variable for the <i>ADM</i> : 0: DEG, 1: D.MS, 2: RAD, 3: MULπ, 4: GRAD.
ALLENG	[A]	
ALPHA		
ALP.IN		
ASLIFT		
AUTOFF		
AUTXEQ		
B		
C		
CARRY	[C]	
CPXj		
CPXRES	[I] (imaginary)	
D		Reserved variable for <i>register D</i> .
DECIM.		
DENANY		
DENFIX		
DENMAX		Reserved integer variable for the maximum denominator, filled by the command DENMAX.
DMY		
FRACT		

Name	Keystrokes	Remarks (see pp. 12ff for general information)
FV		Reserved real variable for the future value of your investment or loan in TVM. ²⁹
GRAMOD		Reserved integer variable determining how the AGRAPH image is displayed: ³⁰ 0: It is merged (OR-ed) with the existing display. 1: It overwrites all pixels in that part of the display. 2: Duplicate “on” pixels get turned “off”. 3: It is XOR-ed with the existing display.
GROW		<i>System flag – see next chapter.</i>
I		Reserved variable for <i>register I</i> .
IGN1ER		<i>System flags – see next chapter.</i>
INTING		
ISM		Reserved integer variable for the <i>integer sign mode</i> : -1: sign and mantissa mode, 0: unsigned, 1: 1's complement, 2: 2's complement.
i%/a		Reserved real variable for the annual interest rate of your investment or loan in TVM. ²⁹
J		
K		
L		
LEAD.0		<i>System flags – see next chapter.</i>
LOWBAT		

²⁹ See Section 5 of the OM.

³⁰ Working like flags 34 and 35 in HP-42S.

Name	Keystrokes	Remarks (see pp. 12ff for general information)
Mat_A	f [MATX] SIM EQ Mat A	
Mat_B	f [MATX] SIM EQ Mat B	
Mat_X	f [MATX] SIM EQ Mat X	Reserved variables for solving systems of linear equations (see Section 2 of the OM).
MDY		System flags – see next chapter.
MULTx		
NPER	g [FIN] TVM n _{PER}	Reserved variable for the <u>total</u> number of <ul style="list-style-type: none"> payment periods for your loan or compounding periods for your investment.
NUM.IN		System flags – see next chapter.
OVERFL	B (big)	
PER/a	g [FIN] TVM f per/a	Reserved variable for the <u>annual</u> number of <ul style="list-style-type: none"> payments for your loan or compounding periods of your investment.
PMT	g [FIN] TVM PMT	Reserved variable for the payment per period for your investment or loan in TVM. ²⁹
POLAR	X	
PRINT		System flags – see next chapter.
PROPFNR		
PRTACT		
PV	g [FIN] TVM PV	Reserved variable for the present value of your investment or loan in TVM. ²⁹
QUIET		System flag – see next chapter.
REALDF		Reserved integer variable for <i>real number</i> display format: 0: ALL, 1: FIX, 2: SCI, 3: ENG.

Name	Keystrokes	Remarks (see pp. 12ff for general information)
RUNIO		
RUNTIM		
SLOW		System flags – see next chapter.
SOLVING		
SPCRES	D (danger)	
SSIZE8		
T		Reserved variables for stack registers T ... Z
X		
Y		
Z		
TDM24		System flags – see next chapter.
TRACE	T	
USB		
USER	f USER	System flag toggled by USER – see next chapter
VMDISP		System flags – see next chapter.
YMD		
αCAP		
↑Lim	f ADV ∫fdx ↑Lim etc.	Reserved real variables for the upper and lower limit of integration (s. the OM, Sect. 4).
#DEC		Reserved integer variable for the number of decimals in real number formatting (actually the parameter specified in last ALL, FIX, SCI, or ENG).

Purposes of System Flags

The *status bar*, especially its indicators (*SBI*), and the command STATUS return visible information about the system status of your WP 43S (cf. Sections 2 and 5 of the OM). Machine readable status information (e.g. for program control) is available via the 40 named *system flags* as listed below, sorted following the *status bar*:

Purpose	SBI	Flag ³¹	Remarks
Time display	Time string	TDM24 (set)	Set for international 24h time display. Expands the date display in the <i>status bar</i> to four digits for the year. Clear for 12h time display: e.g. 1:23 will become 1:23am, 23:45 will become 11:45pm. Shortens the date display in the <i>status bar</i> to two digits for the year.
Date display	Date string	YMD, DMY, MDY (YMD set)	Set for the respective date format chosen. The commands Y.MD, D.MY, and M.DY set the corresponding <i>flag</i> and clear the two others.
Complex results	C / R	CPXRES	Set for allowing <i>complex</i> results also for <i>real</i> input (e.g. $\sqrt{-1}$). Else an error will be thrown in such cases.
Complex letter	—	CPXj	Set for the letter <i>j</i> representing the <i>imaginary</i> number <i>i</i> , clear for <i>i</i> .
Polar notation	E / @	POLAR	Set for polar display of <i>complex numbers</i> , clear for rectangular.

³¹ The flags printed on green may be written by the user, those printed on yellow are written by the system exclusively. *Startup default* status is given in parentheses for the flags set at startup – all others are clear at startup. Grey, orange, and light blue colors are used as in previous chapter.

Purpose	SBI	Flag ³¹	Remarks
Fraction display	—	FRACT	Set if fraction display is chosen, clear for decimal display (a b/c sets FRACT, .d clears it). See next entries.
Fraction kind 1	—	PROPFRACTION (set)	Set for <i>proper fractions</i> , clear for <i>improper fractions</i> (a b/c toggles PROPFRACTION: coming from decimal display, it sets it). PROPFRACTION set allows only <i>proper fractions</i> in display (e.g. $1 \frac{2}{3}$ instead of $\frac{5}{3}$); any <i>reals</i> (with $ x < 10^6$) will be displayed according to the settings of DENANY, DENFIX, and DENMAX as <i>proper fractions</i> .
Fraction kind 2	see the OM	DENANY (set)	Set if <u>any</u> denominator up to DENMAX may appear (DENMAX is indicated in the <i>status bar</i>). ³² For given DENMAX, this is the most precise way of displaying a decimal number as a fraction. See also next entry.
Fraction kind 3	see the OM	DENFIX ³³	Set if the value set by DENMAX is the one and only denominator allowed. Clear if the denominator may be an integer factor of DENMAX. ³⁴
Carry	c or C	CARRY	Reflects the status of the carry bit.
Overflow	o or O	OVERFL	Reflects the status of the overflow bit.
Leading zeros	—	LEAD.0	Set for leading zeros turned on in <i>short integers</i> of bases 2, 4, 8, and 16. ³⁵
Alpha	A or α	ALPHA	Set for AIM, else clear.

³² E.g. for DENMAX = 5 and DENANY set, denominators 1, 2, 3, 4, and 5 are allowed.

³³ DENFIX is evaluated only if DENANY is clear.

³⁴ If DENMAX = 60 and DENFIX is clear, this will allow for denominators 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60 (note 60 was a holy number in ancient Babylon).

³⁵ Works like flag 3 in HP-16C.

Purpose	SBI	Flag ³¹	Remarks
Upper case	A / α	αCAP (set)	Set for capital letters, clear for lower case.
Running timer	⌚	RUNTIM	Set if the timer is running.
Running I/O	⬇️	RUNIO	Set if I/O is in progress.
Printing	🖨️	PRINT	Set if your WP 43S is sending data to the printer.
Tracing	—	TRACE	Set if the print line is in tracing mode. Else prints must be triggered explicitly.
User mode	👤	USER	Set if your WP 43S is in user mode.
USB power	⎓	USB	Set if your WP 43S is connected to a USB power source
Low battery	🔋	LOWBAT	Set if battery voltage is low (cf. BATT? and Section 2 of the OM).
Processor speed	—	SLOW	Kept clear for fresh batteries unless the user intervenes, set for battery low (cf. BATT?). Speed and power consumption will be reduced to ~50% with SLOW set.
Special results	—	SPCRES	Set for allowing special results of calculations (i.e. ±∞ and NaN).
Stack size	—	SSIZE8	Set for eight, clear for four <i>stack registers</i> . Note all <i>register</i> contents will remain unchanged if SSIZE8 is modified (may also be by RCLCFG).
Beeper	—	QUIET	Set for disabled beeper.
Radix mark	—	DECIM. (set)	Set for a decimal point, clear for a comma.
Multiplication symbol	—	MULTx (set)	Set for multiplication symbol ×, clear for ·.

Purpose	SBI	Flag ³¹	Remarks
Overflow from ALL	—	ALLENG	Set if <i>real numbers</i> exceeding the range displayable in ALL or FIX shall be shown in ENGineer's format, clear for SCientific.
Matrix grow mode	—	GROW	Set (e.g. by M.GROW) if matrices may grow, else clear (e.g. by M.WRAP). See the command J+ above and Section 2 of the OM; see also GROW in the HP-42S Owner's Manual, p. 213.
Automatic OFF	—	AUTOFF (set)	Set if automatic shutdown after 600 s is enabled. Else your WP 43S will remain ON until you will turn it off manually or battery voltage will drop below the lower limit (cf. BATT? and Section 2 of the OM).
Automatic execution	—	AUTXEQ	Like flag 11 of HP-42S.
Printer activated	—	PRTACT	Like flag 21 (Print Enable) of HP-42S.
Data entry	—	NUM.IN, ALP.IN	Set for numeric or alphanumeric entry, respectively (like flags 22 and 23 of HP-42S).
Automatic stack lift	—	ASLIFT (set)	Cleared by ENTER, CLX, Σ+, and Σ-, set by all other functions (see the OM, Sect. 1)
Error handling	—	IGN1ER	If set, your WP 43S ignores just 1 arbitrary error and clears IGN1ER then. The operation causing the error will not be executed. This works like flag 25 of HP-42S
Integrating	—	INTING	Set while the <i>Integrator</i> is running.
Solving	—	SOLVING	Set while the <i>Solver</i> is computing a root.
Variable menu	—	VMDISP	Set while the variable menu is displayed.

Nonprogrammable Commands and Keys

The commands marked **violet** in the *I/O* cannot be programmed. The same applies to all operations of the *Matrix Editor* and *Equation Editor*, as well as answers to questions your WP 43S asks.

The *browsers* RBR and STATUS as well as the *application* TIMER use some keys for particular control purposes (e.g. **STO**, **RCL**, **.**, and numeric keys – cf. the *OM, Section 5*). Also RBR, STATUS, as well as the applications TIMER and TVM cannot be programmed.

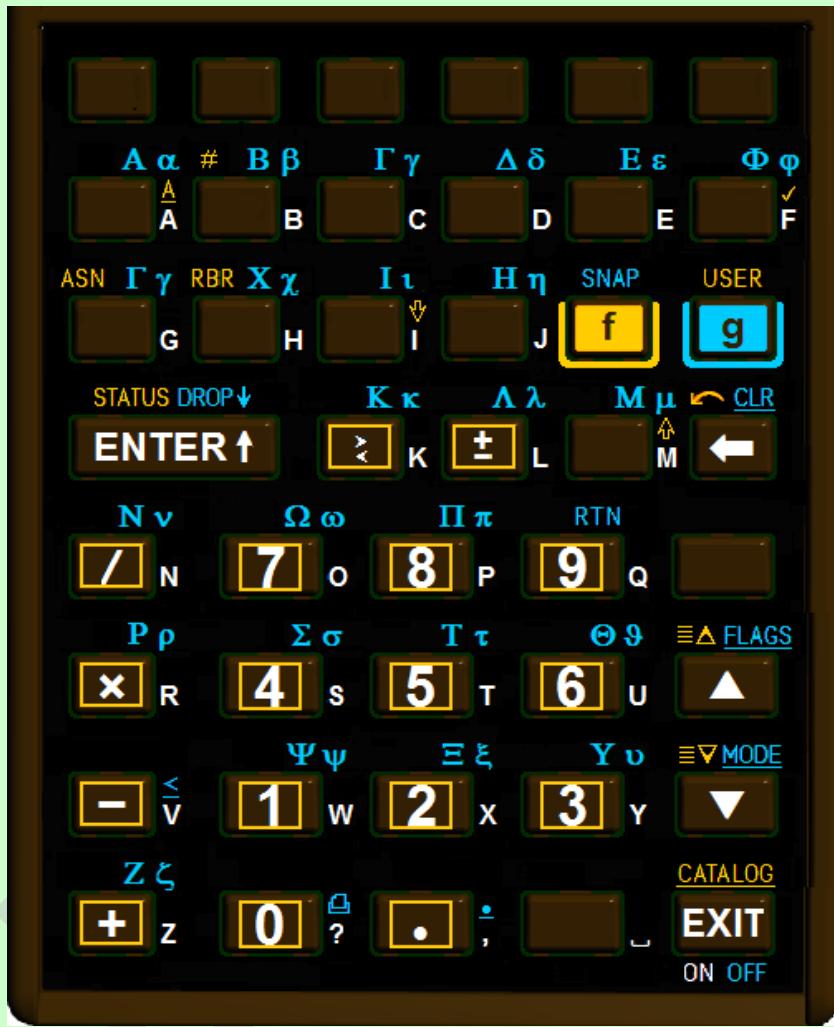
All *catalog* and *menu* calls themselves as well as the operations called by **EXIT**, **P/R**, **α** , **σ** , **Δ** , **$\Delta\Delta$** , **∇** , and **$\nabla\nabla$** in *startup default* condition are neither programmable nor will they show any input echo in the top numeric row as the other commands do (cf. the *OM, Section 2*). See also *Section 2* here (on pp. 114ff) for more about this topic.

Command Parameter Input and Closing It

The following table shows what will happen when particular keys are pressed while command parameter input is not finished yet (see pp. 110ff for input in **X** instead). Note that a “character” may be a letter, digit, punctuation mark, etc. The table below lists the respective keys beginning top left on the keyboard:

Keystrokes	Situation	Meaning
A ... D , I ... L , T , X ... Z	addressing	Enters the address of a <i>global GP register</i> or <i>user flag</i> .
A ... Z	entering a label, a <i>system flag</i> or variable <i>name</i>	Appends the corresponding Latin or Greek letter or digit to the label, <i>system flag</i> or variable <i>name</i> pending. Use ∇ and Δ to switch cases for letters. See the virtual keyboard on p. 109 (cf. the <i>OM, Section 2</i>).
g A ... g O		
f O ... f 9		

Keystrokes	Situation	Meaning
ENTER↑	arbitrary parameter input pending	If there is no input yet, assumes the default, if applicable. Else closes pending input, interprets it as a <i>register</i> or <i>user flag</i> address, a <i>system flag</i> or variable <i>name</i> , a label, or alike, and executes the command. Cf. Section 1 of the OM.
←	arbitrary parameter input pending	Deletes the rightmost character keyed in. If there is nothing left, cancels the pending command, returning to the status of your WP 43S as it was before that input was started.
0 ... 9	addressing or specifying	Enters a numeric parameter, an address, or a local label. See Sections 1 and 3 of the OM for valid number ranges.
.	addressing	Header for <i>local registers</i> or <i>local user flags</i> .
EXIT	arbitrary parameter input pending	If there is an open <i>menu</i> , closes it. Else cancels pending command input, returning to the status of your WP 43S as it was before the current command was called.



Virtual keyboard in *alpha input mode (AIM)*. AIM is also active when a catalog is open, so you can use all accessible characters for alphabetic searching (see pp. 118f). Note there is an 'alpha helper' printed on the rear of your WP 43S.

Alphanumeric Input in X and Closing It

The following table shows what will happen when particular keys are pressed with alphanumeric (incl. numeric) input in X being open still (turn to pp. 107f for command parameter input instead). The table lists the respective keys top left to bottom right on the keyboard:

Keystrokes in mode(s)		Meaning
[A ... [Z	A, α	Appends the corresponding Latin or Greek letter to the <i>text string</i> x . Use ▾ and ▾ to switch cases. See the picture on previous page and cf. Section 2 of the OM.
[g A ... g [O		
[f [#	A, α	Appends # to the <i>text string</i> x .
[f [# base	¬ (A, α)	Closes input of a <i>short integer</i>
[f [d.ms		sexagesimal angle
[f [.d		date
[f [h.ms		sexagesimal time in X. ³⁶
[f [x ² (✓)	A, α	Appends a checkmark ✓ to the <i>text string</i> x .
[CC	¬ (A, α)	Closes input of the first part (i.e. <i>real part</i> or <i>magnitude</i>) of a <i>complex number</i> in X and waits for input of its second part (i.e. <i>imaginary part</i> or <i>phase</i> , see the <i>Key Response Table</i> and Section 2 of the OM). Any additional [CC] in input will be ignored.
[f [R↓ (^)	A, α	Makes the next character a subscript, if applicable.

³⁶ See Section 2 of the OM. After #, any additional # in input will be ignored. At closure, input will be checked – *seconds* (or *minutes*) > 60 will be carried to *minutes* (or *hours*); illegal digits (e.g. 8 in octal input or C in decimal), bases, numbers, or characters found, or out-of-range conditions detected will cause an error thrown (see also the description of **ENTER↑** on next page and the error messages in App. C).

Keystrokes in mode(s)	Meaning
[ENTER↑]	arbitrary input pending If there was input expected but not entered, cancels entry. Else closes input (in X) and checks the following conditions top-down: <ul style="list-style-type: none">• If this input is alphanumeric (i.e. if it contains at least one non-numeric character except E), takes it as a <i>text string</i>.• Else (i.e. if this input is purely numeric) if it contains one CC, takes it as a <i>complex number</i>.• Else if it contains two ,, takes it as a <i>fraction</i>.• Else if it contains one , or one E, takes it as a <i>real number</i>.• Else (i.e. if it contains neither a CC nor a , nor an E), tests it for #:<ul style="list-style-type: none">◦ If it contains one # and a valid base trailing it then takes it as a <i>short integer</i> of said base;◦ else looks up if <u>previous entry</u> was a <i>short integer</i>: if true then takes the new input as another <i>short integer</i> of the same base; else takes the new input as a <i>long integer</i>. Then checks the new input (according to the condition met) as outlined in footnote 36 and interprets it. Finally, unless an error had to be thrown, copies x into Y .
f x>y	A, α Appends > to the <i>text string</i> x .
+/-	$\neg(A, \alpha)$ Changes the sign of the number entered. Affects the exponent if pressed after E .
f +/-	A, α Appends ± to the <i>text string</i> x .
E	$\neg(A, \alpha)$ Closes input of the mantissa and waits for input of the exponent (see Section 1 of the OM). Any additional E in input will be ignored.
f E (^)	A, α Makes the next character a superscript, if applicable.

Keystrokes in mode(s)		Meaning
	arbitrary input pending	Deletes the last (rightmost) character keyed in. If there is nothing left, cancels the pending input, returning to the status of your WP 43S as it was before that input was started.
	A, α	Appends to the <i>text string x</i> .
	A, α	If MULT x is set then appends x , else to the <i>string x</i> .
	A, α	Appends to the <i>text string x</i> .
	A, α	Appends to the <i>text string x</i> .
 A ... F	\neg (A, α)	Numeric input for integer bases >10, appending the corresponding digit to x . See Section 2 of the OM for more. Digits will be checked when input is closed (see the description of above).
	\neg (A, α)	Standard numeric input, appending the corresponding digit to x . Note you can enter ... <ul style="list-style-type: none"> • up to 34 digits plus a sign in the mantissa³⁷ and up to four digits plus a sign in the exponent for a <i>real number</i> or any part of a <i>complex number</i>, • an arbitrary number of digits plus a sign for a <i>long integer</i>, • up to 64 bits for a <i>short integer</i>, or • up to 16 digits for the nominator and up to 4 digits for the denominator of a fraction.
	A, α	Appends to the <i>text string x</i> .
	A, α	Appends the respective digit to the <i>text string x</i> .
	A, α	Appends to the <i>text string x</i> .
	α	Turns to upper case for the letters following.

³⁷ You can enter even more digits but it makes no sense.

Keystrokes in mode(s)		Meaning
[V]	A	Turns to lower case for the letters following.
[.]	\neg (A , α)	For sexagesimal angular input, separates <i>degrees</i> from <i>minutes</i> , <i>seconds</i> , and <i>hundredths of seconds</i> , so input format is dddd.dd.msshh [d.ms] (cf. p. 110 and <i>Section 2</i> of the OM). For <i>time</i> input, separates <i>hours</i> from <i>minutes</i> , <i>seconds</i> , and fractions of <i>seconds</i> , so input format is hhhh.h.mssfffff [h.ms] (cf. p. 110 and <i>Section 2</i> of the OM). Else inserts a radix mark as selected.
Second [.]	\neg (A , α)	A 2 nd [.] in input signals a fraction. See the OM, <i>Section 2</i> for examples. The 2 nd [.] separates nominator and denominator in input. Any additional [.] in input will be ignored. Note you cannot enter [E] after you entered [.] twice but you may delete the 2 nd dot while editing the input.
[.]	A , α	Appends , to the <i>text string x</i> .
[f] [.]	A , α	Appends . to the <i>text string x</i> .
R/S	① , program waiting for input	Closes input and starts its checks and interpretation like [ENTER↑] above. Resumes program execution.
	A , α	Appends a blank space to the <i>text string x</i> .
EXIT	arbitrary input pending	If there is an open <i>menu</i> , closes it. Else closes pending numeric or alphanumeric input and releases it for interpretation.

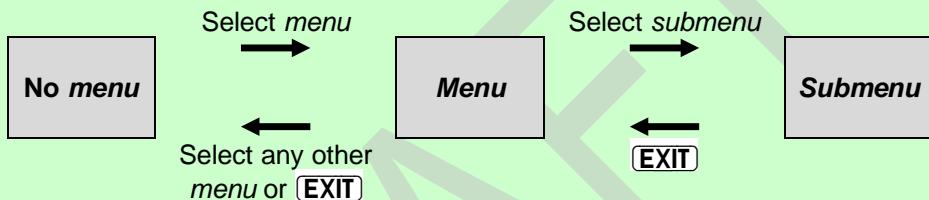
There are many more characters you can enter via the three alpha *menus* or [**CATALOG**] **CHARS** (see pp. 115 and 127ff). Call FBR for browsing the entire character sets provided.

Any other command not specified in the table above will close pending input and release it for interpretation like [**ENTER↑**], [**R/S**], and [**EXIT**] do.

SECTION 2: MENUS AND CATALOGS

Due to the large set of operations your *WP 43S* features, most of them are stored in *menus* as they were discussed in the *OM, Section 1*. Besides operations, numeric constants, or characters (as in the alpha *menus*), there may be also other *items* contained in *menus* (e.g. *submenus*, digits, variables, and program labels).

You may switch *menus* (except *catalogs* – see below) easily by just calling another *menu* accessible in current mode directly from the *menu* you are using – no need to **EXIT** first:



As shown in the picture above, exiting a *submenu* returns you to its ‘parent’ (calling) *menu*; exiting a *menu*, however, returns you to a screen with no *menu* (or **MyMenu** or **Myα**) displayed.

Catalogs are a special kind of *menus* with their contents sorted alphabetically. Your *WP 43S* provides 17 *catalogs*:

- CATALOG'CHARS'αINTL,
- CATALOG'DIGITS,
- CATALOG'FCNS (by far the largest *catalog* at startup),
- CATALOG'MENUS,
- the two *submenus* of CATALOG'PROGS
- CATALOG'SYS.FL,
- the nine *submenus* of CATALOG'VARS and
- CONST.

Within *catalogs*, some special operations ease your path accessing the *items* stored therein (as shown on pp. 118f).

One to Find and Rule Them All – the CATALOG

CATALOG calls a very particular *menu*: **CATALOG** contains all the *items* defined on your *WP 43S* and visible for the user. Many of them are sorted alphabetically in different branches: these *items* we call **catalogued**. Individual *catalogued items* may be accessed quickly in a way demonstrated on pp. 118f.

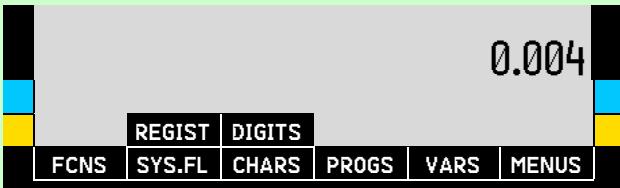
The contents of the various branches of **CATALOG** are presented below. Note they are printed in reverse order compared to the display of your *WP 43S*, taking care of your top-down reading habits:

							Remarks
CATALOG:	FCNS	SYS.FL	CHARS	PROGS	VARS	MENUS	top branches
			REGIST	DIGITS			
FCNS:	$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	10^x	1COMPL	$1/x$	2^x	catalog of the some 740 functions provided
	2COMPL	$\sqrt[3]{\cdot}$	ABS	ACOS	$\text{ac} \rightarrow m^2$	$\text{ac}_{us} \rightarrow m^2$	
	AGM	AGRAPH	ALL	AND	arccos	arcosh	
	...						
	#B				
SYS.FL:	ALLENG	ALPHA	ALP.IN	...			catalog of all system flags (cf. pp. 98ff)
	...						
CHARS:	αINTL	A...Ω		αMATH	Myα	α-	character branches
αINTL:	A	À	Á	Â	Ã	...	catalog of international Latin letters, see p. 127
αMATH:	<	≤	=	≈	...		mathematical operators and symbols, see p. 128
A...Ω:	A	B	Γ	...			Greek letters, see p. 128
α•:	!	;	...				punctuation marks, see p. 129
PROGS:	RAM					FLASH	global labels currently defined

	<input type="checkbox"/>	Remarks					
RAM:	...						both branches are empty at startup; they will be filled with your creations
FLASH:	...						
VARS:	L.INTS	S.INTS	REALS	CPXS	STRING	MATRS	branches for various types of variables
	DATES	TIMES	ANGLES				
ANGLES:	...						catalogs of all variables currently defined, placed following their DTs – most of these submenus are empty at startup but will be filled and grow with your creations
CPXS:	...						
DATES:	...						
L.INTS:	ADM	DENMAX	GRAMOD	REALDF	#DEC	...	
MATRS:	Mat_A	Mat_B	Mat_X	REGS	...		
REALS:	ACC	FV	i%/a	n _{PER}	PER/a	PMT	
	PV	↑Lim	↓Lim	...			
STRING:	...						
S.INTS	...						
TIMES:	...						
MENUS:	ADV	ANGLES	A:	BITS	Binom:	Cauch:	(sub-) menus and sub-menus currently defined (shown here at startup, but also this list will grow with your creations) – see above and below for fix menu contents
	CHARS	CLK	CLR	CONST	CPX	CPXS	
	DATES	DIGITS	DISP	EQN	EXP	Expon:	
	...						
	...	→					
A:	...						(sub-) menus provided unless mentioned above already, see pp. 120ff for predefined contents – here your creations will be inserted as new entries
Binom:	...						
BITS:	...						
...							
...							
REGIST:	A	B	C	D	I	J	register names defined
	K	L	T	X	Y	Z	
DIGITS:	0	1	2	3	4	5	digits defined
	6	7	8	9	A	B	
	C	D	E	F	i		

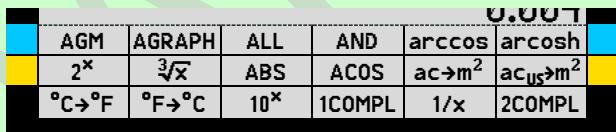
Three branches of **CATALOG** are expandable (**MENUS** and the submenus of **PROGS** and **VARS**) since you may create *items* of these kinds (cf. the OM, Section 6); the other ones are fixed size (**FCNS**, **REGIST**, **SYS.FL**, **DIGITS**, and **CHARS**) since all functions, *register names*, system flags, digits, and characters on your *WP 43S* are predefined.

Calling CATALOG will display its top level branches. The seven labels shown are pointers to the *sub-*



menus containing all the functions, register names, system flags, digits, characters, programs, variables, and menus defined at execution time.

Choosing one of these branches will display its first view of *items* (primary, **f**- and **g**-shifted, as applicable). Pressing the leftmost softkey, for instance, will call the submenu CATALOG/FCNS showing up as pictured here:



Select an *item* by pressing the corresponding *softkey* (headed by **f** or **g** if applicable); e.g.

- call any function via CATALOG'FCNS,
 - call any *menu* via CATALOG'MENUS,
 - test any *system flag* via CATALOG'SYS.FL, or
 - recall any real variable defined via CATALOG'VARS'REALS.

Within **CATALOG** branches, browsing by **▲** will advance by six *items* per keystroke (and **▼** will go back by six) only.³⁸ **EXIT** will just leave the open branch without doing anything.

³⁸ Navigating in catalogs, *AIM* is set as explained in the *OM*. So you may as well use the alphabetic searching method known from *WP 34S catalogs*, but the matching *item* will be displayed together with its up to 17 successors if applicable. See next chapter.

You will find all the over 740 functions available on your *WP 43S* stored in CATALOG'FCNS (most of them may be accessed easier through other *menus* though, see the chapter after next chapter). Remember that each and every command and predefined *menu* featured on your *WP 43S*, the keystrokes calling it, and the necessary particular explanations are printed for your reference in the *IOI* on pp. 12ff. Find the other predefined *items* provided (*system flags* and variable *names*) on pp. 98ff.

See the *OM*, Section 6, to learn how to customize your *WP 43S* by creating your own *menus*, assigning them to your favorite keyboard locations, filling them, and easily accessing the functions you stored therein. You may also assign your favorite functions to almost any location on the keyboard. Actually, you can design your very own *WP 43S* user interface.

Accessing Cataloged Items Rapidly

You can browse a *catalog* like any other *menu* just using \blacktriangle and \blacktriangledown as explained in previous chapter. In CONST and major parts of CATALOG (FCNS, SYS.FL, MENUS, REGIST, DIGITS, CHARS α INTL, and the *submenus* of PROGS and VARS), you may reach your target significantly faster taking advantage of the alphabetic access method demonstrated here. If we are looking for the function FS?S, for **example**:

1 User input

Return

CATALOG FCNS

Your *WP 43S* displays the first view
in this catalog;³⁹ AIM is on.

AGM	AGRAPH	ALL	AND	arccos	arcosh
2^x	$\sqrt[3]{x}$	ABS	ACOS	$ac \rightarrow m^2$	$ac_{us} \rightarrow m^2$
${}^{\circ}\text{C} \rightarrow {}^{\circ}\text{F}$	${}^{\circ}\text{F} \rightarrow {}^{\circ}\text{C}$	10^x	1COMPL	$1/x$	2COMPL

³⁹ ... unless you visited the same *catalog* before – then it will open showing the last view you looked at. The remaining procedure will stay unchanged though.

2 User input	First character of the <i>item</i> desired (e.g. F)																		
Return	Your WP 43S displays a view starting with the first item starting with this character ⁴⁰ e.g.																		
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">FLOOR</td><td style="padding: 2px;">fm.\rightarrowm</td><td style="padding: 2px;">FP</td><td style="padding: 2px;">F_p(x)</td><td style="padding: 2px;">FP?</td><td style="padding: 2px;">fr\rightarrowdB</td></tr> <tr> <td style="padding: 2px;">fēn\rightarrowm</td><td style="padding: 2px;">FF</td><td style="padding: 2px;">FIB</td><td style="padding: 2px;">FILL</td><td style="padding: 2px;">FIX</td><td style="padding: 2px;">FLASH?</td></tr> <tr> <td style="padding: 2px;">FB</td><td style="padding: 2px;">FBR</td><td style="padding: 2px;">FC?</td><td style="padding: 2px;">FC?C</td><td style="padding: 2px;">FC?F</td><td style="padding: 2px;">FC?S</td></tr> </table>	FLOOR	fm. \rightarrow m	FP	F _p (x)	FP?	fr \rightarrow dB	fēn \rightarrow m	FF	FIB	FILL	FIX	FLASH?	FB	FBR	FC?	FC?C	FC?F	FC?S
FLOOR	fm. \rightarrow m	FP	F _p (x)	FP?	fr \rightarrow dB														
fēn \rightarrow m	FF	FIB	FILL	FIX	FLASH?														
FB	FBR	FC?	FC?C	FC?F	FC?S														
3 User input	Second character of the <i>item</i> desired (e.g. S)																		
Return	Your WP 43S displays a view starting with the first item starting with the string you specified e.g.																		
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">f''(x)</td><td style="padding: 2px;">GAP</td><td style="padding: 2px;">GaussF</td><td style="padding: 2px;">GCD</td><td style="padding: 2px;">g_d</td><td style="padding: 2px;">g_d$^{-1}$</td></tr> <tr> <td style="padding: 2px;">fz_{uk}\rightarrowm³</td><td style="padding: 2px;">fz_{us}\rightarrowm³</td><td style="padding: 2px;">F_Δ(x)</td><td style="padding: 2px;">F_Δ(x)</td><td style="padding: 2px;">F⁻¹(p)</td><td style="padding: 2px;">f'(x)</td></tr> <tr> <td style="padding: 2px;">FS?</td><td style="padding: 2px;">FS?C</td><td style="padding: 2px;">FS?F</td><td style="padding: 2px;">FS?S</td><td style="padding: 2px;">ft.\rightarrowm</td><td style="padding: 2px;">ft_{us}\rightarrowm</td></tr> </table>	f''(x)	GAP	GaussF	GCD	g _d	g _d $^{-1}$	fz _{uk} \rightarrow m ³	fz _{us} \rightarrow m ³	F _Δ (x)	F _Δ (x)	F ⁻¹ (p)	f'(x)	FS?	FS?C	FS?F	FS?S	ft. \rightarrow m	ft _{us} \rightarrow m
f''(x)	GAP	GaussF	GCD	g _d	g _d $^{-1}$														
fz _{uk} \rightarrow m ³	fz _{us} \rightarrow m ³	F _Δ (x)	F _Δ (x)	F ⁻¹ (p)	f'(x)														
FS?	FS?C	FS?F	FS?S	ft. \rightarrow m	ft _{us} \rightarrow m														
4 User input	Press the corresponding softkey e.g. for FS?S																		

⁴⁰ This search is case independent (i.e. specifying **A** will find **a** as well). Note, however, that **A** and **a** remain different letters nevertheless. Remember you can search for Greek letters via prefix **g**, e.g. **g** + **A** for α (though watch the sorting order as printed at the beginning of the *IOI*). Also other characters can be specified in a search – please see the *virtual keyboard* printed on p. 110. Note the *items* in the *catalog* you search may be displayed at locations in the *menu section* deviating from the ones you see in simple browsing using **▼** or **▲** exclusively.

You may put in more than one character – though after 3 *seconds* or after pressing **▼** or **▲**, whatever comes first, the search string will be reset. Then you may continue browsing using **▼** or **▲** or start a new search by entering a new first character.

If a character or string specified is not found then the first *item* following alphabetically will be shown – see the sorting order in the *IOI*. If there is no such *item*, then the last *item* in this *catalog* will be displayed.

Return

Your *WP 43S* executes the command, calls the program, recalls the constant or variable, or inserts the command, digit or character selected. *AIM* stays on until this *catalog* is exited – then your *WP 43S* returns to the mode as set before entering this *catalog*.

Result

(in this example after specifying the *flag* number):

true

At the bottom line, this means that ...

- any function provided can be called by **f CATALOG FCNS** + 4 keystrokes maximum if you know its first two characters (i.e. ≤ 7 keystrokes for any function out of more than 740);
- any constant provided can be recalled by **f CONST** + 3 keystrokes maximum if you know its first character;
- any letter provided can be inserted by **f CATALOG CHARS** α INTL (or in *AIM* by **f A**) + 3 keystrokes maximum;
- any system *flag* provided can be tested by **f CATALOG SYS.FL** + 3 keystrokes maximum.

Further Menus and Their Contents

In the table below, all the *menus* provided for you beyond CATALOG are listed in alphabetical sorting order. For each *menu view*, the row of unshifted *softkeys* is listed first, then the **f**-shifted, then the **g**-shifted, following reading habits. Note, however, that on the screen of your *WP 43S* the order of these three rows is reverted with the unshifted row of each *menu view* displayed at the bottom (see the pictures above).

Different *views* within one *menu* are separated by a dashed line, *submenus* by a double line. Individual *items* may appear in more than one *menu* and also on the keyboard.

Menu							Remarks
ADV	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	$\int f dx$	advanced operations, see Sect. 4 of the OM
	PGMSLV		f''(x)			PGMINT	
$\int f dx$			ACC	\downarrow Lim	\uparrow Lim	\int	
BITS	AND	OR	XOR	NOT	MASKL	MASKR	contains all the Boole's and bit operations (first two views) and settings (third view) of HP-16C and WP 34S
	NAND	NOR	XNOR	MIRROR		ASR	
	SB	BS?	#B	FB	BC?	CB	
	SL	RL	RLC	RRC	RR	SR	
	LJ					RJ	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
CLK	DATE	\rightarrow DATE	DATE \rightarrow	WDAY	TIME	x \rightarrow DATE	date and time functions (first view) and settings (second view)
	J \rightarrow D	D \rightarrow J		DAY	MONTH	YEAR	
	SETTIM	TDISP	SETDAT	D.MY	Y.MD	M.DY	
						J/G	
CLR	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX	almost as in HP-42S
	CLREGS	CLPall	CLFall		CLLCD	CLSTK	
	CLall			DELITM		RESET	
CONST							catalog of constants, see pp. 129ff
CPX	dot	cross	UNITV	Re	conj	Re \Rightarrow Im	special complex functions
	CX \rightarrow RE	RE \rightarrow CX	sign	Im	x	\neq	
DISP	FIX	SCI	ENG	ALL	ROUND1	ROUND	display rounding and shifts, formats and settings, mostly for reals
	SDL	SDR			RDP	RSD	
	CHINA	EUROPE	INDIA	JAPAN	UK	USA	
	GAP		RANGE	RANGE?		DSTACK	

Menu							Remarks
<u>EQN</u>	NEW	EDIT	f''	f'	∫f	Solver	equations (see the OM, Sect. 4)
	DELETE						
<u>Solver</u>							show the <i>names</i> of all variables of the current equation and more
<u>∫f</u>							
<u>f'</u>							
<u>f''</u>							
<u>EQ.EDI</u>	←	()	^	:	=	→	Equation Editor
<u>EXP</u>	x ³	√y	log ₁₀ y	lb x	2 ^x	√x	exponential, logarithmic, and hyperbolic functions
	³√x			ln 1+x	e ^{x-1}		
	sinh	arsinh	cosh	arcosh	tanh	artanh	
<u>FIN</u>	%	%MRR	%T	%Σ	%+MG	TVM	financial functions and settings (see the OM, Section 5)
TVM	n _{PER}	i%/a	per/a	PV	PMT	FV	
	Begin					End	
<u>FLAGS</u>	SF	FS?	FF	STATUS	FC?	CF	
	FS?S	FS?C	FS?F	FC?F	FC?S	FC?C	
						CLFall	
<u>INFO</u>	SSIZE?	MEM?	RM?	ISM?	WSIZE?	KTYP?	system information plus one non-binary test (±∞?)
	LocR?	FLASH?	ULP?	NEIGHB	SDIGS?	BATT?	
	WHO?	VERS?	DIM?	±∞?	αPOS?	αLENG?	
	RANGE?	J/G?				BestF?	
<u>INTS</u>	A	B	C	D	E	F	digits for <i>short integers</i> with bases > 10, integer operations
	IDIV	RMD	MOD	*MOD	FLOOR	LCM	
	DBL /	DBLR	DBL ×	^MOD	CEIL	GCD	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
<u>I/O</u>	LOAD	LOADP	LOADR	LOADSS	LOADV	LOADΣ	data exchange and signalling
	BEEP	TONE			RECV	SEND	

Menu							Remarks
LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG	
	DEC					INC	
MATX	NEW	$[M]^{-1}$	$ M $	$[M]^T$	SIM EQ	EDIT	matrix operations (the items sorted almost as in HP-42S)
	dot	cross	UNITV	DIM	INDEX	EDITN	
	ENORM		STOEL	RCLEL	PUTM	GETM	
	I+	I-	STOIJ	RCLIJ	J-	J+	
	RSUM	RNORM	M.LU	DIM?		R N R	
	EIGVAL					EIGVEC	
	M.EDIT	←	↑	OLD	GOTO	↓	→
	INSR			DELR		WRAP	GROW
M.SIMQ	Mat A	Mat B				Mat X	solver for systems of linear equations
MODE	SF	DEG	RAD	GRAD	MULπ	CF	mode settings; SYSTEM is not available on the simulator
			RM		SETSIG	DENMAX	
	SYSTEM						
MyMenu							will pop up out of AIM ⁴¹
Myα							will pop up in AIM ⁴¹
PARTS	IP	FP	MANT	EXPT	sign	DECOMP	some overlaps with HP-42S CONVERT
					x	≠	
					Re	Im	

⁴¹ ... as long as no other menu is called (see Section 6 of the OM).

Menu							Remarks
PRINT							the PRINT commands of the HP-42S
PROB	Norml:	t:	C _{yx}	P _{yx}	F:	χ^2 :	combinations, permutations, random number generators and 14 probability distributions. Selecting one (e.g. Norml) opens a submenu featuring 4 entries for its PDF (or PMF), CDF, error probability, and quantile function
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:	
		NBin:	Geom:	Hyper:	Binom:	Poiss:	
	RAN#	SEED	RANI#		lnΓ	Γ(x)	
	Binom:	Binom _p		Binom _△	Binom _△		
	Cauch:	Cauch _p		Cauch _△	Cauch _△		
	Expon:	Expon _p		Expon _△	Expon _△		
	F:	F _p (x)		F _△ (x)	F _△ (x)		
	Geom:	Geom _p		Geom _△	Geom _△		
	Hyper:	Hyper _p		Hyper _△	Hyper _△		
	LgNrm:	LgNrm _p		LgNrm _△	LgNrm _△		
	Logis:	Logis _p		Logis _△	Logis _△		
	NBin:	NBin _p		NBin _△	NBin _△		
	Norml:	Norml _p		Norml _△	Norml _△		
	Poiss:	Poiss _p		Poiss _△	Poiss _△		
P.FN	t:	t _p (x)		t _△ (x)	t _△ (x)		additional programming functions (avoided a multi-view menu here).
	Weibl:	Weibl _p		Weibl _△	Weibl _△		
	χ^2 :	$\chi^2_p(x)$		$\chi^2_{\Delta}(x)$	$\chi^2_{\Delta}(x)$		
P.FN2	INPUT	END	ERR	TICKS	PAUSE	P.FN2	
	PSTO	PRCL	VARMNU	MVAR	CNST	PUTK	
	R-CLR	R-COPY	R-SORT	R-SWAP	LocR	PopLR	
	MENU	KEYG	KEYX	CLMENU	EXITall	RTN+1	
	SDL	SDR	MSG	NOP			
	BACK	CASE	SKIP	AGRAPH	PIXEL	POINT	

Menu							Remarks
STAT	$\Sigma+$	\bar{x}	s	s	s_m	SUM	for sample statistics.
	$\Sigma-$	\bar{x}_w	s_w	s_w	s_{mw}		
	CLΣ	\bar{x}_G	ε	ε_p	ε_m	PLOT	
	L.R.	r	s_{xy}	Cov	\hat{x}	\hat{y}	for curve fitting and 2d sample statistics.
	$s(a)$	\bar{x}_H					
		\bar{x}_{RMS}	x_{max}	x_{min}		OrthoF	for choosing the fit model(s)
	LinF	ExpF	LogF	PowerF		BestF	
	GaussF	CauchF	ParabF	HypF	RootF		
STK	$x\vec{}$	$y\vec{}$	$z\vec{}$	$t\vec{}$	$\vec{}$	DROPy	stack related operations.
TEST	$x < ?$	$x \leq ?$	$x = ?$	$x \neq ?$	$x \geq ?$	$x > ?$	binary tests.
	INT?	EVEN?	ODD?	PRIME?	LEAP?	FP?	
	ENTRY?	KEY?	LBL?	STRI?	CONVG?	TOP?	
	$x=+0?$	$x=-0?$	$x \approx ?$	MATR?	CPX?	REAL?	
	SPEC?	NaN?		M.SQR?			
TRI	sin	arcsin	cos	arccos	tan	arctan	trigonometric & hyperbolic functions (cf. EXP).
	sinc	sincπ					
	sinh	arsinh	cosh	arcosh	tanh	artanh	
U→	E:	P:	year→s	F&p:	m:	x:	unit conversions (see pp. 139ff).
	°C→°F	°F→°C	s→year		V:	A:	
	power ratio → dB	dB → power ratio	Nm → lbf·ft → Nm		field ratio → dB	dB → field ratio	
	A:	acre → m²	m² → acre	ha→m²	m²→ha	acre _{US} → m²	units of area
		mÜ→m²	m²→mÜ				
E:	cal→J	J→cal	Btu→J	J→Btu	Wh→J	J→Wh	units of energy

Menu							Remarks
F&p:	lbf→N	N→lbf	bar→Pa	Pa→bar	psi→Pa	Pa→psi	units of force and pressure
	in.Hg → Pa	Pa → in.Hg	torr → Pa	Pa → torr	atm→Pa	Pa→atm	
			mmHg → Pa	Pa → mmHg			
m:	lb.→kg	kg→lb.	cwt→kg	kg→cwt	oz→kg	kg→oz	units of mass
	stone → kg	kg → stone	short cwt→kg	kg → sh.cwt	tr.oz → kg	kg → tr.oz	
	ton→kg	kg→ton	short ton → kg	kg → short ton	carat → kg	kg → carat	
P:	hp_E→W	W→hp_E	hp_UK→W	W→hp_UK	hp_M→W	W→hp_M	units of power
V:	gl_UK→m³	m³→gl_UK	qt.→m³	m³→qt.	gl_US→m³	m³→gl_US	units of volume
	floz_UK → m³	m³ → floz_UK	barrel → m³	m³ → barrel	floz_US → m³	m³ → floz_US	
X:	au→m	m→au	l.y.→m	m→l.y.	pc→m	m→pc	units of length
	mi.→m	m→mi.	nmi.→m	m→nmi.	ft.→m	m→ft.	
	in.→m	m→in.			yd.→m	m→yd.	
	lǐ→m	m→lǐ	yǐn→m	m→yǐn	zhàng → m	m → zhàng	
	chǐ→m	m→chǐ	cùn→m	m→cùn	fēn→m	m→fēn	
	fathom → m	m → fathom	point → m	m → point	survey foot_US → m	m → survey foot_US	

X.FN	AGM	B _n	B _n *	erf	erfc	Orthog	advanced mathematical functions like Beta, Bessel, etc.
	FIB	g _d	g _d ⁻¹	I _{xyz}	IΓ _p	IΓ _q	
	J _y (x)	lnβ	lnΓ	max	min	NEXTP	
	W _m	W _p	W ⁻¹	β(x,y)	γ _{xy}	Γ _{xy}	
Orthog	ζ(x)	(-1) ^x					orthogonal polynomials
	H _n	L _m	L _{ma}	P _n	T _n	U _n	
Orthog	H _{np}						

Menu	<input type="checkbox"/>	Remarks					
αINTL	A a	À à	Á á	Â â	Ã ã	Ä ä	[α] catalog of all Latin letters provided. ⁴² All letters but one in this menu will change when case is switched in AIM – note you will see the individual letters displayed in either case only at one time.
	Å å	Æ æ	Ā ā	Ă ĕ	Ą ą	Ɓ b	
	C c	Ҫ ܰ	Ć ܲ	ܴ ܲ	D d	ܵ ܲ	
	ܶ ܲ	ܷ ܲ	E e	ܸ ܲ	ܹ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	F f	G g	ܶ ܲ	H h	I i	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	J j	K k	L l	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	M m	N n	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	O o	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	R r	ܶ ܲ	ܶ ܲ	S s	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	T t	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	X x	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	
	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	ܶ ܲ	

⁴² See https://de.wikipedia.org/wiki/Liste_lateinischer_Alphabete#Erweiterungen.

Menu							Remarks
αMATH	<	≤	=	≈	≥	>	[α] for comparison symbols, parentheses & brackets, as well as more mathematical and related symbols. You can reach every character by 3 keystrokes maximum.
	{	[()]	}	
	x / . ⁴³	÷ / :	∫	∞	∞	∞	
	¬	Λ	¥	≠		&	
	✗	£	⊥	³√	✓	✗	
	✗	✗	✗	✗	✗	✗	
	:=	≈	≡	E	C	R	
	⊗	⊗	⊕				
	±	^	T	-1	ℏ		

α.FN	x→α	αRL	αRR	αSL	αSR	α→x	dedicated functions for <i>alphanumeric strings</i> , plus a font browser
					αLENG?	αPOS?	
	FBR						

Α...Ω	Α α	Β β	Γ γ	Δ δ	Ε ε	Ζ ζ	[α] Greek letters. The keyboard grants direct access to 24 of them. ⁴⁴ Note the two kinds of lower case Σ. See αINTL for more.
Η η	Θ θ	Ι ι	Κ κ	Λ λ	Μ μ		
Ν ν	Ξ ξ	Ο ο	Π π	Ρ ρ	Σ σ		
ζ	Τ τ	Υ υ	Φ φ	Χ χ	Ψ ψ		
Ω ω	ά	έ	ή	ί	ύ		
Ϊ ī	ó	ú	ÿ ü	ö	ώ		

⁴³ With startup default settings, the multiplication dot is found here and the multiplication cross is called via in AIM. If MULTx is clear, however, this dot is called via in AIM and the multiplication cross via [αMATH](#). The symbols : and ÷ will swap, too.

⁴⁴ The Greek alphabet (sic!) goes alpha, beta, gamma, delta, e-psilon, zeta, eta, theta, iota, kappa, lambda, my, ny, xi, o-mikron, pi, rho, sigma, tau, y-psilon, phi, chi, psi, o-mega. About pronunciation, note that ancient Greek H, Θ, and Y are pronounced like Finnish ÄÄ, T, and Y; Finnish Y is spoken like French U or German Ü. Think of Nils Holgersson's goose Yksi (followed by Kaksi, Kolme, Neljä, Viisi, and Kuusi for obvious reasons – these suffice: there is no goose named Seitsemän appearing in that novel).

Menu							Remarks
α•	!	;	:	'	"	✓	[α] for punctuation marks, currency symbols, arrows, and further special characters.
	ı	ż	ſ	ø	~	＼	
	\$	€	%	&	£	¥	
	←	↑	↑↓	↓	→	↖	
	«	»	»	⌚	•	*	
	⌚	⌚	⌚	⌚	⌚	*	
	,	"	...	-			
Σ	n	Σx	Σx^2	Σxy	Σy^2	Σy	all the sums necessary for the statistics in STAT.
		$\Sigma \ln x$	$\Sigma \ln^2 x$	$\Sigma \ln xy$	$\Sigma \ln^2 y$	$\Sigma \ln y$	
	$\Sigma x^2 y$	$\Sigma x \ln y$		$\Sigma \ln y / x$		$\Sigma y \ln x$	
	$\Sigma x^2 / y$	Σ^1 / x	Σ^1 / x^2	$\Sigma x / y$	Σ^1 / y^2	Σ^1 / y	
	Σx^3	Σx^4					
↔	→DEG	→RAD	→GRAD		→D.MS	→MULπ	angular conversions, cf. pp. 147f.
	DEG→	RAD→	GRAD→		D.MS→	MULπ→	
	D→R	R→D		D→D.MS	D.MS→D		

Constants

Your WP 43S contains a *catalog* of 77 physical, astronomical, and mathematical constants:

G	G_0	G_c	g_e	GM_{\oplus}	g_{\oplus}	
c_2	e	e_E	F	F_{α}	F_{δ}	
a	a_0	a_M	a_{\oplus}	c	c_1	

Names of astronomical and mathematical constants are printed on colored background in the table starting overleaf. Values of physical constants (including their relative standard deviations in *red print* below)

are printed on light background if they are exactly defined or almost exactly known – the darker the background, the less precisely the particular value is known.⁴⁵ We use commas as radix marks for better visibility and multiplication dots for space reasons. Formulas are printed where applicable.

Name	Numeric value and <i>rel. SD</i>	Remarks
a (0) ⁴⁶	365,242 5 d <i>(per definition)</i>	<i>Gregorian year</i>
a_0	$5,291\ 772\ 109\ 03 \cdot 10^{-11}$ m <i>($1,5 \cdot 10^{-10}$)</i>	<i>Bohr radius</i> $a_0 = \alpha / 4\pi R_\infty$

⁴⁵ For most of the physical constants in CONST, their precise numeric values (incl. their units) and their relative standard deviations (*rel. SDs*) are from CODATA 2018, copied in May 2019. These are the best values known in the scientific community, agreed on by the national standards institutes worldwide (e.g. by NIST and PTB). Note that the fundamental constants (printed **bold** in the table) of physics all feature less than 16 significant digits.

Relative SDs are included in the table printed here though not contained in CONST. These uncertainties are important for determining the precision of results you obtain using the constants given, through the process of ‘error propagation’ going back to C. F. Gauss (1777 – 1855). This procedure is essential if your results are to be trustworthy – not only in science (remember each and every scientific result shall include the indication of its uncertainty). Please consult suitable reference (e.g. <http://physics.nist.gov/cgi-bin/cuu/Info/Constants/definitions.html>, giving a nice introduction). There is simply no way yardstick measurements can yield results accurate to four decimals.

Apropos, the terms *resolution*, *precision*, and *accuracy* are confused frequently in measuring (and hence in advertising all the more). In a nutshell, *resolution* is the least significant digit a measuring instrument indicates. Using this instrument for measuring the same object under identical conditions multiple times, you get an idea about its *repeatability* (or *precision*); this can be no better than its *resolution* but may be significantly worse – a factor of ten or more may be observed easily in real life. *Accuracy* of a measuring instrument, however, can never be better than its *repeatability*. – Since we cannot know anything about any real-life object or process any better than we can measure it, these considerations are of fundamental importance. We recommend watching them – in your very own interest.

⁴⁶ The numbers in parentheses are to support determination of parameters for CNST – see the IOI.

Name	Numeric value and <i>rel. SD</i>	Remarks
a_{Moon}	$3,844 \cdot 10^8 \text{ m}$ $(1 \cdot 10^{-3})$	Semi-major axis of the Moon's orbit around the earth $\approx 1,3$ <i>light seconds</i> .
a_{\oplus}	$1,495\,979 \cdot 10^{11} \text{ m}$ $(1 \cdot 10^{-6})$	Semi-major axis of the Earth's orbit around the sun. Within the uncertainty stated here, it equals 1 <i>astronomic unit</i> ≈ 499 <i>light seconds</i> ≈ 8 <i>light minutes</i> .
c	$2,997\,924\,58 \cdot 10^8 \text{ m/s}$ <i>(exact)</i>	Speed of light in vacuum $\approx 300\,000 \frac{\text{km}}{\text{s}} = 300 \frac{\text{km}}{\text{ms}} =$ $300 \frac{\text{m}}{\mu\text{s}} = 30 \frac{\text{cm}}{\text{ns}} = 0,3 \frac{\text{mm}}{\text{ps}}$ etc.
c_1 (5)	$3,741\,771\,85 \dots 10^{-16} \text{ W m}^2$ <i>(exact)</i>	First radiation constant $c_1 = 2\pi h c^2$
c_2	$0,014\,387\,768\,77 \dots \text{ m} \cdot \text{K}$ <i>(exact)</i>	Second radiation constant $c_2 = hc/k$
e	$1,602\,176\,634 \cdot 10^{-19} \text{ A s}$ <i>(exact)</i>	Elementary charge $e = \frac{2}{K_J R_K} = \Phi_0 G_0$
e_E	$2,718\,281\,828\,459\,045\,2\dots$	<i>Euler's e.</i>
F	$96\,485,332\,12 \dots \text{ A s/mol}$ <i>(exact)</i>	<i>Faraday constant</i> $F = e N_A$
F_α (10)	$2,502\,907\,875\,095\,892\,8\dots$	
F_δ	$4,669\,201\,609\,102\,990\,6\dots$	<i>Feigenbaum's α and δ</i>
G	$6,674\,30 \cdot 10^{-11} \text{ m}^3/\text{kg s}^2$ $(2,2 \cdot 10^{-5})$	Newtonian constant of gravitation; also known as γ from other authors. See GM_{\oplus} below for a more precise value.
G_0	$7,748\,091\,729 \dots 10^{-5} / \Omega$ <i>(exact)</i>	Conductance quantum $G_0 = 2e^2/h = 2/R_K = eK_j$

Name	Numeric value and <i>rel. SD</i>	Remarks
G_c	0,915 965 594 177 219 0...	Catalan's constant
g_e (15)	-2,002 319 304 362 56 <i>(1,7·10⁻¹³)</i>	Landé's electron g-factor
GM_{\oplus}	$3,986\,004\,418 \cdot 10^{14} \text{ m}^3/\text{s}^2$ <i>(2,0·10⁻⁹)</i>	Newtonian constant of gravitation times the Earth's mass with its atmosphere included (according to WGS84 ⁴⁷)
g_{\oplus}	$9,806\,65 \text{ m/s}^2$ (<i>per def.</i>)	Standard earth acceleration (note the actual values vary from $9,780\,4 \text{ m/s}^2$ at equator to $9,832\,2 \text{ m/s}^2$ at the poles)
h	$6,626\,070\,15 \cdot 10^{-34} \text{ J s}$ <i>(exact)</i>	Planck constant
\hbar	$1,054\,571\,817 \dots 10^{-34} \text{ J s}$ <i>(exact)</i>	Reduced Planck constant $\hbar = h/2\pi$
k (20)	$1,380\,649 \cdot 10^{-23} \text{ J/K}$ <i>(exact)</i>	Boltzmann constant $k = R/N_A$
K_J	$4,835\,978\,48 \dots 10^{14} \text{ Hz/V}$ <i>(exact)</i>	Josephson constant $K_j = 2e/h$
l_{PL}	$1,616\,255 \cdot 10^{-35} \text{ m}$ <i>(1,1·10⁻⁵)</i>	Planck length $l_{PL} = t_{PL}c$
m_e	$9,109\,383\,701\,5 \cdot 10^{-31} \text{ kg}$ <i>(3,0·10⁻¹⁰)</i>	Electron mass $\doteq 511,00 \text{ keV}$
M_{Moon}	$7,349 \cdot 10^{22} \text{ kg}$ <i>(5·10⁻⁴)</i>	Mass of the Moon

⁴⁷ See http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html

Name	Numeric value and <i>rel. SD</i>	Remarks
m_n (25)	$1,674\,927\,498\,04 \cdot 10^{-27}$ kg $(5,7 \cdot 10^{-10})$	Neutron mass $\triangleq 939,57$ MeV
m_n/m_p	1,001 378 419 31 $(4,9 \cdot 10^{-10})$	Neutron to proton mass ratio
m_p	$1,672\,621\,923\,69 \cdot 10^{-27}$ kg $(3,1 \cdot 10^{-10})$	Proton mass $\triangleq 938,27$ MeV
m_{PL}	$2,176\,435 \cdot 10^{-8}$ kg $(1,1 \cdot 10^{-5})$	Planck mass $m_{PL} = \sqrt{\hbar c/G} \approx 22$ µg
m_p/m_e	1 836,152 673 43 $(6,0 \cdot 10^{-11})$	Proton to electron mass ratio
m_u (30)	$1,660\,539\,066\,60 \cdot 10^{-27}$ kg $(3,0 \cdot 10^{-10})$	Atomic mass constant $\approx 10^{-3}$ kg/ N_A
$m_u c^2$	$1,492\,418\,085\,60 \cdot 10^{-10}$ J $(3,0 \cdot 10^{-10})$	Energy equivalent of the atomic mass constant $\approx 931,49$ MeV
m_μ	$1,883\,531\,627 \cdot 10^{-28}$ kg $(2,2 \cdot 10^{-8})$	Muon mass $\triangleq 105,66$ MeV
M_\odot	$1,989\,1 \cdot 10^{30}$ kg $(5 \cdot 10^{-5})$	Mass of the Sun
M_\oplus	$5,973\,6 \cdot 10^{24}$ kg $(5 \cdot 10^{-5})$	Mass of the Earth. See GM_\oplus above for a more precise value.
N_A (35)	$6,022\,140\,76 \cdot 10^{23}$ / mol <i>(exact)</i>	Avogadro's number
NaN	<i>Not a Number</i>	See p. 169 and the corresponding entry in Section 5 of the OM.
p_0	101 325 Pa <i>(per def.)</i>	Standard atmospheric pressure

Name	Numeric value and <i>rel. SD</i>	Remarks
R	8,314 462 618... J/mol K (exact)	Molar gas constant
r _e	2,817 940 326 2·10 ⁻¹⁵ m (4,5·10 ⁻¹⁰)	Classical electron radius $r_e = \alpha^2 a_0$
R _K (40)	25 812,807 45... Ω (exact)	Von Klitzing constant $R_K = h/e^2$
R _{Moon}	1,737 530·10 ⁶ m (5·10 ⁻⁷)	Mean radius of the Moon
R _∞	10 973 731,568 160 / m (1,9·10 ⁻¹²)	Rydberg constant $R_\infty = \frac{\alpha^2 m_e c}{2 h}$
R _⊕	6,96·10 ⁸ m (5·10 ⁻³)	Mean radius of the sun
R _⊕	6,371 010·10 ⁶ m (5·10 ⁻⁷)	Mean radius of the Earth
S _a (45)	6,378 137 0·10 ⁶ m (<i>p. def.</i>)	Semi-major axis
S _b	6,356 752 314 2·10 ⁶ m (1,6·10 ⁻¹¹)	Semi-minor axis
S _{e²}	6,694 379 990 14·10 ⁻³ (1,5·10 ⁻¹²)	First eccentricity squared
S _{e^{1,2}}	6,739 496 742 28·10 ⁻³ (1,5·10 ⁻¹²)	Second eccentricity squared
S _{f⁻¹}	298,257 223 563 (<i>per def.</i>)	Flattening parameter
T ₀ (50)	273,15 K (<i>per definition</i>)	= 0°C, standard temperature

Name	Numeric value and <i>rel. SD</i>	Remarks
T_p	$1,416\,785 \cdot 10^{32} \text{ K}$ <i>(1,1·10⁻⁵)</i>	<i>Planck temperature</i> $T_p = \frac{c^2}{k} \sqrt{\frac{\hbar c}{G}} = \frac{M_p c^2}{k} = \frac{E_p}{k}$
t_{PL}	$5,391\,245 \cdot 10^{-44} \text{ s}$ <i>(1,1·10⁻⁵)</i>	<i>Planck time</i> $t_{PL} = l_{PL}/c$
V_m	$0,022\,413\,969\,5\dots \text{ m}^3/\text{mol}$ <i>(exact)</i>	Molar volume of an ideal gas at standard conditions $V_m = \frac{RT_0}{P_0} \approx 22,4 \text{ l/mol}$
Z_0	$376,730\,313\,668 \Omega$ <i>(1,5·10⁻¹⁰)</i>	Characteristic impedance of vacuum
α (55)	$7,297\,352\,569\,3 \cdot 10^{-3}$ <i>(1,5·10⁻¹⁰)</i>	Fine-structure constant $\alpha = \frac{e^2}{2\varepsilon_0 h c} \approx \frac{1}{137}$
γ	$6,674\,30 \cdot 10^{-11} \text{ m}^3/\text{kg s}^2$ <i>(2,2·10⁻⁵)</i>	Newtonian constant of gravitation; also known as G from other authors. See GM_\oplus below for a more precise value.
γ_{EM}	$0,577\,215\,664\,901\,532\,9\dots$	<i>Euler-Mascheroni constant</i>
γ_p	$2,675\,221\,874\,4 \cdot 10^8 \text{ Hz/T}$ <i>(4,2·10⁻¹⁰)</i>	Proton gyromagnetic ratio $\gamma_p = 4\pi \mu_p/h$
$\Delta\nu_{Cs}$	9 192 631 770 Hz <i>(exact)</i>	Hyperfine transition frequency of ^{133}Cs
ε_0 (60)	$8,854\,187\,812\,8 \cdot 10^{-12} \frac{\text{As}}{\text{Vm}}$ <i>(1,5·10⁻¹⁰)</i>	Vacuum electric permittivity $\varepsilon_0 = 1/\mu_0 c^2$ (note the so-called Coulomb's constant is just $1/4\pi\varepsilon_0$)

Name	Numeric value and <i>rel. SD</i>	Remarks
λ_c	$2,426\,310\,238\,67 \cdot 10^{-12} \text{ m}$ $(3,0 \cdot 10^{-10})$	
λ_{cn}	$1,319\,590\,905\,81 \cdot 10^{-15} \text{ m}$ $(5,7 \cdot 10^{-10})$	Compton wavelengths of the electron $\lambda_c = h/m_e c$, neutron $\lambda_{cn} = h/m_n c$, and proton $\lambda_{cp} = h/m_p c$, respectively
λ_{cp}	$1,321\,409\,855\,39 \cdot 10^{-15} \text{ m}$ $(3,1 \cdot 10^{-10})$	
μ_0	$1,256\,637\,062\,12 \cdot 10^{-6} \frac{\text{Vs}}{\text{Am}}$ $(1,5 \cdot 10^{-10})$	Vacuum magnetic permeability
μ_B (65)	$9,274\,010\,078\,3 \cdot 10^{-24} \text{ J/T}$ $(3,0 \cdot 10^{-10})$	Bohr magneton $\mu_B = e\hbar/2m_e$
μ_e	$-9,284\,764\,704\,3 \cdot 10^{-24} \text{ J/T}$ $(3,0 \cdot 10^{-10})$	Electron magnetic moment
μ_e/μ_B	$-1,001\,159\,652\,181\,28$ $(1,7 \cdot 10^{-13})$	Ratio of electron magnetic moment to Bohr's magneton
μ_n	$-9,662\,365\,1 \cdot 10^{-27} \text{ J/T}$ $(2,4 \cdot 10^{-7})$	Neutron magnetic moment
μ_p	$1,410\,606\,797\,36 \cdot 10^{-26} \text{ J/T}$ $(4,2 \cdot 10^{-10})$	Proton magnetic moment
μ_u (70)	$5,050\,783\,746\,1 \cdot 10^{-27} \text{ J/T}$ $(3,1 \cdot 10^{-10})$	Nuclear magneton $\mu_u = e\hbar/2m_p$
μ_μ	$-4,490\,448\,30 \cdot 10^{-26} \text{ J/T}$ $(2,2 \cdot 10^{-8})$	Muon magnetic moment

Name	Numeric value and <i>rel. SD</i>	Remarks
σ_B	$5,670\,374\,41\dots \cdot 10^{-8} \frac{W}{m^2 K^4}$ <i>(exact)</i>	Stefan-Boltzmann constant $\sigma_B = \frac{2\pi^5 k^4}{15 h^3 c^2}$
Φ	$1,618\,033\,988\,749\,894\,8\dots$	Golden ratio $\Phi = \frac{1}{2}(1 + \sqrt{5})$
Φ_0	$2,067\,833\,848\dots \cdot 10^{-15} V s$ <i>(exact)</i>	Magnetic flux quantum $\Phi_0 = \frac{\hbar}{2e}$
ω (75)	$7,292\,115 \cdot 10^{-5} \text{ rad/s}$ <i>($2 \cdot 10^{-8}$)</i>	Angular velocity of the Earth according to WGS84 (see footnote 47 on p. 132).
$-\infty$	$-\infty$	Note both these 'constants' are counted as numeric values in your WP 43S.
∞	∞	They can be recalled and used with SPCRES set, else an error will be thrown.

Note that these constants will appear in routines with a heading # (e.g. # **c**) as shown in the OM, Section 3.

A few more constants with unknown accuracy, not stored in CONST, follow here for your convenience:

Numeric value	Remarks
$11,2 \cdot 10^3 \text{ m/s}$	Escape speed from Earth
$1,217 \text{ kg/m}^3$	Air density at standard conditions
331 m/s	Speed of sound in air at standard conditions
$28,97 \cdot 10^{-3} \text{ kg/mol}$	Molar mass of air
1370 W/m^2	Solar constant (average incident power at the mean distance of Earth to the Sun outside the atmosphere)
$1,62 \text{ m/s}^2$	Gravity acceleration on the Moon

Some more values found in nature are known with up to 1, 2, or 3 digits precision only – they may be helpful for quick estimates nevertheless:

Radius of an atomic nucleus	$\sim 10^{-15}$ m
Radius of an atom ⁴⁸	$\sim 10^{-10}$ m
Radius of our home galaxy	$\sim 10^5$ l.y. $\approx 10^{21}$ m
Radius of the observable universe ⁴⁹	$\sim 45 \times 10^9$ l.y. $\approx 4.3 \times 10^{26}$ m
Age of the universe	13.8×10^9 a = 4.35×10^{17} s
Hubble ‘constant’	~ 70 (km/s)/Mpc $\approx 1/(4.4 \times 10^{17}$ s)
Baryonic mass ⁵⁰ in observable universe	$\sim 10^{53}$ kg
Number of stars in our home galaxy ⁵¹	$\sim 2 \times 10^{11}$
Number of neuron connections in human brain	$\sim 10^{14}$
Number of stars in observable universe	$\sim 10^{23}$

⁴⁸ So the nucleus takes far less than a billionth of the volume of an atom. Electrons are even smaller. Thus, an atom is almost completely empty space. Our world as we know and see it every day is built of atoms. You can even touch many of these real-world objects. Think about it!

By the way, these facts also give some hand-waving arguments why cancer therapy using heavy ion beams works at all (and often significantly better than using X-rays).

⁴⁹ For more information, please see https://en.wikipedia.org/wiki/Observable_universe (or https://de.w.../Beobachtbares_Universum or https://fr.w.../Univers_observable or https://es.wikipedia.org/wiki/Universo_observable etc. – the latter site includes a picture https://es.wikipedia.org/wiki/Universo_observable#/media/Archivo:Universoobservable.PNG explaining the difference between the radius printed above and the naïve assumption of 13.8×10^9 l.y. for it; this picture is not found on the other sites).

Note the radius of the observable universe divided by the radius of our home galaxy returns a value in the same ballpark as the radius of an atom divided by the radius of a nucleus.

⁵⁰ I.e. the mass of all common matter (i.e. atoms etc.). Physicists think this cannot be everything (for good reasons) and talk about ‘dark matter’ but this was not detected yet.

⁵¹ Assume our home galaxy fills a huge flat cylinder 1000 l.y. high, what will be the average distance between its stars? So what will happen when two such galaxies collide? (Calculate yourself, then feel free looking up footnote 55 for an estimate.)

Amount of atoms in the Sun	⁵²	$\sim 10^{57}$
Amount of atoms in observable universe		$\sim 10^{80}$

Note these quantities are all far within the range of *reals* allowed on your *WP 43S* (see *App. B*). Physical constants are seldom more precisely known than twelve digits. Please take both facts into account when assessing very small numeric differences as well as when talking about very large numbers.

Unit Conversions

Your *WP 43S* features 14 angular conversions provided in **A** (cf. p. 129) and 94 unit conversions in **U**. The latter is subdivided into various branches as explained in the OM, Section 5. Its top view looks like this:



with

- **E:** standing for the *submenu* of energy unit conversions,
- **P:** for power,
- **F&p:** for force and pressure,
- **m:** for mass,
- **x:** for length,
- **A:** for area, and
- **V:** for volume.

See pp. 125f for more details of the structure.

⁵² Take this number for the amount of atoms in our solar system as well – the stuff beyond the sun is neglectable here.

The seven basic *SI* units are: *kelvin*, *meter*, *kilogram*, *second*, *ampere*, *mol*, and *candela* (the latter measuring *luminous intensity*⁵³). Beyond these and products or powers of them, knowledge of some *SI derived units* carrying special names is helpful. Each conversion contained in U either begins or ends in one of these *SI* units:

Quantity	Unit	Symbol and formula
Temperature	<i>degree Celsius</i>	$\vartheta[\text{°C}] = T[\text{K}] - 273.15$
Force	<i>newton</i>	$1 \text{ N} = 1 \text{ kg m/s}^2$
Pressure	<i>pascal</i>	$1 \text{ Pa} = 1 \text{ N/m}^2 = 1 \text{ kg/m s}^2$
Energy	<i>joule</i>	$1 \text{ J} = 1 \text{ N m} = 1 \text{ kg m}^2/\text{s}^2$
Power	<i>watt</i>	$1 \text{ W} = 1 \text{ V A} = 1 \text{ J/s}$
Electric potential	<i>volt</i>	$1 \text{ V} = 1 \text{ W/A}$
Charge	<i>coulomb</i>	$1 \text{ C} = 1 \text{ A s}$
Capacitance	<i>farad</i>	$1 \text{ F} = 1 \text{ C/V} = 1 \text{ As/V}$
Conductance	<i>siemens</i>	$1 \text{ S} = 1 \text{ A/V}$
Resistance	<i>ohm</i>	$1 \Omega = 1 \text{ V/A}$
Magnetic flux	<i>weber</i>	$1 \text{ Wb} = 1 \text{ Vs}$
Magnetic flux density	<i>tesla</i>	$1 \text{ T} = 1 \text{ Wb/m}^2 = 1 \text{ Vs/m}^2$
Inductance	<i>henry</i>	$1 \text{ H} = 1 \text{ Wb/A} = 1 \text{ Vs/A}$
Frequency	<i>hertz</i>	$1 \text{ Hz} = 1/\text{s}$
Absorbed dose	<i>gray</i>	$1 \text{ Gy} = 1 \text{ J/kg}$
Plane angle	<i>radian</i>	$1 \text{ rad} = 1 \text{ m/m}$

⁵³ Translator's note for German readers: Das heißt *Lichtstärke*. Umseitig entspricht *luminous flux* dem *Lichtfluss* oder *Lichtstrom* und *luminance* der *Leuchtdichte*.

Quantity	Unit	Symbol and formula
Solid angle	<i>steradian</i>	$1 \text{ sr} = 1 \frac{\text{m}^2}{\text{m}^2}$
Luminous flux	<i>lumen</i>	$1 \text{ lm} = 1 \text{ cd sr}$
Luminance, illuminance	<i>lux</i>	$1 \text{ lx} = 1 \frac{\text{cd}}{\text{m}^2}$

For talking about inputs and results, knowing also the symbols and names of SI prefixes is beneficial. They cover 36 orders of magnitude:

Prefix	Name	Factor
h	<i>hecto-</i>	10^2
k	<i>kilo-</i>	10^3
M	<i>mega-</i>	10^6
G	<i>giga-</i>	10^9
T	<i>tera-</i>	10^{12}
P	<i>peta-</i>	10^{15}
E	<i>exa-</i>	10^{18}

Prefix	Name	Factor
d	<i>deci-</i>	10^{-1}
c	<i>centi-</i>	10^{-2}
m	<i>milli-</i>	10^{-3}
μ	<i>micro-</i>	10^{-6}
n	<i>nano-</i>	10^{-9}
p	<i>pico-</i>	10^{-12}
f	<i>femto-</i>	10^{-15}
a	<i>atto-</i>	10^{-18}

All the conversions featured in $\text{U}\rightarrow$ and $\text{x}\rightarrow$ are explained in alphabetical order below. Numeric values are either exact (printed on white background) or rounded to six significant digits for your orientation (your WP 43S uses more precise values wherever applicable). Commas are printed as radix marks for better visibility.

Softkey	Calculation	Remarks	Branch
${}^\circ\text{C} \rightarrow {}^\circ\text{F}$	$\times 1,8 + 32$	$\theta [{}^\circ\text{C}] = (T [\text{K}] + T_0)$ and $\tau [{}^\circ\text{F}] = (t [{}^\circ\text{R}] + T_0 \times 1,8)$	$\text{U}\rightarrow$
${}^\circ\text{F} \rightarrow {}^\circ\text{C}$	$- 32) / 1,8$		

Softkey	Calculation	Remarks	Branch
acre → m ²	× 4 046,86	This acre is based on the 'internat. foot', see below	
acre _{US} → m ²	× 4 046,87	This acre is based on the U.S. survey foot, see below	U → f A:
atm → Pa	× 1,013 25 × 10 ⁵	Atmosphere	U → F&p:
au → m	× 1,495 98 × 10 ¹¹	Astronomic unit	U → x:
barrel → m ³	× 0,158 987	(U.S.) barrel of oil, abbr. bbl	U → f V:
bar → Pa	× 100 000	1 mbar = 1 hPa	U → F&p:
Btu → J	× 1 055,06	British thermal unit	
cal → J	× 4,186 8	Calory	U → E:
carat → kg	× 0,000 2		U → m:
chǐ → m	/ 3	'Chinese foot' 1 市尺 := 10 cùn	U → x:
cùn → m	/ 30	'Chin. inch' 1 市寸 := 10 fēn	
cwt → kg	× 50,802 4	1 (long) hundredweight := 112 lbs	U → m:
dB → field ratio	10 ^{R_{dB}/20}	Decibel	
dB → power ratio	10 ^{R_{dB}/10}		U → ▼
fathom → m	× 1,828 8	1 fathom := 2 yards := 6 feet	
fēn → m	/ 300	1 市分 := 0,1 cùn	U → x:
field ratio → dB	20 lg(a ₁ /a ₂)	Also known as amplitude ratio	U → ▼
floz _{UK} → m ³	× 2,841 31 × 10 ⁻⁵	Fluid ounce	
floz _{US} → m ³	× 2,957 35 × 10 ⁻⁵		U → f V:
ft. → m	× 0,304 8	This is the so-called 'international foot' of 1959 1 foot := 12 inches	U → x:

Softkey	Calculation	Remarks	Branch
gl _{UK} →m ³	× 4,546 09×10 ⁻³	Gallon; 1 (Imperial) gallon := 4 quarts	[U] [f] [V:]
gl _{US} →m ³	× 3,785 42×10 ⁻³		
ha→m ²	× 10 000	Hectare (see m ² below)	[U] [f] [A:]
hp _E →W	× 746	Electric horsepower	
hp _M →W	× 735,499	So-called 'metric' horsepower (equivalent to PS in German)	[U] [P:]
hp _{UK} →W	× 745,700	British Imperial horsepower	
in.→m	× 0,025 4	1 inch := 1000 mil	[U] [x:]
in.Hg → Pa	× 3 386,39	Inch of mercury	[U] [F&p:]
J→Btu	/ 1 055,06	Joule	[U] [E:]
J→cal	/ 4,186 8		
J→Wh	/ 3 600		
kg → carat	/ 0,000 2		
kg→cwt	/ 50,802 4	1 t [(metric) ton] := 1000 kg	[U] [m:]
kg→oz	/ 0,028 349 5		
kg→lb.	/ 0,453 592		
kg → sh.cwt	/ 45,359 2		
kg → short ton	/ 907,185		
kg → stone	/ 6,350 29		
kg→ton	/ 1 016,05		
kg → tr.oz	/ 0,031 103 5		
lbf→N	× 4,448 22	Pound force	[U] [F&p:]
lbf·ft → Nm	× 1,355 82	Pound force times foot	[U] [▼]
lb.→kg	× 0,453 592	Pound; 1 lb := 16 ounces	[U] [m:]

Softkey	Calculation	Remarks	Branch
lǐ→m	× 500	'Chin. mile' 1 市里 := 15 yǐn	U→ x:
l.y.→m	× 9,460 73×10 ¹⁵	Light year	
m ² →acre	/ 4 046,86		
m ² →acre _{us}	/ 4 046,87	Square meter; 1 a [are] := 100 m ² ,	
m ² →ha	/ 10 000	1 ha [hectare] := 10 000 m ² , 1 km ² = 100 ha = 10 ⁶ m ²	U→ f A:
m ² →mǔ	× 0,001 5		
m ³ →barrel	/ 0,158 987		
m ³ →floz _{uk}	/ 2,841 31×10 ⁻⁵		
m ³ →floz _{us}	/ 2,957 35×10 ⁻⁵	Cubic meter; 1 liter := 1 dm ³ = 10 ⁻³ m ³ ,	
m ³ →gl _{uk}	/ 4,546 09×10 ⁻³	1 ml [milliliter] = 1 cm ³	U→ f v:
m ³ →gl _{us}	/ 3,785 42×10 ⁻³		
m ³ →qt.	/ 1,136 52×10 ⁻³		
mi.→m	× 1 609,344	1 mile := 1 760 yards	U→ x:
mmHg → Pa	× 133,322	See Pa below.	U→ F&p:
mǔ→m ²	/ 0,001 5	平方亩 (píng fāng mǔ), Chinese unit for areas, in particular for pieces of land	U→ f A:
m→au	/ 1,495 98×10 ¹¹		
m→chǐ	× 3		
m→cùn	× 30		
m → fathom	/ 1,828 8		
m→fēn	× 300		
m→ft.	/ 0,304 8		
m→in.	/ 0,025 4		
m→lǐ	/ 500		

Softkey	Calculation	Remarks	Branch
m→l.y.	/ $9,460\,73 \times 10^{15}$		
m→mi.	/ 1 609,344	Meter	[U↔] [x:]
m→nmi.	/ 1 852		
m→pc	/ $3,085\,68 \times 10^{16}$		
m → point	/ $352,778 \times 10^{-6}$		
m → survey foot _{us}	/ 0,304 801		
m→yd.	/ 0,914 4		
m→yīn	× 0,03		
m → zhàng	× 0,3		
nmi.→m	× 1 852	Nautical mile	
Nm → lbf·ft	/ 1,355 82	Newton meter	[U↔] [▼]
N→lbf	/ 4,448 22	Newton	[U↔] F&p:
oz→kg	× 0,028 349 5	Ounce	[U↔] [m:]
Pa→atm	/ $1,013\,25 \times 10^5$	Pascal; 1 hPa = 1 mbar	
Pa→bar	/ 100 000		
Pa → in.Hg	/ 3 386,39	For all real-world purposes (i.e. within experimental errors), torr are equivalent to millimeters of Hg. Possible differences are merely calculatory. Please check also the document linked below this table.	[U↔] F&p:
Pa → mmHg	/ 133,322		
Pa→psi	/ 6 894,76		
Pa → torr	/ 133,322		
pc→m	× $3,085\,68 \times 10^{16}$	Parsec	
point → m	× $352,778 \times 10^{-6}$	1 (typographical) point := 1/72 inch	[U↔] [x:]
power ratio → dB	$10 \lg(p_1/p_2)$		[U↔] [▼]
psi→Pa	× 6 894,76	Pound per square inch	[U↔] F&p:

Softkey	Calculation	Remarks	Branch
qt. \rightarrow m ³	$\times 1,136\ 52 \times 10^{-3}$	1 (<i>Imperial</i>) quart := 40 (<i>Imp.</i>) fluid ounces	
short cwt \rightarrow kg	$\times 45,359\ 2$	1 <i>short hundredweight</i> := 100 lbs	
short ton \rightarrow kg	$\times 907,185$	1 <i>short ton</i> := 2000 lbs	m:
stone \rightarrow kg	$\times 6,350\ 29$		
survey foot _{US} \rightarrow m	$\times 0,304\ 801$	1 U.S. <i>survey foot</i> := $\frac{1200}{3937}\ m$	x:
s \rightarrow year	/ 31 556 952		
ton \rightarrow kg	$\times 1\ 016,05$	1 <i>Imperial ton</i> := 200 cwt	m:
torr \rightarrow Pa	$\times 133,322$	See Pa above.	F&p:
tr.oz \rightarrow kg	$\times 0,031\ 103\ 5$	<i>Troy ounce</i>	m:
Wh \rightarrow J	$\times 3\ 600$	<i>Watt-hour</i>	E:
W \rightarrow hp _E	/ 746	Watt	
W \rightarrow hp _M	/ 735,499		P:
W \rightarrow hp _{UK}	/ 745,700		
yd. \rightarrow m	$\times 0,914\ 4$	1 <i>yard</i> := 3 <i>feet</i>	
yīn \rightarrow m	/ 0,03	1 市引 := 10 zhàng	x:
year \rightarrow s	$\times 31\ 556\ 952$	= $365,242\ 5 \times 24 \times 60^2$	
zhàng \rightarrow m	/ 0,3	1 市丈 := 10 chǐ	x:

The *British Imperial* units in this table cover what is most popular in the UK and its ex-colonies and territories. The Chinese units are a selection taken from the market standard *Shizhì* (市制) valid for mainland China.

Some more, really simple conversions (hence not included in U→) are listed here:

- $1 \text{ \AA} = 10^{-10} \text{ m}$ for atomic diameters and wavelengths,
- $1 \text{ barn} = 10^{-28} \text{ m}^2$ for cross sections in nuclear and particle physics (note the name),
- $1 \text{ tex} = 10^{-6} \text{ kg/m}$ for yarn density in textile industry,
- $1 \text{ muggeseggele} \approx 2 \dots 0.5 \mu\text{m}$ for Swabian precision mechanics.⁵⁴

 ...	Remarks (see also the OM, Section 2)
DEG→	Takes an integer or <i>real</i> ⁵⁵ x as an angular input in <i>decimal</i> or <i>sexagesimal degrees</i> , resp., and converts it to the current ADM.
D.MS→D	Takes an integer or <i>real</i> ⁵⁵ x as an angular input in <i>sexagesimal degrees</i> (formatted dddd.dd.msshh) and converts it to an <i>angle</i> in <i>decimal degrees</i> (corresponding to the old command H.MS→H).
D→R	Takes an integer or <i>real</i> ⁵⁵ x as an angular ... radians. ⁵⁶
D→D.MS	... sexagesimal degrees (corresponding to the old command H→H.MS).
GRAD→	... grades/gon ...
MULπ→	Takes an integer or <i>real</i> ⁵⁵ x as an angular ... multiples of π ... to the current ADM.
RAD→	... radians ⁵⁶ ...

⁵⁴ See <https://en.wikipedia.org/wiki/Muggeseggele> for more about this. Find further special and exotic units of the (mainly English speaking part of the) Western world in a 90-page guide of NIST (<https://physics.nist.gov/cuu/pdf/sp811.pdf>); this text covers many outdated and weird units, too, but also tells comprehensively how to deal with them. The angular conversions following are found therein as well.

⁵⁵ If x is neither integer nor *real* (i.e. neither DT 1 nor 2), error 24 will be thrown. Conversions of integer values to *angles* in *sexagesimal degrees* do not make real sense but are allowed nevertheless.



...

Remarks (see also the OM, Section 2)

R→D	Takes an integer or <i>real</i> ⁵⁵ x as an angular input in <i>radians</i> and converts it to <i>decimal degrees</i> . ⁵⁶
→DEG	Takes an integer (<i>DT 1</i>) or <i>real</i> (<i>DT 2</i>) x as an angular input in the current <i>ADM</i> and converts it to <i>decimal degrees, sexagesimal degrees, grades/gon, multiples of π</i> , or <i>radians</i> , respectively. ⁵⁶
→GRAD	If x is a tagged <i>real</i> (<i>DT 4</i>), on the other hand, this information is used in conversion (e.g. if $x = 1.5\pi$ then →GRAD will return 300° regardless of current <i>ADM</i>).
→MULπ	If x is neither of <i>DT 1</i> , 2, nor 4, error 24 will be thrown ('illegal input data type for this operation').
→RAD	

Angular output is tagged always.⁵⁷

⁵⁶ Note that *real* angles given in *radians* cannot represent full circles (or multiples of them, as well as simple fractions of π like $\pi/2$, $\pi/3$, $\pi/4$, etc.) exactly but with an accuracy of 34 digits 'only'. If you want to avoid rounding errors caused by that but must keep π in play, *multiples of π* may be a better choice for the *angular display mode* here.

Note that large numeric inputs in trigonometric functions are reduced to values between $-\pi$ and $+\pi$ before calculating (as mentioned in Section 2 of the OM). Such reductions may easily introduce inaccuracies when *radians* are used – all other angular units are uncritical in this aspect.

⁵⁷ Solution of footnote 50: In this very simple model, a volume of 150 (*l.y.*)³ is available for each star in our galaxy on average, i.e. a sphere with a diameter of 6 *l.y.* So in zeroth order approximation, nothing spectacular is expected.

For estimating the consequences of a galactic collision a bit more realistically, take the orientation of both galaxies relative to each other into account when they collide. And star density is highest in the galactic centers and decreases significantly from there outwards. Furthermore, there used to be giant amounts of free gas (although very thin) and dust between the stars.

SECTION 3: CALLING AND EXECUTING OPERATIONS

As mentioned at the beginning of *Section 2* and in the *OM*, the number of *items* featured on your *WP 43S* is far too large to fit them on the keyboard. Hence, there are several ways to call such an *item*.

You know how to call *items* appearing on the keyboard or in *menus* (including *catalogs*). In *Section 6* of the *OM*, you have learned about storing *items* in user *menus* and/or assigning them to specific locations on your *WP 43S*. There is one more way you can use for calling and executing operations: take **[XEQ]** followed by the *name* of the operation typed in *AIM*.

In the two chapters following thereafter, we will list all the functions requiring parameters and those changing *data types*.

Using XEQ for Executing Operations

Instead of picking an operation from a *menu* or *catalog*, you can also call it by *name* using **XEQ** as follows:

1. Press **[XEQ]**.
2. Press **[α]**. You are in *AIM* then; see *Section 2* of the *OM* for the *virtual keyboard* applying in this mode.
3. Key in the *name* of the function wanted. Case may be important, subscript or superscript is not.
4. Press **[ENTER \uparrow]**. Your input will be checked – if the operation specified exists, ...
 - a. it will be checked for required parameters (cf. overleaf);
 - i. if true, you will be prompted for these parameters. Then the function will be executed. End.
 - ii. else the function will be executed. End.
 - b. else error 7 (**No such function**) will be thrown (see *App. C*). End.

Operations Requiring Trailing Parameters

Many functions require at least one trailing (numeric or alphanumeric) parameter specifying what they shall do precisely (see the OM, Section 1). The following three lists summarize these operations:

Operations requiring one trailing parameter	Numeric parameter	Alpha par.
AGRAPH CONVG? DEC DSE DSL DSZ INC INPUT ISE ISG ISZ KEY? KTyp? PUTK RCL RCLCFG RCLS RCL+ RCL- RCLx RCL/ RCL [†] RCL [‡] STO STOCFG STOS STO+ STO- STOx STO/ STO [†] STO [‡] t [‡] VIEW x [‡] x=? x≠? x≈? x<? x≤? x≥? x>? y [‡] z [‡] aLEN [?] aPOS? aRL aRR aSL a→x	Register number	Variable name
ALL ENG FIX GAP RDP RSD SCI SDL SDR	Number of decimals	
ASR MASKL MASKR RL RLC RR RRC SL SR WSIZE	Number of bits	
BACK CASE SKIP	Number of program steps	
BC? BS? CB FB SB	Bit number	
BestF	Fit model code	
CF FC? FC?C FC?F FC?S FF FS? FS?C FS?F FS?S SF	User flag number	System flag name
CONST	Constant number	
DSTACK	Number of stack registers	
ERR MSG	Error number	
f'(x) f''(x) GTO LBL LBL? PGMINT PGMSLV XEQ Π _n Σ _n		Program label
GTO.	Number of program step	Label

Operations requiring one trailing parameter	Numeric parameter	Alpha par.
INDEX MVAR M.DIM M.EDIN SOLVE VARMNU ∫		Variable name
LocR	Number of local registers	
PAUSE ⌂DLAY	Number of ticks	
RM ⌂MODE	Mode number	
SIM_EQ	Number of unknowns	
TDISP	Time format number	
TONE	Tone number	
→INT	Base	
⌂CHAR	Character code	
⌂TAB	Column number	
⌂#	Byte	

Note for any command **XYZ** requiring one trailing parameter, you can enter

XYZ → X

and it will fetch its parameter from **X** like a good old *RPN* command instead.

Operations requiring two trailing parameters	First parameter	Second parameter
ASSIGN	<i>Item</i>	Sequence of keystrokes
KEYG KEYX	Key number (1 ... 18)	Program label

Operation requiring four trailing parameters	1 st to 4 th parameter
⌘	Name of stack register

Operations Changing Data Types

Most functions will return data of the same type they operate on. Some, however, will change the *DT* of the contents of the lowest *stack register(s)* regardless of specific input values, as mentioned at various locations in the *OM*. These operations are collected here:

Input <i>DT</i>	Operation(s)	Output <i>DT</i>	Output <i>registers</i>
1	1/x $\sqrt[3]{x}$ AGM ALL cos ENG erf erfc e ^x FIX f' f'' gd gd ⁻¹ J _y (x) LN LN β LN Γ LOG ₁₀ LOG ₂ LOG _{xy} MANT POISS... SCI sin tan W _m W _p W ⁻¹ \sqrt{y} $\beta(x,y)$ Γ_{xy} γ_{xy} $\Gamma(x)$ Δ% →REAL % %MRR %T %Σ %+MG \sqrt{x} as well as all unit conversions and all orthogonal polynomials	2^{58}	X
	→POL →REC		X, Y
	SLVQ		X, Y, Z
	f'(x) f''(x) SOLVE		X, Y, Z, T
	all angular conversions		X
	→H.MS		X
	J→D →DATE		X
	x→α		X
	M.GET M.NEW		X
	→INT		X

⁵⁸ The functions printed on yellow background will return *long integers* (*DT* 1) wherever possible.

Input DT	Operation(s)	Output DT	Output registers
2	AND CEIL DATE→ DAY D→J EXPT FLOOR IDIV IDIVR IP MONTH NAND NEXTP NOR NOT OR ROUNDI SIGN WDAY XNOR XOR YEAR ±∞?	1	X
	DECOMP	1	X, Y
	SLVQ	2 or 3	X, Y, Z
	RE→CX	3	X
	arccos arcsin arctan and all angular conversions	4	X
	→H.MS	5	X
	x→DATE →DATE	6	X
	x→α	7	X
	M.GET M.NEW	8 or 9	X
	→INT	10	X
3	ABS CROSS IM RE x 4	2	X
	CX→RE	2	X, Y
	SLVQ	2 or 3	X, Y, Z
4	cos sin tan	2	X
5	→HR	2	X
6	DAY D→J MONTH WDAY YEAR	1	X
	DATE→	1	X, Y, Z
	→REAL	2	X
7	α→x	1	X

Input <i>DT</i>	Operation(s)	Output <i>DT</i>	Output <i>registers</i>
8	M.DIM?	1	X, Y
	DET DOT ENORM M	2	X
	$\Sigma+$	2	X, Y, statistic registers
	V ₄	4	X
9	M.DIM?	1	X, Y
	ENORM	2	X
	DET DOT M	3	X
	ABS IM RE ROUND I x	8	X
10	SIGN	1	X
	e ^x LN LOG _x y →REAL	2	X
	x→α	7	X

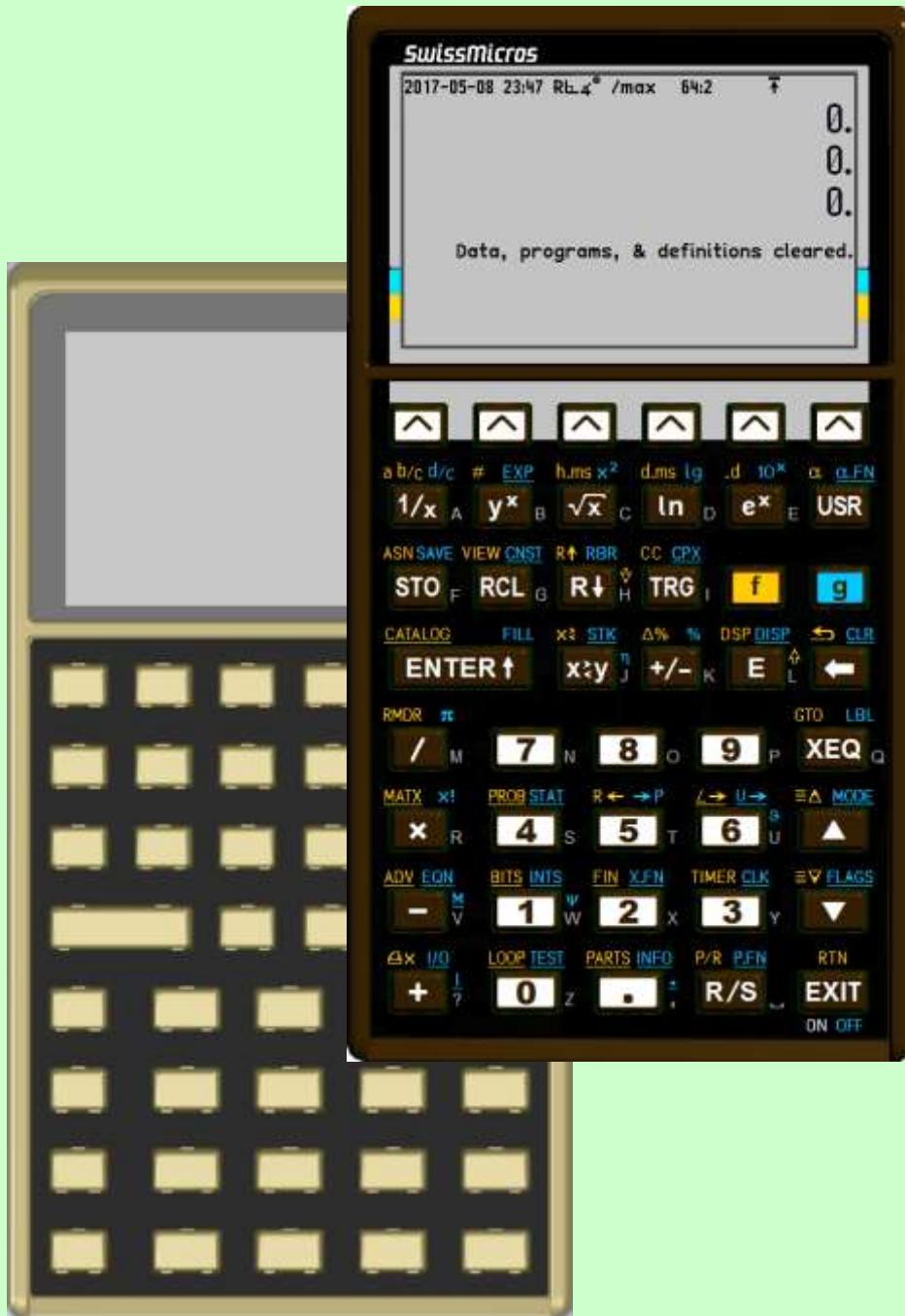
APPENDIX A: HARDWARE

- Dimensions: wedge-shaped 77 mm × 144 mm × 13 mm or 8 mm (see a picture on next page).
- Mass: 180 g incl. battery.
- LCD: monochrome high contrast (14:1) transreflective memory display with an active area of 58.8 mm × 35.3 mm, 400 × 240 quadratic pixels, and dot pitch 147 µm.
- Processor: low power ARM Cortex-M4F (*STMicroelectronics STM32L476*) incl. real-time clock running at 25 MHz on battery power or 80 MHz when power is supplied through *USB* (see below; look up <https://www.st.com/en/evaluation-tools/32l476gdiscovery.html> for the development board used by *SwissMicros*).
- Memory: 1 MB *FM*, 128 kB *RAM* (see *App. B* on pp. 160ff), 32 MB additional external *FM* on a *QSPI* chip; user *RAM* is 96 kB, user *FM* is 32 MB. xxx
- Power supply: 3 V by one *CR2032* lithium coin cell (battery life up to 3 years, non-rechargeable); optionally powered through *USB* port (good battery shall be installed); typical average currents drawn: power on & busy: 4.2 mA; idle: 0.1 mA; power off: 3 µA.
- Buzzer: piezo-electric with 4 kHz resonance frequency, tunable from 1 Hz up to 20 kHz in steps of 1 Hz.
- Connectivity: infrared port compatible with *HP-82240A/B* printers,
micro-USB-B port connecting to a host computer as external mass storage device.
- Keyboard overlays: Three short slots on either side of the keyboard are provided in the calculator frame for easily fixing your overlay sheets with your personal layouts printed on. See *App. F* for more (on p. 217). Look up *Section 6* of the *OM* for inspiration.



Seven pictures of the hardware are displayed here and on the pages following. The default keyboard layout as delivered by SwissMicros for the *DM42* (bottom right) and the front views on next page are printed approximately to scale. Find the printed circuit board (*PCB*) displayed thereafter.







See here the internals.
Unfasten two bolts at
the top of the calculator
backside to get there.

To access the keyboard side of the *PCB* carrying the switching domes, carefully release the *LCD* connection Ⓐ, then unfasten the two Philips bolts Ⓑ pictured best below.

Note all these operations are at your own risk.



This picture shows the *PCB* of an early *DM42* of spring 2017. *RESET* is the top left button here.



Please use the following link to find a discussion of the various hardware components used on the *DM42 PCB* (of February 2018) as pictured on previous page: <https://www.hpmuseum.org/forum/thread-10143.html>.

APPENDIX B: MEMORY MANAGEMENT

Memory Map

0x08 00 00 00 – 0x08 0F FF FF	1024 KiB <i>FM</i> for the <i>WP 43S</i> program and its <u>constant</u> data (incl. fonts, <i>menus</i> , messages, etc.) – this memory is in the <i>CPU</i> .
0x10 00 00 00 – 0x10 00 7FFF	32 KiB system <i>RAM</i> for the headers ⁵⁹ of the 112 global <i>registers</i> and for other global <i>WP 43S</i> variables like expandable <i>submenus</i> (cf. pp. 115ff).
0x20 00 00 00 – 0x20 01 FF FF	96 KiB user <i>RAM</i> , i.e. space for user programs, <i>registers</i> , <i>flags</i> , <i>menus</i> , and variables. STATUS and MEM? tell about the free space there.
0x90 00 00 00 – 0x90 1F CF FF	2036 KiB <i>QSPI</i> (= <i>Quad Serial Peripheral Interface</i>) <i>FM</i> for the operating system
0x90 1F D0 00 – 0x90 1F DF FF	4 KiB <i>QSPI</i> user <i>FM</i>
0x90 1F E0 00 – 0x90 1F FF FF	8 KiB <i>QSPI</i> <i>FM</i>
0x90 20 00 00 – 0x90 7F FF FF	6144 KiB <i>FM</i> for the <i>FAT</i> (= <i>File Allocation Table</i>) system, where backups and snapshots are stored. This part of the memory of your <i>WP 43S</i> is visible on your computer and may accessed by it as long as 'activate <i>USB disk</i> ' is selected.

Of these 9 1/8 MB, we printed the sections writable by the *WP 43S* system only (some 1 MB) on yellow, the user-writable sections on green. Within the 96 KiB of user *RAM*, global and local *registers*, *flags*, variables, and statistical sums use its beginning part and user programs occupy its end.

⁵⁹ Containing variable pointers (see overleaf).

Data Types

There are ten *DTs* you know from Section 2 of the OM. Some more had to be defined for internal use, e.g.:

- 7-character strings for all kinds of *labels*, also including *names* of commands and all other *menu items*; this is the reason why such *names* are confined to 7 characters,
- system integers in the range of $\pm 2\ 147\ 483\ 648$ (i.e. 32 *bits*),
- *flag* words for storing 144 (i.e. 112 global plus 32 local) *user flags* and 112 *system flags* in 256 *bits* in total,⁶⁰
- two *DTs* for two kinds of *menus*,
- one *DT* of variable length for storing *configurations* (see RESET, STOCFG, and RCLCFG),
- another *DT* for *expressions* in EQN (see Section 4 of the OM), and
- two more for program steps and routines (see Section 3 of the OM).

Generally, data – outside of routines – can be contained in (global or local) *registers* or named variables.

A 4-byte *register header* is specified for each of the 112 global *registers*, containing a pointer to the data contained in this register⁶¹, the *internal DT* (see overleaf), and some data information.⁶²

For each (sub-) routine level, 12 *bytes* are reserved in RAM ensuring proper return. For every (sub-) routine requiring *n* local *registers*, 4 + 4 *n bytes* are allocated in addition, with the first 4 *bytes* for the 32 local flags.

⁶⁰ We need far less than 112 *system flags* so far.

⁶¹ I.e. to the first memory block number of the *register* content. This allows access to $65\ 535 \text{ blocks} \times 4 \text{ bytes / block} = 262\ 140 \text{ bytes} = 256 \text{ kB} - 4 \text{ bytes}$. A special value is reserved for the NULL pointer.

⁶² Like display base for a *short integer* (2 - 16), sign for a *long integer* (0: zero, 1: negative, 2: positive), angular unit for an *angle* or *real number* (0: *degree*, 1: *gradian* or *gon*, 2: *radian*, 3: *multiple of π* , 4: *sexagesimal degree*, 5: no angle = plain *real number*).

An 8-byte *named variable header* is specified for each named variable, containing a pointer to the data contained in this variable, the *internal DT*, some data information,⁶² a pointer to the name of the variable, and its length.

The data themselves are then stored the following way:

DT number and meaning		Size [bytes]
1	0	Z Long integer ⁶³
2	1	R Real number ⁶⁴ (real34)
3	2	C Complex number, always stored in rectangular notation (complex34)
4	1	Angle ⁶⁵
5	3	Time ⁶⁶
6	4	Date ⁶⁷
7	5	Alphanumeric string (a.k.a. text string)

⁶³ This DT is for number theory kind of problems. 4 bytes (= 1 block) heading are for the size (in blocks) of the integer following. 1 block following allows for signed integers up to $2^{31} \approx 2 \times 10^9$. Size is increased in 1-block steps when required. Max. long integer size is 416 bytes equivalent to 1001 decimal digits. Look up the display limits further below.

⁶⁴ Deviating from the WP 34S, standard *reals* on your WP 43S feature 128 bits and 34-digits precision. See the chapter after next.

⁶⁵ A tagged *angle* is stored as a *real number* but with a special header (cf. previous page).

⁶⁶ Internally, a *time* or time interval is stored as a *real number* of *seconds*, just with a specific header. A day corresponds to 86 400 s, a year to 31 556 952 s. The format allows for expressing intervals of some 100 billion years with *femtoseconds* precision.

⁶⁷ A *date* is stored as *real number* of *seconds* passed since -4713-01-01 12:00:00 (noon). This is date and time zero for *Julian Day Number* counting.

⁶⁸ The length of a *text string* in *blocks* is $L = (j + 2k + 1) / 4$, with j being the number of characters contained requiring only 1 byte and k being the number of characters needing 2 bytes (bit 15 will be set always for them). There is a 1-block header indicating L and one trailing byte containing 00. The minimum size of such a string is 2 blocks = 8 bytes, minimum size increment is 1 block.

DT number and meaning			Size [bytes]
8	6	Real matrix (of n rows and m columns) ⁶⁹	$4 + n \times m \times 16$
9	7	Complex matrix (of n rows and m columns) ⁶⁹	$4 + n \times m \times 32$
10	8	Short integer ⁷⁰	8
11	9	Configuration (as stored by STOCFG)	$4 + M^{71}$
12	5	Label (or name – up to 7 characters) ⁶⁸	$1 + j + 2 k = 2 - 15$
13		Constant ⁷²	$4 + 0 = 4$
14		System and user flags (112 + 144 flags)	32
15		Extended precision real (39 digits, for internal use only, not exposed to the user) ⁷³	$4 + 32 = 36$
16		System integer (for internal use only)	4
17		User-created menu (limited to 1 view)	$4 + 18 \times 7 \times 2 = 256$
18		Predefined menu (featuring n views)	$4 + n \times 18 \times 14$
19		Program step, see pp. 178f	See pp. 178f
20		Program, containing n program steps	$4 + M$
21		Expression (for members of EQN), may be stored as text string (DT 7) as well	$4 + M$
22		Directory (proposed by P. 2012-12)	$4 + M$

In programs, *text strings* are shorter: there is just 1 *byte* heading and no trailing 00. See also the last chapter of this App.

⁶⁹ 2 bytes for n , 2 bytes for m , then follow the individual matrix elements row by row, taking either 2 or 4 bytes each.

⁷⁰ This DT is for computer science problems.

⁷¹ The size will vary with the number of user assignments being part of the configuration (see Section 6 of the OM).

⁷² A pointer to the constant is sufficient here, regardless of its precision.

⁷³ There are even longer (i.e. more precise) *reals* used in internal computations. See further below.

The *DTs* 7 - 9, 11, and 19ff are of ‘infinite’ size limited by available memory (**M**) only. The size of each individual object is fixed though.

As mentioned above, any object of any *DT* will take one storage space only: one *register* or one variable. In consequence, *register* lengths in your *WP 43S* may vary considerably. You do not have to bother – the operating system of your *WP 43S* will take care of all the necessary administration.

Thus, the amount of *RAM* required for data storage is not fixed. Data and programs allocate their memory from the same large pool – monitoring the free space available by *MEM?*, you may experience differences following e.g. changing contents of your *stack*.

Statistical Summation Registers

Your *WP 43S* features a block of 23 special *registers* for accumulating and storing statistical sums (like the 14 summation *registers* of the *WP 34S* and *WP 31S* before) plus 4 for x_{\min} , x_{\max} , y_{\min} , and y_{\max} . These statistical *registers* neither overlap nor interfere with any *GP registers* unlike they did on *HP*’s pocket calculators. Their contents can be recalled individually using their names (cf. pp. 56 and 88f).

And like on our *WP* calculators before, this block of *registers* is allocated from the pool of free memory available as soon as the first statistical data are entered via **[Σ+]** or **[Σ-]**; it is de-allocated and its memory is returned to the pool space by **[CLΣ]**.

Each statistical register requires 60 *bytes*. Hence, this block occupies $27 \times 60 = 1620$ *bytes*. We need them twice to allow for UNDO.

SAVE and LOAD...

SAVE stores the calculator *configuration*, all *flags* and *registers* (including the *summation registers*, if allocated), and all programs

present in **RAM** in a file in the **FAT system in FM**. Thus, this storage will survive **RESETs**, even hard ones via the RESET button, and is battery fail safe.⁷⁴ So you can use it as on-board backup of your programs and data, for instance.

A second SAVE will overwrite the first one.

The various flavours of LOAD allow resuming calculator operation after such events, according to your requests:

- LOADSS recovers the *configuration* only, similar to RCLCFG,⁷⁴
- LOADΣ recalls just the *summation registers*,
- LOADR all the numbered *registers*,
- LOADP the programs, and
- LOAD recalls the entire data set stored by SAVE at once.

Turn to the *IOI* for more information about these individual commands.

Range of Real Numbers

Your *WP 43S* could calculate with *real numbers* of more than 12 000 orders of magnitude. Within a range of $10^{-6143} \leq |x| < 10^{+6145}$, it computes with 34 digits precision. Its software is based on the *decNumber library* supporting arbitrary precision *BCD* numbers.

As mentioned at some places in the *IOI*, internal computations are usually carried out with 39 digits.⁷⁵ Results should be accurate within

⁷⁴ Note STOCFG stores the calculator *configuration* in a *register* or variable, i.e. in **RAM** instead.

⁷⁵ Actually, 39 digits are the minimum; some modulo calculations for trigonometric functions are performed with more than a thousand digits to avoid cancellation.

Feel free to test any calculator you may use with this: $729^{33.5} / 3^{201} - 1$. Your *WP 43S* will return precisely 0.

There is a quasi-standard established to find out about processors, firmware, and accuracy of calculators – compute $\arcsin\{\arccos[\arctan(\tan\{\cos[\sin(9^\circ)]\})]\}$

$\pm 1 \times 10^{-33}$ per operation (e.g. **n** **1/x** **2n** **x** returns a plain 2 for all primes < 500 – except **7** **1/x** **1** **4** **x** **2** **-** returning 1×10^{-33} , and for 67 and 83 the respective results are -1×10^{-33}). Note that rounding errors due to calculating with a finite number of digits will accumulate (as other errors do) – see two examples in the footnote here.

Results $|x| < 10^{-6176}$ are set to zero. For results $|x| \geq 10^{6145}$, error 4 or 5 will appear unless SPCRES is set (see App. C).

All these effects are caused by the **internal representation of reals**: Standard floating point numbers are stored on your WP 43S in sixteen bytes using an internal format coarsely following decimal128 packed coding,⁷⁶ though with some exceptions:

- Real zero is stored as integer zero, i.e. all bits cleared.

(although this formula is mathematical nonsense). An ideal brainless number cruncher featuring infinite internal precision would return exactly 9° or equivalent (see e.g. [https://www.wolframalpha.com/input/?i=arcsin\(arccos\(arctan\(tan\(cos\(sin\(pi/20\)\)\)\)\)\)](https://www.wolframalpha.com/input/?i=arcsin(arccos(arctan(tan(cos(sin(pi/20)))))))).

Real calculators (computing with a finite number of digits) deviate for obvious reasons. Your WP 43s returns

- 8,999 999 999 999 999 999 999 999 999 937 535 = 9 – $6,246 \cdot 10^{-29}$ for $\text{arcsin}\{\text{real}(\text{arccos}\{\text{real}[\text{arctan}(\tan\{\cos[\sin(9^\circ)]\})]\})\}$.

If you are interested how other calculators have performed in that test, look at <http://www.rskey.org/~mwsebastian/miscpri/results.htm>.

Another – far simpler – test discussed in the internet works as follows: Enter 1,000 000 1 and then press **x²** just 27 times. Your WP 43s will return

- 674 530,470 741 084 559 382 689 **184 727 772 2**.

These are the most precise results known for a pocket calculator so far (the WP 34S with DBLON and Free42 concur, the latter for this last test only). Nevertheless, only the first 25 digits of this output are correct! Calculating with unlimited precision returns

- 674 530,470 741 084 559 382 689 **178 029 746 8** instead for the first 34 of the 10^9 digits of the entire result (you get > 1600 decimals after 8 presses of **x²** already).

Please take this information into account when assessing small deviations or many decimals returned by your WP 34S, in particular after long calculations. It will return up to 34 digits always but not all of them are guaranteed being true.

⁷⁶ It comes close to what is called *quadruple (precision)* in this text about floating point arithmetic: <https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF>.

Find out about decimal128 in https://en.wikipedia.org/wiki/Decimal128_floating-point_format.

- The *significand* (not mantissa⁷⁷) of a *real number* is encoded as pure integer in eleven groups of three digits. Each such group is packed into ten bits straight forward, meaning e.g. $555_{10} = 10\ 0010\ 1100_2 = 22B_{16}$ or $999_{10} = 11\ 1110\ 0111_2 = 3E7_{16}$. So the 33 rightmost decimal digits of the *significand* take its least significant 110 bits. Trailing zeros are omitted, so the *significand* will be right adjusted.
- The most significant (128th) bit takes the sign of the *significand*.
- The remaining 17 bits are used for the exponent and the leftmost digit of the *significand*. Of those 17, the lowest twelve are reserved for the exponent (< 4096). For the top five bits (below the sign bit) it becomes complicated – following the standard *IEEE 754*. If these five bits read...
 - $00ttt$,
 - $01ttt$, or
 - $10ttt$ then ttt takes the leftmost digit of the *significand* ($0 - 7$), and the top two bits will be the most significant bits of the exponent;
 - $11uut$ then u will be added to 1000_2 and the result (8_{10} or 9_{10}) will represent the leftmost digit of the *significand*.
If uu reads 00 , 01 , or 10_2 then these will represent the two most significant bits of the exponent. If it reads 11_2 , there are bit patterns specified for encoding special numbers (see below).

Thus, the maximum absolute value of the stored exponent is $10\ 1111\ 1111\ 1111_2 = 12\ 287_{10}$. For reasons becoming obvious below, 6176_{10} must be subtracted from the stored value to get the true exponent of the floating point number represented. Thus and since $12287 - 6176 + 34 = 6145$ as well as $-6176 + 34 = -6142$, *DT2* can support 34-digit numbers within $10^{-6143} \leq |x| < 10^{+6145}$.

Rewarding your patience so far, we will show you some illustrative **examples** of the encoding in your *WP 43S* instead of telling you more theory. *SE* stands for *stored exponent* in the following:

⁷⁷ The *significand* does not contain a radix mark.

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	SE
1.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0010 0000 1000 00	6176
-1.	a2 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		1010 0010 0000 1000 00	6176
111.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 6f		0010 0010 0000 1000 00	6176
111.111 (111 111×10 ⁻³)	22 07 40 00 00 00 00 00 00 00 00 00 00 01 bc 6f	06f 06f	0010 0010 0000 0111 01	6173
-123.000 123 (-123 000 123 ×10 ⁻⁶)	a2 06 80 00 00 00 00 00 00 00 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0000 0110 10	6170
9.99×10 ⁹⁹ (999×10 ⁹⁷)	22 20 40 00 00 00 00 00 00 00 00 00 00 00 03 e7		0010 0010 0010 0000 01	6273
1×10 ⁻⁹⁹	21 ef 40 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0001 1110 1111 01	6077
-1×10 ⁻⁶¹⁴³	80 08 40 00 00 00 00 00 00 00 00 00 00 00 00 01		1000 0000 0000 1000 01	33

You will lose one digit precision if you divide 10^{-6143} by 10 and one more for each such division following. At 10^{-6176} , only one digit will be left in the *significand*, stored as 1.

Divide this by 1.999 999 999 999 999 999 999 999 999 999 and the result will remain 10^{-6176} in default rounding mode (and in RM 1, 2, 3, and 5, see the command RM). Divide it by 2 instead and the result will become zero.

Let us look at the upper end of our numeric range now:

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	SE
9.999 999 999 999				
999 999 999 999	77 ff be 7f	9 3e7 3e7		
999 999 999 ×10⁶¹⁴⁴	9f e7 f9 fe	3e7 3e7 3e7	0111 0111	
(= 9 999 ... 999 999 ×10 ⁶¹¹⁰)	7f 9f e7 f9 fe 7f 9f e7	3e7 3e7 3e7 3e7 3e7 3e7	1111 1111 10	12 286

This *real number* (featuring 34 times the digit 9) is the maximum which can be keyed in directly. It will be displayed as ∞ in *startup default* format; $9.999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\times10^{6144}$ will be only unveiled by SHOW (see p. 70). The greatest legal *significand* on your WP 43S is $9\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999 = 10^{34} - 1$.

Additionally, your WP 43S features three ‘special reals’:

Floating point 'number'	Hexadecimal value stored	Top 8 bits in binary notation
∞	78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0111 1000
$-\infty$	F8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1111 1000
NaN	7C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0111 1100

Neither an exponent nor a *significand* is applicable here. Note that **these three ‘special reals’ (∞ , $-\infty$, and NaN) may be legal outputs of your WP 43S if SPCRES is set** – no error will be thrown then. NaN (i.e. ‘Not a Number’) covers poles as well as regions where a function result is not defined at all (see the corresponding entry in Section 5 of the OM and examples in next chapter). **∞ and $-\infty$ may be also legal numeric inputs on your WP 43S.**

Remember not every 34-digit number displayed will be true to 34 digits – cf. footnote 75 on pp. 165f. And errors accumulate as explained in

footnote 45 on p. 130. As mentioned above, some internal calculations are executed in “*internal high precision*”, employing even more digits than 34: it may mean 39, 72, 75, or even > 1000 digits in special cases.

- ☞ Rounding mode settings (see RM) may affect results of high precision calculations!

Limitations

Maximum numeric input for *data type* ...

- 1: $\pm 10^{1001}$ (if your patience will suffice for the input) or $\pm 2^{3326}$ (if you are less patient)
- 2: $\pm 9.999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999 \times 10^{\text{RANGE}-1}$ (absolute maximum for **RANGE** is 6145)
- 3: $\pm 9.999\dots \times 10^{\text{RANGE}-1}$ for either part
- 4: $\pm 9.999\dots \times 10^{\text{RANGE}-1}$ in arbitrary *ADM*, so the actual absolute maximum is $9.999\dots \times 10^{\text{RANGE}-1} \pi$
- 5: $\pm(10^{19} - 1) \text{ s} \approx \pm 317 \times 10^9 \text{ years}$ (with 10^{-15} s precision).
- 6: dates xxx
- 8: $\pm 9.999\dots \times 10^{\text{RANGE}-1}$ for each matrix element
- 9: $\pm 9.999\dots \times 10^{\text{RANGE}-1}$ for either part of each matrix element
- 10: For arbitrary word size *n* and unsigned mode, $x_{\max} = 2^n - 1$ (i.e. $\sim 1.8 \times 10^{19}$ for *n*=64); for signed modes, $x_{\max} = 2^{n-1} - 1$.

Internal limits:

- PRIME? works like the other binary tests described in the OM. Above 3 317 044 064 679 887 385 961 981 (i.e. 3.3×10^{24}), however, NEXTP cannot find primes anymore but ‘probable primes’ only (cf. p. 59). If you want to apply NEXTP or PRIME? above this limit, consult appropriate reference literature.⁷⁸

⁷⁸ See https://en.wikipedia.org/wiki/Miller%20-%20Rabin_primality_test for a start (also available in other languages). Whatever is a prime will be confirmed by PRIME?

- Trigonometric functions actually operate between $+\pi$ and $-\pi$ for *radians* exclusively. The necessary modulo 2π reduction ensures that these functions return correct 34-digit results for the entire legal range of angles. *ADMs* other than *radians* are easier to deal with.

Maximum numeric output for *data type* ...

- **1:** $\pm 10^{1001}$ with full 1001-digit precision (though you can see and read up to 296 digits only of such numbers – cf SHOW, p. 70).
- **2, 3, 8, and 9:** The maxima are as specified for input above. Any (intermediate) result exceeding $-10^{\text{RANGE}} < x < 10^{\text{RANGE}}$ will be displayed as $-\infty$ or ∞ , and any (intermediate) result within $-10^{-\text{RANGE}} < x < 10^{-\text{RANGE}}$ will be displayed as '0.'.
But any (intermediate) result within $-10^{-6176} < x < 10^{-6176}$ will be assessed and displayed as 0. (cf. previous chapter). Any (intermediate) result exceeding $-10^{6145} < x < 10^{6145}$ will be assessed as $-\infty$ or ∞ , respectively, and will then be treated according to the system settings at display time: if SPCRES is set, it will be shown as $-\infty$ or ∞ , else an overflow error will be thrown.
- **4:** For angular conversions, the maxima are as specified for input above. The functions ARCSIN, ARCCOS, and ARCTAN return values between $-\pi$ and π (or their equivalents) only.
- **5:** The maximum is as specified for input above.
- **6:** **xxx**
- **10:** The maxima are as specified for input above.

– a composite may be falsely assessed as being prime with a probability of 9×10^{-16} . With some care and a lot of time, NEXTP can find ‘probable primes’ for *long integers* up to some 6.3×10^{1001} , and PRIME? will check them and their neighbouring numbers (*USB*-power recommended) but you cannot see most of their 1002 digits directly. On the other hand, you can get a 296-digit prime within some 40 *seconds* – and read it entirely on the screen of your *WP 43S* using SHOW.

Further output limitations for *data type* ...

- 11: *Configurations* cannot be displayed. Thus, if any register or variable containing such data is on screen, **Configuration data** is shown instead of its content.

What Happens when Changing Word Size?

WSIZE affects *short integer* contents of the allocated *stack registers* and **L** exclusively: Reducing word size will truncate all *short integer* values therein down to the new size. Increasing word size will fill empty bits in the significant side of each *short integer* concerned, up to the new size specified.

ATTENTION: Increasing word size will just add empty bits. Thus, a negative *short integer* will immediately become positive then. There is no automatic sign extension! If you want it, take care of it yourself.

All other memory content of your WP 43S will stay as it is. This differs from the implementation as known from the HP-16C, where all registers were remapped on word size changes.

When *short integers* are recalled from memory which are longer than current word size, these integers will be truncated during recall. When *short integers* are recalled from memory which are shorter than current word size, these integers will be filled during recall.

Special Results

Throughout this chapter, SPCRES is presumed to be set. Thus, infinities and non-numeric results are legal – no error messages will be thrown if such results happen to occur (cf. p. 169).⁷⁹

The following monadic functions will return either ∞ , $-\infty$, or **NaN** under the conditions stated below:

Input x	Operation(s)	Output for \mathbb{R} lit
-1.	artanh	
0 or 0.	ln , lg , $\text{lb } x$	$-\infty$
0.	$\frac{1}{x}$	
1.	artanh	∞
0 or 0.	$\Gamma(x)$	
$\text{Re}(x) < 1$	arcosh	
$ \text{Re}(x) > 1$	arccos , arcsin , artanh	NaN
$\pm 90^\circ$ or equivalents in other ADM	\tan	

And the following monadic functions operate also on infinities:

Input x	Operation(s)	Output for \mathbb{R} lit
$-\infty$	x^3 , $\sqrt[3]{x}$	$-\infty$
	arctan	-90° or equivalents
	\tanh	-1.
	$\frac{1}{x}$, e^x , 10^x , 2^x , sinc	0.
	x^2 , arsinh	∞

⁷⁹ Results were crosschecked against *WP 34S* wherever possible. Additionally, *Wolfram Alpha* was used for checking results with finite arguments. Where deviations are observed we are confident your *WP 43S* returns the correct results.

Input x	Operation(s)	Output for ℝ lit
$-\infty$	arcosh	NaN
$-\infty \leq x < 0$	ln , lg , $\text{lb } x$	NaN
∞	$\frac{1}{x}$, sinc	$0.$
	\tanh	$1.$
	arctan	$90.^{\circ}$ or equivalents
	ln , e^x , x^2 , \sqrt{x} , lg , 10^x , $\text{lb } x$, x^3 , $\sqrt[3]{x}$, \sinh , \cosh , arsinh , arcosh	∞
$-\infty$ or ∞	\cos , \sin , \tan , artanh	NaN

For dyadic functions, we combined the respective tables:

Input y	x	Op.(s)	Output for ℝ lit
∞	$x \neq -\infty$	+	∞^{80}
	$x \neq \infty$		$-\infty^{80}$
$-\infty$	∞	+	NaN^{80}
	$x \neq \infty$		∞^{81}
$-\infty$	$x \neq -\infty$	-	$-\infty^{81}$
	$-\infty$		NaN
∞	∞	x	∞^{80}
	$x > 0$		$-\infty^{80}$
$-\infty$	$x < 0$	x	∞^{80}
	$x < 0$		$-\infty^{80}$
$-\infty$	$x > 0$	x	NaN^{80}
	0 or $0.$		NaN^{80}

⁸⁰ Swapping x and y will return the same result here.

⁸¹ Swapping x and y will return this result times -1.

Input	y	x	Op.(s)	Output for \mathbb{R} lit
0 < $y \leq \infty$	0.		$/$	∞
				$-\infty$
- $\infty \leq y < 0$	0.		$/$	NaN
- ∞ or ∞	- ∞ or ∞		$/$	NaN
0 or 0.	0.		$/$, y^x	NaN
- ∞ or ∞	0. or 0		y^x	NaN
- $\infty \leq y < 0$	- ∞ or ∞		$\sqrt[x]{y}$	NaN
- $\infty < y < 0$	non-integer x		y^x , $\sqrt[x]{y}$	NaN
- ∞		odd $x > 0$		$-\infty$
- ∞	even $x > 0$		y^x	∞
			$\sqrt[x]{y}$	NaN
y ≠ 0	- ∞		y^x	0.
	∞		y^x	∞
0.	- $\infty \leq x < 0$		y^x	∞
	0 < $x \leq \infty$		y^x	0.
	- $\infty < x < 0$		$\sqrt[x]{y}$	∞
	0 ≤ $x < \infty$		$\sqrt[x]{y}$	0.
0 ≤ $y \leq \infty$	- ∞ or ∞		$\sqrt[x]{y}$	1.
∞	- $\infty \leq x < 0$		y^x	0.
	0 < $x \leq \infty$		y^x	∞
	- $\infty < x < 0$		$\sqrt[x]{y}$	0.
	0 ≤ $x < \infty$		$\sqrt[x]{y}$	∞
0.	0 < $x < \infty$		$\log_x y$	$-\infty$

The functions printed on light yellow background in the three tables above will also return NaN (or NaN+i×NaN) with complex results allowed (i.e. CPXRES set). Others will change their output when C is lit.

Some particular returns of elementary transient functions operating at the edge of the complex plane at $\pm\infty$ or close to it (or returning a value at the edge) are listed here:

Input ⁸² Re(x)	Im(x)	r(x)	$\varphi(x)$	Op.	Output for C lit
$-\infty$	—	—	—		
$-\infty$	0	∞	180°	fx	$\infty \neq 90^\circ = 0 + i \times \infty$
$-\infty$	0	∞	180°		$\infty \neq 360^\circ = \infty + i \times 0.$
0.	∞	∞	90°		$\infty \neq 180^\circ = -\infty + i \times 0.$
$-\infty$	—	—	—		$-\infty$
$-\infty$		∞			$-\infty + i \times 0.$ ⁸³
-10^{999}	0	10^{999}	180°	$\sqrt[3]{x}$	$1 \times 10^{333} \neq 60^\circ = 0.5 \times 10^{333} + i \times 0.866\ 025\ 404 \times 10^{333} = 5 \times 10^{332} (1 + i \times \sqrt{3})$
$-\infty$	—	—	—		$-\infty$
$-\infty$	0	∞	180°	x^3	$-\infty + i \times 0.$
$-\infty$	—	—	—		
$-\infty$	0	∞	180°	\ln	$\infty + i \times 3.141\ 592\ 65\dots = \infty + i\pi$
$-\infty$	∞	∞	135°		$\infty + i \times 2.356\ 194\ 49\dots = \infty + i^{3\pi/4}$
0.	∞	∞	90°	\ln	$\infty + i \times 1.570\ 796\ 32\dots = \infty + i^{\pi/2}$
∞	∞	∞	45°		$\infty + i \times 0.785\ 398\ 16\dots = \infty + i^{\pi/4}$

⁸² Complex infinities should be treated in polar notation (see an article by HP in <http://hparchive.com/Journals/HPJ-1984-07.pdf>, p. 27, left column for reasons).

⁸³ This result may be calculatory correct but is mathematically incorrect – compare next row. For mathematical reasons, similar cases may occur with other roots or powers operating near the edge of complex plane. Note *complex numbers* are stored in Cartesian coordinates in your WP 43S.

Input ⁸²		Re(x)	Im(x)	r(x)	$\varphi(x)$	Op.	Output for \mathbb{C} lit
—	—	—	—	—	—	\ln	∞
∞	0	∞	0°	∞	0°		$\infty + i \times 0.$
∞	- ∞	∞	-45°	∞	-45°		$\infty - i \times 0.785\ 398\ 16\dots = \infty - i \pi/4$
0.	- ∞	∞	-90°	∞	-90°		$\infty - i \times 1.570\ 796\ 32\dots = \infty - i \pi/2$
- ∞	- ∞	∞	-135°	∞	-135°		$\infty - i \times 2.356\ 194\ 49\dots = \infty - i^{3\pi/4}$
0.	0.	0.	0.	0.	0.		$-\infty + i \times 0.$
0.	—	—	—	—	—		$-\infty$
- ∞	0	∞	180°	∞	180°	e^x	$0. + i \times 0.$
-10 ⁹⁹⁹	10 ⁹⁹⁹	10 ⁹⁹⁹	135°	∞	135°		$0. + i \times 0.$
- ∞	∞	∞	—	∞	—		NaN+i×NaN
0.	∞	∞	90°	∞	90°		NaN+i×NaN
∞	∞	∞	45°	∞	45°		NaN+i×NaN
∞	0	∞	0°	∞	0°	e^x	$0. + i \times 0.$
∞	- ∞	∞	-45°	∞	-45°		NaN+i×NaN
0.	- ∞	∞	-90°	∞	-90°		NaN+i×NaN
- ∞	- ∞	∞	-135°	∞	-135°		$0. + i \times 0.$
-10 ⁹⁹⁹	-10 ⁹⁹⁹	10 ⁹⁹⁹	—	∞	—		

Computation of lg and $\text{lb } x$ is derived from \ln . The same applies in analogy for e^x , 10^x , and 2^x .

At the bottom line, we hope confusion is limited (and recommend keeping off $\pm\infty$ in *complex* plane).

Program Step Size

Program steps are 1 to 3 *bytes* long typically but may be significantly longer. Popular commands (like $1/x$, 2^x , ARCCOS, CF, DEC, ENTER \uparrow , FILL, GTO, IP, ISG, LN, MOD, NEXTP, RTN, R \downarrow , STO, TAN, x^2 , XEQ, $x \gtrless y$, y^x , +, \sqrt{x} , and almost all binary tests) take 1 *byte*, less frequently used ones take 2 *bytes* each – see the list overleaf. Commands with numeric parameters (like RCL+ 02), lettered registers (like $x < ? Y$), or local labels (like LBL 55) take one additional byte for the parameter; using indirect addressing costs another byte more.

Within programs, numeric constants (*reals*) take 4 to 18 *bytes* – see the examples below. Constants recalled from CONST take 2 *bytes* only. Short integers take 11 *bytes*, complex constants 7 to 34 *bytes*. Long integers (like 42) take 4 to **xxx** *bytes*.

Global LBL steps specifying an *n*-letter label take *n* + 3 *bytes*. Same applies to commands operating on named variables. Also pure *text strings* in programs consisting of *n* characters take *n* + 3 *bytes*. These statements hold for characters below U+0080₁₆ only; any character above requires 2 *bytes* (cf. DT7 as described on p. 162).

Examples:

GTO 17 is encoded in 2 *bytes* as follows:

- 1 *byte* for the GTO statement itself;
- 1 *byte* for the local label 17.

RCL+ →X is encoded in 3 *bytes* as follows:

- 1 *byte* for the RCL+ statement itself;
- 1 *byte* signaling indirect access;
- 1 *byte* for the destination X (= **R100**).

LBL ‘Prime’ is encoded in 8 *bytes* as follows:

- 1 *byte* for the LBL statement itself;
- 1 *byte* signaling a global label;
- 1 *byte* indicating the length of the label following (here: 5 usual letters);
- 5 *bytes* for these letters.

STO 'Count_1' is encoded in 10 *bytes* as follows:

- 1 *byte* for the STO statement itself;
- 1 *byte* signaling a named variable;
- 1 *byte* indicating the length of the variable following (7 usual letters);
- 7 *bytes* for these letters.

1e3 is encoded in 6 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its data type (short *real*);
- 1 *byte* indicating its length (3);
- 3 *bytes* for the characters constituting the number.

-1.5 is encoded in 7 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its data type (short *real*);
- 1 *byte* indicating its length (4);
- 4 *bytes* for the characters.

4.160 643 081 028 110 401 606 014 040 140 401 $\times 10^{-6083}$ (and any similarly large and/or precise *real number*) is encoded in 18 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its data type (long *real*);
- 16 *bytes* = 128 *bits* for the number (cf. p. 165ff for the representation).

8 07 06 05 04 03 02 01₁₆ is encoded in 11 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its data type (*short integer*);
- 1 *byte* telling its display base (here 16);
- 8 *bytes* for the digits (any *short integer* ≤ 64 *bits* can be stored therein).

-12 345 is encoded in 9 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its data type (shorter *long integer*);
- 1 *byte* indicating its length (6);
- 6 *bytes* for the sign and digits.

12 345 is encoded in 8 *bytes* as follows:

- 1 *byte* signaling a numeric constant following;

1 byte telling its data type (*longer long integer*);

1 byte indicating its length (5);

5 bytes for the digits.

Here is the list of all commands taking 1 *byte* only:

- the four basic arithmetic operators +, -, ×, and ÷,
- **sin, sinh, cos, cosh, tan, tanh**, and their inverses,
- the basic flag commands CF, FC?, FF, FS?, and SF,
- all of LOOP: DEC, DSE, DSL, DSZ, INC, ISE, ISG, and ISZ,
- the four basic Boolean operations AND, OR, XOR, and NOT,
- most *stack* commands: DROP, ENTER↑, FILL, R↑, R↓, x~~y~~, and x~~y~~y,
- RCL, RCL+, RCL-, RCL×, RCL/ as well as ST0, ST0+, ST0-, ST0×, and ST0/,
- seven angular conversions: D→R, R→D, →DEG, →D.MS, →GRAD, →MULπ, and →RAD,
- nine basic programming commands: GTO, INPUT, LBL, PAUSE, ROUND, RTN, STOP, VIEW, and XEQ,
- the basic transcendent functions: 10^x , 2^x , $\sqrt[3]{x}$, e^x , LN, LOG₁₀, LOG₂, LOG_{xy}, x^2 , x^3 , \sqrt{y} , y^x , and \sqrt{x} ,
- 1/x, CEIL, CLX, COMB, FLOOR, FP, GCD, IDIV, IP, LCM, max, min, MOD, NEIGHB, NEXTP, PERM, RMD, x!, π, +/-, and |x|,
- almost all of TEST: CONVG?, CPX?, ENTRY?, EVEN?, FC?, FP?, FS?, INT?, KEY?, MATR?, M.SQR?, NaN?, ODD?, PRIME?, REAL?, SPEC?, STRI?, TOP?, ±∞?, and all nine comparisons with x.

All other commands take 2 *bytes*.

APPENDIX C: MESSAGES AND ERROR CODES

There are some commands generating *temporary information* (as specified in *Section 2* of the *OM*), e.g. CORR, DAY, ERR, L.R., MSG, s, VERS, WDAY, \bar{x} , \hat{x} , \hat{y} , $\Sigma+$, $\Sigma-$, σ , \rightarrow POL, \rightarrow REC, and the binary test commands.

Furthermore, there are a number of error messages issued by the operating system. Depending on conditions, one of the following messages will be thrown (listed alphabetically – *EC* means *error code* here):

	<i>EC</i>	Explanations, countermeasures and examples
An argument exceeds the function domain	1	{1, 2, 3, 4, 10} An argument exceeds the domain of the mathematical function called. May be caused by roots of negative numbers or logs of $x \leq 0$ (unless CPXRES is set), by 0^0 , $x/0$, $0/0$, $\Gamma(0)$, $\tan(\pm 90^\circ)$ and equivalents, by $\text{artanh}(x)$ for $ \text{Re}(x) \geq 1$, by $\text{arcosh}(x)$ for $\text{Re}(x) < 1$, etc. ⁸⁴
Bad time or date input	2	{2, 5, 6} Invalid date format or incorrect <i>date</i> or time in input, e.g. month > 12, day > 31. Will be thrown as soon as input is closed.
Cannot delete a predefined item	27	Predefined variables or menus cannot be deleted.
Distribution parameter out of valid range	16	{1, 2} A parameter specified in I , J , or K is out of valid range for the distribution function called (e.g. LGNRM is called with $j < 0$).
Flash memory is full	23	Delete a program from <i>FM</i> to regain space.

⁸⁴ Note that e.g. $\tan(90^\circ)$ and logs of 0 are legal operations on {1, 2, 3} if SPCRES is set. See the end of this appendix.

	EC	Explanations, countermeasures and examples
Flash memory is write protected	19	There was an attempt to edit or delete program steps in <i>FM</i> . See PRCL and PSTO to circumvent.
Function to be coded for that data type	30	Functions may not be coded yet during FW development.
Illegal digit in integer input for this base	9	{10} E.g. 2 in binary or 9 in octal input. Will be thrown as soon as input is closed.
Input data types do not match	31	Attempt to operate on different <i>DTs</i> (e.g. for a <i>Boolean</i> operation: real AND short integer).
Input is too long	10	{7} Keyboard input is too long for the buffer.
Invalid input data type for this operation	24	Convert what is necessary, if possible (see also “ <i>operation is undefined in this mode</i> ”). But this error may also appear in attempts to calculate with a <i>configuration</i> .
Invalid or corrupted data	18	Thrown when there is a checksum error either in <i>FM</i> or as part of a serial download. Also thrown if a <i>FM</i> segment is otherwise not usable.
Item to be coded	29	Functions may not be coded yet during FW development.
I/O error	17	See Section 3 of the OM.
Matrix mismatch	21	{8, 9} <ul style="list-style-type: none"> • A matrix isn't square although it should be. • Matrix sizes aren't miscible.

	EC	Explanations, countermeasures and examples
No backup data found	35	Futile attempt to LOAD something from <i>FM</i> .
No root found	20	{2} The <i>Solver</i> did not converge.
No such function	7	Thrown when calling a nonexistent function via XEQ α ... ENTER↑ (check for typos!) or running a routine containing a nonprogrammable command.
No such label found	6	Attempt to address an undefined label.
No summation data present	28	Attempt to address an un-allocated summation register.
Operation is undefined in this mode	13	Caused e.g. by calling a <i>real-number</i> operation in <i>AIM</i> . Cf. “ <i>illegal input data type for this operation</i> ”.
Out of range	8	<p>{1, 2, 3, 10}</p> <ul style="list-style-type: none"> A number exceeds the valid range. This can be caused by specifying more decimals than 34, <i>word size</i> > 64, short integers $\geq 2^{64}$, <i>hours</i> or <i>degrees</i> > 9 000, invalid <i>dates</i> or <i>times</i>, denominators > 9 999, etc. A <i>register</i> or <i>flag</i> address exceeds the valid range of currently allocated <i>registers</i> or <i>flags</i>. May also happen in indirect addressing or when calling nonexistent local addresses. An R-operation (e.g. R-COPY) attempts accessing invalid <i>register</i> addresses.
Output would exceed 196 characters	33	{7} Maximum length of <i>alphanumeric strings</i> .

	EC	Explanations, countermeasures and examples
Overflow at $+\infty$	4	<p>{1, 2, 3, 8, 9} unless SPCRES is set</p> <ul style="list-style-type: none"> • Division of a number > 0 by 0. • Divergent sum or product or integral. • Positive overflow (see p. 164).
Overflow at $-\infty$	5	<p>{1, 2, 3, 8, 9} unless SPCRES is set</p> <ul style="list-style-type: none"> • Division of a number < 0 by 0. • Divergent sum or product or integral. • Negative overflow (see p. 164).
Please enter a NEW name	26	Attempt to define a new variable or user <i>menu</i> with a <i>name</i> already in use.
RAM is full	11	May be caused by attempts to write too large routines, allocate too many variables, and the like (see pp. 160ff for the space required by different <i>DTs</i>). May happen also in program execution due to dynamic allocations (see <i>Section 3</i> of the <i>OM</i>).
Singular matrix	22	<p>{8, 9}</p> <ul style="list-style-type: none"> • Attempt to use a LU decomposed matrix for solving a system of equations. • Attempt to invert a matrix which isn't of full rank.
Stack clash	12	STOS or RCLS attempts using <i>registers</i> that would overlap the <i>stack</i> (see <i>Section 1</i> of the <i>OM</i>). Will happen with e.g. ssIZE8 set and STOS 93 .
This does not work with an empty string	34	{7} Self-explanatory.

	EC	Explanations, countermeasures and examples
This system flag is write protected	32	Self-explanatory.
Too few data points for this statistic	15	{2} A statistical calculation was attempted with too few data, e.g. <i>regression</i> or <i>standard deviation</i> for less than 2 points.
Undefined op-code	3	An instruction with an undefined operation code occurred. Should never happen – but who knows?
Word size is too small	14	{10} User input or <i>register</i> content is too great to be handled by the word size currently set.
	25	Left unused for WP 34S compatibility

If SPCRES is set, errors 4 and 5 will not occur at all, and error 1 will happen less frequently, since $\pm\infty$ and NaN are legal results then (cf. the corresponding entries in CONST on pp. 129ff and the tables on pp. 169ff). E.g., **0** **In** will return $-\infty$ then.

Each error message will be displayed in Z numeric row and is *temporary information* (see Section 2 of the OM). So **C** or **EXIT** will erase it and allow continuation most easily. Any other key pressed will erase the message as well, but will also – if applicable – execute with the stack contents present.



APPENDIX D: COMPARISON TO THE FUNCTION SETS OF HP-42S, HP-16C, HP-21S, AND WP 34S

In the *IOI*, the corresponding functions of vintage *HP* calculators were mentioned under the respective entry of your *WP 43S*. The tables below revert this in a way. The first table shows the functions of the *HP-42S* and the corresponding ones of your *WP 43S* unless they carry identical names and are either both keyboard accessible or both stored in a *catalog* or *menu*. There is an analog table for *HP-16C* functions starting on p. 193, one for the *HP-21S* on p. 195, and another one for the *WP 34S* on p. 197. Functions newly introduced with *WP* calculators are compiled on pp. 201ff.

Functional differences of homonymous commands are covered in the *IOI* (on pp. 12ff).

Corresponding Operations on *HP-42S*

Remarks printed on light grey indicate commands being either default settings or keyboard accessible on your *WP 43S* while you must use a *menu* on the *HP-42S*.

<i>HP-42S</i>	<i>WP 43S</i>	Remarks
ACOSH	arcosh	In <u>EXP</u>
ADV	�ADV	In <u>PRINT</u>
AIP	Dispensable	You can merge text and numeric data easily as described in <i>Section 2</i> of the OM.
ALENG	�ALENG	In <u>�.FN</u>
ALL�	Dispensable	Your <i>WP 43S</i> runs in ALL� mode always. The summation <i>registers</i> do not overlap with <i>GP registers</i> .
ALPHA	�	See the description of <i>AIM</i> in <i>Sect. 2</i> of the OM.

HP-42S	WP 43S	Remarks
AOFF	CF ALPHA	
AON	SF ALPHA	
ARCL	Disposable	Any register or variable can take a <i>text string</i> . Simply press RCL instead.
AROT	αRL or αRR	In <u>$\alpha.FN$</u>
ASHF	αSL	
ASINH	$arsinh$	In <u>EXP</u>
ASTO	Disposable	Any register or variable can take a <i>text string</i> . Simply press STO instead.
ATANH	$artanh$	In <u>EXP</u>
ATOX	$\alpha \rightarrow x$	In <u>$\alpha.FN$</u>
AVIEW	Disposable	Any register or variable can take a <i>text string</i> . Simply press VIEW instead.
BASE	INTS or BITS	
BASE+	Disposable	Your WP 43S executes these arithmetic commands automatically for <i>short integer</i> inputs.
BASE-		
BASE \times		
BASE \div		
BASE $+\!-\!$		
BINM	Disposable	Press # 2 for converting any closed integer number or integer part in x to binary.
BIT?	BS?	In <u>BITS</u>
BST	 ()	Shortcut works if no <i>multi-view menu</i> is open.
CLA	0 STO K	
CLD	Disposable	Any keystroke will clear <i>temporary information</i> .
CLEAR	CLR	
CLKEYS	n/a	See Section 6 of the OM.
CLRG	CLREGS	In <u>CLR</u>
CLST	CLSTK	Press 0 FILL in run mode.

HP-42S	WP 43S	Remarks
CLV	See remark	Variables are cleared as specified in <i>Section 6</i> of the OM.
COMPLEX	CC	You can also enter <i>complex numbers</i> directly using CC as explained in <i>Section 2</i> of the OM.
CONVERT	L→ & PARTS	
CUSTOM	n/a	You can create as many <i>menus</i> as memory will hold – not only one CUSTOM menu. See <i>Section 6</i> of the OM.
DECM	Disposable	Any input featuring a 0 or an E is interpreted as a <i>real</i> (decimal) number.
DEL	n/a	Not featured. Too dangerous, in our opinion.
DELAY	DLAY	In <u>PRINT</u>
DELR	M.DELR	In <u>MATX</u>
DET	 M 	
DIM	M.DIM	
DIM?	M.DIM?	
EDIT	M.EDI	
EDITN	M.EDIN	
FCSTX	x	In <u>STAT</u>
FCSTY	y	
FNRM	ENORM	In <u>MATX</u> . Euclid is older than Frobenius.
GAMMA	Γ(x)	In <u>PROB</u>
GETKEY	KEY?	In <u>P.FN</u>
GETM	M.GET	In <u>MATX</u>
GROW	M.GROW	
HEXM	Disposable	Press # H for converting any closed integer number or integer part in x to hexadecimal.
H.MS+	Disposable	Your WP 43S executes the respective command automatically for sexagesimal times in x and y when + or - is pressed.
H.MS-		

HP-42S	WP 43S	Remarks
INSR	M.INSR	In <u>MATX</u>
INTEG	\int	In <u>ADV</u>
INVRT	$[M]^{-1}$	In <u>MATX</u>
KEYASN	Disposable	Not needed since no CUSTOM menu is featured (see CUSTOM).
LASTx	RCL L	
LBL		Press LBL .
LCLBL	Disposable	Obsolete since no CUSTOM menu is featured (see CUSTOM). Nevertheless, your WP 43S provides local labels (see Section 3 of the OM).
LINΣ	Disposable	Your WP 43S runs in ALLΣ mode always.
LIST	n/a	Use PROG instead.
LOG	\log_{10}	Press lg .
MAN	CF T	Manual print mode is <i>startup default</i> here.
MAT?	MATR?	In <u>TEST</u>
MEAN	\bar{x}	In <u>STAT</u>
MOD		Press MOD .
MODES	MODE	
N!	$x!$	
NEWMAT	M.NEW	In <u>MATX</u>
NORM	n/a	Not featured.
OCTM	Disposable	Press # 8 for converting any closed integer number or integer part in x to octal.
OLD	RCLEL	In <u>MATX</u>
ON	n/a	Programmable ON is not featured.
PGM.FCN	P.FN	GTO , LBL , RTN , VIEW are on the keyboard.
PI	π	Press π .
POSA	α POS	In <u>α.FN</u>

HP-42S	WP 43S	Remarks
PRA		In <u>PRINT</u>
[PRGM]	[P/R]	
PRLCD		In <u>PRINT</u>
PROFF		
PROMPT	Disposable	Use , instead.
PRON		
PRP		
PRSTK		
PRUSR		
PRV		
PRX		Press .
PRΣ		In <u>PRINT</u>
PUTM	M.PUT	In <u>MATX</u>
PWRF	PowerF	In <u>STAT</u>
RAN	RAN#	In <u>PROB</u>
RND	ROUND	In <u>PARTS</u>
RNRM	RNORM	In <u>MATX</u>
ROTXY	RL, RLC, RR, and RRC	In <u>BITS</u>
RTN		Press .
SDEV	s	In <u>STAT</u>
SIZE	Disposable	There are 100 global <i>GP registers</i> always.
SLOPE	L.R.	In <u>STAT</u>
SOLVE	SLV	In <u>ADV</u>
SQRT		
[SST]	()	Shortcut works if no <i>multi-view menu</i> is open.
STR?	STRI?	In <u>TEST</u>

HP-42S	WP 43S	Remarks
TOP.FCN	Disposable	Obsolete – no top functions are overwritten.
TRACE	SF	
TRANS	[M] ^T	In <u>MATX</u>
UVEC	UNITY	In <u>MATX</u> and <u>CPX</u>
VARMENU	VARMNU	Truncated to 6 characters to fit the <i>menu</i> space.
VIEW		Press VIEW .
WMEAN	\bar{x}_w	In <u>STAT</u>
WRAP	M.WRAP	In <u>MATX</u>
XTOA	$x \rightarrow \alpha$	The conversion is done in X .
$X < 0?, X < Y?$	$x < ?$	In <u>TEST</u>
$X \leq 0?, X \leq Y?$	$x \leq ?$	
$X = 0?, X = Y?$	$x = ?$	
$X \neq 0?, X \neq Y?$	$x \neq ?$	
$X \geq 0?, X \geq Y?$	$x \geq ?$	
$X > 0?, X > Y?$	$x > ?$	
YINT	L.R.	In <u>STAT</u>
y^x		Press y^x .
ΣREG	Disposable	There are 100 global <i>GP registers</i> always.
$\Sigma REG?$		Statistical registers are separate.
$\rightarrow DEC$	$\rightarrow INT$ 10	Press # D
$\rightarrow HR$		Press .d
$\rightarrow H.MS$		Press h.ms ... for closed input.
$\rightarrow OCT$	$\rightarrow INT$ 8	Press # 8
$\rightarrow POL$		Press $\rightarrow P$.
$\rightarrow REC$		Press R\leftrightarrow .
%CH	$\Delta\%$	Press $\Delta\%$.
		Cf. ISO 80000-2: "The symbol \div should not be used."

Corresponding Operations on HP-16C

The table for the functions of the *HP-16C* is sorted following its keyboard layout, starting top left. As for the *HP-42S*, only functions carrying different names on both calculators are listed.

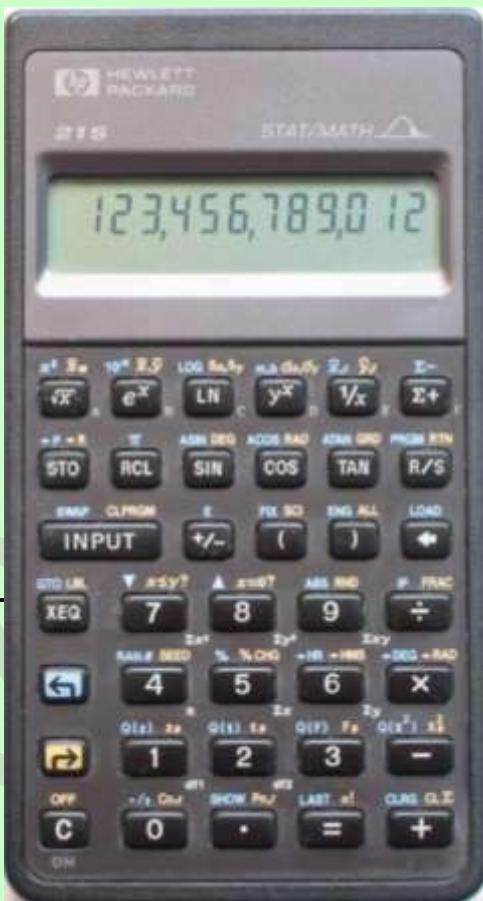
<i>HP-16C</i>	<i>WP 43S</i>	<i>Remarks</i>
[RL], [RLn]	RL	In <u>BITS</u>
[RR], [RRn]	RR	
[RLC], [RLCn]	RLC	
[RRC], [RRCn]	RRC	
÷	/	(see also ISO 80000-2: "The symbol ÷ should not be used.")
DBL÷	DBL/	In <u>INTS</u>
[x ⁱ (i)]	Disposable	Any register may be used for indirection.
[x ⁱ]		
SHOW HEX	n/a	
SHOW DEC		
SHOW OCT		
SHOW BIN		
B?	BS?	In <u>BITS</u>
GSB	XEQ	
HEX	# H	
DEC	# D	
OCT	# 8	
BIN	# 2	
SF 3, CF 3	SF L, CF L	Leading zeros.
SF 4, CF 4	SF C, CF C	Carry.
SF 5, CF 5	SF B, CF B	Overflow.
F?	FS?	In <u>FLAGS</u>
(i)	Disposable	Any register may be used for indirection.
I		

HP-16C	WP 43S	Remarks
CLEAR PRGM	CLP	In <u>CLR</u> . Note here is also CLPALL.
CLEAR REG	CLREGS	In <u>CLR</u>
CLEAR PREFIX	Dispensable	See Section 2 of the OM.
WINDOW	Dispensable	64 bits can be displayed in one row.
SET COMPL 1S	1COMPL	
SET COMPL 2S	2COMPL	In <u>MODE</u> and <u>BITS</u> . Note here is also SIGNMT.
SET COMPL UNSGN	UNSIGN	
SST		(works if no multi-view menu is open.)
BSP		
BST		(works if no multi-view menu is open.)
x≤y	x≤ ?	
x<0	x< ?	In <u>TEST</u> . Note far more tests are covered here.
x>y , x>0	x> ?	
FLOAT	FIX	In <u>DISP</u>
MEM	STATUS	In <u>FLAGS</u>
CHS		
<, >	Dispensable	64 bits can be displayed in one row.
LSTX	RCL L	
x≠y , x≠0	x≠ ?	In <u>TEST</u> . Note far more tests are covered here.
x=y , x=0	x= ?	

Corresponding Operations on HP-21S

The table for the functions of *HP-21S* follows the same rules as the one for *HP-16C*. It is, however, an algebraic calculator; hence its keys **INPUT**, **(**, **)**, and **=** have no direct equivalent on your *WP 43S*.

Consult the *HP-21S OM* for additional information about the four most important continuous statistical distributions and their applications.



<i>HP-21S</i>	<i>WP 43S</i>	Remarks
\bar{x}_w	\bar{x}_w	In <u>STAT</u>
$\bar{x} \cdot \bar{y}$	\bar{x}	
S_x, S_y	s	
m.b	L.R.	
σ_x, σ_y	σ	
$\hat{x} \cdot r$	r, \hat{x}	
$\hat{y} \cdot r$	r, \hat{y}	
PRGM	P/R	
SWAP	x>y	
CLPRGM	CLP	In <u>CLR</u>
INPUT	Dispensable	This functionality is contained in ENTER. Also your <i>WP 43S</i> features a command called INPUT but this works in programs.
(,)	Dispensable	You can forget these keys in <i>RPN</i> .

HP-21S	WP 43S	Remarks
LOAD	n/a	Loads predefined programs in the <i>HP-21S</i> . Also your <i>WP 43S</i> features a command called LOAD but this recalls data from backup.
ABS	 x 	In <u>PARTS</u>
RND	ROUND	
FRAC	FP	
÷	/	Cf. ISO 80000-2: "The symbol \div should not be used."
SEED	SEED	In <u>PROB</u>
%CHG	Δ%	
Q(z)	Norm_e	In submenus of <u>PROB</u> . Note your <i>WP 43S</i> features the <i>normal</i> distribution for <u>arbitrary</u> μ and σ instead of the <u>standardized</u> one ($\mu=0$, $\sigma=1$). And the implementation of inverse distribution functions deviates on both calculators: your <i>WP 43S</i> calculates with the probability P while the <i>HP-21S</i> calculates with the error probability $Q = 1 - P$ as input (cf. the <i>OM</i> , Section 2). Labeling these functions on the <i>HP-21S</i> using the letter p may add some confusion.
zp	Norm⁻¹	
Q(t)	t_Δ(x)	
tp	t⁻¹(p)	
Q(F)	F_Δ(x)	
Fp	F⁻¹(p)	
Q(x²)	x²_Δ(x)	
X²p	(x²)⁻¹	
Cn.r	COMB	In <u>PROB</u>
Pn.r	PERM	
LAST	RCL L	
=	Dispensable	You can forget this key in <i>RPN</i> .
n!	x!	
CLRG	CLREGS	In <u>CLR</u>

Corresponding Operations on WP 34S

The WP 34S and WP 43S share over 90% of their function sets. It was our objective that your WP 43S is equal or better than the WP 34S in every aspect. Most of the discrepancies between both calculators are caused by their different displays. Thus, your WP 43S allows for *softkeys* – the WP 34S can only carry four *hotkeys* instead. Also dealing with matrices is greatly eased by the large high resolution dot matrix display of your WP 43S; thus some elementary matrix commands of the WP 34S are not required anymore on your WP 43S.

Remarks printed on light grey indicate commands being either default settings or obsolete on your WP 43S while you must use them on the WP 34S.

WP 34S	WP 43S	Remarks
ANGLE	4	
Binom	Binom _▲	
Binom _u	Binom _▲	
Binom	Binom _▲	
Cauch _u	Cauch _▲	
CL _a	0 [STO] [K]	Check the OM for the conditions when this register is used.
CONST	CNST	For keyboard space reasons.
CONV	U→	
DBLOFF	Disposable	Your WP 43S features 34-digit DTs per default – it does neither need nor feature a <i>double precision</i> mode.
DBLON		
Expon	Expon _▲	
Expon _u	Expon _▲	
F(x)	F _▲ (x)	
F _u (x)	F _▲ (x)	

WP 34S	WP 43S	Remarks
dRCL	Disposable	Your WP 43S features various <i>data types</i> .
gCLR, gDIM, gDIM?, gFLP, gPIX?, gPLOT, gSET	n/a	The <i>LCD</i> of your WP 43S features 240×400 px rows compared to 6×43 px of HP-30b – the graphic paradigm of WP 34S makes no sense on your WP 43S. On the other hand, it was not our objective designing a graphing calculator. Thus, we include just the basic graphic support of HP-42S (AGRAPH, CLLCD, PIXEL) plus POINT.
Geom	Geom _△	
Geom _u	Geom _△	
GTO α	Disposable	Use GTO with an appropriate parameter instead.
H.MS+, H.MS-	Disposable	Your WP 43S features a dedicated <i>DT</i> for <i>times</i> , so + and - suffice for adding or subtracting sexagesimal times, respectively.
INTM?	Disposable	Your WP 43S features dedicated <i>DTs</i> for integers – it does neither need nor feature an integer mode.
iRCL	Disposable	Your WP 43S features various <i>data types</i> .
I _x	I_{xyz}	This is a triadic function after all.
Lgnrm	LgNrm _△	
Lgnrm _u	LgNrm _△	
L _n	L_m	Renamed to avoid search conflict with LN.
L _{na}	L_{ma}	Renamed in consequence to L _m .
Logis	Logis _△	
Logis _u	Logis _△	
MROW+ \times , MROW \times	Disposable	Obsolete matrix commands.
MROW \Leftarrow	M.R\LeftarrowR	
M+ \times	Disposable	Obsolete matrix command.
M $^{-1}$	[M]$^{-1}$	

WP 34S	WP 43S	Remarks
M-ALL, M-COL, M-DIAG, M-ROW	Disposable	Obsolete matrix commands.
Mx	Disposable	Your WP 43S features two dedicated DTs for matrices. Thus you can simply multiply two matrices using X and copy matrices like any other objects.
M.COPY	Disposable	
M.IJ, M.REG	Disposable	Obsolete matrix commands.
nBITS	#B	
nCOL, nROW	Disposable	Obsolete matrix commands.
Norml	Norml _Δ	
Norml _u	Norml _Δ	
Poiss	Poiss _Δ	
Poiss _u	Poiss _Δ	
REALM?	Disposable	Your WP 43S features a dedicated DT for <i>reals</i> – it does not need a <i>real</i> mode.
REGS, REGS?	Disposable	The number of global GP registers is fixed to 100 on your WP 43S.
SENDA, SENDP, SENR, SENDΣ	SEND	SEND combines all those four commands of the WP 34S.
SEPOFF, SEPON	GAP	
SHOW	RBR	
sRCL	Disposable	Your WP 43S features various <i>data types</i> .
TRANSP	[M] ^T	
TSOFF	GAP 0	
TSON	GAP 3	

WP 34S	WP 43S	Remarks
$t(x)$	$t_{\Delta}(x)$	
$t_u(x)$	$t_{\Delta}(x)$	
VIEWa, VWa+	Disposable	Use VIEW instead; <i>alphanumeric strings</i> are just another <i>DT</i> . Combine text and numeric data easily using + as shown in the OM, Section 2.
Weibl	Weibl _Δ	
Weiblu	Weibl _Δ	
XEQa	Disposable	Use XEQ with an appropriate parameter instead.
XTAL?	Disposable	A quartz crystal is installed by default.
YDOFF, YDON	Disposable	Your <i>WP 43S</i> displays <i>y</i> whenever possible and wanted. See DSTACK.
αDATE, αDAY	Disposable	You can combine text and numeric data easily using + as shown in <i>Section 2</i> of the OM.
αGTO	Disposable	Use GTO w/ an appropriate parameter instead.
αIP, αMONTH	Disposable	You can combine text and numeric data easily using + as shown in <i>Section 2</i> of the OM.
αRCL, αRC#	Disposable	Your <i>WP 43S</i> features various <i>data types</i> and 'knows' which type is in the <i>register</i> specified. Appending texts is done by + .
αSTO	Disposable	Simply press STO instead (any <i>register</i> can take a <i>text string</i>).
αTIME	Disposable	See αDATE.
αXEQ	Disposable	Use XEQ with an appropriate parameter instead.
β	β(x,y)	
Γ	Γ(x)	
ΔDAYS	Disposable	Simply subtract two <i>dates</i> .
ζ	ζ(x)	
Φ(x) ...	Disposable	Use NORML... with $\mu=0$ and $\sigma=1$ instead.
$\chi^2(x)$	$\chi^2_{\Delta}(x)$	

WP 34S	WP 43S	Remarks
$\chi^2_u(x)$	$\chi^2_{\Delta}(x)$	
$\rightarrow H$	$\rightarrow HR$	
$\blacksquare PLOT$	n/a	See gCLR.
$\blacksquare C_{XY}$	Disposable	Use $\blacksquare r$ instead.
$\blacksquare \alpha,$ $\blacksquare \alpha +,$ $\blacksquare + \alpha$	Disposable	Combine text and numeric data easily using $\blacksquare +$ as shown in <i>Section 2</i> of the OM. Then use $\blacksquare r$.
$\blacksquare ?$	Disposable	A quartz crystal and the proper firmware for printing are installed by default.

New Commands on your WP 43S

The following table lists the commands and pseudo-commands created for your *WP 43S* (and for preceding *WP* calculators, if applicable), offering new or extended functionality compared to earlier *HP RPN* and algebraic pocket calculators. In total, these are more than 350 operations, not counting the unit conversions and constants provided; 80 of them are even new or extended compared to earlier *WP* calculators. The commands are printed below as spelled on your *WP 43S*.

Command	WP 43S	WP 31S	WP 34S
2^x AGM	●	—	new
ALL	●	●	extended
AND ASR NOT OR XOR	●	—	extended
BACK CASE SKIP	●	—	new
BATT?	●	●	new
BC? FB	●	—	new
BestF	extended	●	●
BestF?	new	—	—

Command	WP 43S	WP 31S	WP 34S
Binom _p Binom _A (of Binomial distribution)	●	●	new
B _n B _n * CEIL FLOOR	●	—	new
Cauch _p Cauch _A Cauch _A Cauch ⁻¹	●	●	new
CauchF GaussF HypF ParabF RootF	new	—	—
CLCVAR	new	—	—
CLFall CLPall CONJ CONVG? COV	●	—	new
CX→RE RE→CX	new	—	—
DATE TIME	●	—	(●)
DATE→ DAY MONTH YEAR →DATE	●	—	new
DEC DSL INC ISE	●	—	new
DECOMP	●	●	new
DEG→ D.MS→ GRAD→ RAD→	●	—	new
DELITM	new	—	—
DROP	●	—	new
DROPy DSTACK	new	—	—
D→J J→D	●	—	new
EIGVAL EIGVEC	new	—	—
ENTRY?	●	—	new
EQ.DEL EQ.EDI EQ.NEW	new	—	—
erf erfc ERR MSG	●	—	new
EVEN? ODD?	●	—	new
Expon _p Expon _A Expon _A Expon ⁻¹	●	●	new
EXPT MANT	●	—	new
FBR	new	—	—
FC?F FC?S FF FS?F FS?S	●	—	new
FIB	●	—	new
FILL	●	●	new
FLASH? FP?	●	—	new

Command	WP 43S	WP 31S	WP 34S
$F_p(x)$ $F_{\Delta}(x)$ (of F distribution)	●	●	new
f' f''	new	—	—
$f'(x)$ $f''(x)$	extended	—	new
GAP	extended	●	new
GCD LCM	●	●	new
g_d g_d^{-1}	●	—	new
Geom _p Geom _△ Geom _△ Geom ⁻¹	●	—	new
H _n H _{nP} L _m L _{ma} P _n T _n U _n	●	—	new
Hyper _p Hyper _△ Hyper _△ Hyper ⁻¹	new	—	—
IDIV	●	—	new
IDIVR IM RE	new	—	—
INT? ISM? I _{xyz} IΓ _p IΓ _q	●	—	new
J _y (x) J/G?	new	—	—
J/G	extended	●	new
KEY? KTyp? LBL? LEAP?	●	—	new
LgNrm _p LgNrm _△ LgNrm _△ LgNrm ⁻¹	●	—	new
LNB LNF LOADP LOADR LOADSS LOADΣ LocR LocR? LOG ₂ LOG _{xy}	●	—	new
LOAD SAVE	●	●	new
Logis _p Logis _△ Logis _△ Logis ⁻¹	●	●	new
max min MIRROR	●	—	new
MOD	●	●	new
MULπ MULπ→	new	—	—
M.LU M.SQR? NAND NaN? NEIGHB NOR	●	—	new
NBin _p NBin _△ NBin _△ NBin ⁻¹	new	—	—
NEXTP PRIME?	extended	●	new
Norml _p Norml _△ Norml _△ Norml ⁻¹	●	●	new
nΣ (callable by name)	●	●	new

Command	WP 43S	WP 31S	WP 34S
OrthoF PLOT POINT	new	—	—
PAUSE	●	—	extended
Poiss _p Poiss _Δ Poiss _Δ Poiss ⁻¹	●	●	new
PopLR PRCL PSTO PUTK	●	—	new
RANGE RANGE? RANI#	new	—	—
RBR	●	●	new
RCLCFG STOCFG	extended	—	new
RCLS STOS	●	—	new
RCL↑ RCL↓ STO↑ STO↓	●	—	new
RDP RECV SEND	●	—	new
Re ^z Im	new	—	—
RJ	●	—	new
RL RLC RR RRC	●	—	extended
RMD	●	●	extended
RM RM? ROUNDI RSD RTN+1 R-CLR R-COPY R-SORT R-SWAP	●	—	new
SDIGS? SETSIG	new	—	—
SDL SDR SETCHN SETEUR SETIND SETJPN SETUK SETUSA	●	—	new
SETDAT SETTIM	●	—	(●)
SIGNMT sinc	●	—	new
sincπ	new	—	—
SL SR	●	—	extended
SLVQ SPEC?	●	—	new
S _m S _{mw} S _w	●	●	new
SNAP	new	—	—
SSIZE?	●	●	new
STATUS	extended	—	extended
S _{xy}	●	—	new
s(a) TDISP	new	—	—

Command	WP 43S	WP 31S	WP 34S
TICKS	●	—	new
TIMER	●	—	(●)
TOP? ULP?	●	—	new
$t_p(x) \ t_{\Delta}(x)$ (of t distribution)	●	●	new
$t_x^y \ y_x^z \ z_x^y$	●	—	new
(UNDO)	●	new	—
∇_x	new	—	—
VERS? WDAY WHO?	●	●	new
Weibl _p Weibl _Δ Weibl _Δ Weibl ⁻¹	●	●	new
$W_m \ W_p \ W^{-1} \ WSIZE? \ \bar{x}_G \ XNOR$	●	—	new
$\bar{x}_H \ \bar{x}_{RMS} \ x \rightarrow DATE$	new	—	—
$x < ? \ x \leq ? \ x = ? \ x \neq ? \ x \geq ? \ x > ?$	extended	—	extended
$x = +0? \ x = -0? \ x \approx ?$	●	—	new
y^x	extended	●	●
Y.MD	●	●	new
αLENG?	extended	—	●
αPOS?	extended	—	—
αRL αRR αSL αSR	●	—	extended
$\beta(x,y) \ \Gamma_{xy} \ \gamma_{xy} \ \varepsilon \ \varepsilon_m \ \varepsilon_p \ \zeta(x) \ \Pi_n$ $\Sigma_n \ \sigma_w$	●	—	new
$\Sigma^1/x \ \Sigma^1/x^2 \ \Sigma^1/y \ \Sigma^1/y^2 \ \Sigma \ln y/x$ $\Sigma x^2/y \ \Sigma x^3 \ \Sigma x^4 \ \Sigma x/y$	new	—	—
$\Sigma \ln^2 x \ \Sigma \ln^2 y \ \Sigma \ln x \ \Sigma \ln xy \ \Sigma \ln y \ \Sigma x$ $\Sigma x^2 \ \Sigma x^2 y \ \Sigma x \ln y \ \Sigma xy \ \Sigma y \ \Sigma y \ln x \ \Sigma y^2$ (callable by names)	●	●	new
$\chi^2_p(x) \ \chi^2_{\Delta}(x)$ (of chi-square distribution)	●	●	new
$(-1)^x \ xMOD \ ^MOD$	●	—	new
$\pm \omega?$	new	—	—
→DEG →RAD	●	●	new
→D.MS →MULπ	new	—	—

Command	WP 43S	WP 31S	WP 34S
→GRAD	●	—	new
→INT →REAL	new	—	—
■ ADV ■ CHAR ■ r ■ REGS ■ TAB ■ # ■ MODE	●	—	(new)
■ WIDTH	extended	—	(new)
	●	●	new

The statements in parentheses in the rightmost column refer to the WP 34S with optional quartz and capacitors installed (see its manual).

Reference Literature

As mentioned above, some advanced functionality of your WP 43S is taken over from previous HP calculators. The following vintage HP material is recommended as source of in-depth information (as far as calculating, programming, and applications are concerned) about the topics listed, from a calculator point of view. All the manuals listed below are entirely contained in a document set distributed by the *Museum of HPC* (see <http://www.hpmuseum.org/cd/cddesc.htm>). They can be also found in the internet, some even provided by HP still.

Topic	Recommended literature
General calculation examples & applications	All vintage HP calculator manuals can be recommended.
Statistical distributions and their application	<i>HP-21S Owner's Manual</i> , especially pp. 63 – 105. ⁸⁵
Manipulating bits and short integers	<i>HP-16C Owner's Handbook</i> ⁸⁶

⁸⁵ Download from <https://literature.hpcalc.org/community/hp21s-om-en.pdf>

⁸⁶ Download from <http://www.hp41.net/forum/fileshp41net/hp16c.pdf>

Topic	Recommended literature
Root finding and numeric integration	<i>HP-34C Owner's Handbook & Programming Guide</i> ⁸⁷ <i>HP-15C Owner's Handbook</i> ⁸⁸ <i>HP-15C Advanced Functions Handbook</i> ⁸⁹ <i>HP-42S Programming Examples & Techniques</i> ⁹⁰
Accuracy of numeric calculations	<i>HP-15C Advanced Functions Handbook</i> , pp. 172 – 211. ⁸⁹
Programming	<i>HP-42S Owner's Manual</i> ⁹¹ <i>HP-42S Programming Examples & Techniques</i> ⁹⁰
Financial calculations	<i>HP-17BII+ User's Guide</i> ⁹²

Depending on your educational background and professional qualification, textbooks about various mathematical, scientific, or engineering topics may be helpful in addition. Ensure you know enough about what you compute (and check footnote 100 on p. 231 below as well as the last paragraph on p. 16 of the OM).

 The floating point standard *IEEE 754* was developed in 1985, after most of the calculators mentioned above were launched already (see https://en.wikipedia.org/wiki/Floating-point_arithmetic as a starter, also about floating point numbers in general).

⁸⁷ Read <https://www.yumpu.com/en/document/read/19323790/hp34c-slide-rule-museum> or download from <https://literature.hpcalc.org/community/hp34c-oh-en.pdf>

⁸⁸ Download a reprint from <http://h10032.www1.hp.com/ctg/Manual/c03030589.pdf>

⁸⁹ Download a reprint from <http://h10032.www1.hp.com/ctg/Manual/c03308725.pdf>

⁹⁰ Download from <https://literature.hpcalc.org/community/hp42s-prog-en.pdf>

⁹¹ Download from <http://www.hp41.net/forum/fileshp41net/manual-hp42s-us.pdf>

⁹² Download a reprint from <http://h10032.www1.hp.com/ctg/Manual/c00363348.pdf>

APPENDIX E: EMULATING A WP 43S ON YOUR COMPUTER

1) Under Windows, you can ...

1.a) use **MSYS2 MinGW 64-bit**, a runtime environment for *gcc*. You get it at <https://www.msys2.org>. Install it following the on-site instructions (but in **c:/msys64**) until step 7. Then enter in the black *MinGW* window:

```
pacman -S mingw-w64-x86_64-gcc git base-devel mingw-w64-x86_64-gtk3  
This step may take many minutes.
```

```
git clone https://gitlab.com/Over_score/wp43s.git
```

```
git config pull.rebase false
```

```
PATH=/mingw64/bin:$PATH
```

```
PKG_CONFIG_PATH=/mingw64/lib/pkgconfig:$PKG_CONFIG_PATH
```

```
cd wp43s          for changing to the proper directory.  
                  Continue at 5).
```

1.b) alternatively do the following:

Open the folder

https://gitlab.com/Over_score/wp43s/tree/master/windows%20binaries

Open README.md and proceed as described therein.

Eventually run wp43s.exe. Continue at 6).

2) Under Linux:

You have to install the standard development tools, git, gtk+ 3 dev, and libgmp dev packages.

Then, the first time, simply run:

```
git clone https://gitlab.com/Over_score/wp43s.git
```

```
cd wp43s          for changing to the proper directory.  
                  Continue at 5).
```

3) Under OSX (Mac):

Here is *Harald Overbeek's* step by step solution:

- a) Install **XCode** from the App Store.
- b) Clone the source code of the *WP 43S* project by opening Xcode and clone it using https://gitlab.com/Over_score/wp43s.git. Cloning the project has some advantages over downloading the code. It makes sure *XCode* tracks the changes, for example.
- c) Install **MacPorts** using
<https://guide.macports.org/chunked/installing.macports.html>
- d) Thereafter install the following *MacPorts* one by one:

```
sudo port install gcc9
sudo port install gtk3
sudo port install freeType
sudo port install pkgconfig
sudo port install x11
sudo port install dbus
```

- e) Then run the **makefile** form the root directory of the project (probably **wp43s**)
- f) When the project has successfully compiled, run the program, for example like this: **./wp43s** Continue with 6).

In the terminal window the following warning may appear:

Gtk-WARNING **: 11:23:52.903: Locale not supported by C library.
Using the fallback 'C' locale.

Solve this by entering:

```
export LC_ALL="en_US"
export LANG="en_US"
export LANGUAGE="en_US"
export C_CTYPE="en_US"
export LC_NUMERIC=
export LC_TIME=en_US" ... or similar locale settings.
```

If you get this error:

```
dbus[1369]: Dynamic session lookup supported but failed: launchd did not
provide a socket path, verify that org.freedesktop.dbus-session.plist
is loaded!
```

use this:

```
sudo launchctl load -w /Library/LaunchDaemons/org.freedesktop dbus-
system.plist
launchctl load /Library/LaunchAgents/org.freedesktop dbus-session.plist
export DBUS_SESSION_BUS_ADDRESS="launchd:env= DBUS_FINK_
SESSION_BUS_SOCKET"
```

After that I get no more warnings or error messages.

On the first 'make' I got error messages about header files that could not be found. I solved this by adding the following lines to the *makefile*. After:

```
else ifeq ($(detected_OS),Darwin) # Mac OS X
CFLAGS += -D OSX
```

add:

```
CFLAGS += -I/opt/local/include/
CFLAGS += -I/opt/local/include/glib-2.0/
CFLAGS += -I/opt/local/lib/glib-2.0/include/
CFLAGS += -I/opt/local/include/gtk-3.0/
CFLAGS += -I/opt/local/include/pango-1.0/
CFLAGS += -I/opt/local/include/cairo/
CFLAGS += -I/opt/local/include/gdk-pixbuf-2.0/
CFLAGS += -I/opt/local/include/atk-1.0/
CFLAGS += -I/opt/local/include/freetype2
```

On my machine I put the project into `~/wp43s`. However, the application searches for the `wp43s_pre.css` in the local home directory. If no calculator window comes up, copy the css-file:

```
cp wp43s_pre.css ~
```

Let me know if this does not work.

4) Updating the simulator:

4.a) The following is for *Linux*, only if necessary:

`cd wp43s` Continue at 4b).

4.b) The following is for *Windows* (within the *MinGW* window) & *Linux*:

`git pull` for pulling all the changed files from *gitlab* repository.⁹³ Continue at 4c)

4.c) Optionally enter

`rm backup.bin` if you want to start the simulator reset to default.
Continue at 5).

5) Enter

`make` for compiling and building a new executable.⁹⁴

`./wp43s.exe` for starting the simulator.

6) The **simulator window will open**, looking like one of the two pictures overleaf though larger.

⁹³ Sometimes, this step may terminate with an error due to conflicting local changes. The message reads “[Please commit or stash your changes before you merge](#)” (or a bad translation into your language). Then enter `git reset --hard` and try again thereafter.

⁹⁴ There may be files updated by `git pull` but no new build possible sometimes. Then `make` will throw a corresponding message.

There may be also other obstacles or error messages in compilation (do not care for warnings or notes); then `make rebuild` will clean the field before it starts a fresh compilation. True software errors will remain, however.

Operate the simulator with the mouse. The ten digits as well as $.$, **ENTER** \uparrow , $+$, $-$, \times , and $/$ may also be entered via the numeric keypad of your computer directly, \blacktriangle and \blacktriangledown via the cursor keys, and \blackleftarrow via [\leftarrow Backspace]. Further computer keyboard shortcuts to simulator keys are presented on next page.

(A vertical screen size of ≥ 980 px is required for the portrait window; else the landscape window will open needing 1000×568 px.
The lower picture shows an old calculator keyboard here.)





Right clicking will call **g**-shifted labels directly in any calculator mode.

Pressing ...

- ... **h** copies the entire simulator screen image to the clipboard.
- ... **x** copies the full content of **X** thereto.⁹⁶
- ... **z** copies the full contents of all 12 lettered *registers* thereto.
- ... **Z** copies the full contents of all 112 global *registers* thereto.

Current content of *register L* is shown top left in the simulator window. Instead of the low-battery indicator **L** making no sense on a computer application, 'SL' is displayed far right in the *status bar* whenever ASL is enabled (cf. *Section 1* of the OM).

⁹⁵ Capitals are printed green here for better differentiation.

⁹⁶ This is the way to output very long integer results $> 10^{296}$ in their full glory.

APPENDIX F: FLASHING AND UPDATING YOUR WP 43S

There are two ways to get your hands on a *WP 43S*:

1. You can buy a *WP 43S* off the shelf or
2. you can flash an existing *DM42* (or *DM41X*).

Way 2 allows you to repurpose a *DM42* (or *DM41X*) you own already, so you may save costs – but you will have to live with stickers on 17 keys at least then. This way is explained in next chapter.

The chapter thereafter (beginning on p. 216) shows how to update your existing *WP 43S*, be it bought or flashed, when a new firmware becomes available.

How to Create Your *WP 43S* by Flashing a *DM42*

1. Start your computer. Take a *DM42* and turn it on; then press



SETUP

- 5** System
- 2** Enter system menu
- 4** Reset to *DMCP* menu

Now connect your computer to the calculator *Micro USB* socket using a suitable data cable. Ensure it connects properly on the calculator side – cutting back the plastic isolation a bit may be necessary.

- 6** Activate *USB* disk. The flash disk of your *DM42* should show up as an external mass storage volume on your computer now.

2. Start the internet browser on your computer and go to https://gitlab.com/Over_score/wp43s/tree/master/DM42_binary.



Copy WP43S.pgm to
the *DM42* flash disk.

(Option: Copying also
keymap.bin to the *DM42*
flash disk will reassign keys
to match the *WP 43S* layout
also after leaving *WP 43S*.
Unless you have done this,
EXIT/ON stays bottom left.)

3. Press **2** **4** **3** **ENTER↑** **WP43S.pgm** **ENTER↑**. Wait some 15 s for flashing completed. Then press **EXIT** **1** **EXIT**. Now, your WP 43S is up and waiting for your commands.

As long as you rely on a converted DM42, the graphic files supplied in https://gitlab.com/Over_score/wp43s/-/tree/master/artwork may ease



To leave the **WP 43S** program, enter **MODE SYSTEM** to return to the *DMCP* system. If you chose the option above, the key assignments will stay as they were in **WP 43S** when navigating therein (due to *keymap.bin*); else **EXIT/ON** returns to the bottom left key now.

To retrieve the original **DM42 keyboard layout** (cf. p. 156), copy the file `original_DM42_keymap.bin` to the **DM42** flash disk, rename it `keymap.bin` and **RESET** the **DM42**. Look here for more information: <https://technical.swissmicros.com/dm42-devel/dmcp-devel-manual/>

How to Update Your **WP 43S**

If you have *Free42* still running on your **DM42** then proceed as demonstrated in previous chapter.

Else **WP 43S** is installed on your calculator already. Then:

1. Start your computer.
2. Take your calculator and turn it on. **SAVE** your programs and data. Then leave **WP 43S** by pressing **MODE** **SYSTEM** to return to the **DMCP** system. If you copied `keymap.bin` of **WP 43S** before last flashing, key assignments will stay as they were in your **WP 43S** when navigating in the **DMCP** system – else the **DM42** assignments will become valid (cf. p. 156).
3. Connect your computer to your calculator *Micro USB* socket using a suitable data cable (ensure it connects properly to the calculator – cutting the plastic isolation back a bit may be necessary). The flash disk of your calculator should show up as an external mass storage volume after you pressed ...

6 Activate *USB Disk*

4. Start the internet browser on your computer and go to
https://gitlab.com/Over_score/wp43s/tree/master/DM42%20binary.

Copy `WP43S.pgm` to the root directory of the **DM42** flash disk.

Option: Copy `keymap.bin` to the **DM42** flash disk if you have not done so far. This will reassign the keys to match the **WP 43S** layout also when leaving **WP 43S**. Else **ON** remains bottom left.

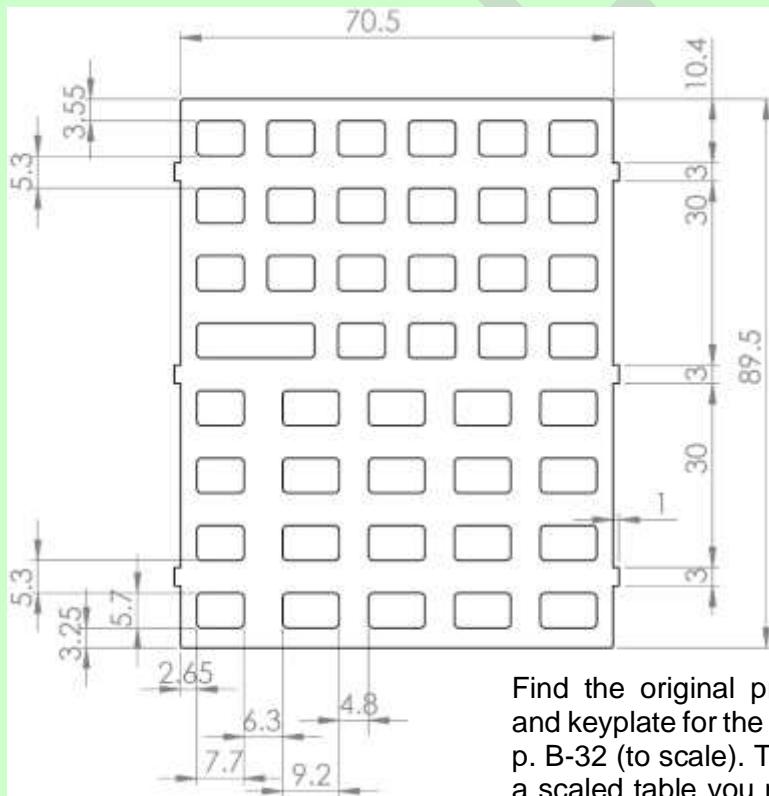
5. Press **EXIT** **ENTER↑** **3** (Load Program), use the cursor keys to select `WP43S.pgm`, and press **ENTER↑** **ENTER↑**.

6. Wait for flashing completed (~15 s). Then press **EXIT** **EXIT** **1** **+**. Recall your saved programs and data via **I/O LOAD**. You can resume your work with your updated *WP 43S* now.

Sometimes, you may need an update of the *DMCP* on your *WP 43S*; see https://technical.swissmicros.com/dm42/doc/dm42_user_manual/#DMCP_update_guide for how to do this.

Overlays

See here the drawing for a blank overlay. All dimensions are given in *millimeters*. Note the overall width is 72.5 mm.



Find the original print of keys and keyplate for the *WP 43S* on p. B-32 (to scale). There is also a scaled table you may use for creating your own overlay.

APPENDIX G: TROUBLESHOOTING GUIDE

Calculator Frozen

There are several ways to put your calculator in a freeze state wherein it will not react on any keys you press, even without flashing *WP 43S*. Usually, pressing the RESET button on its rear side should bring it back to life. If this does not work, however, the following should do:

1. Open your calculator by unfastening the two bolts at the top of its backside. You will find a printed circuit board (*PCB*) with its top probably looking like this ➔ (cf. p. 159 for an earlier *PCB*).

In any case, you will see two small buttons, one labeled **RESET**, the other one called **PGM** or **BOOT0**.



2. Now:
 - a. Press and hold the PGM (or BOOT0) button.
 - b. Press and release the RESET button.
 - c. Release the PGM (or BOOT0) button.

This shall reset your *DM42* and put it in bootloader mode.⁹⁷

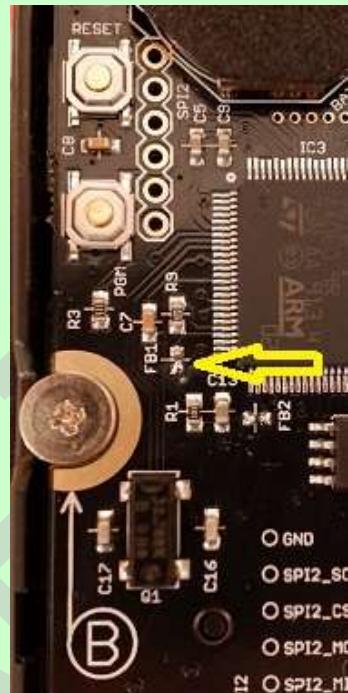
3. Then you can reflash your calculator using *dm_tool.exe* as explained in https://www.swissmicros.com/dm42/doc/dm42_user_manual/.

⁹⁷ If this method should not work, however, this may point to a real hardware problem. We recommend contacting SwissMicros then.

Fresh Battery Constantly Low

This was observed with factory-fresh calculators in 2020: The low-battery indicator  is lit and BATT? stubbornly returns 1.21 V, although the battery is actually good and measures over 3 V. If resetting the calculator, removing the battery, updating the firmware, etc. does not change the output of BATT?, open the calculator and inspect the PCB. There may be an improper soldering at FB1 (compare previous picture).

If you have the right tools and experience, feel free to fix this yourself – else contact SwissMicros.



Keymap Trouble

If you find you have loaded a `keymap.bin` not matching the layout you wanted or you do not find the keys properly assigned then proceed this way (assuming both calculator and computer running and connected):

1. Find where MODE went on the calculator keyboard and call it.
2. With MODE open, enter  **SYSTEM** to return to the *DMCP* system.
3. Press **6** (Activate *USB Disk*).
4. Start the internet browser on your computer and go to https://gitlab.com/Over_score/wp43s/tree/master/DM42%20binary.
5. Copy `original_DM42_keymap.bin` to the flash disk, rename it `keymap.bin` and **RESET** the *DM42*.
6. Press **EXIT**, then the bottom left key – else you will run into the same trouble again.

Then your *WP 43S* is up and waiting for your orders.

APPENDIX H: ADVANCED MATHEMATICAL FUNCTIONS AND TASKS

Your WP 43S contains several operations covering advanced mathematics. Most of them are taken over from WP 34S, some are implemented here for the first time on an RPN calculator. Find those functions collected here and described in more detail than in the *IOI*, together with a few traditional pocket calculator functions matching the topic.

For reasons explained in *Section 1*, we assume you are able to read and understand mathematical formulas for *real* and *complex* domain functions.

Ensure you understand the respective fundamental mathematical concepts; else leave these functions aside. By experience, it is only beneficial to use something you overview and know the background of – else it may even become dangerous for you and your fellow men.

Number Generating Functions

The following are all *monadic* functions except COMB and PERM.

Name	Remarks (see pp. 12ff for general information)
B_n , B_n^*	B_n returns the Bernoulli number for an integer $n > 0$ given in X : $B_n = (-1)^{n+1} \cdot n \cdot \zeta(1 - n)$ B_n^* works with the old definition instead: $B_n^* = 2 \cdot \frac{(2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$ See p. 259 for $\zeta(x)$.

Name	Remarks (see pp. 12ff for general information)
COMB, PERM	<p>For $y \geq x \geq 0$ and $x, y \in \mathbb{N}$, $C_{y,x} = \binom{y}{x} = \frac{y!}{x!(y-x)!}$ is the number of <i>combinations</i> and $P_{y,x} = \frac{y!}{(y-x)!} = x!C_{y,x}$ the number of <i>permutations</i> of x and y as explained in the IOP (see pp. 26 and 57, respectively).</p> <p>Note $C_{y,0} = 1$, $C_{y,1} = y$, and $C_{y,2} = \frac{1}{2}y(y-1)$.</p> <p>$C_{y,x}$ applies to the <i>binomial distribution</i> (see p. 222): In a <i>Galton box</i>⁹⁸ (a.k.a. <i>bean machine</i>) featuring y rows of pins and fed with 2^y balls, $C_{y,x}$ is the number of balls expected in column x of that box (start column counting with zero).</p> <p>Generally, $P_{y,x} = \frac{\Gamma(y+1)}{\Gamma(y-x+1)}$ and $C_{y,x} = \frac{\Gamma(y+1)}{\Gamma(x+1) \cdot \Gamma(y-x+1)}$ work also for non-integer numbers and in <i>complex domain</i>.</p>
FIB	<p>For integers, FIB returns the <i>Fibonacci</i> number f_n with $n = x$. The <i>Fibonacci</i> numbers are defined as $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$. With UNSIGN, f_{93} is the maximum before an overflow occurs. For <i>long integers</i>, f_{4791} is the maximum on your WP 43S.</p> <p>For non-integers, FIB returns the extended Fibonacci number</p> $F_x = \frac{1}{\sqrt{5}} [\Phi^x - \Phi^{-x} \cos(x\pi)]$ <p>for an arbitrary <i>real</i> or <i>complex number</i> x, with $\Phi = \frac{1+\sqrt{5}}{2}$ denoting the <i>golden ratio</i>.</p>

⁹⁸ Translator's note: This is called «Planche de Galton» in French, “Galtonbrett” in German, and “macchina di Galton” in Italian. Note the subtle differences in naming. Galton invented his box in 1889.

Statistical Distribution Functions (PMF, PDF, CDF, etc.)

Stack-wise, the following are all *monadic* functions, stored in PROB. Actually, they feature more parameters though. Those are supplied in the *registers* **I**, **J**, and **K** as applicable and mentioned below.

In the following text, the five **discrete distributions** are covered first, the eight continuous ones thereafter. Typical plots are shown for the *PMF's* or *PDF's*.

Binom: *Binomial* distribution with the *number of successes g* in **X**, the *gross probability of a success p₀* in **I** and the *sample size n* in **J**.

BINOM_P returns

$$p_B(g; n; p_0) = \binom{n}{g} p_0^g (1 - p_0)^{n-g} = C_{n,g} p_0^g (1 - p_0)^{n-g} \quad (\text{see COMB on p. 221 for the explanation of the notation}).$$

BINOM_▲ returns $F_B(m; n; p_0) = \sum_{g=0}^m p_B(g; n; p_0)$ with the maximum number of successes **m** in **X**.

The *binomial* distribution is fundamental for error statistics in industrial sampling, e.g. for designing test plans.

Example: What is the probability for finding no faulty item in a sample of 15 items drawn from a batch of 300 wherein you expect 3% defective items overall? This will tell you:

.03 [STO] J 15 [STO] K 0 [PROB] g [Binom: Binom_▲]

returning 0.633 – so the odds are almost two out of three that you

will not detect any defect in your sample! ⁹⁹

Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm>.

Geom: Geometric distribution:

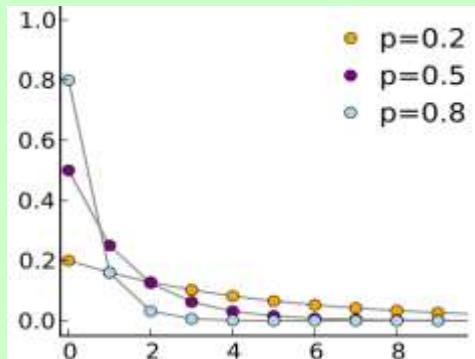
GEOM_P returns

$$p_{Ge}(n) = p_0(1 - p_0)^n$$

GEOM_M returns

$$F_{Ge}(m) = 1 - (1 - p_0)^{m+1}, \text{ being}$$

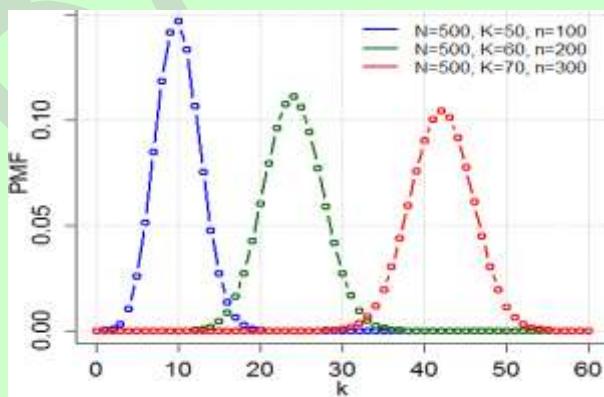
the probability for a first success after $m = x$ Bernoulli experiments. The probability p_0 for a success in each such experiment must be specified in I.



Start reading here for more:

http://en.wikipedia.org/wiki/Geometric_distribution.

Hyper: Hypergeometric distribution with the number of successes g in X, gross probability of a success p_0 in I, sample size n in J, and batch size n_0 in K (in the diagram, $g=k$, $p_0=K/N$, and $n_0=N$).



⁹⁹ The exact result for said boundary conditions is 0.626, calculated using the hypergeometric distribution. These results show nicely that two significant digits are a typical accuracy of theoretical statistical statements – frequently the (often simplified) statistical model used matches reality no better than that.

HYPERP returns $p_H(g; n; p_0; n_0) = \frac{\binom{n_0 p_0}{g} \cdot \binom{n_0(1-p_0)}{n-g}}{\binom{n_0}{n}}$ (see COMB on p. 221 for the explanation of the notation).

While the *binomial* distribution assumes that each sample part is returned to the batch after checking, the *hypergeometric* distribution lets you keep your samples out of the batch. This is found more often in real life, but may be neglected in so-called 'large' batches ($n_0 > 10$) and for small sample sizes (<10% of n_0). Start reading here for more: http://en.wikipedia.org/wiki/Hypergeometric_distribution.

NBin: Negative binomial distribution with the *total number of failures f* (in n draws) in **X**, the *gross probability of a success in a single draw* p_0 in **I**, and n in **J**.

NBINP returns $p_{NB}(f; n; p_0) = \binom{n-1}{f-1} \times p_0^f \times (1-p_0)^{n-f} = C_{n-1; f-1} \times p_0^f \times (1-p_0)^{n-f}$ (s. COMB on p. 221 and cf. BINOM).

Start reading here for more:

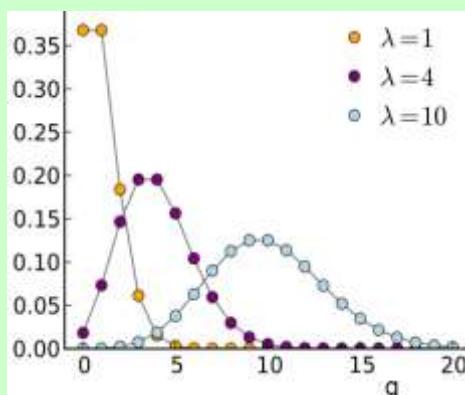
http://en.wikipedia.org/wiki/Negative_binomial_distribution.

Poiss: Poisson distribution with the *number of successes g* in **X** and the *Poisson parameter λ* in **J**.

POISSP computes

$$p_P(g; \lambda) = \frac{\lambda^g}{g!} e^{-\lambda}$$

and POISS returns the corresponding *CDF* for the maximum number of successes m in **X**.



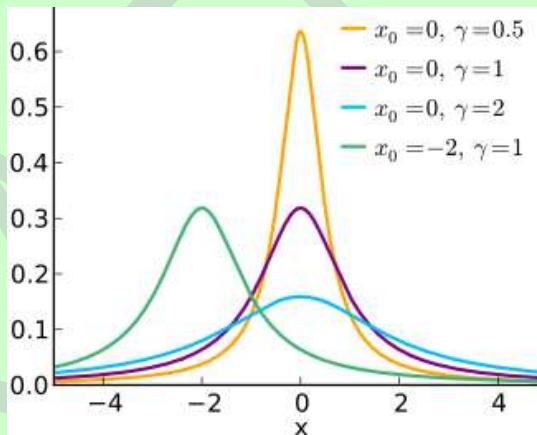
The *Poisson* distribution provides the mathematically simplest model for industrial sampling tests – use $\lambda = np_0$ with the gross error probability p_0 and the sample size n (cf. BINOM). For the example introduced with BINOM above, POISS returns 0.638.

Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm>.

Continuous distributions:

Cauch: Cauchy-Lorentz distribution (also known as Lorentz or Breit-Wigner distribution) with the *location* x_0 specified in **I** and the *shape* γ in **J**.



CAUCH_P returns $f_{Ca}(x) = \left\{ \pi\gamma \cdot \left[1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right] \right\}^{-1}$,

CAUCH_A returns $F_{Ca}(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x - x_0}{\gamma}\right)$,

CAUCH⁻¹ returns $F_{Ca}^{-1}(p) = x_0 + \gamma \tan\left[\pi \cdot \left(p - \frac{1}{2}\right)\right]$.

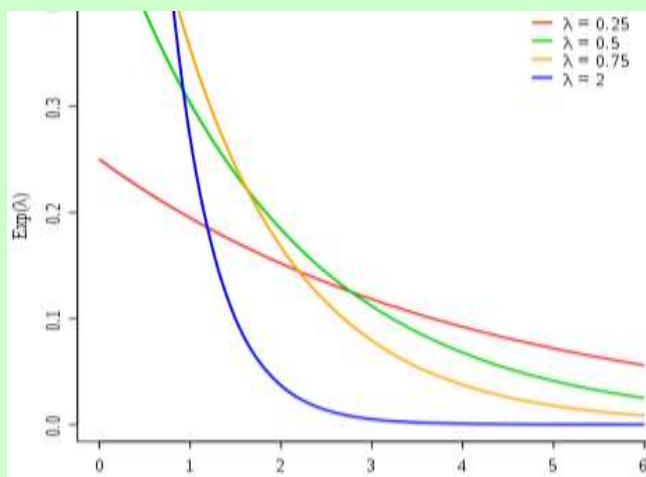
This distribution is quite popular in physics. It is a special case of Student's *t* distribution. Start reading here for more:

http://en.wikipedia.org/wiki/Cauchy_distribution.

Expon: Exponential distribution with the rate λ in **I**.

EXPON_P returns $f_{Ex}(x) = \lambda e^{-\lambda x}$.

EXPON_{Δ} returns $F_{Ex}(x) = 1 - e^{-\lambda x}$.



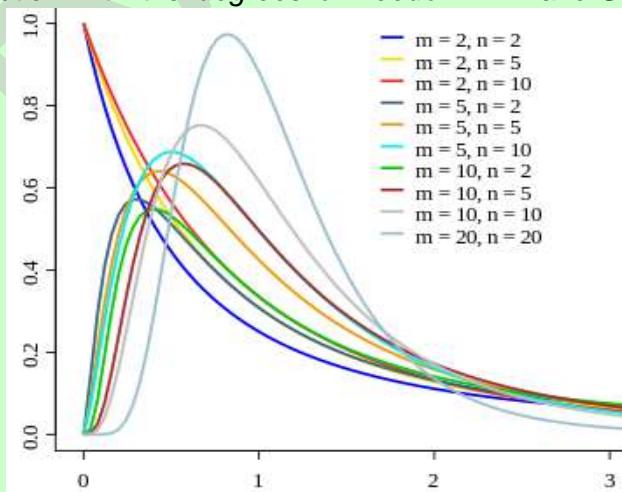
Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3667.htm>

F(x): Fisher's F distribution with the degrees of freedom in **I** and **J**.

It is used e.g. for analyses of variance (ANOVA).

The graph presents the PDFs plotted for different degrees of freedom **m** and **n** corresponding to *i* and *j*.



Read here for more information:

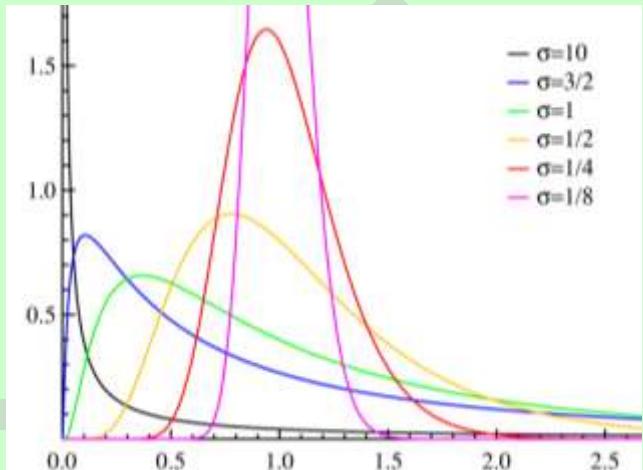
<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3665.htm>

LgNrm: Log-normal distribution with the parameters $\mu = \ln \bar{x}_g$ in **I** and $\sigma = \ln \varepsilon$ in **J** (see some PDF plots below).

$$\text{LGNRM}_{\text{P}} \text{ returns } f_{Ln}(x) = \frac{1}{x \sigma \sqrt{2\pi}} e^{-\frac{[\ln(x)-\mu]^2}{2\sigma^2}}.$$

LGNRM_A: returns $F_{Ln}(x) = \Phi\left(\frac{\ln(x)-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized*

normal CDF as presented on p. 228.



Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3669.htm>

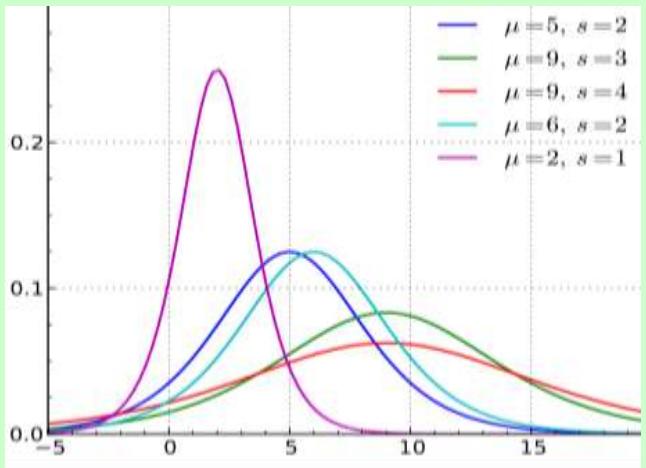
Logis: Logistic distribution with an arbitrary *mean* μ given in **I** and a *scale parameter* s in **J**.

$$\text{Substituting } \xi = \frac{x-\mu}{s},$$

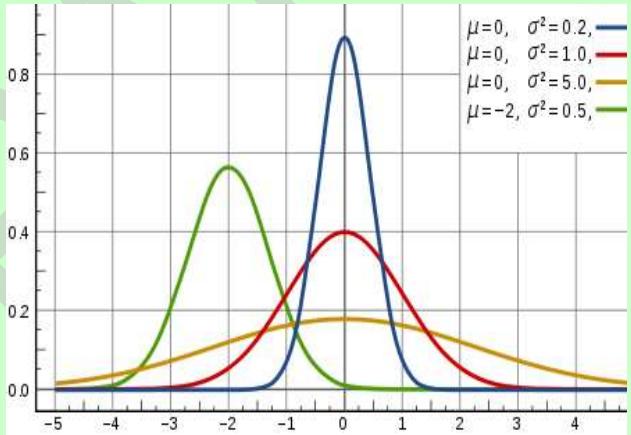
$$\text{LOGIS}_{\text{P}} \text{ returns } f_{Lg}(x) = \frac{e^{-\xi}}{(1+e^{-\xi})^2 s} \text{ (plotted overleaf) and}$$

$$\text{LOGIS}_{\text{A}} \text{ returns } F_{Lg}(x) = \frac{1}{1+e^{-\xi}}.$$

$$\text{LOGIS}^{-1} \text{ returns } F_{Lg}^{-1}(p) = \mu + s \ln\left(\frac{p}{1-p}\right).$$



Normal: Normal distribution with an arbitrary mean μ given in I and an arbitrary standard deviation σ in J. The red curve (for $\mu=0$ and $\sigma=1$) represents the *standardized normal* (a.k.a. Gaussian) distribution φ .



NORML_P returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \varphi\left(\frac{x-\mu}{\sigma}\right)$ and

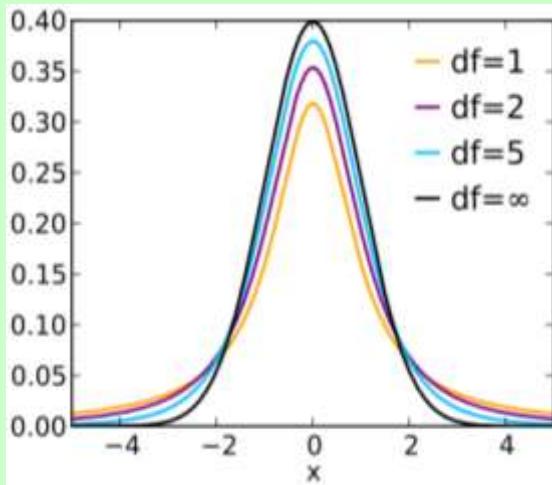
NORML_A returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized normal CDF* (cf. the *error function* on p. 255).

Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3661.htm>

t(x): Standardized Student's *t*distribution with its *degrees of freedom* in **I**.

It is used for hypothesis testing and calculating confidence intervals e.g. for means. The diagram shows its *PDFs* plotted for different *degrees of freedom*. For $df \rightarrow \infty$, the shoulders of $t(x)$ shrink and it approaches the *PDF* of the *standard normal distribution* (compare the red *Gaussian* curve at NORML on p. 228).



Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3664.htm>

Weibl: Weibull distribution with its *shape parameter* **b** in **I** and its *characteristic lifetime* **T** in **J**.

WEIBL_P returns $f_W(t) = \frac{b}{T} \cdot \left(\frac{t}{T}\right)^{b-1} e^{-(t/T)^b}$ for $t \geq 0$, else 0. This is a very flexible function – see the curves plotted overleaf.

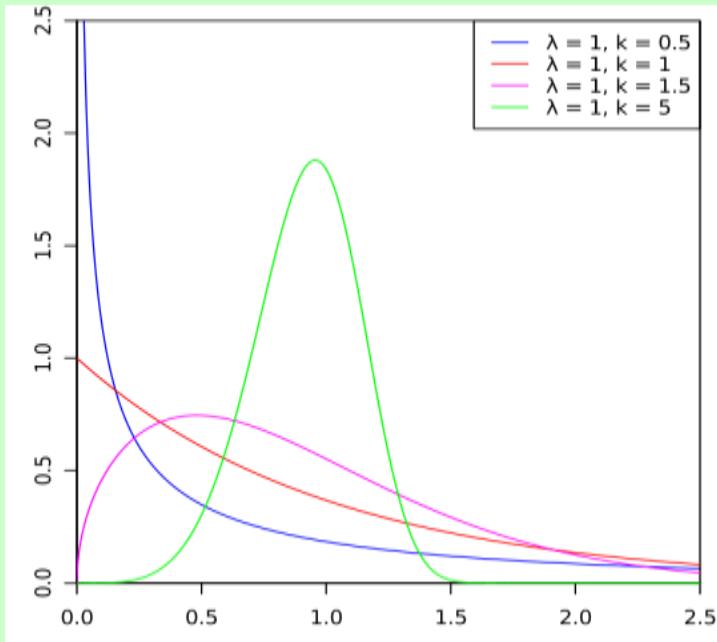
WEIBL_A returns $F_W(t) = 1 - e^{-(t/T)^b}$

This distribution is widely used e.g. for analyzing tool and product lifetimes.

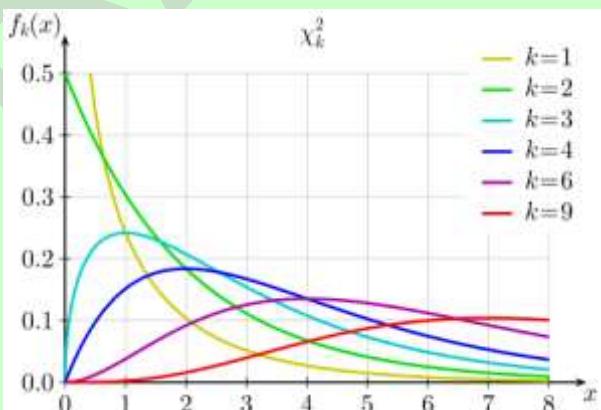
Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm>

You may even find some more application fields mentioned in https://en.wikipedia.org/wiki/Weibull_distribution#Applications.



$\chi^2(\mathbf{x})$: Chi-square distribution with its *degrees of freedom* given in \mathbf{I} . It is used for calculating confidence intervals for *standard deviations*, *variances*, *process and machine capabilities*, and the like. The graph shows PDF's for different degrees of freedom.



Read here for more information:

<http://www.itl.nist.gov/div898/handbook/eda/section3/eda3666.htm>

More Statistical Formulas, also for Curve Fitting

The following equations are for data measured at samples of n specimens (i.e. n is the *sample size*). Note that a complete measurement result must include both: information about the expected value and about its uncertainty.

- For samples drawn out of a *normally* distributed (additive) process, the expected value is the *arithmetic mean* (or *average*) of the sample values and its uncertainty is given by its *standard error* (see \bar{x} and s_m).
- For samples drawn out of a *log-normally* distributed (multiplicative) process, the expected value is the *geometric mean* and its uncertainty is given by its *scattering factor* (see \bar{x}_g and ε_m).
- For samples drawn out of other kinds of processes other measures apply.

Generally, the statistical model shall be chosen that matches observations best – within their statistical errors.¹⁰⁰ Be assured not everything is *normal* or *Gaussian* in real world.¹⁰¹ Process characteristics can be detected (and should be checked well in advance of calculating e.g. means) using suitable tests – turn to applicable statistical reference literature.

¹⁰⁰ In real-life cases, dramatic deviations from the model distribution are frequently found – then you cannot expect the calculated consequences matching reality any better.

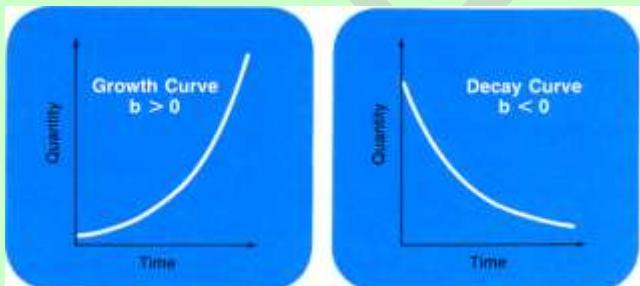
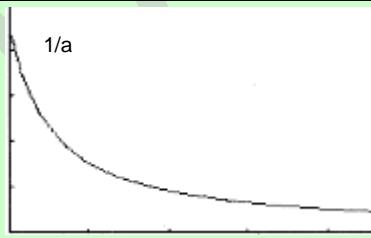
As mentioned in the main text, we recommend you look deeply into statistics textbooks to ensure you fully understand what you do with the functions provided in your WP 43S. The real world shows lots of sad examples where people full of good will caused large damages by applying tools they did not know sufficiently – or applied standard tools in areas where those are not applicable. “*Wenn Dumme fleißig werden, wird's gefährlich*” (i.e. ~ “*It's getting dangerous with fools becoming busy*”), a former boss of mine used to say (compare also D.T. recently).

¹⁰¹ Since the *PDF* of a *normal* (a.k.a. *Gaussian*) distribution will never reach zero, this statistical model tells you to expect individual items far, far away from the mean value when your sample becomes large enough. This, however, does not match reality. So we must conclude nothing at all is really *Gaussian* in real world. Nevertheless, the *Gaussian* distribution is a very successful model describing a lot of real-world observations very well. Just never forget the limits of such models.

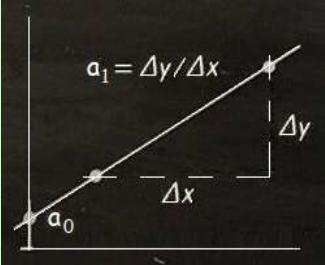
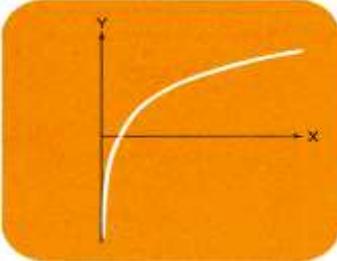
The following functions as named in the left column (sorted alphabetically) are all found in STAT:

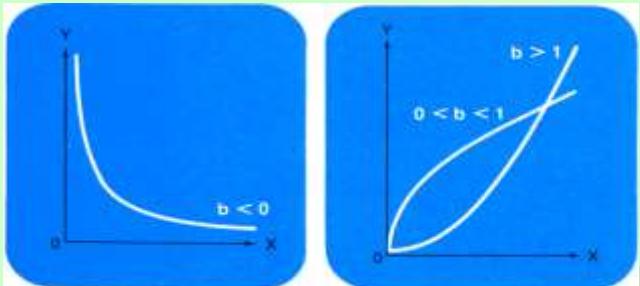
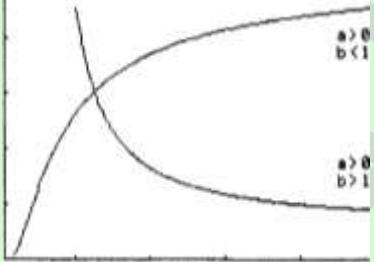
Name	Remarks (see pp. 12ff for general information)
CauchF	Selects a <i>Cauchy</i> (a.k.a. <i>Lorentz</i> , <i>Breit-Wigner</i>) peak fit model $R(x) = \frac{1}{[a_0(x + a_1)^2 + a_2]}$ for least squares regression. ¹⁰² See p. 225 for shapes of such peaks.
CORR	For any set of data points (x_i, y_i) , the <i>coefficient of correlation</i> is $r = \frac{s_{xy}}{s_x s_y}$. See s_{XY} and s below. For an arbitrary fit model $R(x)$, $r^2 = 1 - \frac{\sum [R(x_i) - y_i]^2}{\sum (\bar{y} - y_i)^2}$ is its <i>coefficient of determination</i> indicating the fraction of the variation of the dependent data y determined by the variation of the independent data x . For $r^2 = 1$, y is fully determined by x ; for $r^2 = 0$, y is completely independent of x ; and e.g. $r^2 = 0.85$ means 85% of the variation of y is due to x . Note BESTF picks the fit model showing the maximum r^2 out of the models allowed. A two-parameter regression (like the majority of the fit models provided on your WP 43S) is said being (statistically) <i>significant</i> at a 99% <i>confidence level</i> if $\sqrt{\frac{r^2}{1 - r^2}(n - 2)} > t_{n-2}^{-1}(0.99)$ with the right side being the inverse of the t distribution for the <i>degrees of freedom</i> $n - 2$ (see p. 229).

¹⁰² Note that *least squares regression* is best for data point errors in vertical direction y being significantly greater than the errors in horizontal direction x . See pp. 232ff for the formulas and more about the fit models provided.

Name	Remarks (see pp. 12ff for general information)
COV, S _{XY}	<p>For any set of data points (x_i, y_i), the <i>population covariance</i> is</p> $COV_{xy} = \frac{1}{n^2} \left(n \sum x_i y_i - \sum x_i \sum y_i \right)$ <p>and the <i>sample covariance</i> is</p> $s_{xy} = \frac{1}{n(n-1)} \left(n \sum x_i y_i - \sum x_i \sum y_i \right).$
ExpF	<p>Selects the exponential curve fit model $R(x) = a_0 e^{a_1 x}$ for least squares regression.¹⁰² Generally, this will be a good choice if the measured data follow the shape of one of the two curves pictured here (think of human population growth or nuclear decay, for example).¹⁰³</p> 
GaussF	<p>Selects a <i>Gauss peak</i> fit model $R(x) = a_0 e^{\frac{(x-a_1)^2}{a_2}}$ for least squares regression.¹⁰² See p. 228 for the shapes of such peaks.</p>
HypF	 <p>Selects the hyperbolic fit model $R(x) = 1/(a_0 + a_1 x)$ for least squares regression.¹⁰²</p>

¹⁰³ Color plots on this and the next page are taken from the HP-27 manual; therein, a equals a_0 and b equals a_1 , on your WP 43S.

Name	Remarks (see pp. 12ff for general information)
LinF	 <p>Selects the linear fit model $R(x) = a_0 + a_1 x$ for least squares regression.¹⁰² Generally, this will be a good choice if the measured data follow a straight line, raising or falling (but compare ORTHOF).</p>
LogF	 <p>Selects the logarithmic curve fit model $R(x) = a_0 + a_1 \ln(x)$ for least squares regression.¹⁰² Generally, this will be a good choice if the measured data follow a curve looking like drawn at left.</p>
L.R.	<p>Uses the fit model selected and computes the two or three parameters of the regression for the data accumulated.</p> <p>For all curve fit models provided on your WP 43S, a regression parameter is (statistically) <i>significant</i> at a 99% <i>confidence level</i> if</p> $\left \frac{a_i}{s(a_i)} \right > t_{n-2}^{-1}(0.995),$ <p>with the right side being the inverse of the <i>t</i> distribution for the <i>degrees of freedom</i> $n - 2$ (cf. p. 229).</p>
OrthoF	<p>Selects the linear fit model $R(x) = a_0 + a_1 x$ like LINF but assuming data point errors in x are equal to those in y (precisely: their variances are equal). The sum of squared distances of the data points to the fit line will be minimized. This model is called <i>orthogonal regression</i>. See pp. 237ff for more and the OM for application examples.</p>
ParabF	<p>Selects a parabolic fit model $R(x) = a_0 + a_1 x + a_2 x^2$ for least squares regression.¹⁰²</p>

Name	Remarks (see pp. 12ff for general information)
PowerF	Selects the power curve fit model $R(x) = a_0 x^{a_1}$ for least squares regression. ¹⁰² Generally, this will be a good choice if measured data follow the shape of one of the curves pictured here (look for Tower of Pisa in the OM). ¹⁰³ 
RootF	 Selects the root curve fit model $R(x) = a b^{1/x} = a_0 a_1^{1/x}$ for a least squares regression. ¹⁰²
s, s_m	The <i>sample standard deviation (SD)</i> is the positive square root of the <i>sample variance</i> $s_x^2 = \frac{1}{n(n-1)} \left[n \sum x_i^2 - \left(\sum x_i \right)^2 \right] = \frac{1}{n-1} \left(\sum x_i^2 - n \bar{x}^2 \right)$ And the <i>standard error</i> (i.e. the SD of the mean \bar{x}) is $s_m = s / \sqrt{n}$
s_w, s_{mw}	The <i>sample SD for weighted data</i> (where the weight y_i of each data point x_i was entered via $\Sigma+$) is $s_w = \sqrt{\frac{\sum y_i \sum y_i x_i^2 - (\sum y_i x_i)^2}{\sum y_i (\sum y_i - 1)}}$

Name	Remarks (see pp. 12ff for general information)
	And the corresponding <i>standard error</i> (the <i>SD</i> of the <i>mean</i> \bar{x}_w) is $s_{mw} = \frac{1}{\sum y_i} \sqrt{\frac{\sum y_i \sum y_i x_i^2 - (\sum y_i x_i)^2}{\sum y_i - 1}}$
s_{XY}	See COV above.
\bar{x}	The <i>arithmetic mean</i> is calculated as $\bar{x} = \frac{1}{n} \sum x_i$
\bar{x}_G	The <i>geometric mean</i> is calculated as $\bar{x}_G = \sqrt[n]{\prod x_i} = e^{[\frac{1}{n} \sum \ln(x_i)]}$
\bar{x}_H	The <i>harmonic mean</i> is calculated as $\bar{x}_H = n / \sum \frac{1}{x_i}$
\bar{x}_{RMS}	The <i>quadratic mean</i> is calculated as $\bar{x}_{RMS} = \sqrt{\frac{1}{n} \sum x_i^2}$
\bar{x}_w	The <i>arithmetic mean</i> for <u>weighted</u> data (where the weight y_i of each data point x_i was entered via $\Sigma+$) is $\bar{x}_w = \sum x_i y_i / \sum y_i$
ε	The <i>scattering factor</i> ε_x for a sample of <i>log-normally</i> distributed data is calculated via: $\ln(\varepsilon_x) = \sqrt{\frac{1}{n-1} \left[\sum \ln^2(x_i) - 2n \ln(\bar{x}_G) \right]}$ Compare s.

Name	Remarks (see pp. 12ff for general information)
ε_m	The <i>scattering factor</i> ε_m of the <i>geometric mean</i> (compare s_m) is $\varepsilon_m = \varepsilon^{1/\sqrt{n}}$
ε_p	The <i>scattering factor</i> ε_p for a population of <i>log-normally</i> distributed data is calculated via: $\ln(\varepsilon_p) = \sqrt{\frac{n-1}{n}} \ln(\varepsilon)$ Compare σ .
σ	The <i>SD</i> of a population of <i>normally</i> distributed data is calculated via $\sigma = \sqrt{\frac{n-1}{n}} s$
σ_w	The <i>SD</i> of the population for <u>weighted</u> data (where the weight y_i of each data point x_i was entered via $\Sigma +$) is $\sigma_w = \sqrt{\frac{\sum y_i (x_i - \bar{x}_w)^2}{\sum y_i}}$

Curve Fitting Models Provided

Actually, a proper linear regression is computed for LINF and ORTHOF only. For the other three standard models (EXPF, LOGF, and POWERF) the same method is applied to transformed data. Your data might follow a straight line if you plot...

- the logarithm of your **y**-data over your **x**-data (then EXPF will fit);
- the logarithm of your **y**-data over the logarithm of your **x**-data (then POWERF will fit);

- your y -data over the logarithm of your x -data (then LOGF will fit).

This is what your *WP 43S* does when you enter statistical data points and compute the parameters of a fit curve thereafter:

1. It accumulates the 22 sums listed on pp. 87ff and increments the number of data points n . Some of these 22 sums may turn non-numeric for negative entries – just do not care.
2. The subsequent evaluation will depend on the fit model you select (cf. pp. 233ff):
 - a. If you choose LINF then the least squares regression line parameters a_0 and a_1 will be computed following the formulas:

$$a_0 = \frac{\sum x_i^2 \cdot \sum y_i - \sum x_i \cdot \sum x_i y_i}{n \sum x_i^2 - (\sum x_i)^2}$$

$$a_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2} = \frac{s_{xy}}{s_x^2} = r \frac{s_y}{s_x}$$

Their *standard errors* can be calculated using the formulas

$$s(a_1) = \frac{s_y}{s_x} \sqrt{\frac{1-r^2}{n-2}} \quad \text{and} \quad s(a_0) = s(a_1) \cdot \sqrt{\frac{n-1}{n} s_x^2 + \bar{x}^2} \quad \text{with}$$

$$r = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \cdot \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

The so-called *standard error of estimate*, which may be helpful for testing the slope and forecasts (see pp. 97ff in the *HP-21S OM*), is computed using

$$s_{y|x} = \sqrt{\frac{n-1}{n-2} (s_y^2 - a_1^2 s_x^2)}$$

- b. If you choose EXPF then the least squares regression line parameters for the transformed data $x_i, \ln(y_i)$ will be computed using

$$a_{0,tEXP} = \frac{\sum x_i^2 \cdot \sum \ln(y_i) - \sum x_i \cdot \sum x_i \ln(y_i)}{n \sum x_i^2 - (\sum x_i)^2}$$

$$a_{1,tEXP} = \frac{n \sum x_i \ln(y_i) - \sum x_i \sum \ln(y_i)}{n \sum x_i^2 - (\sum x_i)^2}$$

$$r_{tEXP} = \frac{n \sum x_i \ln(y_i) - \sum x_i \sum \ln(y_i)}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \cdot \sqrt{n \sum \ln^2(y_i) - [\sum \ln(y_i)]^2}}$$

The standard errors of $a_{0,tEXP}$ and $a_{1,tEXP}$ can be calculated using the formulas for LINF on p. 238 with the transformed results.

The parameters of the fit curve $R(x) = a_0 e^{a_1 x}$ turn out being $a_0 = e^{a_{0,tEXP}}$ and $a_1 = a_{1,tEXP}$.

- c. If you choose POWERF then the least squares regression line parameters for the transformed data $\ln(x_i), \ln(y_i)$ will be computed in analogy to the method shown for EXPF. Thus they will be

$$a_{0,tPOW} = \frac{\sum \ln^2(x_i) \cdot \sum \ln(y_i) - \sum \ln(x_i) \cdot \sum \ln(x_i) \ln(y_i)}{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$a_{1,tPOW} = \frac{n \sum \ln(x_i) \ln(y_i) - \sum \ln(x_i) \sum \ln(y_i)}{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$r_{tPOW} = \frac{n \sum \ln(x_i) \ln(y_i) - \sum \ln(x_i) \sum \ln(y_i)}{\sqrt{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2} \cdot \sqrt{n \sum \ln^2(y_i) - [\sum \ln(y_i)]^2}}$$

The standard errors of $a_{0,tPOW}$ and $a_{1,tPOW}$ can be calculated using the formulas for LINF on p. 238 with the transformed results.

The parameters of the fit curve $R(x) = a_0 x^{a_1}$ turn out being $a_0 = e^{a_{0,tPOW}}$ and $a_1 = a_{1,tPOW}$.

- d. If you choose LOGF then the least squares regression line parameters for the transformed data $\ln(x_i)$, y_i will be computed in analogy to the method shown for EXPF. Thus they will be

$$a_{0,tLOG} = \frac{\sum \ln^2(x_i) \cdot \sum y_i - \sum \ln(x_i) \cdot \sum y_i \ln(x_i)}{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$a_{1,tLOG} = \frac{n \sum y_i \ln(x_i) - \sum \ln(x_i) \sum y_i}{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$r_{tLOG} = \frac{n \sum y_i \ln(x_i) - \sum \ln(x_i) \sum y_i}{\sqrt{n \sum \ln^2(x_i) - [\sum \ln(x_i)]^2} \cdot \sqrt{n \sum y_i^2 - [\sum y_i]^2}}$$

The standard errors of $a_{0,tLOG}$ and $a_{1,tLOG}$ can be calculated using the formulas for LINF on p. 238 with the transformed results.

The parameters of the fit curve $R(x) = a_0 + a_1 \ln(x)$ are just $a_0 = a_{0,tLOG}$ and $a_1 = a_{1,tLOG}$.

- e. If you choose HYPF then the parameters of the least squares regression curve $R(x) = \frac{1}{a_0 + a_1 x}$ are computed to be

$$a_{0,HYP} = \frac{\sum x_i^2 \cdot \sum \frac{1}{y_i} - \sum x_i \cdot \sum \frac{x_i}{y_i}}{n \sum x_i^2 - (\sum x_i)^2} \text{ and } a_{1,HYP} = \frac{n \sum \frac{x_i}{y_i} - \sum x_i \cdot \sum \frac{1}{y_i}}{n \sum x_i^2 - (\sum x_i)^2}$$

$$r_{HYP}^2 = \frac{a_{0,HYP} \sum \frac{1}{y_i} + a_{1,HYP} \sum \frac{x_i}{y_i} - \frac{1}{n} \left(\sum \frac{1}{y_i} \right)^2}{\sum \frac{1}{y_i^2} - \frac{1}{n} \left(\sum \frac{1}{y_i} \right)^2}$$

- f. If you choose ROOTF then the least squares regression curve parameters will be computed using

$$A = n \sum \frac{1}{x_i^2} - \left(\sum \frac{1}{x_i} \right)^2$$

$$B = \frac{1}{A} \left[\sum \frac{1}{x_i^2} \cdot \sum \ln(y_i) - \sum \frac{1}{x_i} \cdot \sum \frac{\ln(y_i)}{x_i} \right]$$

$$C = \frac{1}{A} \left[n \sum \frac{\ln(y_i)}{x_i} - \sum \frac{1}{x_i} \cdot \sum \ln(y_i) \right]$$

The parameters of the fit curve $R(x) = a_0 a_1^{1/x}$ turn out being just $a_{0,t\sqrt{-}} = e^B$ and $a_{1,t\sqrt{-}} = e^C$.

$$r_{t\sqrt{-}}^2 = \frac{B \sum \ln(y_i) + C \sum \frac{\ln(y_i)}{x_i} - \frac{1}{n} [\sum \ln(y_i)]^2}{\sum [\ln(y_i)]^2 - \frac{1}{n} [\sum \ln(y_i)]^2}$$

g. If you choose PARABF then the least squares regression curve parameters will be computed using

$$\begin{aligned} A &= n \sum x_i^2 - (\sum x_i)^2, & B &= n \sum x_i^2 y_i - \sum x_i^2 \cdot \sum y_i, \\ C &= n \sum x_i^3 - \sum x_i^2 \cdot \sum x_i, & D &= n \sum x_i y_i - \sum x_i \cdot \sum y_i, \\ E &= n \sum x_i^4 - (\sum x_i^2)^2 \end{aligned}$$

The parameters of the fit curve $R(x) = a_0 + a_1 x + a_2 x^2$ will then be

$$a_{2,PAR} = \frac{A B - C D}{A E - C^2}, \quad a_{1,PAR} = \frac{D - a_2 C}{A},$$

$$\text{and } a_{0,PAR} = \frac{1}{n} \left(\sum y_i - a_{2,PAR} \sum x_i^2 - a_{1,PAR} \sum x_i \right).$$

$$\text{And } r_{PAR}^2 = \frac{a_{0,PAR} \sum y_i + a_{1,PAR} \sum x_i y_i + a_{2,PAR} \sum x_i^2 y_i - \frac{1}{n} (\sum y_i)^2}{\sum y_i^2 - \frac{1}{n} (\sum y_i)^2}$$

- h. If you choose GAUSSF then the least squares regression curve parameters will be computed using the auxiliary terms A, B, C, D, and E exactly as for PARABF. Furthermore,

$$F = \frac{A B - C D}{A E - C^2}, \quad G = \frac{D - F C}{A},$$

$$\text{and } H = \frac{1}{n} \left(\sum \ln(y_i) - F \sum x_i^2 - G \sum x_i \right).$$

The parameters of the fit curve $R(x) = a_0 e^{(x-a_1)^2/a_2}$ will then be

$$a_{2,GAU} = \frac{1}{F}, \quad a_{1,GAU} = -\frac{G}{2} a_{2,GAU} \quad \text{and} \quad a_{0,GAU} = e^{H - F a_{1,GAU}^2}.$$

$$r_{GAU}^2 = \frac{H \sum \ln(y_i) + G \sum x_i \ln(y_i) + F \sum x_i^2 \ln(y_i) - \frac{1}{n} [\sum \ln(y_i)]^2}{\sum [\ln(y_i)]^2 - \frac{1}{n} [\sum \ln(y_i)]^2}$$

- i. If you choose CAUCHF then the least squares regression curve parameters will be computed using the auxiliary terms A and E exactly as in PARABF. The other terms will be

$$B = n \sum \frac{x_i^2}{y_i} - \sum x_i^2 \cdot \sum \frac{1}{y_i}$$

$$C = n \sum x_i^3 - \sum x_i \cdot \sum x_i^2$$

$$D = n \sum \frac{x_i}{y_i} - \sum x_i \cdot \sum \frac{1}{y_i}$$

F and G will be calculated as for GAUSSF but with the components computed here; and

$$H = \frac{1}{n} \left(\sum \frac{1}{y_i} - R_{12} \sum x_i - R_{13} \sum x_i^2 \right)$$

The fit curve $R(x) = \frac{1}{[a_0(x + a_1)^2 + a_2]}$ will be specified by:

$$a_{0,CAU} = F, \quad a_{1,CAU} = \frac{G}{2a_0}, \quad \text{and} \quad a_{2,CAU} = H - F a_1^2.$$

$$r_{CAU}^2 = \frac{H \sum \frac{1}{y_i} + G \sum \frac{x_i}{y_i} + F \sum \frac{x_i^2}{y_i} - \frac{1}{n} \left(\sum \frac{1}{y_i} \right)^2}{\sum \left(\frac{1}{y_i} \right)^2 - \frac{1}{n} \left(\sum \frac{1}{y_i} \right)^2}$$

- j. If you choose BESTF then the correlation coefficient will be computed with your data for model a and with the transformed data for models b through i, if allowed (cf. the /OI). The model delivering the greatest absolute r value will be selected.
- k. If you choose ORTHOF then the least squares regression line parameters a_0 and a_1 will be computed following the formulas:

$$a_1 = \frac{1}{2s_{xy}} \left[s_y^2 - s_x^2 \pm \sqrt{(s_y^2 - s_x^2)^2 + 4s_{xy}^2} \right] \quad \text{and} \quad a_0 = \bar{y} - a_1 \bar{x}$$

The other formulas can be taken from model a (i.e. from LINF).

The output of L.R. is formatted like this, allowing for up to 12 significant digits displayed for each model parameter:

Linear	$a_1 = -312.345\ 678\ 901$
$y = a_0 + a_1 x$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$
Exponential	$a_1 = -12.345\ 678\ 901$
$y = a_0 e^{(a_1 x)}$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$
Power	$a_1 = -12.345\ 678\ 901$
$y = a_0 x^{a_1}$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$
Logarithmic	$a_1 = -12.345\ 678\ 901$
$y = a_0 + a_1 \ln(x)$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$
Hyperbolic	$a_1 = -12.345\ 678\ 901$
$y = (a_0 + a_1 x)^{-1}$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Root	$a_1 = -12.345\ 678\ 901$
$y = a_0 a_1^{(1/x)}$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Parabolic	$a_2 = -2.345\ 678\ 901\ 2$
$y =$	$a_1 = -12.345\ 678\ 901$
$a_0 + a_1 x + a_2 x^2$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Gauss peak	$a_2 = -2.345\ 678\ 901\ 2$
$\ln(y) =$	$a_1 = -12.345\ 678\ 901$
$a_0 + (x - a_1)^2 / a_2$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Cauchy peak	$a_2 = -2.345\ 678\ 901\ 2$
$1/y =$	$a_1 = -12.345\ 678\ 901$
$a_0 (a_1 + x)^2 + a_2$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Orthogonal	$a_1 = -312.345\ 678\ 901$
$y = a_0 + a_1 x$	$a_0 = -1.234\ 567\ 8 \times 10^{-3}$

Error Propagation in Calculations

Experimental data are always attended with errors (cf. footnote 45), caused by e.g. the uncertainty of the measuring method, the instrument used, and/or environmental variations. Even under controlled environmental and measuring conditions, random errors remain. These errors must be taken into account for a proper estimation of the uncertainty of your results computed using those experimental data. For about 200 years, *Gauss' least squares method* can be employed for this task.

Assume you know that your result R depends on several experimental parameters x_1 through x_n . Each such parameter x_i has an uncertainty or error Δx_i . Now, if $R = f(x_1, \dots, x_n)$ then its error is

$$\Delta R = f(x_1 \pm \Delta x_1, \dots, x_n \pm \Delta x_n) - f(x_1, \dots, x_n)$$

$$= \pm \sqrt{\left(\frac{df}{dx_1}\right)^2 \Delta x_1^2 + \cdots + \left(\frac{df}{dx_n}\right)^2 \Delta x_n^2}$$

Often, however, the differential terms under the square root are tedious to determine analytically.

But this root can be written simpler: $\Delta R = \pm \sqrt{\Delta f_1^2 + \cdots + \Delta f_n^2}$.

And with your *WP 43S*, the following algorithm will do for computing ΔR , even if f should be ‘strongly curved’:

1. Program the function $R = f(x_1, x_2, \dots, x_n)$ in a way you can vary its parameters easily.
2. Let your *WP 43S* compute $f(x_1, x_2, \dots, x_n)$.
3. Let it compute $R_{1+} = f(x_1 + \Delta x_1, x_2, \dots, x_n)$ and $\Delta R_{1+} = R_{1+} - R$.
4. Let it compute $R_{1-} = f(x_1 - \Delta x_1, x_2, \dots, x_n)$ and $\Delta R_{1-} = R_{1-} - R$.
5. Let it compute $R_{2+} = f(x_1, x_2 + \Delta x_2, \dots, x_n)$ and $\Delta R_{2+} = R_{2+} - R$.
6. Let it compute $R_{2-} = f(x_1, x_2 - \Delta x_2, \dots, x_n)$ and $\Delta R_{2-} = R_{2-} - R$.
7. Repeat the last two steps for each remaining parameter.

Being through with all n parameters, you will end with

$$\Delta R = \pm \sqrt{\frac{1}{2} (\Delta R_{1+}^2 + \Delta R_{1-}^2 + \Delta R_{2+}^2 + \Delta R_{2-}^2 + \cdots + \Delta R_{n+}^2 + \Delta R_{n-}^2)}$$

So the terms under the square root have become simple differences which are determined most easily with the help of your *WP 43S*.

For ‘small’ errors or less curvature of f , the following simpler algorithm will do, requiring down to half as many steps:

1. Program the function $R = f(x_1, x_2, \dots, x_n)$ in a way you can vary its parameters easily.
2. Let your *WP 43S* compute $R = f(x_1, x_2, \dots, x_n)$.
3. Let it compute $R_1 = f(x_1 + \Delta x_1, x_2, \dots, x_n)$ and $\Delta R_1 = R_1 - R$.

4. Let it compute $R_2 = f(x_1, x_2 + \Delta x_2, \dots, x_n)$ and $\Delta R_2 = R_2 - R$.
5. Repeat the last step for each remaining parameter.

Being through with all n parameters, you will end with

$$\Delta R = \pm \sqrt{\Delta R_1^2 + \Delta R_2^2 + \dots + \Delta R_n^2}$$

You might know this formula from your university or lab classes.

Solving Differential Equations

The method applied to the examples in the respective chapter in *Section 3* of the *OM* develops as explained below:

First, we solve 1-dimensional problems of the kind

$$\frac{d^2f}{dt^2} = a - b \left(\frac{df}{dt} \right)^2$$

This is the equation of motion for a body falling through a medium featuring drag proportional to said body's velocity squared. For earthly problems, take $a = 9.81 \frac{m}{s^2} = g$ and $b = \delta/M$ with M being the mass of said body and the constant parameter δ taking care of the viscosity of the medium (e.g. air) as well as the size and shape of the falling body as a whole.

For a first guess, let us assume $b = 0$. So there will be no drag at all, the body will be just accelerated by $a = g$. Then for two arbitrary subsequent points in time, the development of

- vertical velocity will be like $\left(\frac{df}{dt} \right)_{i+1} \approx \left(\frac{df}{dt} \right)_i + a\Delta t$ and
- position over ground will be like $f_{i+1} \approx f_i + \left(\frac{df}{dt} \right)_i \Delta t$.

Proceeding from time zero in small, constant time steps $\Delta t = t_{i+1} - t_i$:

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_0 \Delta t \text{ and } \left(\frac{df}{dt}\right)_1 \approx \left(\frac{df}{dt}\right)_0 + a\Delta t ,$$

$$f_2 \approx f_1 + \left(\frac{df}{dt}\right)_1 \Delta t \text{ and } \left(\frac{df}{dt}\right)_2 \approx \left(\frac{df}{dt}\right)_1 + a\Delta t , \text{ etc.}$$

Principally, a better approximation of the slope of f is achieved using the so-called *half-step method*:

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + a \frac{\Delta t}{2}$$

$$\left(\frac{df}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{df}{dt}\right)_{i-\frac{1}{2}} + a\Delta t$$

$$f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i+1/2} \Delta t$$

Proceeding from time zero in small steps Δt again, we get

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + a \frac{\Delta t}{2}$$

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_{1/2} \Delta t \text{ and } \left(\frac{df}{dt}\right)_{3/2} \approx \left(\frac{df}{dt}\right)_{1/2} + a\Delta t$$

$$f_2 \approx f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t , \text{ etc.}$$

Let us drop the restriction for b now. Replacing a in the previous set of equations by the right side of the differential equation on p. 246, we will get the following new set:

$$\frac{df}{dt}_{1/2} \approx \frac{df}{dt}_0 + \left[a - b \left(\frac{df}{dt}\right)_0^2 \right] \frac{\Delta t}{2}$$

$$\left(\frac{df}{dt}\right)_{i+1/2} \approx \left(\frac{df}{dt}\right)_{i-\frac{1}{2}} + \left[a - b \left(\frac{df}{dt}\right)_{i-\frac{1}{2}}^2 \right] \Delta t$$

$$f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i+1/2} \Delta t$$

Proceeding from time zero in small steps Δt again, we get

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + \left[a - b\left(\frac{df}{dt}\right)_0^2\right] \frac{\Delta t}{2}$$

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_{1/2} \text{ and } \left(\frac{df}{dt}\right)_{3/2} \approx \left(\frac{df}{dt}\right)_{1/2} + \left[a - b\left(\frac{df}{dt}\right)_{1/2}^2\right] \Delta t$$

$$f_2 \approx f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t, \text{ etc.}$$

This half-step method as explained above can be applied easily to all ordinary differential equations of 2nd order which can be written like

$$\frac{d^2f}{dt^2} = h\left(t, f, \frac{df}{dt}\right)$$

with an arbitrary real function h depending on time, the function itself and its first derivative. The equations applicable in this general case are then

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + h\left(t_0, f_0, \left[\frac{df}{dt}\right]_0\right) \frac{\Delta t}{2}$$

$$\left(\frac{df}{dt}\right)_{i+1/2} \approx \left(\frac{df}{dt}\right)_{i-1/2} + h\left(t_{i-1/2}, f_{i-1/2}, \left[\frac{df}{dt}\right]_{i-1/2}\right) \Delta t$$

$$f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i-1/2} \Delta t$$

For solving a 2-dimensional problem like e.g. finding the orbit of a satellite in the gravitational field of the earth, we need two differential equations, one for x and one for y :

$$\frac{d^2x}{dt^2} = \frac{F_x}{m} = -\frac{F}{m} \frac{x}{\sqrt{x^2 + y^2}} \quad \text{and} \quad \frac{d^2y}{dt^2} = \frac{F_y}{m} = -\frac{F}{m} \frac{y}{\sqrt{x^2 + y^2}}.$$

And we know that $F = G m M / (x^2 + y^2)$, thus

$$\frac{d^2x}{dt^2} = -\frac{GM}{(x^2 + y^2)^{3/2}} x = K_x \quad \text{and} \quad \frac{d^2y}{dt^2} = -\frac{GM}{(x^2 + y^2)^{3/2}} y = K_y$$

This is a pair of coupled differential equations. It is solved as follows:

$$\left(\frac{dx}{dt}\right)_{1/2} \approx \left(\frac{dx}{dt}\right)_0 + K_x \frac{\Delta t}{2} \quad \left(\frac{dy}{dt}\right)_{1/2} \approx \left(\frac{dy}{dt}\right)_0 + K_y \frac{\Delta t}{2}$$

$$\left(\frac{dx}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dx}{dt}\right)_{i-\frac{1}{2}} + K_x \Delta t \quad \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dy}{dt}\right)_{i-\frac{1}{2}} + K_y \Delta t$$

$$x_{i+1} \approx x_i + \left(\frac{dx}{dt}\right)_{i+1/2} \Delta t \quad y_{i+1} \approx y_i + \left(\frac{dy}{dt}\right)_{i+1/2} \Delta t$$

DRAFT

Orthogonal Polynomials

The following polynomials are all collected in X.FN'ORTHOG.

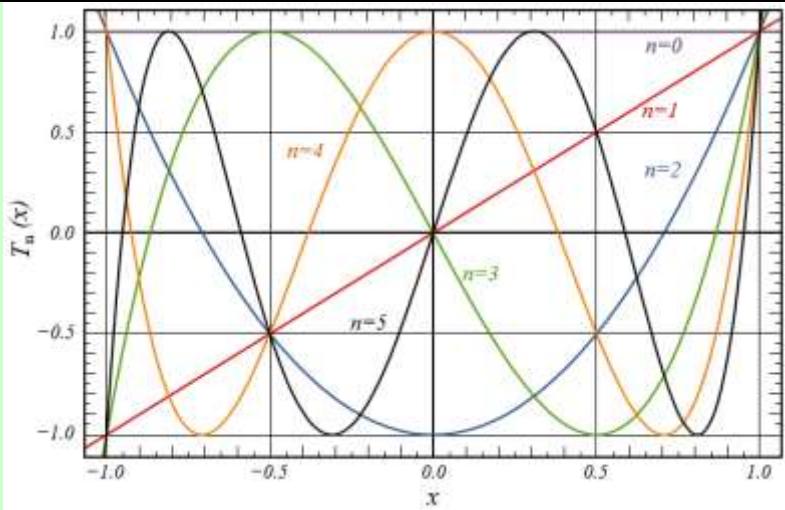
Name	Remarks (see pp. 12ff for general information)
H_n	<p>Hermite polynomials for <u>probability</u>: $H_n(x) = (-1)^n \cdot e^{x^2/2} \cdot \frac{d^n}{dx^n} \left(e^{-x^2/2} \right)$</p> <p>with n in Y, solving the differential equation</p> $f''(x) - 2x \cdot f'(x) + 2n \cdot f(x) = 0 .$ <p>See the first five polynomials plotted overleaf.</p>

Name	Remarks (see pp. 12ff for general information)
H _{np}	<p>Hermite polynomials for physics: $H_{np}(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n} (e^{-x^2})$ with n in Y, solving the same differential equation. See the first five polynomials plotted below.</p>

Name	Remarks (see pp. 12ff for general information)
L_m	<p>Laguerre polynomials (compare $L_{n\alpha}$ below):</p> $L_n(x) = \frac{e^x}{n!} \cdot \frac{d^n}{dx^n} (x^n e^{-x}) = L_n^{(0)}(x)$ <p>with n in \mathbb{Y}, solving the differential equation $x \cdot f''(x) + (1-x) \cdot f'(x) + n \cdot f(x) = 0$.</p> <p>See the first five <i>Laguerre polynomials</i> plotted here.</p>
$L_{m\alpha}$	<p>Laguerre's generalized polynomials (compare L_n above):</p> $L_n^{(\alpha)}(x) = \frac{x^{-\alpha} e^x}{n!} \cdot \frac{d^n}{dx^n} (x^{n+\alpha} e^{-x})$ <p>with n in \mathbb{Y} and α in \mathbb{Z}. Some of them are plotted below ($k = \alpha$).</p>

Name	Remarks (see pp. 12ff for general information)
P_n	<p>Legendre polynomials: $P_n(x) = \frac{1}{2^n n!} \cdot \frac{d^n}{dx^n} [(x^2 - 1)^n]$ with n in \mathbb{Y}, solving the differential equation</p> $\frac{d}{dx} \left[(1 - x^2) \cdot \frac{d}{dx} f(x) \right] + n(n+1)f(x) = 0.$ <p>See the first six polynomials plotted here:</p>
T_n	<p>Chebyshev (a.k.a. Čebyšev, Tschebyschow, Tschebyscheff) polynomials of first kind</p> $T_n(x) = \begin{cases} \cos(n \arccos(x)) & \text{for } -1 \leq x \leq 1 \\ \cosh(n \operatorname{arcosh}(x)) & \text{for } x > 1 \\ (-1)^n \cosh(n \operatorname{arcosh}(-x)) & \text{for } x < -1 \end{cases} \quad \text{with } n \text{ in } \mathbb{Y}, \text{ solving}$ <p>the differential equation</p> $f''(x) - \frac{x}{1-x^2} f'(x) + \frac{n^2}{1-x^2} f(x) = 0$ <p>The plot overleaf shows $T_0(x) \dots T_5(x)$.</p>

Name	Remarks (see pp. 12ff for general information)
------	--

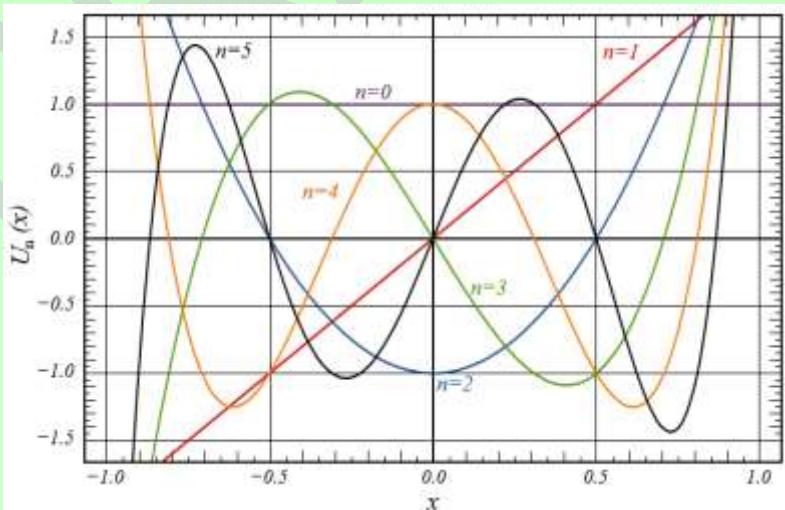


U_n

Chebyshev polynomials of second kind $U_n(x)$ with n in Y, solving the differential equation

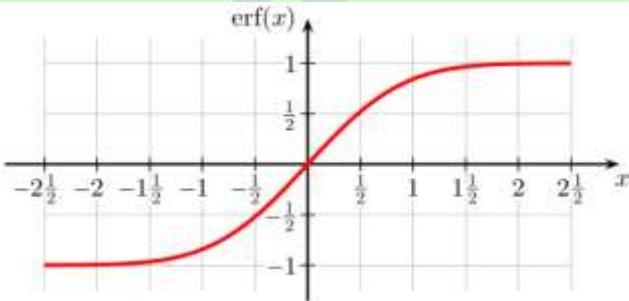
$$f''(x) - \frac{3x}{1-x^2}f'(x) + \frac{n(n+2)}{1-x^2}f(x) = 0$$

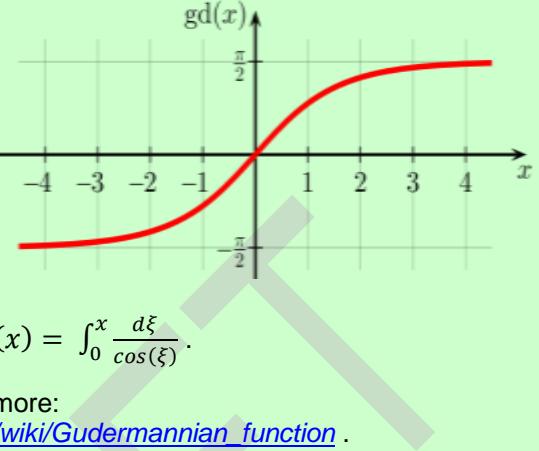
The plot below shows $U_0(x) \dots U_5(x)$:



Even More Mathematical Functions

All the following functions are found in X.FN. Some of them are for pure mathematics only but were useful at some stages of the *WP 34S* or *WP 43S* projects, so we made them accessible for the public.

Name	Remarks (see pp. 12ff for general information)
AGM	Returns the <i>arithmetic-geometric mean</i> . Find more about it here: http://mathworld.wolfram.com/Arithmetic-GeometricMean.html .
erf	<p>Returns the <i>error function</i> $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau$.</p> <p>Note that</p>  <p>$\text{erf}\left(\frac{x}{\sqrt{2}}\right) = 2 \Phi(x) - 1$ with $\Phi(x)$ representing the <i>standardized normal CDF</i> as described on p. 228.</p> <p>Beyond statistics, the <i>error function</i> may be helpful in heat conduction and diffusion problems, for instance.</p>
erfc	This command returns the <i>complementary error function</i> $\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-\tau^2} d\tau$. This function is related to the <i>error probability</i> of the <i>standardized normal distribution</i> .

Name	Remarks (see pp. 12ff for general information)
g_d , g_d^{-1}	<p>Returns the <i>Gudermannian function</i></p> $g_d(x) = \int_0^x \frac{d\xi}{\cosh(\xi)}$ <p>linking hyperbolic and trigonometric functions. See the plot for its real values. The inverse of this function is $g_d^{-1}(x) = \int_0^x \frac{d\xi}{\cos(\xi)}$.</p> <p>Start reading here for more: http://en.wikipedia.org/wiki/Gudermannian_function .</p> 
I_{xyz}	<p>Returns the <i>regularized (incomplete) Beta function</i> $\beta_x(x, y, z) / B(y, z)$</p> <p>with $\beta_x(x, y, z) = \int_0^x \tau^{y-1} (1-\tau)^{z-1} d\tau$ being the <i>incomplete Beta function</i> and $B(y, z)$ being <i>Euler's Beta function</i> (see p. 86 and https://en.wikipedia.org/wiki/Beta_function).</p>
$I\Gamma_p$	<p>Returns the <i>regularized Gamma function</i> $P(x, y) = \frac{\gamma(x, y)}{\Gamma(x)}$</p> <p>See γ_{XY} below for $\gamma(x, y)$ and p. 86 for $\Gamma(x)$.</p>
$I\Gamma_q$	<p>Returns the <i>regularized Gamma function</i> $Q(x, y) = \frac{\Gamma_u(x, y)}{\Gamma(x)}$</p> <p>See Γ_{XY} below for $\Gamma_u(x, y)$ and p. 86 for $\Gamma(x)$.</p>

See here for more:
<https://en.wikipedia.org/wikilink>
[complete gamma function](https://en.wikipedia.org/wikilink)

Name	Remarks (see pp. 12ff for general information)
$J_y(x)$	<p>Generally, the <i>Bessel functions</i> solve the differential equation</p> $x^2 f''(x) + xf'(x) + (x^2 - \nu^2)f(x) = 0 \quad \text{with } \nu \in \mathbb{C}.$ <p>$J_y(x)$ returns the <i>Bessel function of first kind</i> and order $y = \nu$. For arbitrary ν, this is</p> $J_\nu(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m! \Gamma(m + \nu + 1)} \left(\frac{x}{2}\right)^{2m+\nu}$ <p>For integer ν, this is also</p> $J_\nu(x) = \frac{1}{\pi} \int_0^\pi \cos[\nu t - x \sin(t)] dt$ <p>Start reading here for more information: http://en.wikipedia.org/wiki/Bessel_function.</p>

Name	Remarks (see pp. 12ff for general information)
sinc, sinc π	<p>The graph shows two functions plotted against x from -20 to 20. The x-axis has major ticks at $-6\pi, -4\pi, -2\pi, 0, 2\pi, 4\pi, 6\pi$. The y-axis ranges from -0.2 to 1.0. A red curve represents $\frac{\sin(x)}{x}$, which has a sharp vertical asymptote at $x=0$ and oscillates around the x-axis. A blue curve represents $\frac{\sin(\pi x)}{\pi x}$, which is smooth and oscillates around the x-axis without a singularity at $x=0$. A legend in the upper right identifies the curves.</p>
W_p , W_m	<p>Return Lambert's W with its principal branch (called W_p here) and its negative branch (called W_m for minus). The connecting point is $(x, y) = (-1/e, -1)$. The graph shows the <i>real</i> values of both branches.</p> <p>Start reading here for more information: http://en.wikipedia.org/wiki/Lambert_W_function. Learn more here: http://mathworld.wolfram.com/LambertW-Function.html.</p> <p>The graph shows the real branches of the Lambert W function, $W_p(x)$ and $W_m(x)$, plotted against x from -1 to 4. The y-axis ranges from -3 to 1. The horizontal axis is labeled x and the vertical axis is unlabeled. The $W_p(x)$ branch is a blue curve starting at $(-1/e, -1)$ and increasing towards positive infinity as $x \rightarrow 4$. The $W_m(x)$ branch is a red curve passing through $(-1/e, -1)$ and decreasing towards negative infinity as $x \rightarrow -1$.</p>

Name	Remarks (see pp. 12ff for general information)
γ_{xy}	Returns the <i>lower incomplete Gamma function</i> $\gamma(x, y) = \int_0^y t^{x-1} e^{-t} dt .$ Required for $I\Gamma_p$ above.
Γ_{xy}	Returns the <i>upper incomplete Gamma function</i> $\Gamma_u(x, y) = \int_y^\infty t^{x-1} e^{-t} dt .$ Required for $I\Gamma_q$ above.
$\zeta(x)$	Returns <i>Riemann's Zeta</i> for <i>real</i> arguments, with $\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^x}$ for $x > 1$, and its analytical continuation for $x < 1$: $\zeta(x) = 2^x \pi^{x-1} \sin\left(\frac{\pi}{2}x\right) \cdot \Gamma(1-x) \cdot \zeta(1-x).$ <p>Note the different vertical scales for negative and positive x in the plot below. And $\sqrt{6 \zeta(2)} = \pi$.</p> <p>Look here for more: http://mathworld.wolfram.com/RiemannZetaFunction.html .</p>

Note the *error function* as well as *Laguerre*, *Legendre*, and *Bessel functions* were provided 1976/77 already on the *Commodore M55* pocket calculator (featuring 55 keys).

Beyond what is printed in this appendix, you will also find lots of information about the special functions implemented in your *WP 43S* in the internet. Generally, *Wikipedia* is a good starter – we recommend checking the articles in different languages since they may well contain different material and use different approaches. For applied statistics, the *NIST Sematech* online handbook (quoted on pp. 222ff) is a competent source. And *Mathworld* (quoted on pp. 255ff) or *WolframAlpha* may contain more details than you ever want to know. Further references are found at these sites.

DRAFT

APPENDIX I: INFORMATION FOR ADVANCED USERS

Recursive Programming

Using local registers allows for creating a subroutine that calls itself recursively. Each invocation deals with its local data only. Of course, the *RPN stack* is global so be careful not to corrupt it.

Below is a recursive implementation of the factorial. It is an **example** for demonstration purposes only, since this routine will neither set the *stack* correctly nor will it work for input greater than some hundred:

LBL 'FACT'

IP

x> 1 ?

GTO 00

1

RTN

LBL 00

LocR 01

STO .00

DEC X

XEQ 'FACT'

RCLx .00

RTN

Assume $x = 4$ when you call FACT. Then it will allocate one local register (**R.00**) and store **4** therein. After decrementing x , FACT will call itself.

Then FACT₂ will allocate a local register (**R.00₂**) and store **3** therein. After decrementing x , FACT will call itself again.

Then FACT₃ will allocate a local register (**R.00₃**) and store **2** therein. After decrementing x , FACT will call itself once more.

Then FACT₄ will return to FACT₃ with $x = 1$. This x will be multiplied by **r.00₃** there, returning to FACT₂ with $x = 2$. This x will be multiplied by **r.00₂** there, returning to FACT with $x = 6$, where it will be multiplied by **r.00** and will finally become 24.

Building WP 43S Almost from Scratch

How to build the simulator on Windows:

1. Navigate to <https://www.msys2.org/>. Download and run msys2-x86_64-yyyymmdd.exe
2. Click **Next**
3. Enter the installation folder **C:\msys64** and click **Next**
4. Tick **Run MSYS2 now** and click **Finish**

The following **commands printed blue** must be executed in the black MSYS2 window:

5. **pacman -Syu** . Confirm each question with ENTER and wait until finished.
6. Close the black window by Alt-F4.
7. Reopen *MSYS2 MinGW 64-bits* (every time you need to open MSYS2, open the 64-bits version).
8. **pacman -Syu** . Confirm each question with ENTER.
9. **pacman -S mingw-w64-x86_64-gcc git base-devel mingw-w64-x86_64-gtk3** . Confirm each question with ENTER.
10. **cd ..** to navigate to your home directory.
11. **git clone https://gitlab.com/Over_score/wp43s.git** to get a local copy of the gitlab source repository on your PC.
12. **cd wp43s** to navigate to the *WP 43S* program sources.
13. **git pull** to get the latest version from gitlab.com.
14. **make mrproper** to clean the build environment (this is not required but recommended). Alternatively, you can enter **make rebuild** and skip step 15.
15. **make** to build the *WP 43S* simulator.
16. **./wp43s.exe** to run the simulator.
17. Return to step 13. (or to *App. F*) to get a new version of the sources.

How to build the WP43S.pgm for the *DM42* hardware:

1. Navigate to <http://gnutoolchains.com/arm-eabi/>, download and run **arm-eabi-gcc9.2.1.exe** or a more recent version.
2. Installation directory **C:\msys64\arm-eabi** shall be the same base directory as in step 3 on previous page.
3. Select **Current user**
4. Tick **Hardlink duplicate files**
5. Untick **Add binary directory to %PATH%**
6. Tick **I accept the terms of the license agreement**
7. Click **Install**
8. Click **OK** on the Installation succeeded dialog.
9. Start *MSYS2 64 bits* if not yet started.
10. **cd ~/wp43s/DMCP_build** to navigate to the *DMCP* build directory.
11. **./build_GMP_static_ARM_library** this is a long process: about 12 minutes on my PC.
12. **cd ~/wp43s/DMCP_build** to navigate to the *DMCP* build directory if not yet there.
13. **git pull** to get the latest version from gitlab.com
14. **./build_WP43S.pgm_for_DM42_hardware** to build *WP43S.pgm* for the *DM42*. Maybe the first time the process ends with an error – then run the same command a second time.
15. Copy the file *~/wp43s/DMCP_build/build/WP43S.pgm* via *USB* to your *DM42* and flash it (cf. *App. F*).
16. Loop to step 12.

Index of Everything Provided

This index lists the *name* of each and every *item* provided, be it a command, constant (# ...), menu or submenu (M unless trailed by a colon), predefined label (L) or register/variable (V), reserved character (c), or system flag (S). The *names* of *items* are sorted here as they are in your WP 43S:

$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	16	arctan	19	Binom _p	22	CLP	25
$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	16	artanh	19	BITS	22	CLPALL	25
10^x	16	ASIN	19	B _n	22	CLR (M)	25
1COMPL	16	ASLIFT (S)	106	B _n *	22	CLREGS	25
1/x	17	ASR	19	BS?	22	CLSTK	25
2COMPL	17	ASSIGN	19	Btu \rightarrow J	22	CLX	26
2 ^x	17	ATAN	20	C (V)	99	CLΣ	26
$\sqrt[3]{x}$	17	atm \rightarrow Pa	21	cal \rightarrow J	23	CNST	26
A (V)	99	AUTOFF (S)	106	CARRY (S)	105	COMB	26
ABS	17	AUTXEQ (S)	106	CASE	23	CONJ	26
$a_{\text{us}} \rightarrow m^2$	17	au \rightarrow m	21	CATALOG	23	CONST (M)	26
$a_c \rightarrow m^2$	17	A...Ω (M)	86	Cauch ⁻¹	24	CONVG	26
ACC (V)	99	A:	20	Cauch _Δ	24	CORR	27
ACOS	17	B (V)	99	Cauch _Δ	24	cos	27
ADM (V)	99	BACK	20	Cauch:	24	cosh	27
ADV	17	bar \rightarrow Pa	20	CauchF	24	COV	27
AGM	17	BATT?	20	Cauch _p	24	CPX (M)	27
AGRAPH	17	bbl \rightarrow m ³	20	CB	24	CPXj (S)	104
ALL	18	BC?	20	CEIL	24	CPXRES (S)	104
ALLENG (S)	106	BEEP	21	CF	24	CPXS (M)	27
ALPHA (S)	105	BeginP	21	CHARS (M)	24	CPX?	27
ALP.IN (S)	107	BestF	22	CLALL	24	CROSS	28
AND	18	BestF?	22	CLCVAR	25	ct \rightarrow kg	28
ANGLES (M)	18	Binom _Δ	22	CLFALL	25	cwt \rightarrow kg	28
arccos	18	Binom ⁻¹	22	CLK (M)	25	CX \rightarrow RE	28
arcosh	19	Binom _Δ	22	CLLCD	25	D (V)	99
arcsin	19	Binom:	22	CLMENU	25	DATE	28

DATES (M)	28	D→R	32	FCNS (M)	35	GaussF	37
DATE→	28	EIGVAL	32	FF	35	GCD	37
DAY	29	EIGVEC	32	FIB	35	g_d	37
DBLR	29	END	32	FILL	35	g_d^{-1}	37
DBLx	29	ENDP	32	FIN (M)	35	Geom_p	38
DBL/	29	ENG	32	FIX	35	Geom_Δ	38
dB→fr	29	ENORM	32	FLAGS (M)	35	Geom_Δ	38
dB→pr	29	ENTER↑	33	FLASH (M)	35	Geom^{-1}	38
DEC	29	ENTRY?	33	FLASH?	35	Geom:	38
DECIM. (S)	106	EQN (M)	33	FLOOR	35	$gl_{US} \rightarrow m^3$	38
DECOMP	29	EQ.DEL	33	fm.→m	35	$gl_{UK} \rightarrow m^3$	38
DEG	29	EQ.EDI	33	FP	36	GRAD	38
DEG→	29	EQ.NEW	33	$F_p(x)$	36	GRAD→	38
DELITM	29	erf	33	FP?	36	GRAMOD (V)	100
DENANY (S)	104	erfc	33	fr→dB	36	GROW (S)	106
DENFIX (S)	105	ERR	33	FRACT (S)	104	GTO	38
DENMAX	30	EVEN?	33	FS?	36	GTO.	39
DENMAX (V)	99	e ^x	33	FS?C	36	ha→m ²	39
DET	30	EXITALL	33	FS?F	36	H_n	39
DIGITS (M)	30	EXP (M)	34	FS?S	36	H_{n_P}	39
DISP (M)	30	ExpF	34	ft.→m	36	$hp_E \rightarrow W$	39
DMY (S)	104	Expon	34	$ft_{US} \rightarrow m$	36	$hp_M \rightarrow W$	39
DOT	30	Expon _e	34	FV (V)	100	$hp_{UK} \rightarrow W$	39
DROP	30	Expon _p	34	$fz_{UK} \rightarrow m^3$	36	Hyper_p	39
DROPy	30	Expon ⁻¹	34	$fz_{US} \rightarrow m^3$	36	Hyper_Δ	39
DSE	30	Expon:	34	$F_\Delta(x)$	36	Hyper_Δ	39
DSL	31	EXPT	34	$F_\Delta(x)$	36	Hyper^{-1}	39
DSTACK	31	e ^{x-1}	34	$F^{-1}(p)$	36	Hyper:	40
DSZ	31	E:	34	F:	36	HypF	40
D.MS	31	FB	34	f' (M)	36	I (V)	100
D.MS→	31	FBR	34	$f'(x)$	37	IDIV	40
D.MS→D	31	FC?	34	$f''(M)$	36	IDIVR	40
D.MY	31	FC?C	35	$f''(x)$	37	IGN1ER (S)	107
D→D.MS	31	FC?F	35	F&p:	37	iHg→Pa	40
D→J	32	FC?S	35	GAP	37	Im	40

INC	40	KEY?	44	LOADP	47	Mat_A (V)	101
INDEX	40	kg→ct	44	LOADR	47	Mat_B (V)	101
INFO (M)	40	kg→cwt	44	LOADSS	47	Mat_X	50
INPUT	41	kg→lb.	44	LOADV	48	Mat_X (V)	101
INT?	41	kg→oz	44	LOADΣ	48	max	50
INTING (S)	107	kg→scw	44	LocR	48	MDY (S)	104
INTS (M)	41	kg→sto	44	LocR?	48	MEM?	50
INVRT	41	kg→s.t	44	LOG ₁₀	48	MENU	50
in.→m	41	kg→ton	44	LOG ₂	48	MENUS (M)	50
IP	41	kg→trz	44	LogF	48	min	50
ISE	41	KTYP?	45	Logis _p	48	MIRROR	51
ISG	42	L (V)	100	Logis _Δ	48	mi.→m	51
ISM (V)	100	LASTx	45	Logis _Δ	48	mmHPa	51
ISZ	42	lb.→kg	45	Logis ⁻¹	48	MOD	51
I _{xyz}	42	lbf→N	45	Logis:	48	MODE (M)	51
IΓ _p	42	lbf→Nm	45	LOG _{xy}	48	MONTH	51
IΓ _q	42	LBL	45	LOOP (M)	48	MSG	51
I+	42	LBL?	45	LOWBAT (S)	105	MULTx (S)	106
I-	42	LCM	45	ly→m	49	MULπ	51
I/O (M)	42	LEAD.0 (S)	105	L.INTS (M)	49	MULπ→	51
i%/a (V)	100	LEAP?	46	L.R.	49	MVAR	51
J (V)	100	LgNrm _p	46	m ² →ac	49	MyMenu	51
J _y (x)	43	LgNrm _Δ	46	m ² →ac _{us}	49	Myα (M)	51
J+	42	LgNrm _Δ	46	m ² →ha	49	M.DELR	52
J-	42	LgNrm ⁻¹	46	m ³ →bbl	49	M.DIM	52
J/G	42	LgNrm:	46	m ³ →fz _{UK}	49	M.DIM?	52
J/G?	42	LinF	46	m ³ →fz _{us}	49	M.DY	52
J→Btu	43	LJ	46	m ³ →gl _{UK}	49	M.EDI	52
J→cal	43	L _m	46	m ³ →gl _{us}	49	M.EDIN	52
J→D	43	L _{ma}	46	MANT	49	M.EDIT (M)	52
J→Wh	43	LN	46	MASKL	50	M.GET	52
K (V)	100	LNβ	47	MASKR	50	M.GOTO	53
KEY	44	LNΓ	47	MATRS (M)	50	M.GROW	53
KEYG	44	LN(1+x)	47	MATR?	50	M.INSR	53
KEYX	44	LOAD	47	MATX (M)	50	M.LU	53

M.NEW	53	Norml:	56	Poiss ⁻¹	58	RCLEL	61
M.PUT	53	NOT	56	Poiss:	58	RCLIJ	61
M.RZR	53	NPER (V)	101	POLAR (S)	104	RCLS	61
M.SIMQ (M)	53	NUM.IN (S)	107	PopLR	58	RCL+	62
M.SQR?	54	nΣ	56	PowerF	58	RCL-	62
M.WRAP	54	N→lbf	56	PRCL	58	RCL×	62
m:	54	ODD?	56	PRIME?	59	RCL/	62
m→au	54	OFF	56	PRINT (M)	59	RCL↑	62
m→fm.	54	OR	56	PRINT (S)	105	RCL↓	62
m→ft _{US}	54	OrthoF	56	PROB (M)	59	RDP	62
m→ft.	54	ORTHOG (M)	56	PROG (M)	59	Re	62
m→in.	54	OVERFL (S)	105	PROGS (M)	59	REAL?	62
m→ly	54	oz→kg	56	PROPFR (S)	104	REALDF (V)	102
m→mi.	54	ParabF	56	PRTACT (S)	107	REALS (M)	62
m→nmi.	54	PARTS (M)	56	pr→dB	59	RECV	62
m→pc	54	PAUSE	57	psi→Pa	59	REGS (V)	102
m→pt.	54	Pa→atm	57	PSTO	59	RESET	63
m→yd.	54	Pa→bar	57	pt.→m	59	RE→CX	63
NAND	54	Pa→iHg	57	PUTK	59	Re ∇ Im	63
NaN?	54	Pa→mmH	57	PV (V)	101	RJ	63
NBin _p	54	Pa→psi	57	P.FN (M)	60	RL	64
NBin _Δ	54	Pa→tor	57	P.FN2 (M)	60	RLC	64
NBin _Δ	54	pc→m	57	P:	60	RM	64
NBin ⁻¹	54	PERM	57	qt.→m ³	60	RMD	65
NBin:	54	PER/a (V)	101	QUIET (S)	106	RM?	65
NEIGHB	55	PGMINT	57	RAD	60	RNORM	65
NEXTP	55	PGMSLV	57	RAD→	60	RootF	65
nmi.→m	55	PIXEL	57	RAM (M)	60	ROUND	65
Nm→lbf	55	PLOT	58	RANGE	60	ROUNDI	65
NOP	55	PMT (V)	101	RANGE?	60	RR	65
NOR	55	P _n	58	RANI#	60	RRC	65
Norml _p	55	POINT	58	RAN#	61	RSD	66
Norml _Δ	55	Poiss _p	58	RBR	61	RSUM	66
Norml _Δ	55	Poiss _Δ	58	RCL	61	RTN	66
Norml ⁻¹	55	Poiss _Δ	58	RCLCFG	61	RTN+1	66

RUNIO (S)	105	sinh	71	s _w	74	ULP?	77
RUNTIM (S)	105	SKIP	71	s _{xy}	75	U _n	77
R-CLR	66	SL	71	SYSTEM	75	UNITV	78
R-COPY	67	SLOW (S)	106	SYS.FL (M)	75	UNDO	78
R-SORT	67	s _m	72	s(a)	75	UNSIGN	78
R-SWAP	67	SMODE?	72	S.INTS (M)	75	USER (S)	105
R>D	67	s _{mw}	72	s.t→kg	75	U→ (M)	78
R↑	67	SNAP	72	s→year	75	VAR (M)	78
R↓	67	SOLVE	72	T (V)	102	VARMNU	78
s	68	Solver (M)	72	T ₀	75	VARS (M)	78
SAVE	68	SOLVING (S)	107	tan	75	VERS?	78
SB	68	SPCRES (S)	106	tanh	75	VIEW	79
SCI	68	SPEC?	72	TDISP	76	VMDISP (S)	107
scw→kg	68	SR	73	TDM24 (S)	103	V:	79
SDIGS?	68	SSIZE8 (S)	106	TEST (M)	76	V ₄	79
SDL	68	SSIZE?	73	TICKS	76	WDAY	79
SDR	68	STAT (M)	73	TIME	76	Weibl _p	79
SEED	69	STATUS	73	TIMER	76	Weibl _A	79
SEND	69	STK (M)	73	TIMES (M)	76	Weibl _A	79
SETCHN	69	STO	73	T _n	76	Weibl ⁻¹	79
SETDAT	69	STOCFG	73	TONE	76	Weibl:	79
SETEUR	69	STOEL	73	ton→kg	76	WHO?	79
SETIND	69	STOIJ	73	TOP?	76	Wh→J	79
SETJPN	69	STOP	73	tor→Pa	77	W _m	80
SETSIG	69	STOS	73	t _p (x)	77	W _p	80
SETTIM	69	STO+	74	TRACE (S)	105	WSIZE	80
SETUK	69	STO-	74	TRANS	77	WSIZE?	80
SETUSA	69	STO×	74	TRI (M)	77	W ⁻¹	80
SF	69	STO/	74	trz→kg	77	W→hp _{UK}	80
SHOW	70	STO↑	74	TVM (M)	77	W→hp _E	80
SIGN	70	sto→kg	74	t _A (x)	77	W→hp _M	80
SIGNMT	70	STO↓	74	t _A (x)	77	X (V)	102
SIM_EQ	70	STRING (M)	74	t ⁻¹ (p)	77	ŷ	81
sin	70	STRI?	74	t:	77	ȳ	81
sinc	70	SUM	74	t _Z	77	x ²	81

x^3	81	y^x	84	$\Sigma 1/y^2$	87	$x \text{MOD}$	91
XEQ	81	Y.MD	84	$\Sigma \ln^2 x$	88	/	91
\bar{x}_G	81	$y \bar{x}$	84	$\Sigma \ln^2 y$	88	${}^{\wedge} \text{MOD}$	91
\bar{x}_H	81	Z (V)	102	$\Sigma \ln x$	88	$\rightarrow (c)$	91
$x \text{IN}$	81	$z \bar{x}$	84	$\Sigma \ln xy$	88	$\rightarrow \text{DATE}$	91
x_{\max}	81	$\alpha \text{CAP (S)}$	105	$\Sigma \ln y$	88	$\rightarrow \text{DEG}$	91
x_{\min}	81	$\alpha \text{INTL (M)}$	84	$\Sigma \ln y/x$	88	$\rightarrow \text{D.MS}$	91
XNOR	81	$\alpha \text{LENG?}$	84	Σ_n	88	$\rightarrow \text{GRAD}$	91
XOR	81	$\alpha \text{MATH (M)}$	85	σ_w	88	$\rightarrow \text{HR}$	92
$x \text{OUT}$	82	$\alpha \text{POS?}$	85	Σx	88	$\rightarrow \text{H.MS}$	92
\bar{x}_{RMS}	82	αRL	85	Σx^2	88	$\rightarrow \text{INT}$	92
\bar{x}_w	82	αRR	85	$\Sigma x^2 y$	88	$\rightarrow \text{MUL}\pi$	92
$\ddot{\forall}y$	82	αSL	85	$\Sigma x^2/y$	88	$\rightarrow \text{POL}$	92
$x!$	82	αSR	85	Σx^3	88	$\rightarrow \text{RAD}$	92
X.FN (M)	82	$\alpha \cdot$ (M)	85	Σx^4	88	$\rightarrow \text{REAL}$	93
$x:$	82	$\alpha \text{FN (M)}$	85	$\Sigma x \ln y$	88	$\rightarrow \text{REC}$	93
$x \rightarrow \text{DATE}$	83	$A \dots \Omega$ (M)	85	Σxy	88	$\uparrow \text{Lim}$ (V)	102
$x \rightarrow \alpha$	83	$\alpha \rightarrow x$	86	Σy	89	$\downarrow \text{Lim}$ (V)	102
$x \bar{x}$	83	$\beta(x,y)$	86	Σy^2	89	\bar{x}	93
$x \bar{x}y$	83	Γ_{xy}	86	$\Sigma y \ln x$	89	$ M $	94
$x < ?$	83	γ_{xy}	86	$\Sigma +$	89	$ x $	94
$x \leq ?$	83	$\Gamma(x)$	86	$\Sigma -$	89	$ $	94
$x = ?$	83	δx (L)	86	$\chi^2_p(x)$	90	$\%$	94
$x \neq ?$	83	$\Delta \%$	86	$\chi^2_{\Delta}(x)$	90	$\% \text{MRR}$	94
$x \approx ?$	83	ε	86	$\chi^2_{\Delta}(x)$	90	$\% T$	95
$x \geq ?$	83	ε_m	87	$\chi^2:$	90	$\% \Sigma$	95
$x > ?$	83	ε_p	87	$(\chi^2)^{-1}$	90	$\% + \text{MG}$	95
$x = +0?$	83	$\zeta(x)$	87	$(-1)^x$	90	\sqrt{x}	95
$x = -0?$	83	π	87	$[M]^T$	90	\int	95
Y (V)	102	Π_n	87	$[M]^{-1}$	90	$\int f$ (M)	96
\hat{y}	84	Σ (M)	87	$+$	90	$\int f dx$ (M)	96
$y d \rightarrow m$	84	σ	87	$+/-$	90	$\not \exists$	96
YEAR	84	$\Sigma 1/x$	87	$\pm \infty?$	91	$\not \exists \rightarrow$ (M)	96
year-s	84	$\Sigma 1/x^2$	87	$-$	91	$\blacksquare \text{ADV}$	96
YMD (S)	104	$\Sigma 1/y$	87	\times	91	$\blacksquare \text{CHAR}$	96

DLAY	96	# F_δ	133	# N_A	135	# Δv_{Cs}	137
LCD	96	# G	133	# NaN	135	# ε_0	137
MODE	97	# G_0	133	# p_0	135	# λ_C	138
PROG	97	# G_C	133	# R	135	# λ_{Cn}	138
r	97	# g_e	134	# r_e	136	# λ_{Cp}	138
REGS	97	# G_{M_\oplus}	134	# R_K	136	# μ_0	138
STK	98	# g_\oplus	134	# R_{Moon}	136	# μ_B	138
TAB	98	# h	134	# R_∞	136	# μ_e	138
USER	98	# \hbar	134	# R_\odot	136	# μ_e/μ_B	138
WIDTH	98	# k	134	# R_\oplus	136	# μ_n	138
Σ	98	# K_J	134	# S_a	136	# μ_p	138
#	98	# l_{PL}	134	# S_b	136	# μ_u	138
#	98	# m_e	134	# Se^2	136	# μ_μ	138
# a	132	# M_{Moon}	134	# Se'^2	136	# σ_B	139
# a_0	132	# m_n	134	# Sf^{-1}	136	# Φ	139
# a_{Moon}	132	# m_n/m_p	135	# T_0	136	# Φ_0	139
# a_\oplus	133	# m_p	135	# T_p	136	# ω	139
# c	133	# m_p/m_e	135	# t_{PL}	137	# $-\infty$	139
# c_1	133	# m_{PL}	135	# V_m	137	# ∞	139
# c_2	133	# m_u	135	# Z_0	137	# B	98
# e	133	# $m_u c^2$	135	# α	137	# DEC (V)	103
# e_E	133	# m_p	135	# γ	137		
# F	133	# M_\oplus	135	# γ_{EM}	137		
# F_α	133	# M_\odot	135	# γ_p	137		

APPENDIX J: RELEASE NOTES

	Date	Release notes
0	29.11.12	Official project start with first publication of the 43S concept and a layout on one of the forums of the <i>Museum of HP Calculators</i> (https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685). Though there are found far older traces of a '43S' denoting a 'Super HP-42S', though in various more or less fictional cases – pure vapourware™.
0.1	2.2.14 23.5.15	Manual setup based on the one of <i>WP 34S</i> . Passed to <i>Jake Schwartz</i> , <i>Eric Smith</i> , and <i>Richard Ottosen</i> for first information.
0.2	3.10.15	Update based on <i>Jake's</i> feedback and further thoughts, distributed to <i>Eric</i> , <i>Jake</i> , <i>Marcus</i> , and <i>Pauli</i> .
0.3	21.3.16	Split the manual in three; moved LBL onto the keyboard, renamed STOM to STOCFG, RCLM to RCLCFG, SERR to s_m , and SERR _w to s_{mw} ; refined the <i>Key Response Table</i> . Passed to <i>Michael</i> for information.
0.4	28.3.16	Renamed LOGS to EXP and EEX to E . Added hardware information from 2 nd manufacturer.
0.5	29.10.16	Returned EEX . Changed keyboard layout .
0.6	22.8.17	Merged the Applications and Owner's Manual. Changed the input order of complex number parts on Pauli's request. Changed keyboard layout introducing D.MS, SST, BST, and % while removing ý, RAN#, 'FRC, and 'CFIT . Put 'CFIT into 'STAT and 'FRC into 'MODE. Placed OFF below EXIT for easier customizing. Renamed cc to C5, EEX to E , STOPW to TIMER, SHOW to REGS, 'SOLVE to 'ADV, DLINES to DSTACK, 12h to CLK12, and 24h to CLK24. Replaced IND by →. Deleted %MG since covered by Δ%, added EIGVAL and EIGVEC. Swapped CNST and CONST. Defined the echo rows for alphanumeric and command input. Expanded and modified the character sets for better use of display space. Added the QRG.
0.7		Changed keyboard layout. Replaced the labels BST by $\triangleleft\triangleright$, SST by $\exists\forall$, and UNDO by \square; added some alpha input mode reminders on the keyboard. Added AGRAPH, CLLCD, EQ.xxx, HYP, J/G, M.GOTO, ORTHOF, PIXEL, POINT, TDISP, and \blacksquare USER. Moved the background considerations out of <i>ReM App. D</i> . Introduced K as <i>alpha register</i> for alphanumeric constants in programs. Removed <i>fraction data type</i> . Extended <i>items</i> from 6 to 7 characters to match <i>HP-42S</i> .

Date	Release notes
2.4.18	Specified <i>data types</i> more precisely in <i>ReM App. D</i> . Reduced the maximum number of <i>local registers</i> from 888 to 100. Deleted JG1582 and JG1752. Renamed two commands for TVM. Replaced the heading apostrophe for <i>menu names</i> . Put <u>SUMS</u> in <u>STAT</u> . Renamed the trigonometric and hyperbolic functions according to mathematical standards, and $\blacksquare\text{CHR}$ to $\blacksquare\text{CHAR}$. Redistributed the chapter about constants. Modified STATUS display. Refined the unit conversions to ensure S/on one side. Specified 0 SEED. Expanded <i>ReM App. A</i> . Added formula output for L.R. Modified CPX?, DBL?, and REAL?. Changed output of binary tests for compatibility with HP-42S.
0.8 7.5.18 20.9.18	Changed keyboard layout: introduced <u>TRG</u> containing trigonometric functions, removed <u>HYP</u> into <u>EXP</u> and $\blacksquare\pi$ to g-shifted $\blacksquare\theta$, swapped some shifted labels. Refined the chapters about register arithmetic, <i>Command Parameter Input</i> , <i>Alphanumeric Input</i> , <i>Matrix Calculations</i> , and <i>Orthogonal Polynomials</i> . Introduced CLCVAR and more vintage examples. Rearranged <i>temporary information</i> on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE. Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9 3.1.19	Removed <i>angle data type</i> . Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded <i>App. B</i> . Added SHOW for displaying full precision of DP numbers and FBR for browsing our two fonts. Split a chapter. Expanded some titles. Added the overlay drawing. Modified functionalities of <u>EXIT</u> and <u>Vx</u> to match HP-42S. Added a chapter about curve fitting. Modified functionalities of <u>ENTER↑</u> and <u>GT</u> . Expanded <i>App. K</i> . Renamed DOUBLE to →DP. Added →SP and conversions of <i>quarts</i> . Rearranged X.FN. Replaced <u>USR</u> by <u>UM</u> . Changed keyboard moving <u>UM</u> , <u>JX</u> , and <u>TRI</u> . Moved \blacksquare to $\blacksquare f$ <u>R/S</u> . Added XIN and XOUT. Added a chapter in <i>App. E</i> and information about infinite integers. Extended the domain of GCD and LCM. Refined and corrected.
0.10 3.3.19	Returned <i>angle data type</i> and aSR. Added IDIVR and VANGLE. Refined FP, IP, IMPFRC, PROFRC, SDIGS?, →DP, →HR, →INT, →REAL, →SP, explanation of ALL, the summary of integer functions, and handling of long alpha strings. Modified contents of <u>CPX</u> , <u>MATX</u> , and <u>g•</u> . Added a summary of matrix functions. Removed the <u>ON</u> -key combinations. Modified MEM?. Rewrote the angular conversions. Renamed infinite and finite integers to <i>long</i> and <i>short integers</i> . Added a chapter about $\pm\infty$ and NaN. Modified RBR and the menu for STO and RCL. Removed \blacksquare from the keyboard. Renamed X_u to X_e for the distributions.

	Date	Release notes
0.11	8.5.19	<p>Changed keyboard making CC primary and user mode shifted, removing x^2, \sqrt{x}, and DSP, adding x , DROP, and SHOW, and moving some shifted labels. Modified <u>BITS</u>, <u>CLREGS</u>, <u>CNST</u>, <u>CPX</u>, <u>DISP</u>, <u>EXP</u>, <u>INTS</u>, <u>MODE</u>, <u>PARTS</u>, <u>SHOW</u>, <u>STAT</u>, <u>U\rightarrow</u>, <u>aMATH</u>, the division matrix, <i>data type</i> conversions, and the <i>Quick Reference Guide</i>. Added conversions of <i>barrels</i>, <i>carats</i>, and <i>fathoms</i>. Deleted DSP. – Separated predefined variables. Refined Sect. 6. Added \bar{x}_H, \bar{x}_{RMS}, nine statistical sums and five curve fit models. Split <u>STAT</u> in <u>STAT</u> and <u>SUMS</u>; renamed RMDR to RMD, L_n to L_m, L_{na} to L_{ma}, Π to Π_n, Σ to Σ_n, and some constants to avoid search ambiguities. Refined App. J, Sect. 3 and 4, \rightarrowINT, <u>CLR</u>, and the functions of Δ and ∇. Put <u>SUMS</u> instead of <u>RMD</u> on the keyboard, moved <u>ADV</u>, <u>BITS</u>, <u>CATALOG</u>, <u>EQN</u>, <u>FILL</u>, <u>INTS</u>, <u>MATX</u>, <u>MODE</u>, <u>PROB</u>, <u>RTN</u>, <u>SHOW</u>, <u>STAT</u>, and <u>a.FN</u>. Rearranged $A \dots \Omega$ and Sect. 2 of the OM.</p>
0.12	16.10.19	<p>Rearranged the appendices of the <i>ReM</i> from App. D on. Expanded App. A of the OM and App. K. Deleted the standardized normal distribution Φ and rearranged <u>PROB</u>. Updated <u>CNST</u> following CODATA 2018. Renamed the angular conversions. Changed the composing and cutting functionality of CC. Refined exiting <i>short integer</i> input. Expanded App. D. Specified maximum size of <i>long integers</i>. Changed keyboard adding \sqrt{x}, moving CPX, FIN, RBR, R\uparrow, and SHOW, removing %. Renamed VANGLE to V\sqrt{x}. Modified <u>CPX</u>, <u>MATX</u>, <u>TRI</u>, and <u>X.FN</u>. Rearranged Section 1 of the OM. Added some internal <i>data types</i> to App. B; reduced the range of <i>long integer</i> results and DP real inputs to $10^{\pm 999}$. Defined the domains of e^{x-1}, IDIVR, LN(1+x), MOD, and RMD according to the HP-42S; modified PLOT and $\Sigma+$. Refined the <i>Addressing Tables</i>. Added a <i>data type</i> matrix for IDIVR. Refined the <i>Special Results</i> in App. B.</p>
0.13	30.11.19	<p>Expanded the alpha keyboard and App. I. Modified <u>CPX</u>, <u>INTS</u>, <u>MODE</u>, <u>PROB</u>, <u>STK</u>, <u>TEST</u>, <u>a.</u>, <u>SHOW</u>, and <u>STATUS</u>. Refined the sorting order of <i>items</i>, ALL, CX\rightarrowRE, MEM?, RE\rightarrowCX, RBR, RM, SLVQ, and <u>U\rightarrow</u>. Started filling App. F and G. Refined App. 2. Added a <i>long integer</i> example, CPXR?, LZ?, Δv_{Cs}, conversions of <i>hectares</i>, and a proposal for system status information.</p>
0.14	7.3.20	<p>Introduced <i>system flags</i> for status information. Split <u>I/O</u>. Added <u>CATALOG'SYS.FL</u>, <u>PRINT</u>, <u>PROG</u>, <u>RANI#</u>, <u>VAR</u>, auxiliary constants, some predefined variables, and an index in App. I. Changed keyboard swapping MODE and FLAGS, U\rightarrow and $\sqrt{x}\rightarrow$, moving CPX, FILL, RBR, R\uparrow, USER, a.FN, aINTL, \sqrt{x}, and \bar{x}, displaying PRINT, RMD, STATUS, x^2, and Σ, and removing c/d, $\bar{x}x$, \rightarrowSP, and \rightarrowDP. Renamed <u>DISP</u> to <u>DSP</u> and <u>SUMS</u> to Σ, changed Σ to Σ. Refined the addressing tables and catalog access, <u>a b/c</u>, <u>ADV</u>, <u>BATT?</u>,</p>

Date	Release notes
	BITS, CATALOG'CHARS and 'MENUS, CLALL, CLFALL, CPX, EXP, GAP, INTS, I/O, MODE, NEIGHB, PARTS, PRIME?, P.FN, SHOW, STAT, STK, X.FN, αINTL, and α•. Deleted all 16-digit (i.e. SP) data types as well as A...Z and the commands CLK12, CLK24, CPXi, CPXj, CPXRES, CPXR?, DBL?, DENANY, DENFAC, DENFIX, ENGOVR, FAST, IMPFRC, LZOFF, LZON, LZ?, MULTx, MULT+, POLAR, PROFRC, QUIET, RDX., RDX,, REALRE, RECT, SCIOVR, SLOW, SSIZE4, SSIZE8, →DP, and →SP. Corrected.
0.15 14.6.20	Added BESTF?, RANGE, RANGE?, REGIST, SNAP, and s(a), as well as errors 28 and 31 – 35. Changed DSZ and ISZ to comply with HP-16C. Changed keyboard shifting N, O, P, and Q, swapping ? and Z, moving CNST, CPX, FLAGS, RBR, RTN, RT↑, VIEW, and □, removing :, and adding MOD, √, and SNAP. Renamed DSP to DISP, CNST to CONST, CONST to CNST, ASL.BLK to ASLIFT, SSIZE to SSIZE8, TDM to TDM24, and the left and right sided probabilities. Refined ASSIGN, CATALOG, CNST, DISP, INFO, NEXTP, PRIME?, PROB, RBR, RESET, SHOW, SINC, STAT, U→, VIEW, x=+0?, x=-0?, y^x, α→x, √x, pp. 54 – 57 and 205 – 207 (and consequences) as well as Section 6 of the OM, pp. 108 – 117, App. B, C, and E of the ReM, and some looping and statistical explanations. Reduced the maximum number of local registers from 100 to 99. Changed ALLSCI to ALLENG and RECTN to POLAR. Added data type matrices for powers. Corrected.
0.16 3.11.20	Added torque and mmHg conversions, ISM, LOADV, x _{max} and x _{min} . Added UNDO to the I/O. Refined I/O and the descriptions of LOAD, LOADSS, RESET, and UNDO. Marked the not-undoable items in the I/O. Renamed the constants according to the OM and kicked them out of the I/O. Added USB. Refined Basic Kinds of Program Steps, App. E and F. Changed bit numbering from 1 ... 64 to 0 ... 63. Renamed SMODE? to ISM?. Refined IM, RE, SINC, TRI, WSIZE, X.FN and the labels of torque conversions. Added SINCTT and more vintage pictures. Expanded App. G. Refined the power matrix. Corrected.
0.17 21.2.21	Empty menus show up. Refined 3√x, AUTOFF, BATT?, CLFALL, CLP, CLREGS, e ^x , GTO., IP, J/G, LOAD, LOCR, LOCR?, LN, MEM?, M.EDI, M.EDIN, RCLEL, RDP, RSD, SAVE, SDL, SDR, TDISP, WSIZE, √y, y ^x , Σ+, Σ-, √x, □r, labels in U→, distributions, the DT matrices, Sect. 3 and 4 and App. 1, as well as App. B, E, F, and I. Renamed ST.X etc. to X etc. Increased the number of local flags to 32. Added Chinese units of length and area to U→. Deleted M.OLD, added DELITM, J/G?, and □x. Corrected.

WP 43S QUICK REFERENCE GUIDE

USING MENUS

A *menu* defines the top row of keys by displaying up to three *softkeys* above each . If the current *menu view* is limited by a dashed line this indicates it is a *multi-view menu* and or can be used to browse the additional views of this *menu*. To execute a softkey in the lowest row, press the corresponding . To execute one in 2nd or 3rd row, press the corresponding headed by or , respectively.

MEMORY

The *stack* is a workspace for calculations. Each *stack register* may contain any type of data. Choose a *stack* of four (**X**, **Y**, **Z**, and **T**) or eight *registers* (**X**, **Y**, **Z**, **T**, **A**, **B**, **C**, and **D**). Last *x* is saved in *register L*.

General purpose registers: There are 100 numbered global *GP registers* (00 ... 99). Furthermore, there are **I**, **J**, and **K** serving special purposes in matrix handling (see p. Q-7), probability distributions (see p. Q-8), and programming but may be used globally otherwise. Also **A**, **B**, **C**, and **D** may be used this way unless being part of *stack*. Each *register* may contain any type of data. **STO nn** stores a copy of *x* into **Rnn**, **RCL nn** recalls a copy of the contents of **Rnn** into **X**, and **x↔ nn** swaps *x* and the contents of **Rnn**.

Variables are named storage locations that may contain any type of data. E.g. for storing *x* into a variable named **XYZ**, enter **STO** **α XYZ** **ENTER↑**. Variable *names* shall be unique, ≤7 characters long, and contain ≥1 letter.

Flags: There are 112 global *user flags*. and some 35 named *system flags*.

Programs consist of ≥4 program steps: LBL with a global label, at least one action step, RTN, and END. Each program may contain subroutines (up to 8 levels deep). See p. Q-6 for more.

Available memory: **INFO MEM?** (or **FLAGS STATUS**) displays the amount of free memory. Use CLP for clearing programs or clear variables to free memory that is no longer needed.

DATA TYPES

Long integers: are the simplest type. Any number you enter without using **.(dot)**, **E**, **CC**, or **#** is taken as a long integer of base 10.

Real numbers: Any number you enter using **.(dot)** and/or **E** is a real number.

Complex numbers: A complex number consists of two real numbers combined to represent its real and imaginary part like $1.23-i\times 4.56$ in rectangular mode (CF POLAR and press **1.23 CC 4.56 +/- ENTER↑**) or its magnitude and phase like $-7.89 \angle 120^\circ$ in polar mode (SF POLAR and press **7.89 +/- CC 120 ENTER↑**).

Angles: Any real number input trailed by **d.ms** is interpreted as an *angle* in *sexagesimal degrees*. Angles may be entered as well in *decimal degrees*, *radians*, *multiples of π* , or *grades*. Choose the appropriate angular display mode via **MODE** (see overleaf).

Times: Any real number input trailed by **h.ms** is interpreted as a sexagesimal *time*. It will be displayed like $23:45:43.210\ 9$ with as many decimals of *seconds* as needed.

Dates: Any real number input trailed by **.d** is interpreted as a *date* in the format selected (yyyy.mmdd for Y.MD or dd.mmyyyy for D.MY or mm.ddyyyy for M.DY).

Matrices: see pp. Q-7 f.

Short integers: Any purely numeric input trailed by **#** and number 2 ... 16 is interpreted as a *short integer* of the base specified. **D** and **H** are shortcuts for base 10 and 16, respectively. *Short integers* may occupy 1 ... 64 bits.

Alphanumeric (or text) strings: Enter *alpha input mode (AIM)* by pressing **@**. Data entered in AIM become an *alphanumeric string* when closed (unless they are function parameters). All available Latin letters (incl. accented ones) are found in **f A**. Greek letters are accessed via **g** plus the corresponding Latin letter (see calculator backside). Turn to lower case by **▼** and back to upper by **▲** for all letters.

f plus one of the keys **+**, **-**, **x**, **.**, and **0** ... **9** will enter the corresponding character. Special characters are found in **g -** and **g .**

f R makes the subsequent character entered a subscript, **f E** makes it a superscript, if applicable.

MODES

MODE	SYSTEM		RM	SETSIG	DENMAX	
	SF	DEG	RAD	GRAD	MULπ	CF

SF *n*, CF *n* Set (or clear) the flag specified.

DEG Selects *degrees* as angular display mode (ADM).

RAD Selects *radians* as ADM.

GRAD Selects *grades*, a.k.a. *gon*, as ADM.

MULπ Selects *multiples of π* as ADM.

RM *n* Sets rounding mode.

SETSIG *n* Sets calculator precision (1 ... 34 significant digits).

DENMAX *n* Sets the maximum denominator for calculating with fractions.

SYSTEM Returns the calculator to the DMCP system for updating.

DISPLAY FORMATS

DISP	GAP		RANGE	RANGE?		DSTACK	
	CHINA	EUROPE	INDIA	JAPAN	UK	USA	
	SDL	SDR			RDP	RSD	
	FIX	SCI	ENG	ALL	ROUNDI	ROUND	

FIX *n* Fixed number of *n* decimals.

SCI *n*, ENG *n* Scientific (or engineering) notation.

ALL *n* Displays all digits present as far as possible.

ROUND Rounds a *time*, real, or complex *x* to current display format.

ROUNDI Rounds to next integer.

RDP *n* Rounds *x* to *n* decimal places (1 ... 99, think of FIX)

RSD *n* Rounds *x* to *n* significant digits (1 ... 34, think of SCI).

SDL *n*, SDR *n* Shifts digits left (right) by *n* decimal positions.

CHINA, EUROPE, INDIA, JAPAN, UK, USA Set local display preferences.

GAP *n* Selects a digit group gap inserted after every *n* digits.

RANGE *n* Sets the maximum exponent to be displayed for real numbers

RANGE? Returns the range setting

DSTACK *n* Sets the number of stack registers to be displayed (1 ... 4).

PROGRAMMING

Program Entry

- P/R** toggles *program entry mode*.
- GTO**  moves the *program pointer* to a new program space.
- GTO**  *nnnn* moves it to step number *nnnn* within the *current program*.
-  moves it to previous step
 moves it to next step
 deletes the *current program step* entirely.
- EXIT** exits *program entry mode*.

Labels

A program label is a marker used to identify an entire program or a section within a program. Each program must begin with a global label (cf. p. Q-4).

Global labels can be accessed from anywhere in memory (thus, they should be unique). Global labels are alphanumeric and ≤ 7 characters long.

Local labels can be accessed only within the current program (thus, they should be unique within this program). Local labels are numeric (00 ... 99).

Local registers

... are allocated via **LOCR n** with the amount of *registers* specified (≤ 100). 16 *local flags* come with them. Local data are valid in the current routine only.

Tests (Do if True, Skip if False)

When a binary test step is executed, the program step immediately following it is executed if the test result is “true”; if the result is “false”, the step following the test step is skipped.

Looping

ISE, ISG, ISZ, DSE, DSL, and DSZ (found in **LOOP**) control looping. Each accesses a variable or *register* containing a **loop control number** in the form ccccc.fffii with ccccc being the current counter value, fff the final counter value, and ii the increment (or decrement) size (default is 1); DSZ and ISZ count to 0 in steps of 1. As long as the count is not complete, the step following the instruction is executed (usually a branch to the top of the loop). The example pictured counts from 1 to 52 by threes (executing the loop 18 times) and then beeps.

```
...
1.05203
STO 'Count'
LBL 01
...
ISG 'Count'
GTO 01
BEEP
...
```

Using a Variable Menu

A *variable menu* may be displayed by the *Solver* or *Integrator* (see pp. Q-10f) or by VARMNU within a program. Each label in this *menu* represents a variable. While this *menu* is displayed, you can:

Store a value into a variable: Key in the value and then press the *softkey*.

Recall the contents of a variable: Press **RCL** and then the *softkey*.

View the contents of a variable without recalling it: Press **VIEW** and then the *softkey*.

Select a variable: Press the corresponding *softkey* without keying in a number first (for the *Solver*, this is how you select the unknown variable; for the *Integrator*, this is how you select the variable of integration).

You can call and use any function *menu* without exiting from the *variable menu*.

EXECUTING FUNCTIONS AND PROGRAMS

Any function or program can be executed via **XEQ** **α name** **ENTER↑** where **name** is the function *name* or the program label. If **name** is not unique, the global label closest to the permanent end (.END.) has precedence. If **name** is a local label, WP 43S searches in the current program only.

Smart program menu: **XEQ PROG** displays all programs (actually: all global labels) defined. Specify the required program by pressing the corresponding *softkey*.

Single stepping: To execute the current program step, press **≡V** (or **▼** if no *multi-view menu* is displayed). Press **≡A** (or **▲**) for browsing backwards.

Run/Stop key: Press **R/S** to run the current program beginning with the current step or to stop the running program after the current step is executed completely.

The catalog of functions: Browse **CATALOG FCNS** and execute the required function by pressing the corresponding *softkey*. This catalog can also be searched alphabetically. Avoid using function names twice!

Specifying Function Parameters

Numeric parameters: Functions accepting numeric parameters prompt you with a cursor for each digit expected. To key in a numeric parameter, just enter its digits. If you provide a digit for each underscore, the function will execute. You can also provide less digits and finish input with **ENTER↑**.

Alphanumeric parameters: Many functions accept alphanumeric parameters as well. The parameter you want will often be an object already existing, so your *WP 43S* will display a *menu* for quick entry. If it does not exist yet, type it. E.g. for creating a variable **ABC** just type **STO** **α ABC** **ENTER↑**.

Stack parameters: Any function accepting a ‘usual’ *register* as parameter also accepts a *stack register*. Just press the corresponding *softkey* for **X** ... **T** or the keys in second row for **A** ... **D**, if applicable.

Indirect addressing: Rather than keying in an actual parameter, you can specify the variable or *register* containing the parameter. Just press the *softkey* **→**. E.g. to display the contents of the variable or *register* specified in **R12**, key in **VIEW** **→ 12**. This works with *stack registers* as well.

CLEARING AND DELETING

CLR	CLall		DELITM		RESET	
	CLREGS	CLPall	CLFall		CLLCD	CLSTK
	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX

- | | |
|--|---|
| CLS | Clears all statistical data. |
| CLP | Clears (deletes) the <i>current program</i> . |
| CF <i>n</i> | Clears <i>flag n</i> . |
| CLMENU | Clears the <i>programmable menu</i> . |
| CLCVAR | Clears all variables used in the <i>current program</i> . |
| CLX | Clears <i>stack register X</i> . |
| CLREGS | Clears all <i>registers</i> (except the <i>stack</i> and statistical data). |
| CLPALL | Clears (deletes) all programs in <i>RAM</i> . |
| CLFALL | Clears all numbered user <i>flags</i> . |
| CLLCD | Clears the <i>LCD</i> above and to the right of pixel <i>x, y</i> . |
| CLSTK | Clears the entire <i>stack</i> (i.e. fills all its <i>registers</i> with zero). |
| CLALL | Clears almost everything but the modes set. |
| RESET | Resets the <i>WP 43S</i> to <i>startup default configuration</i> . |
| DELITM CATALOG VARS ... <input type="checkbox"/> | deletes the user variable selected. |
| DELITM CATALOG MENUS ... <input type="checkbox"/> | deletes the user <i>menu</i> selected. |
| DELITM CATALOG PROGS ... <input type="checkbox"/> | deletes the user program selected. |

MATRIX OPERATIONS

A matrix is an array with m rows and n columns of real or complex elements.

To create a new $m \times n$ matrix, enter its dimensions (m **ENTER↑** n) and press

[MATX] NEW for a matrix in X or

[MATX] DIM α *name* **ENTER↑** for a matrix in a named variable. If the variable already exists, DIM re-dimensions it.

To edit the matrix in X, use **[MATX] EDIT**.

To edit a named matrix, use **[MATX] EDITN** *name*.

When a matrix is being edited it is said to be *indexed* (to index a named matrix without editing it, use INDEX). Whenever there is an indexed matrix, two pointers are used to indicate the row and column of the current element: they are stored in I and J, respectively. If I and J are pointing to the last element (bottom right) in a matrix and you press **[→]** then ...

- ...the pointers wrap around to the first element of the matrix (**Wrap mode**, automatically set whenever you enter or exit the *Matrix Editor*) or ...
- ...the matrix grows by one complete row and the pointers move to the first element in the new row (**Grow mode**).

WRAP and GROW are in the **f**-shifted row of the *Matrix Editor menu*.

Matrix arithmetic: **[+]**, **[-]**, **[×]**, and **[/]** work for matrices just as for individual numbers. Advanced functions often operate on the individual matrix elements. Any time a matrix is used in a mathematical operation with a complex object, the result will be a complex matrix.

To solve a system of simultaneous linear equations represented by the matrix equation $(A)\vec{X} = \vec{B}$:

- Enter **[MATX] SIM EQ** n with n being the number of unknowns. Your WP 43S will automatically create or re-dimension the matrix variables **Mat_A** and **Mat_B**.
- Press **[Mat A]**; fill the matrix; press **EXIT**.
- Press **[Mat B]**; fill the matrix; press **EXIT**.
- Press **[Mat X]** to compute the solution matrix.

PROBABILITY

	RAN#	SEED	RANI#			$\Gamma(x)$
PROB		NBin:	Geom:	Hyper:	Binom:	Poiss:
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:
	Norml:	t:	C _{yx}	P _{yx}	F:	χ^2 :
Binom:	Binom _p		Binom _▲	Binom _△		Binom ⁻¹

- C_{yx}, P_{yx} Returns the number of possible combinations (or permutations, a.k.a. arrangements) of x items taken out of a set of y items.
 RAN# Returns a random real number between 0 and 1.
 SEED Stores a seed for RAN#.
 RANI# Returns a random integer number between x and y .
 $\Gamma(x)$ Returns the *Gamma function* value of x .

These 14 continuous and discrete distributions (d) are provided:

- Binom:** Binomial d. ($i = p_0$ = gross probability of success, $j = n$ = sample size)
Cauch: Cauchy-Lorentz (a.k.a. Breit-Wigner) d. (i = location, j = shape)
Expon: Exponential d. (i = rate)
F: Fisher's F d. (i = degrees of freedom 1 (dof_1), j = dof_2)
Geom: Geometric d. ($i = p_0$)
Hyper: Hyperbolic d. ($i = p_0$, $j = n$, k = batch size)
LgNrm: Log-normal d. ($i = \mu$, $j = \sigma$)
Logis: Logistic d. ($i = \mu$, j = scale parameter)
NBin: Negative Binomial d. ($i = p_0$, $j = n$)
Norml: (General) normal d. ($i = \mu$, $j = \sigma$)
Poiss: Poisson d. ($i = n p_0$ = Poisson parameter)
t: Student's t d. ($i = dof$)
Weibl: Weibull d. (i = shape, j = characteristic lifetime)
 χ^2 : Chi-square d. ($i = dof$)

Following naming convention holds for most distributions, e.g. for the *normal d.*: **Norml_p** denotes the *probability density function*, **Norml_▲** the *cumulated d. function*, **Norml_△** the *error probability*, and **Norml⁻¹** the *quantile function*.

Store the required parameters in **I**, **J**, and **K** as listed above; the remaining parameter must be given in **X** before calling the respective function – note the *quantile functions* require a probability input in **X** ($0 \leq x \leq 1$).

STATISTICS

Statistical data are accumulated in 23 dedicated summation *registers*, kept separate from all the other *registers* introduced above.

Clear the statistical registers before doing a new stat. analysis: **STAT CLΣ**.

Then, accumulate the data:

- For each individual data value: **x-value Σ+**.
- For each weighted data value: **weight-value ENTER↑ x-value Σ+**.
- For each x-y data pair or point: **y-value ENTER↑ x-value Σ+**.
- For x-y data pairs stored in a two-column matrix (*x-values* in column 1, *y-values* in column 2): place the complete matrix in **X** and then press **Σ+**.

To undo input errors or remove erroneous data,

- either press **UNP** (for the very last data point or input)
- or recall the (earlier) incorrect y and x data in the *stack* and press **Σ-**.

Data Evaluation and Analysis

	GaussF	CauchF	ParabF	HypF	RootF		
	LinF	ExpF	LogF	PowerF		BestF	
		\bar{x}_{RMS}	x_{max}	x_{min}		OrthoF	
		\bar{x}_H					
	L.R.	r	s_{xy}	COV	\hat{x}	\hat{y}	
STAT	CLΣ	\bar{x}_G	ε	ε_p	ε_m	PLOT	
	Σ^-	\bar{x}_w	s_w	σ_w	s_{mw}		
	Σ^+	\bar{x}	s	σ	s_m	SUM	

\bar{x} , s, σ , s_m Arithmetic mean value, sample standard deviation (SD), population SD, standard error (a.k.a. SD of the mean).

\bar{x}_w , s_w , σ_w , s_{mw} Same for weighted data.

\bar{x}_G , ε , ε_p , ε_m Geometric mean value, sample scattering factor (SF), population SF, SF of the mean.

\bar{x}_H , \bar{x}_{RMS} , x_{max} , x_{min} Harmonic and quadratic means, max. and min. values.

SUM Recalls Σy and Σx .

PLOT See the ReM.

L.R. Computes the parameters a_0 and a_1 (and a_2 , if applicable) of the fit model selected (see below).

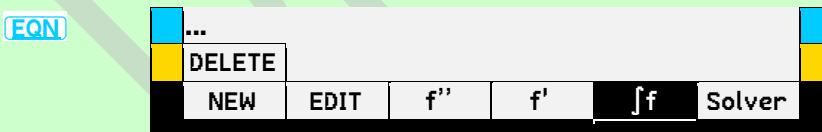
r	Returns the correlation coefficient.
s_{xy} , cov	Return the sample or population covariance.
\hat{x}, \hat{y}	Return the forecast for x or y according to the fit model selected.
LinF	Linear fit model: $y = a_0 + a_1x$.
ExpF	Exponential fit model: $\ln(y) = \ln(a_0) + a_1x$ or $y = a_0 e^{a_1 x}$.
LogF	Logarithmic fit model: $y = a_0 + a_1 \ln(x)$.
PowerF	Power fit model: $\ln(y) = \ln(a_0) + a_1 \ln(x)$ or $y = a_0 x^{a_1}$.
RootF	Root fit model: $y = a_0 a_1^{1/x}$.
HypF	Hyperbolic fit model: $y = 1/(a_0 + a_1 x)$.
ParabF	Parabolic fit model: $y = a_0 + a_1 x + a_2 x^2$.
CauchF	Cauchy peak fit model: $y = 1/[a_0 (x + a_1)^2 + a_2]$.
GaussF	Gauss peak fit model: $y = a_0 e^{\frac{(x-a_1)^2}{a_2}}$.
BestF	Blindly selects the model returning the best correlation coefficient.
OrthoF	Works like LINF but assumes equal errors in x and y. Note that ORTHOF is not part of the fit model pool BESTF investigates.

ADVANCED OPERATIONS

[EQN] is for interactive editing, storing, recalling, solving, integrating, & deriving equations.

[ADV] is for programmed summing, multiplying, solving, integrating, & deriving.

Interactive Operations on Equations



For creating a new equation, press **[NEW]**. The *Equation Editor menu* will open, and the blue row will display the current equation. Press **[EXIT]** when finished.

For browsing existing equations, press **[▲]** or **[▼]**. The equation displayed in **[g]-shifted** row is called the *current equation*.

For editing (or deleting) the current equation, press **[EDIT]** (or **[DELETE]**).

For operating on the current equation, press the respective *softkey*. A *menu* will pop up displaying the *names* of all variables used and more.

Using Advanced Operations in Programs

ADV	PGMSLV	f''(x)				
SOLVE	SLVQ	f'(x)	Π_n	Σ_n	$\int f dx$	

SLVQ solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input stack [c, b, a, \dots]. It returns two real or complex solutions.

Π_n label calculates the product of the terms given by the routine specified, using the loop control number given in **X** (cf. p. Q-4).

Σ_n label calculates the sum of the terms given by the routine specified, using the loop control number given in **X** (cf. p. Q-4).

SOLVE var solves for an unknown variable in an expression, given values for all the other variables. The expression $f(x_1, x_2, \dots)$ shall be written as a program (let's call it **AB**, for example):

- **AB** must begin with a global label.
- The body of **AB** shall evaluate the expression. For an expression to be solved, it must be coded that $f(x_1, x_2, \dots) = 0$ is fulfilled. Recall the variables of the expression as they are required and calculate f .
- **AB** must logically end with RTN.

Then write a program calling the Solver (let's call it **CD**, for example). At the position where you need the expression solved, press **ADV**:

1. Press **PGMSLV** and specify **AB**.
2. Store a value into each known variable, e.g. using **STO**. Optionally store a guess into the unknown variable to direct the *Solver* to a solution.
3. Press **SOLVE** and specify the unknown variable.

When running **CD** later on, **SOLVE** will solve for the unknown.

f'(x) (or f''(x)) calculates the first (or second) derivative of $f(x)$ at location x . The function $f(x)$ shall be written as a program (e.g. called **EF**); it must begin with a global label, take care of all variables used, and evaluate $f(x)$.

Then write a program calling the derivator (let's call it **GH**, for example).

1. Store a value into each of the variables that shall remain constant under derivation.
2. At the position where you need the derivative, put the respective location into **X**, then press **ADV f'(x)** (or **f''(x)**) specifying **EF**.

When running **GH** later on, the derivative will be returned in **X**.

∫fd var numerically computes a definite integral. The integrand $f(x)$ shall be written as a program (e.g. called **IJ**); it must begin with a global label, recall all integration constants used, and evaluate $f(x)$.

Then write a program calling the *Integrator* (let's call it **KL**, for example). At the position where you need the integral, press **ADV**:

1. Press **∫fdx**. A submenu will open.
2. Press **PGMINT** and specify **IJ**.
3. Store a value into each of the variables that shall remain constant under integration, e.g. using **STO**.
4. Store the lower limit (**↓LIM**), the upper limit (**↑LIM**), and the accuracy factor (**ACC**).
5. Press **∫** and specify the variable of integration.

When running **KL** later on, the integral will be returned in **X** and the uncertainty of computation will be returned in **Y**.

OPERATIONS ON SHORT INTEGERS

	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
LJ					RJ		
SL	RL	RLC	RRC	RR	SR		
SB	BS?	#B	FB	BC?	CB		
NAND	NOR	XNOR		MIRROR	ASR		
AND	OR	XOR	NOT	MASKL	MASKR		

AND, OR, XOR, NAND, NOR, XNOR *Boole's binary operators.*

NOT	Inverts every bit in X .
MASKL, MASKR	Creates a mask of x bits on the left (or right) side.
MIRROR	Reflects all bits.
ASR n	Arithm. shifts x to the right by n places = divides x by 2^n .
SB, FB, or CB n	Sets, flips, or clears bit # n in x (counting starts with #0).
BS?, BC?	Checks if bit # n in x is set (or clear).
#B	Returns the number of bits set in x .
SL, SR n	Shifts x left (or right) by n places.
RL, RR n	Rotates x left (or right) by n places.
RLC, RRC n	Rotates x left (or right) by n places through Carry.
LJ, RJ	Adjusts the bits set in x to the left (or right).
1COMPL, 2COMPL	Sets 1's (2's) complement mode.

UNSIGN Sets unsigned mode.
SIGNMT Sets sign-and-mantissa mode.
WSIZE Sets the word size to 1 ... 64 bits.

	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
INTS	DBL /	DBLR	DBL ×	\wedge MOD	CEIL	GCD	
	IDIV	RMD	MOD	\times MOD	FLOOR	LCM	
	A	B	C	D	E	F	

A ... F Digits for input of short integers of bases >10.
IDIV $\mathbb{R} \mathbb{Z}$ Integer divide – works also for real numbers (\mathbb{R}) and long integers (\mathbb{Z}).
RMD, MOD $\mathbb{R} \mathbb{Z}$ Remainder and modulo.
 \times MOD $\mathbb{R} \mathbb{Z}$ Returns $(z \cdot y) \bmod x$.
 \wedge MOD $\mathbb{R} \mathbb{Z}$ Returns $(z^y) \bmod x$.
FLOOR \mathbb{R} Returns the greatest integer $\leq x$.
CEIL \mathbb{R} Returns the smallest integer $\geq x$.
LCM \mathbb{Z} Returns the least common multiple of x and y .
GCD \mathbb{Z} Returns the greatest common divisor of x and y .
DBL /, DBLR, DBLx Double word length commands for division, remainder, and multiplication.

OPERATIONS ON ALPHANUMERIC STRINGS

Connect strings by pressing + . Then x will be appended to the string y . With numeric data in \mathbf{X} , their current display format is taken into account.

a.FN	FBR				α LENG?	α POS?	
	$x \rightarrow \alpha$	α RL	α RR	α SL	α SR	$\alpha \rightarrow x$	

$x \rightarrow \alpha \ s$ Converts a code x to the corresponding character and appends it to the string in s .
 α RL, α RR s Rotates the string in s by x characters to the left (or right).
 α SL, α SR s Deletes the first (or last) x characters of the string in s .
 $\alpha \rightarrow x \ s$ Pushes the code of the first character in s on the stack.
 α LENG? s Pushes the length of the string in s on the stack.
 α POS? s Returns the position where substring x begins in the string in s .
FBR Displays all characters defined in both fonts.

BACKGROUND CONSIDERATIONS AND FACTS

This section is for recording and explaining some of the boundary conditions considered and settings chosen for the *WP 43S* in the course of this project. It is not necessary for operating the *WP 43S* but may foster understanding; and a bit of the product philosophy may be found here, too.

Accessing Items

The hardware offers 43 keys for some 740 *items*. Subtract six for the *softkeys*. Space for primary functions is quickly occupied – the respective functions are mostly set.

Obvious primary functions are the digits **0** ... **9**, **.**, **ENTER↑**, **x^y**, **±**, **E**, **⬅**, **+**, **-**, **x**, **/**, **STO**, **RCL**, **XEQ**, **R/S**, **▲** and **▼**, **EXIT**, **f** and **g**, taking 29 key tops. So eight locations are left. Once you want to deal with *complex numbers* seriously, you will need a primary **CC**.

Other functions may be debated: **R↓**, **1/x**, **y^x**, **x²**, **✓x**, **e^x**, **In**, **10^x**, **lg**, **sin**, **cos**, and **tan** are the most popular – twelve candidates for seven key tops left. Either **x²** or **✓x** shall be primary (we chose **x²** since a shifted **✓x** is of little use); and we can ditch **10^x** and **lg** when we have **e^x** and **In** primary. So the six functions **R↓**, **1/x**, **y^x**, **sin**, **cos**, and **tan** compete for the remaining four key tops. We chose the first three and put the latter three (and their inverses) under one primary *menu* key: **TRI**; thus, you can access each of **sin**, **cos**, **tan**, **arcsin**, **arccos**, and **arctan** with two keystrokes maximum.

The losing candidates **✓x**, **10^x**, and **lg** shall become secondary functions. But is it better having them as shifted keyboard functions or unshifted *softkeys*? No definite answer can be given here since it depends on the time the respective *menu* will stay on screen. For these three functions, we made all **g**-shifted and put **✓x** also in the unshifted row of **EXP** since it is the most popular of these.

WP 43S features 33 *menus* on its keyboard. Of these, CATALOG, CONST, U→, and the three alpha character *menus* shall be separated since they follow special rules. Each of the other 27 *menus* offers six unshifted locations for its most popular *items*. Selecting these can be easy (like in ADV, LOOP, STK, or TRI) or more difficult (like in INFO, P.FN, or X.FN).

Unshifted softkeys are (cf. pp. 121ff):

<u>ADV</u>	SOLVE	SLVQ	$f'(x)$	Π_n	Σ_n	$\int f dx$
<u>BITS</u>	AND	OR	XOR	NOT	MASKL	MASKR
<u>CLK</u>	DATE	\rightarrow DATE	DATE \rightarrow	WDAY	TIME	$x \rightarrow$ DATE
<u>CLR</u>	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX
<u>CPX</u>	dot	cross	UNITV	Re	conj	Re \Rightarrow Im
<u>DISP</u>	FIX	SCI	ENG	ALL	ROUNDI	ROUND
<u>EQN</u>	NEW	EDIT	f''	f'	$\int f$	Solver
<u>EXP</u>	x^3	$\sqrt[3]{y}$	$\log_x y$	lb x	2^x	\sqrt{x}
<u>FIN</u>	%	%MRR	%T	%Σ	%+MG	TVM
<u>FLAGS</u>	SF	FS?	FF	STATUS	FC?	CF
<u>INFO</u>	SSIZE?	MEM?	RM?	ISM?	WSIZE?	KTYP?
<u>INTS</u>	A	B	C	D	E	F
<u>I/O</u>	LOAD	LOADP	LOADR	LOADSS	LOADV	LOADΣ
<u>LOOP</u>	DSE	DSZ	DSL	ISE	ISZ	ISG
<u>MATX</u>	NEW	$[M]^{-1}$	M	$[M]^T$	SIM EQ	EDIT
<u>MODE</u>	SF	DEG	RAD	GRAD	MULπ	CF
<u>PARTS</u>	IP	FP	MANT	EXPT	sign	DECOMP
<u>PRINT</u>	$\blacksquare x$	$\blacksquare r$	$\blacksquare \Sigma$	\blacksquare ADV	\blacksquare LCD	\blacksquare PROG
<u>PROB</u>	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2 :
<u>P.FN</u>	INPUT	END	ERR	TICKS	PAUSE	P.FN2
<u>STAT</u>	$\Sigma +$	\bar{x}	s	g	s_m	SUM
<u>STK</u>	$x \gtrless$	$y \gtrless$	$z \gtrless$	$t \gtrless$	\gtrless	DROPy
<u>TEST</u>	$x < ?$	$x \leq ?$	$x = ?$	$x \neq ?$	$x \geq ?$	$x > ?$
<u>TRI</u>	sin	arcsin	cos	arccos	tan	arctan

<u>U→</u>	E:	P:	year→s	F&p:	m:	x:
X.FN	AGM	B _n	B _n *	erf	erfc	Orthog
α.FN	x→α	αRL	αRR	αSL	αSR	α→x
Σ	n	Σx	Σx ²	Σxy	Σy ²	Σy
→	→DEG	→RAD	→GRAD		→D.MS	→MULπ

All these functions can be accessed via a single keystroke if and when their *menu* is open, else via three keystrokes. Thus, repeating any unshifted *softkey* as a shifted label on the keyboard is of limited value; and there are just two cases where this is done (\sqrt{x} and $\sqrt[4]{x}$ as well as STATUS and **STATUS**).

|x| and ↵ are also featured as shifted softkeys: accessing |x| or ↵ needs two keystrokes while |x| and ↵ require four maximum (and two if the *menu* is open). In consequence, |x| and ↵ are actually not needed in any *menu* but are left in the related menus Cpx and PARTS since space is available there so far.

See also *Layouting* on pp. B-17f.

Alpha Register

For long I thought we could do without a dedicated *alpha register* since each and every *register* is capable holding an *alphanumeric string*. Some special programming functions like KEYG and KEYX, however, seem to require such a *register* – else handling these functions would become more complicated than it was on the *HP-42S*.

Especially direct entry of alphanumeric constants in programs is easier when the destination is automatically defined, and people became used to this method in decades since the *HP-42S* was launched. Thus, I introduced this *register* in v0.7, taking K for it (cf. the OM, Section 3).

Angles

Originally, a separate *DT* for *angles* was planned. It was removed in v0.9 since its scope is quite limited and the opinion rose that ‘*angles* work like *real*

numbers'. It turned out, however, that D.MS data would need special treatment in calculations, so *DT* 4 returned with v0.10 for sake of keeping algebraic operations simple and avoiding dedicated commands like D.MS+, D.MS-, etc.

Actually, *angles* are displayed in five 'modes' (*decimal* and *sexagesimal degrees*, *radians*, *multiples of π* , and *gon* or *grades*). They were represented internally in a fixed format of 1296 units per turn – similar to *short integers* where a fixed bit pattern may be displayed differently depending on *integer sign modes* and bases selected. *Radians*, however, did not fit into this concept due to the need for high precision storage of π for modulo calculations and reduction of rounding errors. And *radians* are inevitable since Taylor series for trigonometric functions are written for angular input in *radians*. So, internally, *angles* are tagged *reals* now.

Generally, trigonometric functions shall actually operate on *angles* within $\pm 180^\circ$ only; thus, angular input beyond this range shall be reduced modulo 360° , then minus 180° (or equivalents in the other *angular display modes* available) before executing the function. Again, the crucial mode are *radians*. *WP 34S* had demonstrated that 451 digits for 2π suffice to warrant 16 digits accuracy of respective function results for the number range of *single precision* reals (see <https://forum.swissmicros.com/viewtopic.php?f=2&t=350#p4349>). *WP 43S* uses 1065 digits for 2π to warrant 34 digits accuracy of respective function results within $\pm 10^{999}$.

Backward Compatibility

Compatibility to *WP 34S* and *HP-42S* was planned to be kept for many years in a way that programs written for both calculators could have run on the *WP 43S* as well (except matrix operations and some flag allocations). It became difficult with the full implementation of the *DT* concept and had to be eventually abandoned officially when introducing named *system flags* with v0.14. On the other hand, we were allowed to eliminate some *HP-42S* bugs this way (e.g. the *Matrix Editor* pushing 0 on the stack).

Nevertheless, *names* of *items* were kept as close as possible to the *names* users are used to unless there were striking reasons for better *names*. Extra entries are provided catching traditional *names*.

Calculation Internals

General powers in \mathbb{R} and \mathbb{C} are calculated as $y^x = e^{x \ln(y)}$ and general roots as $\sqrt[x]{y} = e^{\frac{\ln(y)}{x}}$ for all values of x except the integers 2 and 3. For odd ('integer') roots of $y < 0$, $\sqrt[x]{y} = -e^{\frac{\ln(-y)}{x}}$; here, 'integer' includes DT 1 and 10 and reals with zero fractional part. Some special results in [App. B](#) can be deduced from these calculation paths.

Note that *Free42* may calculate even powers differently (as a series of multiplications using repeated squaring), never employing more than 34 digits (see also p. B-21).

Character Sets

The browser FBR displays the characters of both fonts provided as designed and implemented for the *WP 43S*, sorted according to their hexadecimal codes (most of them following *Unicode*).

The so-called '**numeric**' font uses a matrix of up to 16×32 px (variable width, fixed height). Therein, the punctuation space (2008_{16} , 8 px wide) is employed for separating groups of digits in longer numbers – following ISO 80000-1 for an unambiguous numeric display. This font is generally used for numeric output of the *WP 43S*. It is also employed for echoing numeric input unless too long. It can be used for echoing command input as well – screen space suffices.

In total, six blank characters are provided here allowing for any spacing wanted (standard / em / figure, punctuation, four-per-em, and hair space being 16, 8, 4 and 1 px wide).

Most of the elevated characters are for exponents or fraction numerators. The digits below are for denominators. Numeric indices are for indicating bases of short integers. Non-numeric indices are mainly provided for CONST.

Optionally, narrower digits can be used in *complex numbers*, matrices, or *short integers* of small base where space may be scarce (see pp. B-7ff).

All characters of the **standard** (a.k.a. small) **font** of alphanumeric characters as designed and implemented live in a matrix of up to 14×20 px (variable width, fixed height again). Herein, characters usually start at column one and feature

two empty columns at their right side. There are a few exceptions: see e.g. the multiplication dot at $00B7_{16}$ and the root symbols in row 2210_{16} .

Characters with codes $< 0020_{16}$ are for control purposes; some of them ($4, 10_{10}, 27_{10}$) may be useful for printer control (e.g. of an *HP 82240 A/B*).

Many characters are 8 px wide as digits – they will help where a constant character spacing is wanted.

There is a number of super- and subscripts provided. They allow for displaying all the *items* featured on the *WP 43S*. Arbitrary numeric indices or exponents are possible as well.

Eight blank characters are provided (listed here with their hex addresses and widths). Using them, any spacing is feasible.

	Code	px
Standard space	20_{16}	10
m space	2003_{16}	12
m/3 space	2003_{16}	4
m/4 space	2003_{16}	3
m/6 space	2003_{16}	2
Figure space	2003_{16}	8
Punctuation sp.	2003_{16}	4
Hair space	2003_{16}	1

This small character set allows for correctly spelling the languages of more than 3.5×10^9 people using either Greek or Latin alphabets:

Afrikaans, aymara, Bahasa Indonesia, Bahasa Melayu/Malaysia, Basa Jawa, Basa Sunda, Bhāṣā Bali, bosanski, català, Cebuano (Bisayan), čeština, Cymraeg, dansk, Deutsch, eesti, Ελληνικά, English, español, euskara, Filipino, français, Gaeilge, galego, Hiligaynon, hrvatski, Ilokano, italiano, Kiswahili, kreyòl (ayisien), kurdî, lietuvių, magyar, Malagasy, Nāhuatl (Mexicatlatolli), nederlands, nihongo (rōmaji), norsk, Ӧ'zbek tili, polski, português, quechua (runasimi), română, shqip, slovenščina, slovensky, srpski, suomi, svenska, Tagalog, tatarça, Türkçe, Türkmen dili, Vlaams, walon, Waray, and zhōngwén (hànyǔ pīnyīn).

This makes the *WP 43S* the most versatile multilingual calculator available worldwide. If you know of further living languages covered (with ≥ 1 million speakers) beyond the ones listed here, please tell us.

Turn to the OM for examples where and how these characters are used. See here two sample strings in either font, printed approximately to a common realistic scale:

$$-1.602\ 22\times 10^{-19}\text{ C} \quad -1,602\ 22\cdot 10^{-19}\text{ C}$$
$$-1.602\ 22\times 10^{-19}\text{ As} \quad -1,602\ 22\cdot 10^{-19}\text{ As}$$

Some characters displayed by FBR are not found in any other menu of your WP 43S. They are not required for any item provided so far and may be for future use.

Complex Notation and Storage

Like with angles or short integers, there are different ways a *complex number* can be written: either in Cartesian or polar notation, the latter with all kinds of angular units. As long as you stay away from infinities, any notation will do.

For reasons of mathematical tradition, rectangular notation is found most frequently. We used it for storing *complex numbers* in the WP 34S. Thus, it is used for the WP 43S as well, but care must be taken at **complex infinities**:

Since infinities may be counted as numeric data being part of the *real number* range (see p. 169), also complex infinities may be part of the *complex number* plane the WP 43S operates on. Coming from rectangular notation, there are eight ‘complex infinities’ possible only, listed here aside to their equivalents in polar notation:

$\text{Re}(z)$	$\text{Im}(z)$	$r(z)$	$\varphi(z)$	
$-\infty$	$-\infty$	∞	-135°	$-3\pi/4$
0	$-\infty$	∞	-90°	$-\pi/2$
∞	$-\infty$	∞	-45°	$-\pi/4$
∞	0	∞	0°	0
∞	∞	∞	45°	$\pi/4$
0	∞	∞	90°	$\pi/2$
$-\infty$	∞	∞	135°	$3\pi/4$
$-\infty$	0	∞	180°	π

Note the phase is counted counterclockwise, starting with $\varphi = 0$ at the positive real axis.

Calculating with infinities, any finite number may be neglected in comparison; so for $|x| \neq \infty$ any inputs like e.g. $x + i \times \infty$ may be replaced by $0 + i \times \infty$, easing calculations significantly. Actually, you have to deal with the eight cases listed above only as long as you build your complex calculations on Cartesian notation (see footnote 82 for additional information). With polar notation, on the other hand, an infinite number of complex infinities would have to be treated.

Note, however, that polar notation is advantageous for complex powers and roots: the n^{th} root of a *complex number* ($r; \varphi$) in polar notation will return $(\sqrt[n]{r}; \varphi/n)$. Hence, e.g. $\sqrt[3]{(\infty; 180^\circ)} = (\infty; 60^\circ)$, corresponding to the Cartesian point $\infty \times (1 + i\sqrt{3})$ which the calculator will display as $\infty + i \times \infty$, converted following the calculation rules – but mathematically wrong; and $\sqrt[6]{(\infty; 180^\circ)} = (\infty; 30^\circ)$, corresponding to the Cartesian point $\infty \times (\sqrt{3} + i)$, would return $\infty + i \times \infty$ as well.

Integer powers of $(r; \varphi)$, on the other hand, will return $(r^n; \varphi \times n)$. E.g. $\infty + i \times \infty = (\infty; 45^\circ)$ squared shall return $(\infty; 90^\circ) = 0 + i \times \infty$. Naïve pedestrian's approach: $(\infty + i \times \infty)(\infty + i \times \infty) = \infty - \infty + 2i \times \infty = 0 + i \times \infty$. Note the two ∞ are exactly identical here; but $\infty - \infty$ is generally defined as NaN for good reasons, so the properly calculated result will deviate from the truth here, too.

Thus, the odds are high that roots and powers of *complex numbers* near the far edge of complex plane will return incorrect data, in particular if specified in polar notation. These errors seem to be inevitable.

Display Limits

Due to the character sizes and their design (cf. pp. B-5f), the screen could take inputs of up to 23 digits, a sign, and an 8-px radix mark:

-4.2345678901234567890123 ,

occupying $15 + 23 \times 16 + 8 = 391$ px. Numeric output would allow for the same 23 digits. Without digit group separators, however, this would hardly be readable. With 3-digit separators (*startup default*), 20 digits are displayable in one row instead:

-4.234 567 890 123 456 789 0 ,

taking $15 + 20 \times 16 + 7 \times 8 = 391$ px again. This maximum precision is independent of the position of the radix mark. Scientific or engineering notation allows for a 16-digit mantissa

$-4.234\ 567\ 890\ 123\ 456 \times 10^{-925}$,

taking 395 px ($= 15 + 16 \times 16 + 5 \times 8 + 15 + 16 + 4 \times 13 + 1$) for displaying this number this way. Note that 1 blank pixel column had to be added at right since exponential digits are right adjusted (since used for numerators as well) and the screen is framed in black.

With SHOW, any *real number* can be displayed with 34-digits precision in a single row:

2020-01-06 17:28 Cls.4^r /max 64:2 A S L
-1.428 571 428 571 428 571 428 571 429 $\times 10^{-235}$
 $-1.428\ 571\ 428\ 571\ 429 \times 10^{-235}$

Some *temporary information* may limit output precision, though without limiting its use for real-world applications. E.g. for linear regression, up to 8 digits are viable allowing for 2-digit exponents in SCI or ENG and up to 12 digits in FIX:

Logarithmic* $a_1: -5.234\ 567\ 8 \times 10^{-92}$
 $y = a_0 + a_1 \ln(x)$ $a_0: -1.234\ 567\ 890\ 12$

Complex numbers in Cartesian notation require $1 + 15 + 12 + 15 + 1 = 44$ px for $+jx$ in addition to the space for two reals. Only the real part may need extra space for a 15-px sign. This allows for 8 decimals per part in worst case

$-4.234\ 567\ 89 + j \times 4.234\ 567\ 89$,

since $44 + 15 + 2 \times (16 + 8 + 48 + 8 + 48 + 8 + 32) = 397$ px in total. It applies if both real and imaginary parts are in the same order of magnitude and the multiplication cross is chosen.

With SCI or ENG, a minimum of 3 decimals can be shown ($15 + 2 \times (4 \times 16 + 8 + 15 + 16 + 4 \times 13) + 44 + 1 = 370$ px, but another digit would need 2×16 px at least):

$-4.234 \times 10^{-925} + i \times 4.234 \times 10^{-925}$.

Using 8 px wide multiplication dots instead, only $1 + 15 + 12 + 8 + 1 = 37$ px are necessary for $+i$. We can display one decimal more now since $15 + 2 \times (5 \times 16 + 3 \times 8 + 16 + 4 \times 13) + 37 + 1 = 397$ px:

$$-4,234\ 5 \cdot 10^{-925} + i \cdot 4,234\ 5 \cdot 10^{-925}$$

Alternatively, 13 px wide narrow digits allow for 4 decimals even with multiplication crosses, while 5 decimals are viable with multiplication dots:

$$-6.234\ 5 \times 10^{-925} + i \times 6.234\ 5 \times 10^{-925}$$

$$-6.234\ 56 \cdot 10^{-925} + i \cdot 6.234\ 56 \cdot 10^{-925}$$

With SHOW, any *complex number* can be displayed with 34 digits precision in two rows:

2020-01-06 17:15 CL_E^r /max 64:2 A ↑ S
 $-8.403\ 361\ 344\ 537\ 815\ 126\ 050\ 420\ 168\ 067\ 229 \times 10^{-126}$
 $-i \times 5.882\ 352\ 941\ 176\ 470\ 588\ 235\ 294\ 117\ 647\ 059 \times 10^{-158}$

$$-1 \cdot 10^{-33}$$

$$-8.403 \times 10^{-126} - i \times 5.882 \times 10^{-158}$$

Complex numbers in **polar notation** need $4 + 16 + 4 = 24$ px for \angle plus 16 px for the angular unit in addition to the space for two signed reals. Both magnitude and angle may require a 15 px sign. 7 decimals in FIX occupy $40 + 2 \times (15 + 8 \times 16 + 3 \times 8) = 374$ px, so we can display them this way:

$$-4.234\ 567\ 8 \angle -0.234\ 567\ 8\pi$$

With SCI or ENG, the minimum number of decimals depends on the angular display mode since output is confined to the interval -180° to $+180^\circ$ or its equivalents, e.g. $-\pi$ to $+\pi$ in *radians* or -200^g to $+200^g$ in *gon* (see Section 2 of the OM). Hence, the angular parts can be displayed without exponents always. This allows for a minimum of 4 decimals for *degrees* and *gon*:

$$-4.234\ 5 \times 10^{-925} \angle -120.234\ 5^\circ$$

For *radians* or *multiples of π* , however, 5 decimals are displayable always at least:

$$-4.234\ 56 \times 10^{-925} \angle -0.234\ 56\pi$$

Digits in **fractions** are 13 px wide like in exponents. Thus, a 4-digit numerator and denominator take $4 \times 13 + 8 = 60$ px each; the fraction bar takes another 16 px and the trailer 29 ($= 16 + 12 + 1$). The remaining 235 px would suffice for the optional sign, an 11-digit number, and the 16 px gap between integer and fraction ($15 + 12 \times 16 + 3 \times 8 = 231$ px) in a proper fraction:

-67 890 234 567 2 289/4 567 >

For ***long integers***, up to 21 digits and a sign may be displayed using the usual large digits:

-123 456 789 012 345 678 901 .

taking $(1 + 15 + 21 \times 16 + 6 \times 8 = 400$ px). Larger *long integers* employ the small font, allowing for 42 digits and a sign:

-123 456 789 012 345 678 901 234 567 890 123 456 789 012 .

Even larger *long integers* may be displayed with an exponent replacing as many of their least significant digits as necessary:

-123 456 789 012 345 678 901 234 567 890 123 456 $\times 10^{21}$.

For unsigned ***short integers***, up to 21 bits may be shown in the usual large digits in binary representation:

0 1100 0010 1101 0110 0000,

Base 3 (with narrow blanks every three digits) allows for displaying 20 digits and a sign:

-22 211 200 201 120 001 212₃

In base 4 (with narrow blanks every two digits), 19 digits representing 38 *bits* are displayable:

3 21 23 30 22 11 21 20 32 12₄

Also bases 5, 6, and 7 allow for showing 20 digits and a sign like base 3, base 8 for 19 digits like base 4 (but representing 57 bits in base 8).

Using the narrower digits provided, up to 25 bits are displayable in binary representation:

0 1110 1100 0010 1101 0110 0000₂ .

Then 24 digits and a sign can be shown for bases 3, 5, 6, and 7, as well as 22 digits for bases 4 and 8.

Longer integers in bases 2 through 6 must be displayed using the small font. This allows for showing up to 44 bits in binary notation:

1110 1100 0101 1101 0110 1110 1100 0010 1101 0110 0000,

41 digits and a sign can be displayed for base 3 being already sufficient for 64 bits, as well as the 39 digits theoretically displayable for base 4.

For showing the maximum of 64 bits in base 2, two special 5 px wide characters were created:

Summing up, for given base and word size, the following fonts will do for *short integers*:

Base ▼	Allowable size of digits for display			
	large	narrow	small	special
2	21 bits	25 bits	44 bits	64 bits
3	31 bits	38 bits		
4	38 bits	44 bits		
5	46 bits	55 bits		
6	51 bits	62 bits		
7	56 bits			
8	57 bits			
> 8	64 bits	64 bits		

One row of four arbitrary **real matrix elements** (with absolute values $< 10^{100}$) takes 399 px in small font, SCI 3:

$$[-6,609 \cdot 10^{-19} \quad -6,609 \cdot 10^{-19} \quad -1,609 \cdot 10^{-19} \quad -1,609 \cdot 10^{-19}]$$

using multiplication dots. Else you will lose one decimal. A slightly different notation allows for SCI 4:

$$[-6.609 \, 2 \cdot 10^{-19} \quad -6.609 \, 2 \cdot 10^{-19} \quad -1.609 \, 2 \cdot 10^{-19} \quad -1.609 \, 2 \cdot 10^{-19}]$$

Matrices with more than four columns will need ellipses added on one or both sides:

$$[\dots \quad -6,609 \, 2 \cdot 10^{-19} \quad -6,609 \, 2 \cdot 10^{-19} \quad -1,609 \, 2 \cdot 10^{-19} \quad \dots] .$$

allowing to display a section of three elements in SCI 4 format. Using multiplication crosses will cost one decimal.

Vertically, each such matrix row requires 20 px as other small font strings do. Thus, 5 matrix rows ($5 \times 20 + 4 = 104$ px) can be put in the space taken by 3 standard numeric rows ($3 \times 32 + 2 \times 5 = 106$ px). So, a 5×4 real matrix can be displayed entirely always, using SCI 3 in worst case.

In consequence, any chosen 3×3 section out of a real matrix of arbitrary size can be shown in SCI 3 minimum with surrounding ellipses. In FIX format, 8 decimals can be displayed always.

In analogy, for a **complex matrix** of arbitrary size any chosen 3×2 section can be displayed in FIX 6 format maximum with surrounding ellipses like

$$[\dots \quad -6.609 \, 226+i \cdot 6.609 \, 226 \quad -1.609 \, 226+i \cdot 1.609 \, 226 \quad \dots] .$$

while displaying complex matrix elements featuring large exponents may become inconvenient very soon, regardless of the symbols used:

$$[\dots \quad -6.60E^{-199}+i \cdot 6.60E^{-199} \quad -1.60E^{-199}+i \cdot 1.60E^{-199} \quad \dots] .$$

Also displaying an arbitrary 3×3 section out of a larger complex matrix is viable up to FIX 3 as long as the numbers stay in a reasonable range:

$$[\dots \quad -6,609+i \cdot 6,609 \quad -6,609+i \cdot 1,609 \quad -1,609+i \cdot 1,609 \quad \dots] .$$

One row of **alphanumeric text** will typically take some 40 characters. The actual number will vary depending on their individual widths as mentioned above.

The *status bar* is a good example for such an alphanumeric row: Loaded to maximum, it might look like

2017-05-08 23:49 RL₄π /3 546f 64:u⁰ A X↑O↓G↑U↑

containing 45 characters.

Putting the alphabet in a row allows for

abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOP

i.e. 41 characters.

Echoing command input requires up to 16 characters (the 17th will close input) for a 7-character command indirectly addressing a 7-character variable entered in AIM. This can be done in either font.

Command and variable *names* in menus are discussed in the last paragraph of next chapter. Although seven characters are allowed for such *names*, six may well fill the screen space available there already. Thus, it is recommended to keep such *names* as short as possible, though meaningful.

Display Segmentation

The *LCD* of the WP 43S is full dot matrix: 400 px wide and 240 high. Each pixel is 0.147 mm square. Going top down, you will find

- 20 px for the *status bar*,
- 4 blank pixel rows for separation,
- 147 px for either
 - a) the contents of up to 4 *stack registers*, or
 - b) 7 program steps in *PEM*, or
 - c) up to 7 rows of numeric output of SHOW, and
- 69 px maximum for the menu section.

2017-05-08 23:49		
-12 345 67		
-9.234 56		
-5.678 901		
010 1100 0		
ABCDEF	B	
B		
C		

The reasons for these figures are:

- Each regular alphanumeric row (e.g. for labels, status, a text string, or a program step) requires 20 px vertically (cf. pp. B-5f). There shall be at least one row of blank pixels separating it from the next row of data.

- Hence, each *menu* row takes (counting bottom-up) $1 + 20 + 1 + 1 = 23$ px (the first pixel is for separation from the black frame, the last for its upper frame line). Thus, three *menu* rows require 69 px. In U, extra-high labels may appear: they will require $1 + 20 + 1 + 20 + 1 + 1 = 44$ px for double height or $1 + 20 + 1 + 20 + 1 + 20 + 1 + 1 = 65$ px for triple height then.
- At top of the *LCD*, the *status bar* takes another 20 px plus 4 for separation. Thus, $240 - 69 - 24 = 147$ px remain free on the screen so far.
- Each regular numeric output requires 32 px vertically (cf. p. B-5) plus 4 px separating it from the next such output row. Thus, for 4 rows we need $4 \times 32 + 3 \times 4 = 140$ px. Since there are 4 px above them already, we put the remaining 7 px below this block.
- If there is a short *text string* in an arbitrary *stack register*, its base line should be positioned where the base line of the respective numeric output would have been.
- With a *text string* in **X** needing two rows, $1 + 20 + 1 + 20 + 1 = 43$ px are required vertically matching the $7 + 32 + 4 = 43$ px available for the lowest numeric row. Longer *strings* will need SHOW to be shown entirely.
- In *PEM*, on the other hand, $147 = 7 \times (20 + 1)$ px correspond to a block of 7 alphanumeric program rows.
- With SHOW, also pure numeric output may require more than one row – the small font will be used there as well. Thus, up to 7 rows of small digits are possible again. Cf. pp. 70 and B-11.

In the *menu section*, we also have a horizontal structure for the six *softkeys*. We start one pixel off the black frame at left display edge. On the right edge, the characters themselves contain at least one blank column. A minimum of 2 px separate *softkey* labels from each other (one black and one blank). This way we lose a total of $1 + 5 \times 2$ px. The remaining 389 px mean a width of 65 available for 6 *softkey* labels, corresponding to six standard width letters (though letters may extend from 4 to 14 px in small font) which should be centered as good as possible. Note that labels in *menu views* may be not fully displayed if they are wider than 64 px, so labels deviating only in their very last characters may become visually indistinguishable. Users should avoid such ambiguities.

Echo and Fallback

Almost all key presses are echoed and fall back to NOP. Softkeys shall not be echoed since WYS/WYG applies there always. Furthermore, some functionalities, wherever they may be assigned to, are not echoed (and hence cannot fall back) since

- 1) they are harmless (EXIT, UP, DOWN) or
- 2) reverting or exiting them is a no-brainer (UP, DOWN, P/R, all *menu* calls) requiring no more than one keystroke.

With the presence of UNDO (see p. B-29), the necessity of fallback to NOP becomes debatable overall; there is some benefit remaining in *user mode* as long as an overlay matching the actual assignments is not available. And I admit UNDO is shifted in *startup default configuration*.

Equations

Equations are entered in **EQN** as written (i.e. following algebraic notation and rules). While editing them, punctuation spaces are automatically inserted after each constant or variable (you know a variable *name* ends when the next operator is entered) as well as after = and behind each operator like +, -, ·, ×, /, and ! (except $\hat{}$); a standard space is inserted after :. There is no implicit multiplication.

Other functions like absolute values, roots, or trigs shall be written using the parentheses softkey, e.g. pressing **()**, then stepping back into the parentheses for specifying the argument. The same applies to dyadic (like **C_{xy}**) and triadic operations in analogy – their arguments shall be separated by blank spaces inserted via **R/S**.

Terminating the *Equation Editor*, numeric exponents are automatically converted from e.g. **xy ^23** to **xy²³**. For easier handling, this will be reverted when editing such an equation again.

Layouting

After drawing many fictional calculators just for fun, we gained our real-world layout experience with the *WP 34S*. Based on this and seeing a forthcoming better display than the very limited one of the *HP-20b/30b* at the horizon, I conducted a poll on the forum of the *Museum of HP Calculators* about the community's general preferences for the placement of the four basic arithmetic operators on a hypothetical portrait RPN pocket calculator (<https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234505>). In return, I published the basic concept and first layout of the *WP 43S* (<https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685>) in November 2012 (the picture shows my last draft before said post, wishfully assuming the good old slanted keys of the Seventies). We wanted to create an optimized pocket calculator based on our experience instead of resurrecting or copying an existing layout.

Even after years, the order and position of the four basic arithmetic operators may become subject to repeated vivid discussions still. All the early *HP pocket* calculators (from the world's first scientific pocket calculator, the *HP-35* of 1972, to the *HP-41C*, *CV*, and *CX*) presented them beneath **ENTER↑** in the order **-**, **+**, **x**, **÷** (top down);

though actually no one knows the reason for this old sequence anymore. With the launch of the *Voyagers* in 1981, these operators had to be moved and *HP*



abandoned its proprietary sequence for pocket calculators turning to the common order $+$, $-$, \times , \div (bottom up). The poll mentioned above resulted in a majority for placing them beneath **ENTER↑** for ergonomic reasons.

Everything after the first layout of November 2012 was and is just patient refinement and careful tuning of the basic idea. It even survived an hardware switch – we were waiting for *Eric Smith's* and late *Richard Ottosen's* so-called *Reptiles* (see the *HHC* meetings until 2015) still when *Michael* mailed me the concept of his and *David's* *DM42* in March 2016.

Actually, development of the *DM42* overtook the *WP 43S* – it was launched late in spring 2017 as a resurrection of the *HP-42S* while *Pauli* and me were looking for willing software engineers and actual support beyond friendly words still in vain. Despite its popularity in the community, qualified manpower for coding *WP 43S* was the lasting bottleneck in our project since 2012.

Menus

The community does not like deep *menus*: it prefers the *HP-32SII* to the *HP-32S*. On the other hand, it wants a large function set. So *menus* become inevitable but shall be designed carefully.

Menu size corresponds to keystroke efficiency; optimum for our user interface is a *menu* encompassing three *views* containing up to 54 functions in total: the top *view*, one *view* going up via **▲**, and one going down via **▼**. Larger *menus* lack efficiency, smaller *menus* lack functionality. Besides its (in)visibility, a function presented in the unshifted row of the top *menu view* is more efficient than a shifted function presented on the keyboard – if used more than just once (cf. pp. B-1ff).

Generally, I separated status setting from ‘acting’ operations in different *menu views* or rows at least.

Number Range

A number range up to 10^{99} is sufficient for almost all real-world problems – else common scientific calculators would feature a larger numeric range generally (cf. also pp. 138f). So we can conclude that the *real number* range supported

(cf. p. 164) suffices by far for solving what has to be solved. Saving display space gave reason for RANGE.

For sake of consistency, maximum numbers within different *DTs* should match. I.e. the maximum absolute value allowed for a *long integer* should be approximately equal to the respective values for a *real* and a *complex number*.

Note the number range determines the precision required for calculating accurately (see p. B-21).

Plotting?

It is mentioned elsewhere we are not out for creating a graphing calculator. There is, however, a very useful application where a basic scatter plot of measured data points would support decision making significantly (see the third industrial statistics application example in *Section 2* of the OM). This would require the statistical data (e.g. max. 100 experimental data points, i.e. 100 pairs of *x* and *y* values) to be stored point by point in a matrix, not just summed up as in earlier *RPN* calculators.

Plotting could be called by a command PLOT stored in STAT displaying the data points collected in a quadratic diagram. Both axes shall reach from minimum value measured to maximum value measured (plus a little extension which can be calculated based on the data points). Axis scales are not required for analysis so I omitted them. Drawing area has to be quadratic (240×240 px for data, 242×240 px incl. the vertical axis).

Hence softkeys can be positioned on one side of the diagram (max. 3×2 labels) still. The status bar would be partially overwritten by the diagram. See the sketch (to scale) for general screen layout and various data point symbols for checking visibility.



CLLCD was modified for clearing just the screen section required for the diagram. Although axis and diagonal can be created using AGRAPH and PIXEL alone, four characters are provided in the small font for 'drawing' the vertical axis and the 45° line:



Data points can then be plotted using POINT (containing 3x3 px); positioning them properly in the diagram, however, will require some background calculations best performed by a program. For POINT, some of the graphic control modes of *HP-42S* shall be implemented in analogy (the table below is copied from the *HP-42S OM*, p. 137):

Flag 34	Flag 35	How the AGRAPH Image is Displayed
Clear*	Clear*	The image is merged with the existing display (logical OR).
Clear	Set	The image overwrites all pixels in that portion of the display.
Set	Clear	Duplicate "on" pixels get turned "off."
Set	Set	All pixels are reversed (logical XOR).

* Default setting.

Settings 1 (0, 0) and 3 (1, 0) herein should suffice for our plotting (cf. **GRAMOD** on p. 100).

Softkey functions could be ...

- CENTRL for fitting the center line to the data points and plotting it for checking deviations from the 45° line (some background calculations in L.R. using ORTHOF for orthogonal regression are required for the plot, setting 1 in the table above will do while plotting);
- S_{mi} for calculating the minimum experimental standard deviation of the measuring instrument (some background calculations required again); and
- CLOSE for closing the plot screen, returning to normal display.

(DEL1PT turned obsolete.) It may be beneficial to define a general origin for graphics at a location deviating from pixel 0, 0 (i.e. the bottom left corner of the LCD) – the point 158, 0 may be a useful origin. This would allow for creating also other graphics than just the correlation diagrams mentioned above, while

reserving a ‘protected screen space’ left for up to six softkeys. Any user may do his own in this almost quadratic drawing area then, using the commands AGRAPH, CENTRL, CLLCD, CLOSE, PIXEL, PLOT, POINT, and S_{mi}.

Precision and Accuracy

As mentioned above more than once, there are inevitable errors in each numeric calculation step, frequently caused by rounding to the internal finite precision the calculator features. Already a simple fraction like $1/3$ stored as a *real number* deviates from the truth by more than 3×10^{-35} . During calculations, such errors accumulate (cf. footnote 75).

In real-world problems, usually the least accurate of all input (real) parameters determines the accuracy of the result. In the standard test mentioned in said footnote starting with 9° , you can nevertheless get 28-digits precision in the result since the input of 9° is exact (but note one digit precision is lost with each trigonometric function calculated here).

Internally, for instance, the *WP 34S* computes with 39 digits and rounds the results to 34 or 16 digits, respectively (cf. footnote 75). Consequently, *WP 43S* also works with 39 digits internally most times and rounds results to 34 digits. SLVQ calculates using 72 digits. The statistical summation registers are 75 digits wide (used also for the initial steps of variance calculation). Range reductions for trigonometric functions use many more digits (cf. pp. B-3f).

Luckily, real-world problems are usually far less precisely defined than the internal precision of *WP 43S*. Compare also the set of physical and astronomical constants provided (cf. pp. 129ff).

Prefixes

Prefixes  and  passed without any discussion for more than six years until 2019-06. Alternatively, prefixes  and  could have been chosen but their typography leaves less freedom for placing the corresponding golden and blue labels.

Sorting in Detail

There is no international standard for sorting characters; we had to invent our own order. Sorting of *items*, variable *names*, *text strings*, *system flags*, etc. on WP 43S works as listed below, top down and left to right.

Note that sorting is a two-step procedure: step 1 sorts the text strings under consideration just according to column 1 of this table, comparing them; if two strings are rated equal in this aspect, step 2 takes the columns following into account. Applying this algorithm, a section of CATALOG'FCNS looks like e.g.

Sorting is illustrated for the small font here. It holds also for the large font as far as characters are applicable. The 4-digit number trailing each character in the table is its hexadecimal *Unicode*. Characters printed on grey background are inaccessible for users; those printed on darker grey are not used at all so far.

?	0020	2003	2004	...	2008	200a	2423
0	0030	220e	00b0	0 2070	0 2080		
1	0031	1 2027	½ 00bc	¼ 00bd	1 2071	1 2081	1 2460
2	0032	2 00b2	2 2082	2 2461			
3	0033	3 00b3	3 2083	3 2462	3/ 221b		
4	0034	4 2074	4 2084	4 2463			
5	0035	5 2075	5 2085	5 2464			
6	0036	6 2076	6 2086	6 2465			
7	0037	7 2077	7 2087	7 2466			
8	0038	8 2078	8 2088	8 2467			
9	0039	9 2079	9 2089	9 2468			
10	2491	10 2469					
11	246a						
12	246b						
13	246c						
14	246d						
15	246e						

16	246f					
A	0041	a 0061	ä 00aa	A 24b6	ä 2090	ä 249c
		À 00c0	à 00e0	À 00c1	à 00e1	À 00c2
		Ã 00c3	ã 00e3	Ã 00c4	ã 00e4	Ã 00c5
		Æ 00c6	æ 00e6	Ā 0100	ā 0101	Ā 0102
						ä 0103
					À 0104	ä 0105
B	0042	b 0062	B 24b7	b 249d		
C	0043	c 0063	c 24b8	c 249e	ç 00c7	ç 00e7
		Ć 0106	ć 0107	Ć 010c	ć 010d	Ć 2102
D	0044	d 0064	D 24b9	d 249f	đ 00d0	đ 00f0
				Đ 010e	đ 010f	Đ 0110
E	0045	e 0065	E 24ba	e 2091	e 24a0	È 00c8
		É 00c9	é 00e9	Ê 00ca	ë 00ea	Ë 00cb
		Ē 0112	ē 0113	Ĕ 0114	ě 0115	Ĕ 0116
		Ę 0118	ę 0119	Ě 011a	ě 011b	Ě 2073
F	0046	f 0066	f 24a1	F 24bb		
G	0047	g 0067	g 24a2	G 24bc	გ 011e	გ 011f
H	0048	h 0068	h 210e	h 24a3	ჰ 24bd	ჰ 2095
						ჰ 0127
						ჰ 210f
I	0049	i 0069	I 24be	i 24a4	ି 00cc	ି 00ec
		ି 00cd	ି 00ed	ି 00ce	ି 00ee	ି 00cf
		ି 012a	ି 012b	ି 012c	ି 012d	ି 012e
						ି 0130
						ି 0131
J	004a	j 006a	J 24bf	j 24a5		
K	004b	k 006b	K 24c0	k 24a6	k 2096	
L	004c	l 006c	L 24c1	l 24a7	ლ 2097	լ 0139
				ლ 013d	լ 013e	լ 0141
M	004d	m 006d	M 24c2	m 24a8	m 2098	

N 004e	n 006e	n 24c3	n 24a9	n 2099	Ñ 00d1	ñ 00f1
	Ń 0143	ń 0144	Ń 0147	ń 0148	Ń 2115	
O 004f	o 006f	o 00ba	ø 00a9	o 24c4	o 24aa	o 2092
	ò 00d2	ò 00f2	ó 00d3	ó 00f3	ò 00d4	ó 00f4
	ő 00d5	ő 00f5	ö 00d6	ö 00f6	ő 00d8	ø 00f8
	ō 014c	ō 014d	ő 014e	ő 014f	œ 0152	œ 0153
P 0050	p 0070	p 24c5	p 24ab	p 209a		
Q 0051	q 0071	q 24c6	q 24ac	Q 211a		
R 0052	r 0072	r 24ad	r 24c7	Ŕ 0154	ŕ 0155	
				Ŕ 0158	ŕ 0159	Ŕ 211d
S 0053	s 0073	s 24c8	s 24ae	s 209b	Ś 015a	ś 015b
	ſ 015e	ſ 015f	ſ 0160	ſ 0161	þ 00df	
T 0054	t 0074	t 24af	t 22a4	t 24c9	t 209c	
			t 0162	ẗ 0163	ẗ 0164	ẗ 0165
U 0055	u 0075	u 24ca	u 24b0	u 1d64	Ü 00d9	ü 00f9
	ú 00da	ú 00fa	ú 00db	ú 00fb	Ü 00dc	ü 00fc
	ű 0168	ű 0169	ű 016a	ű 016b	ű 016c	ű 016d
			ű 016e	ű 016f	ų 0172	ų 0173
V 0056	v 0076	v 24cb	v 24b1			
W 0057	w 0077	w 24cc	w 24b2	ŵ 0174	ŵ 0175	
X 0058	x 0078	x 1d61	x 24cd	x 24b3	x 2093	
			᷇ 0379	᷈ 0378	᷉ 037f	᷊ 221c
Y 0059	y 0079	y 24ce	y 24b4	ÿ 00dd	ý 00fd	
	ÿ 0176	ÿ 0177	ÿ 0178	ÿ 00ff	ÿ 0233	ÿ 0232
Z 005a	z 007a	z 24cf	z 24b5	ž 0179	ž 017a	ž 017b
			ž 017c	ž 017d	ž 017e	ž 2124
A 0391	α 03b1	α 2065	á 03ac			
B 0392	β 03b2					

Γ	0393	γ	03b3				
Δ	0394	δ	03b4	δ	2066		
Ε	0395	ε	03b5	έ	03ad		
Ζ	0396	ζ	03b6				
Η	0397	η	03b7	ή	03ae		
Θ	0398	θ	03b8				
Ι	0399	ι	03b9	ί	03af	ϊ	03aa
				ϊ	03ca	ΐ	0390
Κ	039a	κ	03ba				
Λ	039b	λ	03bb				
Μ	039c	μ	03bc	μ	00b5	μ	2067
Ν	039d	ν	03bd				
Ξ	039e	ξ	03be				
Ο	039f	ο	03bf	ό	03cc		
Π	03a0	π	220f	π	03c0		
Ρ	03a1	ρ	03c1				
Σ	03a3	σ	03c3	ς	03c2		
Τ	03a4	τ	03c4				
Υ	03a5	υ	03c5	ύ	03cd	ΰ	03ab
				ΰ	03cb	ΰ	03b0
Φ	03a6	φ	03c6				
Χ	03a7	χ	03c7				
Ψ	03a8	ψ	03c8				
Ω	03a9	ω	03c9	ώ	03ce		
(0028)	0029				
[005b	τ	23a1		23a2	λ	23a3
]	005d		23a4		23a5
{	007b	}	007d]	23a6
„	2430	„	2431	„	2432	„	2433
+	002b	+	207a	+	208a	±	00b1

-	002d	-	207b	-1	2072	-	208b	≠	2213	
x	00d7	.	00b7	•	2219	◦	2218	*	002a	* 208f
/	002f	\	005c							
^	005e									
,	002c	j	2429							
.	002e	i	2428	...	2026					
!	0021	i	00a1							
?	003f	ż	00bf							
:	003a	:	2236	÷	00f7					
;	003b									
'	0027	'	2018	'	2019	,	201a	'	201b	104
"	0022	"	201c	"	201d	,,	201e	"	201f	« 00ab
@	0040									» 00bb
-	005f	*	2427							
~	007e									
→	2192	→	21c0							
←	2190	⌚	21cd							
↑	2191	↑	21e7	▲	21c9	≠	242b			
↓	2193	▼	21e9	▼	21cb					
⤠	21c4									
⤡	2195									
☰	21cc									
¬	00ac									
⤢	2227	⤢	2228	⤢	22bb	⤢	22bc	⤢	22bd	
&	0026									
	007c		2223		2224		2225		2226	

¹⁰⁴ Look up https://de.wikipedia.org/wiki/Anführungszeichen#Andere_Sprachen or https://en.wikipedia.org/wiki/Quotation_mark#Summary_table for properly using these characters in different languages.

« 226a	< 003c	≤ 2264	≡ 2261	:= 2254	= 003d	≈ 2243
	≈ 2248	≡ 2258	△ 2259	≠ 2260	≥ 2265	> 003e
						» 226b
% 0025	\$ 0024	€ 20ac	¢ 00a2	£ 00a3	¥ 00a5	₪ 00a7
✓ 221a	✗ 221d					
∞ 221e	∞ 209e	∞ 209f				
∫ 222b	ʃ 222c	ʃ 222d	ʃ 222e	ʃ 222f	ʃ 2230	
◎ 2299	◎ 229a	◎ 2068				
⊕ 2295	⊕ 2069					
↳ 221f	↳ 22a5					
↶ 2220	↶ 2221	↶ 2222				
↑ 2308	↑ 2309					
↓ 230a	↓ 230b					
☒ 2399	☒ 231b	☒ 231a	☒ 242a	☒ 242c	☒ 242f	☒ 2434
# 0023						
UK 242d	US 242e					
▼ 2200	▼ 2202	▼ 2203	▼ 2204	▼ 2205	▼ 2206	▼ 2207
	▼ 2208	▼ 2209	▼ 220b	▼ 220c	▼ 2229	▼ 222a
↓ 2421	↓ 2422	↓ 2425	↓ 2426			
✓ 2713						

Stack Size

At a very early stage of this project (2013), stack size was discussed. An RPL-like ‘infinite’ stack would allow for saving (pushing) everything thereon before calling a (sub-) routine and popping it after RTN but makes traditional R↓, R↑, and top level repetition obsolete (and FILL as well). In this context I suggested two new commands called CLOSES and OPENS for closing the bottom section (4 or 8 registers) of an infinite stack for the time when R↓, R↑,

FILL, and top level repetition were required, and opening it thereafter. At the bottom line, eight *stack registers* turn out being sufficient for solving any real-world mathematical, scientific, or engineering problem (cf. Section 1 of the OM as well as field experience with *WP 34S* and *WP 31S* since 2011).

After all, we decided sticking to *RPN* as implemented on the *WP 34S* and *WP 31S*. It covers everything needed most easily. For support of special actions, the commands STOS and RCLS are provided.

Stack Lift Disabling Functions

Also these functions were subject of discussion. For sake of backward compatibility, we decided keeping them as they were on the vintage *HP RPN* pocket calculators up to the *HP-42S* (and *WP 34S* and *WP 31S*):

Only ENTER↑, CLX, Σ+, and Σ- disable ASL, all other functions enable it. But compare INPUT on p. 40 as well as M.EDI and M.EDIN on p. 52.

Structured Programming

In 2013, I suggested the following control structures:

- IF ... THEN ... ELSE ... END,
- FOR ... FROM ... TO ... END,
- REPEAT ... UNTIL, and
- WHILE ... END.

Traditional END would need to be called ENDPGM then.

Later, we discussed some *PASCAL*-like structures:

- IF ... THEN BEGIN ... END ELSE BEGIN ... END;
- FOR ... DO BEGIN ... END; and
- WHILE ... DO BEGIN ... END;

We refrained from implementing such commands since we had doubts about the sensibility of mixing keystroke programming and structured programming features.

UNDO

In 2013, UNDO was planned as it works in *HP-48*, recalling just the *stack* as it was before executing last command. The *WP 31S*, on the other hand, features an UNDO recalling the entire calculator state as it was before executing last command. Until 2020, we assumed that such a complete UNDO was viable in *WP 43S* as well, but the overhead turned out forbidding.

Since

- any user *flags* altered inadvertently can be reverted easily and
- any wide-reaching clear commands (CLREGS, CLFALL, CLPALL, etc.) ask for confirmation before executing,

we implemented UNDO recalling not only the *stack* but also the summation *registers* and *system flags* as they were before executing last command, for better user experience (this specification also allows for undoing $\Sigma-$).

For Your Convenience

Please find here the display frame of your *WP 43S* as we designed it (printed to scale), and below its virtual keyboard in alpha input mode (*AIM*) plus a branching helper.

Overleaf, its original keys and keyplate are printed to scale. You will also find there an explanatory picture taken from the back of an *HP-16C* (with C denoting CARRY and G standing for OVERFL).

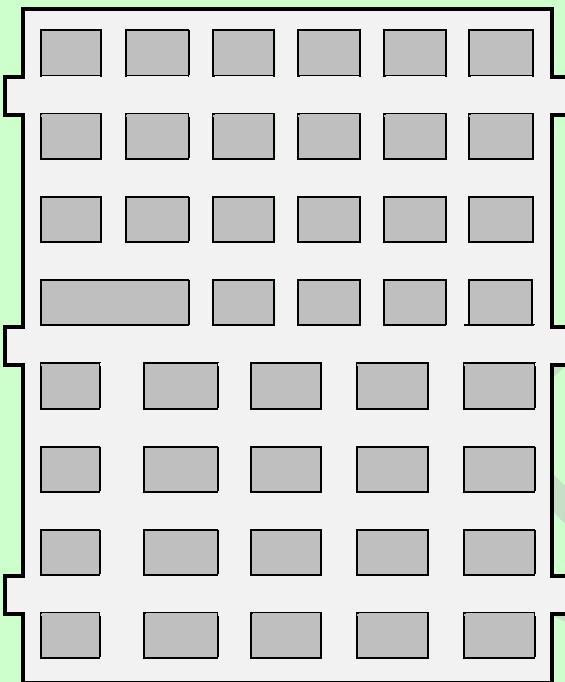
Choose your favorites, cut them out, and use them with your *WP 43S*. If you bought it complete and flashed, keys and keyplate shall be printed properly on its top face and the picture shown here at right is on its rear for your reference.





	C	G	
$+$	x	x	
$-$	--	x	
\times	x	x	$\text{RMD} \neq 0 \rightarrow C$
\div	x	--	$\text{RMD} \neq 0 \rightarrow C$
\sqrt{x}	x	--	$\text{RMD} \neq 0 \rightarrow C$
CHS ABS	--	x	
$\text{DBL } X$	--	o	$y = x + (X \& Y)$
$\text{DBL } Z$	x	o	$(Y \& Z) \div x \rightarrow X ; \text{RMD} \neq 0 \rightarrow C$
SL	x	--	$\square \leftarrow \square \rightarrow \square \rightarrow o$
SR	x	--	$o \rightarrow \square \rightarrow \square \rightarrow \square$
ASR	x	--	$\square \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow C$
RL	x	--	$\square \leftarrow \square \rightarrow \square \rightarrow \square$
RR	x	--	$\square \rightarrow \square \leftarrow \square \rightarrow \square \rightarrow C$
RLC	x	--	$\square \leftarrow \square \rightarrow \square \rightarrow \square \rightarrow C$
RRC	x	--	$\square \rightarrow \square \leftarrow \square \rightarrow \square \rightarrow C$

Template for a quick overlay (almost to scale – see point 3 below):



1. Print out on standard copy paper (80 g/m^2). Fill in labels where, if, and as required.
2. Laminate with 80 mics stock (result is a thickness of about 0.25 mm).
3. Cut out voids for keys and perimeter. Attention: Tabs (a.k.a. tongues) as drawn are too wide and too long, cut smaller.