WP 43S 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*

  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.

The pictures on p. 150 and bottom of p. 151 were kindly supplied by *SwissMicros* as well as the drawing on p. 208. 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 on 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

TABLE OF CONTENTS

Background Considerations and Facts B-1

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 $7{,}089\cdot10^{-12}$) 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:

2D = two-dimensional.

3D = three-dimensional.

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. 154ff).

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 (*IOI*)

All the *items* provided on your *WP 43S* (more than 850) are listed in this section with their *names* (as they are displayed and printed in routines) in column 1 and the way to call them in column 2. Most *items* shall be picked from *menus* (see pp. 108ff). For such *items*, we record the respective *menu* and its *view*, and the label as shown therein (printed on gold or blue if **f** or **g** are required accessing it); we are confident you will find the corresponding *softkey*. *Items* stored in CONST are listed with their *names* only since they are sorted alphabetically; they 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]

□ 0…9 10 11…16 Aa…Zz Αα…Ωω ( ) [ ] { }
\+ - × / \ ^ ± , ; . ⁝ ! ¡ ? ¿ : ÷ ; ' " « » · • * @ _ ~
→ ← ↑ ⤒ ↓ ≷ ⇕ ¬ ∧ ∨ & | ‖
« < ≤ ≡ := = ≃ ≈ ≘ ≙ ≠ ≥ > » % $ € ¢ £ ¥
√ ∞ ∫ ⊙ ⊕ ⊾ ∠ ∡ ⊥ (clock) ⌛ (printer) (battery) # U UK US !

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

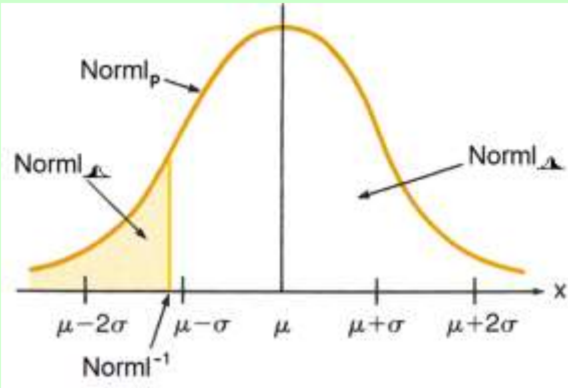
In principle, *WP 43S* operations work as the corresponding ones did on the *WP 34S* where applicable (see *App. E*). Referring to vintage *HP* calculators, most functions and keystroke programming will work as they did on the *HP-42S*, bit and integer functions as on the *HP-16C*, unless specified otherwise.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

[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 own 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.

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 or black – 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 representing:

(0) functions without any effects on the *stack* (e.g. mode setting);

(1) *monadic functions*,

(2) *dyadic functions*, and

(3) *triadic functions* as defined in *Section 1* of the *OM*;

(−1) functions pushing one object on the *stack* and

(−2) functions pushing two objects on the *stack*.

---

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

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**.

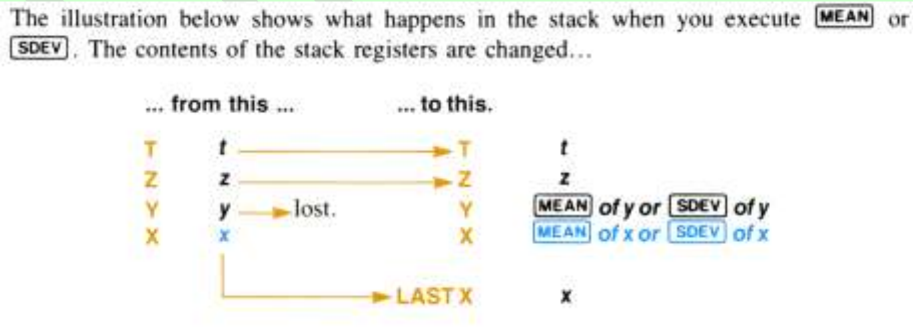
1. ***n*** represents an arbitrary integer number which must be keyed in directly, while

   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;

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

[4] The *HP-42S* does that also with statistical functions returning two values – 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. The picture below shows what the *HP-55*, *HP-19C/29C*, *HP-67/97*, *HP-41C*, and *HP-42S* do there:



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

2. ***r*** (or ***s***) represents an arbitrary *register* address or variable *name* which must be keyed in directly or picked from a *menu*, while

   r (or ***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. ***labl*** represents an arbitrary program label which must be keyed in directly or picked from a *menu*, while

   labl represents such a label which may be specified indirectly (as shown in the addressing table in *Section 3* of the *OM*); and

   labl 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. 154ff. 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 °C→°F, for instance: For *real number* input, output will stay *real*. For integer input, however, output will become *real*.

Some functions operating on ***long*** *integers* will return either such integers or *reals*, depending on the input value. E.g. ∛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*.

ASL is enabled after each command except the following:

1. After CLX, ENTER↑, Σ+, and Σ−, *ASL* is disabled (cf. *Section 1* of the *OM*); thus, (alpha-) 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)
°C→°F °F→°C	U→ °C→°F etc.	(1) {2}; {1} → {2} Convert temperatures. See pp. 123ff.
10^x	10^x	(1) {1, 2, 3, 8*, 9*, 10} Returns 10^x, the inverse of $\lg(x)$.
1COMPL	BITS ▼ 1COMPL INTS ▲ 1COMPL	(0) Sets 1's complement mode for operations on *short integers*. See *Section 2* of the *OM*.
$^1/_x$	$1/x$	(1) {2, 3, 5, 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. 85).

[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↑ is a *stack* lift manually initiated by the user. An additional *ASL* immediately after this command makes no sense.

c) Σ+ and Σ- 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)
2COMPL	BITS ▼ 2COMPL INTS ▲ 2COMPL	(0) Sets 2's complement mode for operations on *short integers*. See *Section 2* of the *OM*.
2^x	EXP 2^x	(1) {1, 2, 3, 8*, 9*, 10} Returns 2^x, the inverse of $\mathrm{lb}(x)$.
$\sqrt[3]{x}$	EXP $\sqrt[3]{x}$	(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	CATALOG FCNS ABS	Points to |x| on p. 89. Maintained for backward compatibility only.
ACOS	CATALOG FCNS ACOS	Points to arccos on p. 18. Maintained for backward compatibility only.
ac→ha	U→ f A: acre → ha	(1) {2}; {1} → {2} Convert areas. See pp. 123ff.
ac_{US}→ha	U→ f A: $acre_{US}$ → ha	(1) {2}; {1} → {2} Convert areas. See pp. 123ff.
ADV	ADV	*Menu*. See p. 114.
AGM	X.FN AGM	(2) {2, 3}; {1} → {2} Returns the *arithmetic-geometric mean* of x and y. Will throw an error for x or y being negative. See p. 247 for more.
AGRAPH	P.FN P.FN2 AGRAPH s	(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 the less characters are required for describing it. – Cf. *HP-42S OM*, pp. 135 – 140, and *HP-42S Programming Examples & Techniques*, pp. 195 – 223.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ALL	DISP ALL *n*	(0) Sets the numeric display format to show all decimals of *real* or *complex numbers* whenever displayable (without trailing decimal zeros). ALL **0** works almost like ALL in *HP-42S*. For $x \geq 10^{16}$ (or earlier for *complex numbers*), 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. 58.
AND	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 non-zero *real number* or *long integer* is 'true'.
ANGLES	CATALOG VARS ANGLES	*Submenu* of tagged angular variables defined at execution time. See pp. 109f.
arccos	TRI arccos	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $arccos(x)$.[7]
arcosh	EXP arcosh TRI arcosh	(1) {2, 3, 8*, 9*} Returns $arcosh(x)$.
arcsin	TRI arcsin	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $arcsin(x)$.[8]

[7] Precisely, ARCCOS returns the principal value of $arccos(x)$, i.e. a *real* part ∈ [0, π] in ∡ʳ, or ∈ [0, 1] in ∡π, or ∈ [0°, 180°] in ∡° or ∡", or ∈ [0ᵍ, 200ᵍ] in ∡ᵍ. Cf. *ISO/IEC 9899*.

[8] Precisely, ARCSIN returns the principal value of $arcsin(x)$, i.e. a *real* part ∈ [−π/2, π/2] in ∡ʳ, or ∈ [−0.5, 0.5] in ∡π, or ∈ [−90°, 90°] in ∡° or ∡", or ∈ [−100ᵍ, 100ᵍ] in ∡ᵍ. Cf. *ISO/IEC 9899*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
arctan	TRI arctan	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $arctan(x)$.[9]
arsinh	EXP arsinh TRI arsinh	(1) {2, 3, 8*, 9*} Returns $arsinh(x)$.
artanh	EXP artanh TRI artanh	(1) {2, 3, 8*, 9*} Returns $artanh(x)$.
ASIN	CATALOG FCNS ASIN	Points to ARCSIN above. Maintained for backward compatibility only.
ASR	BITS 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*.
ASSESS	STAT ▲ g ASSESS	(0) Plots the data points in the statistics registers together with the linear regression curve fitted. If BESTF was used to select >1 fit model then all allowed functions will be displayed sequentially. Use ZOOM for zooming out, NXTFIT or ▲ to proceed to the next model allowed, ▼ to return, EXIT to leave. See pp. 222ff for more.
ASSIGN	ASN ***item, location***	(0) Assigns an *item* (like a function, *menu*, label, or character) to a specific location on the keyboard or in a *menu*. See the *OM*, *Sect. 6*.
ATAN	CATALOG FCNS ATAN	Points to ARCTAN above. Maintained for backward compatibility only.
atm→Pa	U→ F&p: ▼ atm→Pa	(1) {2}; {1} → {2} Convert pressures and distances. See pp. 123ff.
au→m	U→ x: au→m	

[9] Precisely, ARCTAN returns the principal value of $arctan(x)$, i.e. a *real* part ∈ [−π/2, π/2] in ∡r, for example (cf. ASIN), if SPCRES is set. Else the result interval for ATAN becomes (−π/2, π/2) in ∡r, for example. Cf. *ISO/IEC 9899*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
A:	U→ f A:	*Submenu*. See p. 123.
BACK	P.FN P.FN2 BACK ***n***	(0) Jumps ***n*** steps backwards (0 ≤ ***n*** ≤ 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	U→ F&p: bar→Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 123ff.
BATT?	INFO 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 1 mV. Voltages < 2.5 V are considered low, voltages < 2.0 V may lead to your *WP 34S* turning off automatically.
bbl→m³	U→ f V: barrel → m³	(1) {2}; {1} → {2} Converts volumes. See pp. 123ff.
BC?	BITS BC? ***n***	(−1) {10} Tests if the specified bit in ***x*** is clear.
BEEP	I/O BEEP	(0) Sounds a sequence of four tones. See also TONE.
BeginP	FIN TVM Begin	(0) Sets “Begin” mode in *TVM*: payments occur at the beginning of each period. Typical for savings plans and leasing. Compare ENDP.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
BestF	STAT ▼ BestF ***n***	(0) Instructs your *WP 43S* to select the 'best' curve fit model for the current statistical data by picking the one with maximum *correlation* out of the models allowed (almost like BEST in *HP-42S*). Relevant for L.R., CORR, COV, s_{XY}, x̂, and ŷ. You can accelerate computation of these functions significantly by excluding curve fit models making no sense for your data (e.g. for physical or technical reasons). The parameter ***n*** carries this information. Each curve fit model corresponds to a number as listed: • LINF 1 1_2 • EXPF 2 10_2 • LOGF 4 100_2 • POWERF 8 1000_2 • ROOTF 16 $1\ 0000_2$ • HYPF 32 $10\ 0000_2$ • PARABF 64 $100\ 0000_2$ • CAUCHF 128 $1000\ 0000_2$ • GAUSSF 256 $1\ 0000\ 0000_2$ Take the numbers of all models you can exclude and sum them up – the result is ***n***. **Example:** Excluding the three 3-parameter models leads to $\boldsymbol{n} = 64 + 128 + 256 = 448$. Hence, to look for the best-fitting 2-parameter model, call BESTF **448**. Note ORTHOF is not part of the set of models under investigation. See pp. 222ff for more.
BestF?	INFO ▼ BestF?	(−1) { } → {1} Returns the 'best' curve fit model for the current statistical data by picking the one with maximum *correlation* out of the models allowed, encoded as an integer according to the list at BESTF.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$Binom_p$ $Binom_{\triangle}$ $Binom_{\triangle}$ $Binom^{-1}$	PROB g Binom: $Binom_p$ etc.	(1) {2}; {1} → {2} *Binomial distribution* with the number of successes ***g*** in **X**, the probability of a success ***p₀*** in **I**, and the sample size ***n*** in **J**. See p. 213 for more. $Binom^{-1}$ returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, ***p₀*** in **I** and ***n*** in **J**.
Binom:	PROB g Binom:	*Submenu*. See p. 117.
BITS	BITS	*Menu*. See p. 112.
B_n	X.FN B_n	(1) {1, 2} B_n and B_n^* return the Bernoulli number for an integer $n > 0$ given in **X**, working with different definitions (see p. 211 for more).
B_n^*	X.FN B_n^*	
BS?	BITS BS? ***n***	(0) {10} Tests if the specified bit in ***x*** is set.
Btu→J	U→ E: Btu→J	(1) {2}; {1} → {2} Converts energies. See pp. 123ff.
cal→J	U→ E: cal→J	(1) {2}; {1} → {2} Converts energies. See pp. 123ff.
CASE	P.FN P.FN2 CASE ***s***	(0) Works like SKIP but takes the number of steps to skip from ***s***. **Example**: Assume a program section: ... 100 CASE 12 101 GTO 01 102 GTO 02 103 GTO 07 104 GTO 05 105 LBL 01 ... 132 LBL 02 ...

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		153 LBL 05 ... 234 LBL 07 ... 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 $1 \leq r12 \leq 4$. **ATTENTION:** CASE might surprise you for $r12 > 4$ in the example above. Take care of the input you provide! 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.**
CATALOG	CATALOG	*Catalog* of everything. See pp. 109ff.
CauchF	STAT ▼ CauchF	(0) Selects the *Cauchy* (a.k.a. *Lorentz*) peak fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 222ff for more.
Cauch$_p$ Cauch$_{\triangle}$ Cauch$_{\triangle}$ Cauch^{-1}	PROB f Cauch: Cauch$_p$ etc.	(1) {2}; {1} → {2} *Cauchy-Lorentz* (a.k.a. *Lorentz* or *Breit-Wigner*) *distribution* with the location x_0 specified in **I** and the shape γ in **J**. See p. 216 for more. Cauch^{-1} returns x for a given probability p in **X**, with x_0 in **I** and γ in **J**.
Cauch:	PROB f Cauch:	*Submenu*. See p. 117.
CB	BITS CB *n*	(1) {10} Clears the specified bit in x, i.e. sets it to **0**.
CEIL	INTS CEIL	(1) {8*}; {1, 2} → {1} Returns the smallest integer ≥ x. Compare FLOOR.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CENTRL	STAT ▲ g PLOT CENTRL	(0) Fits an *orthogonal regression* center line through the data points plotted by PLOT. See *Section 2* of the *OM* for more.
CF	FLAGS CF ***n***	(0) Clears the *flag* specified, i.e. sets it to **0**.
	MODE CF ***n***	
	CLR CF ***n***	
CHARS	CATALOG CHARS	*Submenu* of characters. See pp. 109ff.
chĭ→m	U→ x: ▼ chĭ→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
CLALL	CLR CLall	(0) Clears all *registers*, numbered *user flags*, variables, and programs in *RAM*. Modes will stay as they are. See also CLCVAR, CLFALL, CLPALL, CLREGS, CLSTK, and RESET.
CLCVAR	CLR CLCVAR	(0) Clears all variables used in the *current program*, i.e. sets all such *real, complex* and integer variables to zero, all *time* variables to **0:00:00**, all *date* variables to **January 1st** of year **0**, all *text strings* to zero length, and all the elements of all matrix variables used to **0.**
CLFALL	FLAGS CLFall	(0) Clears (after confirmation unless in *PEM*) all numbered global and local *user flags* (cf. CF). Lettered flags are not cleared since they may coincide with *system flags.*
	CLR CLFall	
CLK	CLK	*Menu*. See p. 108.
CLLCD	CLR CLLCD	(0) Clears all pixels $\geq x$ and $\geq y$ on the screen.
CLMENU	CLR CLMENU	(0) Clears all *menu* key definitions for the programmable *menu*. See MENU.
	P.FN P.FN2 CLMENU	
CLP	CLR CLP	(0) Clears the *current program* in *RAM* or *FM*. Freed memory is returned to the pool of free space.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CLPALL	CLR CLPall	(0) Clears all programs in *RAM*. Cf. CLP.
CLR	CLR	*Menu*. See p. 115.
CLREGS	CLR CLREGS	(0) Clears (after confirmation unless in *PEM*) all global and local *GP registers* allocated (see also LOCR), i.e. sets all these registers to **0**. The contents of the *stack* and **L** are kept, see CLSTK.
CLSTK	CLR CLSTK 0 FILL	Clears all *stack registers* currently allocated (i.e. either **X** … **T** or **X** … **D**). All other *register* contents are kept. Cf. CLREGS.
CLX	CLR CLX ← (for closed *x*)	(1) Clears *stack register* **X**, disabling *ASL*. Cf. CLREGS and CLSTK.
CLΣ	CLR CLΣ STAT CLΣ	(0) Clears the statistical summation *registers* and releases the memory allocated for them (see p. 158). Remember to call CLΣ before accumulating data for a new statistical analysis.
CNST	P.FN CNST *n*	(−1) { } → {2} Returns the constant stored at position ***n*** in CONST (see below and pp. 133ff). Allows for indirectly addressing these constants, also beyond ∞.
COMB	PROB C_{yx}	(2) {1} Returns the number of possible subsets of ***x*** items taken out of a set of ***y*** items (i.e. choose ***x*** out of ***y***). No item occurs more than once in a subset, and different orders of the same ***x*** items are not counted separately. Compare PERM. (2) {2, 3} See pp. 211f for the formula.
CONJ	CPX conj	(1) {3, 9*} Returns the *complex conjugate* of ***x***.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CONST	CONST	*Menu*. Cf. CNST above and see pp. 133ff.
CONVG?	TEST CONVG? *r*	(0) {2} Checks for convergence by comparing ***x*** and ***y*** as determined by the lowest five bits of ***r***. a) The very lowest 2 bits set the tolerance limit: 0 = 10^{-14}, 1 = 10^{-24}, 2 = 10^{-32}. b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the numbers ***x*** and ***y*** relatively, 1 = compare them absolutely. c) The top bit tells how special numbers will be treated: 0 = NaN and ±∞ are considered converged, 1 = they are not considered converged. Now, $\boldsymbol{r} = a + 4b + 16c$.
CORR	STAT ▲ r	(−1) { } → {2} Returns the *coefficient of correlation* for the current statistical data and curve fit model. See pp. 222ff for more.
cos	TRI cos	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the cosine of the angle in **X** (takes ***x*** as tagged, else assumes the current *ADM*; see *Section 2* of the *OM* for details).
cosh	EXP cosh TRI cosh	(1) {2, 3, 8*, 9*} Returns the hyperbolic cosine of ***x***.
COV	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. 222ff for more.
CPX	CPX	*Menu*. See p. 115.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CPXS	CATALOG VARS CPXS	*Submenu* of *complex* variables defined at execution time. See pp. 109f.
CPX?	TEST ▲ CPX?	(0) Checks if x is *complex*. Returns true if **X** contains data of type 3 or 9 with nonzero *imaginary* part.
CROSS	MATX cross	(2) {8} Requires two *real* 2D or 3D vectors in **X** and **Y** and returns their cross product. Crossing of 2D vectors works as it does for *complex numbers*.
	CPX cross	(2) {3} → {2} When two *complex numbers* are crossed, your *WP 43S* simply returns a *real number* that is equal to the signed *magnitude* of the resulting moment vector. See an example in the *OM*.
crt→g	U→ m: carat → g	(1) {2}; {1} → {2} Convert masses and distances. See pp. 123ff.
cǜn→m	U→ x: ▼ cǜn→m	
cwt→kg	U→ m: cwt→kg	
CX→RE	CC (works in *run mode* only)	(−1) {3} → {2}; {9} → {8} Cuts a closed *complex* object x, putting either • (for ∟) its *real* part in **Y** and its *imaginary* part in **X** or • (for ⊙) its *magnitude* in **Y** and *phase* in **X**.
	CPX CX→RE	
DATE	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 *Section 2* of the *OM*).
DATES	CATALOG VARS f DATES	*Submenu* of *date* variables defined at execution time. See pp. 109f.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DATE→	CLK DATE→	(−2) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real number* in corresponding format) and pushes its three components as integers on the *stack*. Reversible by →DATE.
DAY	CLK 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. See also WDAY.
DBLR	INTS DBLR etc.	{10} Double *word* length commands for remainder, multiplication and division (see the *HP-16C OH*, *Section 4*, pp. 52ff). 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× takes ***x*** and ***y*** as factors as usual but returns the product in **X** and **Y** (most significant bits in **X**).
DBL×		
DBL/		
dB→fr	U→ ▲ dB → field ratio	(1) {2}; {1} → {2} Convert ratios. See pp. 123ff.
dB→pr	U→ ▲ dB → power ratio	
DEC	LOOP DEC *r*	(0) {1, 2, 10} Decrements ***r*** by 1. Does not load **L** even for target address **X**.
DECOMP	PARTS 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	MODE DEG	(0) Sets the *ADM* to *decimal degrees*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DEG→	L→ DEG→	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
DELITM	CLR DELITM	(0) Deletes a user-defined *item* from memory. See the *OM*, *Section 6*.
DENMAX	MODE 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	CAT. FCNS DET	Points to |M| explained on p. 89. Maintained for backward compatibility only.
DISP	DISP	*Menu*. See p. 115.
DOT	CPX dot	(2) {3} → {2} Returns $Re(x)\, Re(y) + Im(x)\, Im(y)$
	MATX 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	DROP↓	Drops ***x*** … from the stack. See *Section 1* of the *OM* for details.
DROPy	STK DROPy	Drops ***y***
DSE	LOOP DSE ***r***	(0) {1, 2, 10} Given ***r*** = `ccccc.fffii`, DSE decrements ***r*** by `ii`, skipping next program step if then `ccccc` ≤ `fff` (cf. the rear side of your *WP 43S*). If ***r*** features no fractional part then `fff` is 0. If `ii` = 0, `ccccc` 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 makes sense with `ccccc` > 0 only. …

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		In run mode, DSE returns **true** for `ccccc` > `fff` and **false** for `ccccc` ≤ `fff`.
DSL	LOOP DSL *r*	(0) {1, 2, 10} Works like DSE but skips if `ccccc` < `fff` .
DSTACK	DISP ▲ DSTACK *n*	(0) Sets the maximum number of *stack registers* displayed. For an input of 1, only ***x*** will be shown directly above the *menu section*; 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 *status bar*. This command is for old-school calculator users who feel distracted by a multitude of *stack registers* displayed changing simultaneously while only the lowest ones are really relevant.
DSZ	LOOP DSZ *s*	(0) {1, 2, 10} Decrements ***s*** by 1 and skips the next step if $-1 < s < +1$ thereafter. Does not load **L** even for target address **X**. Known from *HP-29C*, *HP-67*, and *HP-16C*.
D.MS	d.ms (for closed *x*)	(0) Sets the *ADM* to *sexagesimal degrees*.
D.MS→	L→ D.MS→	(1) {1, 2, 4} → {4} Convert angles as described on p. 132.
D.MS→D	L→ D.MS→D	
D.MY	CLK ▲ D.MY	(0) Sets the format `dd.mm.yyyy` for *dates*.
D→D.MS	L→ D→D.MS	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
D→J	CLK D→J	(1) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real number* in corresponding format) and converts it to a *Julian day number* according to the J/G setting. Please see p. 41.
D→R	L→ D→R	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
EIGVAL	MATX ▲ EIGVAL	(−1) {8, 9} Evaluates the matrix ***x*** and pushes a matrix containing its eigenvalues on the *stack*. See the *OM, Sect. 2.*
EIGVEC	MATX ▲ EIGVEC	(−1) {8, 9} Evaluates the matrix ***x*** and pushes a matrix containing its eigenvectors on the *stack*.
END	P.FN 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	FIN TVM End	(0) Sets "End" mode in *TVM*: payments occur at the end of each period. Typical for loans and investments. Cf. BEGINP.
ENG	DISP ENG ***n***	(0) Sets engineer's display format (see *Section 2* of the *OM*); $0 \le n \le 15$ is the number of digits in the mantissa minus 1.
ENORM	MATX ENORM	(1) {8, 9} → {2} Calculates the Euclidean norm of the matrix in **X**. This norm is defined as the square root of the sum of squares of all matrix elements. For a vector, ENORM returns its length. Compare |x| on p. 89.
ENTER↑	ENTER↑	(−1) Separates two entries in input. Copies ***x*** into **Y**, disabling *ASL*. See p. 105 and the *OM, Sect. 1,* for details.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ENTRY?	TEST ENTRY?	(0) Checks the (internal) entry *flag*. It is set if: • 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	EQN	*Menu*. See p. 115.
EQ.DEL	EQN DELETE	Deletes an equation. (See Section 4 of the OM.)
EQ.EDI	EQN EDIT	Opens the *Equation Editor* to edit an existing equation. (See Section 4 of the OM.)
EQ.NEW	EQN NEW	Opens the *Equation Editor* to enter a new equation. (See Section 4 of the OM.)
erf erfc	X.FN erf etc.	(1) {2}; {1} → {2} Returns the error function or its complement. See pp. 247ff for more.
ERR	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. 175ff for the error codes. Compare MSG.
EVEN?	TEST 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 *Sect. 2* of the *OM* for more.
EXITALL	P.FN P.FN2 EXITall	(0) Exits all *menus*.
EXP	EXP	*Menu*. See p. 115.
ExpF	STAT ▼ ExpF	(0) Selects the exponential 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)
Expon$_P$ Expon$_\triangle$ Expon$_\triangle$ Expon^{-1}	PROB f Expon: Expon$_P$ etc.	(1) {2}; {1} → {2} *Exponential distribution* with the rate λ in **I**. See pp. 213ff for more. Expon^{-1} returns the survival time t_s for a given probability p in **X**, with λ in **I**.
Expon:	PROB f Expon:	*Submenu*. See p. 117.
EXPT	PARTS EXPT	(1) {1, 2} → {1} Returns the exponent ***h*** of the number $x = m \cdot 10^h$ displayed. Compare MANT.
e^x-1	EXP e^x-1	(1) {2, 8*} For ***x*** ≈ *0* , this returns a more accurate result for the fractional part than e^x does.
E:	U→ E:	*Submenu*. See p. 123.
FB	BITS FB ***n***	(1) {10} Inverts ('flips') the specified bit in ***x***.
FBR	a.FN FBR	(0) Font browser. Shows all characters designed for your *WP 43S*.
FCNS	CATALOG FCNS	*Submenu* of all functions. See pp. 109ff.
FC?	FLAGS FC? ***n***	(0) Tests if the specified *flag* is clear.
FC?C FC?F FC?S	FLAGS FC?C ***n*** etc.	(0) Tests if the specified *flag* is clear. Clears, flips, or sets this *flag* after testing, respectively.
fēn→m	U→ x: ▼ fēn→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
FF	FLAGS FF ***n***	(0) Flips the *flag* specified.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FIB	X.FN FIB	(1) {1} Returns the Fibonacci number (maximum input is 4791). (1) {2, 3} Returns the extended Fibonacci number. See pp. 211f.
FILL	FILL	Fills the whole *stack* with x.
FIN	FIN	Menu. See p. 115.
FIX	DISP FIX *n*	(0) Sets fixed point display format (see the *OM*, *Section 2*); $0 \le n \le 15$ is the number of decimals of the mantissa.
FLAGS	FLAGS	*Menu*. See p. 116.
FLASH	CAT. PROGS FLASH	*Submenu* of global labels defined in *FM* at execution time. See pp. 109f.
FLASH?	INFO FLASH?	(−1) { } → {1} Returns the number of free *bytes* in *FM*.
FLOOR	INTS FLOOR	(1) {8*}; {1, 2} → {1} Returns the greatest integer $\le x$. Cf. CEIL.
fm.→m	U→ x: ▲ fathom → m	(1) {2, 11; {1} → {2} Converts distances. See pp. 123ff.
FP	PARTS FP # F (for closed x)	(1) {1, 2, 8*, 10} Returns the fractional part of x. Compare IP.
FP?	TEST FP?	(0) Tests x for having a fractional part $\neq 0$.
$F_p(x)$ $F_{\blacktriangle}(x)$ $F_{\blacktriangle}(x)$ $F^{-1}(p)$	PROB F: $F_p(x)$ etc.	(1) {2}; {1} → {2} *Fisher's F distribution*. $F_{\blacktriangle}(x)$ equals Q(F) on *HP-21S*. The degrees of freedom shall be specified in **I** and **J**. See pp. 213ff for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
fr→dB	U→ ▲ field ratio→dB	(1) {2}; {1} → {2} Converts ratios. See pp. 123ff.
FS?	FLAGS FS? *n*	(0) Tests if the specified *flag* is set.
FS?C	FLAGS FS?C *n* etc.	(0) Tests if the specified *flag* is set. Clears, flips, or sets this *flag* after testing, respectively.
FS?F		
FS?S		
ft.→m	U→ x: ft.→m	(1) {2}; {1} → {2} Convert distances and volumes. See pp. 123ff.
ft$_{US}$→m	U→ x: ▲ survey foot$_{US}$ → m	
fz$_{UK}$→ml	U→ f V: floz$_{UK}$ → ml	
fz$_{US}$→ml	U→ f V: floz$_{US}$ → ml	
F:	PROB F:	*Submenu*. See p. 117.
f'	EQN f'	*Submenus* for computing the 1st or 2nd derivative of a given equation. See the *OM*, *Sect. 4*, for more.
f"	EQN f"	
f'(x)	ADV f'(x) ***labl***	{1, 2} → {2} f'(x) [f"(x)] returns the 1st [2nd] derivative of the function ***f***(***x***) at position ***x***. This ***f***(***x***) must be specified in a routine starting with LBL ***labl***. 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. **ATTENTION:** f'(x) and f"(x) fill all *stack registers* with ***x*** before calling the routine specified.
f"(x)	ADV f"(x) ***labl***	
F&p:	U→ F&p:	*Submenu*. See p. 123.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
GAP	DISP ▲ GAP *n*	(0) Defines the interval for inserting digit group separators in *reals* (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 *reals*. See *Section 2* of the *OM*. After GAP **2**, **1**, or **0**, <u>no</u> group separators will be displayed in any number at all.
GaussF	STAT ▼ GaussF	(0) Selects the *Gauss* peak fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 222ff for more.
GCD	INTS GCD	(2) {1; 10} Returns the Greatest Common Divisor of x and y.[10] This will always be positive.
g_d g_d^{-1}	X.FN g_d etc.	(1) {2, 3} ; {1} → {2} Returns the Gudermannian function or its inverse. See p. 248 for details.
$Geom_p$ Geom◭ Geom◭ $Geom^{-1}$	PROB g Geom: $Geom_p$ etc.	(1) {2}; {1} → {2} *Geometric distribution*: The *CDF* returns the probability for a 1st success after $m = x$ Bernoulli experiments. The probability p_0 for a success in each such experiment must be specified in **I**. See pp. 213ff for more. $Geom^{-1}$ returns the number of failures f before 1st success for given probabilities p in **X**, p_0 in **I**.
Geom:	PROB g Geom:	*Submenu*. See p. 117.
gal_{UK}→l gal_{US}→l	U→ f V: gal_{UK}→l etc.	(1) {2}; {1} → {2} Convert volumes. See pp. 123ff.

[10] $GCD(x,y) = \left| \frac{x\,y}{LCM(x,y)} \right|$. 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)
GRAD	MODE GRAD	(0) Sets the *ADM* to *grades* (a.k.a. *gradians* or *gon*).
GRAD→	L→ GRAD→	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
GTO	GTO ***labl***	(0) In *PEM*, inserts an unconditional branch to ***labl***. Else positions the program pointer to ***labl***.
GTO.	GTO . ***nnn***	(0) Positions the program pointer … to local label ***n*** (specify ≤ 2 digits).
		… to step ***nnn*** in *current pgm* – specify ≥ 3 digits until becoming definite.
	GTO . ***labl***	… to the global label specified (shall be picked from PROG).
	GTO . ▲	… directly after previous END, i.e. to the top of *current program* (see *Sect. 3* of the *OM*); if being there already, jumps to the top of previous program.
	GTO . ▼	… directly after next END, i.e. to the top of next program.
	GTO . .	… to the end of used program memory in *RAM*, i.e. right to the final END.
g→crt	U→ m: g → carat	(1) {2}; {1} → {2} Convert masses. See pp. 123ff.
g→oz	U→ m: g→oz	
g→trz	U→ m: g → tr.oz	
ha→ac	U→ f A: ha → acre	(1) {2}; {1} → {2} Convert areas. See pp. 123ff.
ha→ac$_{US}$	U→ f A: ha → acre$_{US}$	
ha→m^2	U→ f A: ha→m^2	
H_n	X.FN Orthog H_n	(2) {2}; {1} → {2} *Hermite polynomials* for probability (H_n) and physics (H_{np}). See p. 242 for details.
H_{np}	… Orthog f H_{np}	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
hp_E→W hp_M→W hp_{UK}→W	U→ P: hp_E→W etc.	(1) {2}; {1} → {2} Convert powers. See pp. 123ff.
$Hyper_p$ Hyper◭ Hyper◬ $Hyper^{-1}$	PROB Hyper: $Hyper_p$ etc.	(1) {2}; {1} → {2} *Hypergeometric distribution* with the number of successes ***g*** in **X**, the probability of a success ***p₀*** in **I**, the sample size ***n*** in **J**, and the batch size ***n₀*** in **K**. See pp. 213ff for more. $Hyper^{-1}$ returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, ***p₀*** in **I**, ***n*** in **J**, and ***n₀*** in **K**.
Hyper:	PROB g Hyper:	*Submenu*. See p. 117.
HypF	STAT ▼ HypF	(0) Selects the hyperbolic fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 222ff for more.
IDIV	INTS IDIV	(2) {1, 10}; {2} → {1} Integer division, working like / + IP.[11]
IDIVR	CATALOG FCNS IDIVR	{1, 2, 10} Like IDIV but returns also the remainder in **Y**.[11]
iHg→Pa	U→ F&p: in.Hg → Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 123ff.
Im	CPX Im PARTS Im	(1) {2, 3} → {2}; {9} → {8}; Returns the imaginary part of ***x***. Compare RE.
INC	LOOP INC *r*	(0) {1, 2, 10} Increments ***r*** by 1. Does not load **L** even for target address **X**.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
INDEX	MATX INDEX *name*	(1) Indexes a named matrix. You can also index a matrix by editing it (see M.EDIT or M.EDIN). After exiting the *Matrix Editor*, the matrix is no longer indexed. – See also *Matrix Utility Functions* in the *HP-42S OM*, pp. 223ff.
INFO	INFO	*Menu*. See p. 116.
INPUT	P.FN INPUT *r*	Works in programs only: Recalls the content of the source specified into **X**, displays the name of the source along with ***r***, and halts program execution, allowing you to enter or calculate a value; pressing R/S then stores ***x*** 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 *name* undefined at execution time, INPUT automatically creates the variable with an initial value of zero.
INTS	INTS	*Menu*. See p. 116.
INT?	TEST INT?	(0) Tests ***x*** for being an integer, i.e. having a fractional part equal to zero. Cf. FP?.
INVRT	CAT. FCNS INVRT	Works like $[M]^{-1}$ on p. 89. Maintained for backward compatibility only.
in.→mm	U→ x: in.→mm	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
IP	PARTS IP # I (for closed ***x***)	(1) {1, 8*}; {2, 10} → {1} Returns the integer part of ***x***. Cf. FP.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ISE	LOOP ISE *s*	(0) {1, 2, 10} Given cccccc.fffii in the source *s*, ISE increments *s* by ii, skipping next program step if ccccc ≥ fff then (cf. the rear side of your *WP 43S* and DSE on p. 29). If *s* features no fractional part then fff is 0. If ii = 0, ccccc will be incremented by 1. ISE does not load **L** even for target address **X**. Note that neither fff nor ii can be negative, but ccccc can. In run mode, ISE returns **true** for ccccc < fff and **false** for ccccc ≥ fff.
ISG	LOOP ISG *s*	(0) {1, 2, 10} Works like ISE but skips if ccccc > fff.
ISM?	INFO ISM?	(−1) { } → {1} Returns the *integer sign mode* set for *short integers*: true 2 for 2's complement, true 1 for 1's complement, false 0 for unsigned, or true −1 for sign & mantissa mode.
ISZ	LOOP ISZ *s*	(0) {1, 2, 10} Increments *s* by 1, skipping next program step if $-1 < s < +1$ thereafter. ISZ does not load **L** even for target address **X**. Known from *HP-29C*, *HP-67*, and *HP-16C*.
I_{xyz}	X.FN I_{xyz}	(3) {1, 2} Returns the *regularized Beta function*.
$I\Gamma_p$ $I\Gamma_q$	X.FN $I\Gamma_p$ etc.	(2) {1, 2} Returns the *regularized Gamma function* (one of two kinds).

See p. 248.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
I+	MATX ▲ I+	(0) Increments or decrements the row index *i* of the indexed matrix. See INDEX and also J+, J−, RCLEL, STOEL, RCLIJ, and STOIJ.
I-	MATX ▲ I-	
I/O	I/O	*Menu*. See p. 116.
jīn→kg	U→ m: ▲ jīn→kg	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
$J_y(x)$	X.FN $J_y(x)$	(2) {2}; {1} → {2} Returns the *Bessel function of first kind* and order *y*. See p. 249 for details.
J+	MATX ▲ J+	(0) 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-	MATX ▲ J-	
J/G	CLK ▲ J/G	(0) {2} Takes *x* specifying the date the *Gregorian* calendar became valid in the region you are interested in. This shall be entered as a real number formatted according to the *date display mode* set (D.MY, M.DY, or Y.MD) reflecting the crucial *Gregorian* date.
J/G?	INFO ▲ J/G?	(-1) { } → {6} Returns the current setting of J/G (see there).
J→Btu J→cal	U→ E: J→Btu etc.	(1) {2}; {1} → {2} Convert energies. See pp. 123ff.
J→D	CLK J→D	(1) {1} → {6} Takes *x* as a *Julian day number*[12] and converts it to a common *date* according to J/G (see above) and the date format selected.

[12] Translator's note: *Julian day number* translates to «jour Julien» in French and to "Julianische Tageszahl" in German. See the corresponding articles in *Wikipedia* for

Item	Keystrokes	Remarks (see pp. 12ff for general information)
J→Wh	U→ E: J→Wh	(1) {2}; {1} → {2} Converts energies. See pp. 123ff.
KEY		See KEYG and KEYX below.
KEYG	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 KEY ***key#*** GTO ***labl*** or KEY ***key#*** XEQ ***labl*** , respectively. Key numbers go from 1 to 18 with 1 corresponding to F1, 9 to f F3, and 14 to g F2, for example.
KEYX	P.FN P.FN2 KEYX ***key#, <u>labl</u>***	
KEY?	TEST KEY? ***<u>r</u>***	(0) Tests if a key was pressed while a routine was running or paused. If <u>no</u> 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 ***r***. Key codes reflect the rows and columns on the keyboard (cf. *OM, Sect. 3*; cf. GETKEY on *HP-42S*).
kg→cwt	U→ m: kg→cwt	(1) {2}; {1} → {2} Convert masses. See pp. 123ff.
kg→jīn	U→ m: ▲ kg→jīn	
kg→lb.	U→ m: kg→lb.	
kg→liă	U→ m: ▲ kg → liăng	
kg→scw	U→ m: kg → sh.cwt	
kg→sto	U→ m: kg → stone	
kg→s.t	U→ m: ▲ kg → short ton	
kg→ton	U→ m: ▲ kg→ton	

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)
km→mi. km→nmi	U→ m: km→mi. etc.	(1) {2}; {1} → {2} Convert distances. See pp. 123ff.
KTYP?	INFO KTYP? *r*	(−1) { } → {1} Assumes a key code in the address specified (cf. KEY?), checks it, and returns its key type: • 0 … 9 if it corresponds to a digit 0 … 9, • 10 if it corresponds to ., E, or +/-, • 11 if it corresponds to f or g, • 13 if it corresponds to a *softkey*, and • 12 if it corresponds to any other key. Helps in user interaction with routines (see the *OM*, *Section 3*)..
LASTx	RCL L	(−1) See *Sect. 1* of the *OM*. This command will be recorded as RCL L in routines.
lbft→Nm lbf→N	U→ ▼ lbf·ft → Nm U→ F&p: lbf→N	(1) {2}; {1} → {2} Convert torques and forces. See pp. 123ff.
LBL	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?	TEST LBL? ***labl***	(0) Tests for existence of the label specified, anywhere in program memory. See LBL for more.
lb.→kg	U→ m: lb.→kg	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
LCM	INTS LCM	(2) {1; 10} Returns the *Least Common Multiple* of x and y.[13] This will always be positive.

---

[13] $\mathrm{LCM}(x,y) = \left| {}^{x\,y}/_{\mathrm{GCD}(x,y)} \right|$. 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?	TEST 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.
$LgNrm_p$ $LgNrm_e$ $LgNrm_\Delta$ $LgNrm^{-1}$	PROB f LgNrm: $LgNrm_p$ etc.	(1) {2}; {1} → {2} *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. 213ff for more. $LgNrm^{-1}$ returns ***x*** for a given probability ***p*** in **X**, with ***μ*** in **I** and ***σ*** in **J**.
LgNrm:	PROB f LgNrm:	*Submenu*. See p. 117.
LinF	STAT ▼ LinF	(0) Selects the linear fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 222ff for more.
liǎ→kg	U→ m: ▲ liǎng → kg	(1) {2}; {1} → {2} Convert masses and distances. See pp. 123ff.
lǐ→m	U→ x: ▼ lǐ→m	
LJ	BITS ▲ 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 $L_{m\alpha}$	X.FN Orthog L_m etc.	(2) {2}; {1} → {2} *Laguerre polynomials* and *Laguerre's generalized polynomials*. See pp. 229f for more.
LN	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	X.FN lnβ	(2) {2, 3}; {1} → {2} Returns the natural logarithm of *Euler's* Beta function (see p. 82).
LNΓ	X.FN lnΓ PROB ▲ lnΓ	(1) {2, 3}; {1} → {2} Returns the natural logarithm of $\Gamma(x)$ (see p. 82). Allows also for calculating really great factorials (see an example in the *OM*).
LN(1+x)	EXP 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	I/O LOAD	Restores the entire backup from the file in the *FAT* system in *FM* whereto it was written by SAVE. LOAD calls LOADP, LOADR, LOADV, LOADΣ, LOADSS, recalls the lettered *registers* (incl. the *stack*!), and signals `Backup restored`.[14]
LOADP	I/O LOADP	(0) Loads the entire program memory from backup and appends it to the programs already in *RAM* (if there is sufficient space – else an error will be thrown). [14]
LOADR	I/O LOADR	(0) Recalls the numbered *GP registers* from backup (incl. the *local registers* allocated). Lettered *registers* will not be recalled.[14] The number of registers copied is the minimum of the registers held in the backup and allocated in *RAM* at execution time.
LOADSS	f I/O LOADSS	(0) Recovers the system state from backup,[14] meaning the entire calculator *configuration* (user assignments, system variables and *flags* as covered by RESET – see p. 60) plus all global *user flags*.

[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	I/O LOADV	(0) Recalls the user-defined variables from backup. [14]
LOADΣ	I/O LOADΣ	(0) Recovers the statistical summation *registers* from backup. Throws an error if there is none. [14]
LocR	P.FN LocR ***n***	(0) Allocates ***n*** *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?	INFO LocR?	(−1) { } → {1} Returns the number of *local registers* currently allocated for the *current routine*.
LOG_{10}	lg	(1) {1, 2, 3, 8*, 9*, 10} ({1} → {2}) Return the logarithms of ***x*** for base 10 or 2, respectively.
LOG_2	EXP lb x	
LogF	STAT ▼ LogF	(0) Selects the logarithmic fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 222ff for more.
$Logis_p$	PROB f Logis: $Logis_p$ etc.	(1) {2}; {1} → {2} *Logistic distribution* with ***μ*** given in **I** and ***s*** in **J**. See pp. 213ff for details.
$Logis_{\blacktriangle}$		
$Logis_{\triangle}$		
$Logis^{-1}$		
Logis:	PROB f Logis:	*Submenu*. See p. 117.
LOG_xy	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	LOOP	*Menu*. See p. 116.
L.INTS	CATALOG VARS L.INTS	*Submenu* of *long integer* variables defined at execution time. See pp. 109ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
L.R.	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, and GAUSSF). For a straight line, a_0 is its y-intercept and a_1 is its slope. For forecasting, see $\hat{x}$ and $\hat{y}$ (note $\hat{x}$ may return 2 results). See pp. 213ff for more.
l.y.→m	U→ x: l.y.→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
l→gal$_{UK}$	U→ f V: l→gal$_{UK}$ etc.	(1) {2}; {1} → {2} Convert volumes. See pp. 123ff.
l→gal$_{US}$		
l→qt.		
m²→ha	U→ f A: m²→ha	(1) {2}; {1} → {2} Convert areas and volumes. See pp. 123ff.
m²→mŭ	U→ f A: m²→mŭ	
m³→bbl	U→ f V: m³ → barrel	
MANT	PARTS MANT	(1) {2}; {1} → {2} Returns the mantissa ***m*** of the number $x = m \cdot 10^h$ displayed. Cf. EXPT.
MASKL	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. **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.
MASKR	BITS MASKR *n*	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MATRS	CATALOG VARS MATRS	*Submenu* of matrix variables defined at execution time. See pp. 109f.
MATR?	TEST ▲ MATR?	(0) Checks if x is a *real* or *complex* matrix.
MATX	MATX	*Menu*. See p. 116.
Mat_X	MATX SIM EQ Mat X	(−1) Returns the solution vector for a system of linear equations and leaves M.SIMQ (see the *OM*, *Section 2*).
max	X.FN max	(2) {1, 2, 4, 5, 6, 7, 10} Returns the maximum of x and y.
MEM?	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	P.FN P.FN2 MENU	Displays the programmable *menu*. See the *OM*, *Sect. 3*, and the *HP-42S OM, Part 2, Sect. 10*, p. 146.
MENUS	CAT. MENUS	*Submenu* of all *menus* defined at execution time. See pp. 109ff.
min	X.FN min	(2) {1, 2, 4, 5, 6, 7, 10} Returns the minimum of x and y.
MIRROR	BITS MIRROR	(1) {10} Reflects the bit pattern in x (e.g. $0001\ 0111_2$ would become $1110\ 1000_2$ for *word* size 8).
mi.→km	U→ x: mi.→m	(1) {2}; {1} → {2} Convert distances, heights, volumes, and pressures. See pp. 123ff.
ml→fz$_{UK}$	U→ f V: ml → floz$_{UK}$ etc.	
ml→fz$_{US}$		
mmH→Pa	U→ F&p: ▲ mmHg → Pa	
mm→in.	U→ x: mm→in.	
mm→pt.	U→ x: mm → point	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MOD	MOD INTS MOD	(2) {1, 2, 10} Returns $y \bmod x$ (modulo, see *Section 2* of the *OM* for examples). Cf. RMD.
MODE	MODE	*Menu*. See p. 116.
MONTH	CLK MONTH	(1) {2, 6} → {1} Assumes x containing a *date* in the format selected (or a *real number* in corresponding format) and extracts the month.
MSG	P.FN P.FN2 MSG	(0) {1, 2} Throws the (*temporary*) *error message* specified by the integer part of x. Cf. ERR. See *App. C* on pp. 175ff for the respective error codes.
MULπ	MODE MULπ	(0) Sets the *ADM* to *multiples of π*.
MULπ→	L→ MULπ→	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
mŭ→m²	U→ f A: mŭ→m²	(1) {2}; {1} → {2} Converts areas. See pp. 123ff.
MVAR	P.FN MVAR ***name***	(0) Defines a *menu variable*. Such variables are required for VARMNU. Works in *PEM* only. See the *OM*, *Section 3*.
MyMenu	CAT. MENUS MyMenu	User *menu*. See the *OM, Section 6.*
Myα	CAT. MENUS Myα	User *menu* in *AIM*. See the *OM, Section 6.*
M.DELR	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	MATX DIM ***name***	(0) {1, 2} Creates a new named matrix or re-dimensions an existing matrix to IP(y) rows and IP(x) columns. Cf. DIM in the *HP-42S OM*, p. 217.

Item	Keystrokes		Remarks (see pp. 12ff for general information)
M.DIM?	INFO DIM?		{8, 9} → {1}
	MATX ▲ DIM?		Returns the dimensions of the matrix ***x*** (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	CLK ▲ M.DY		(0) Selects the format `mm/dd/yyyy` for *dates*.
M.EDI	MATX EDIT		(2) {8, 9} Opens ***x*** using the *Matrix Editor* (like MATRIX EDIT in *HP-42S*).[15] See *Section 2* of the *OM*.
M.EDIN	MATX EDITN ***name***		(2) Opens a named matrix using the *Matrix Editor* (like MATRIX EDITN in *HP-42S*).[15] See *Section 2* of the *OM*.
M.EDIT			*Submenu* for matrix editing, called by M.EDI and M.EDIN. See p. 116.
M.GET	MATX GETM		(0) {1, 2} → {8, 9} Gets a sub-matrix with IP(***y***) rows and IP(***x***) columns out of the indexed matrix, starting with its element ***i j***, into **X**. Cf. M.PUT.
M.GOTO	GOTO	with M.EDIT displayed	(0) Asks for target row and column and moves to this matrix element.
M.GROW	GROW	with M.EDIT displayed	(0) Allows the indexed matrix to grow automatically (cf. J+ and *Section 2* of the *OM*; cf. also GROW in the *HP-42S OM*, p. 213.). See M.WRAP.
M.INSR	INSR	with M.EDIT displayed	(0) Inserts a new row of elements containing 0., left of the current cursor position in the matrix.

15 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. x⇄y) 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.LU	MATX ▲ M.LU	(1) *WP 34S*: Transforms x into its LU decomposition in-situ. The value in $\mathbf{X}$ is replaced by a *descriptor* 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	f MATX NEW	(2) {1, 2} → {8} Creates a new matrix. Its number of rows shall be supplied in $\mathbf{Y}$ and its number of columns in $\mathbf{X}$. M.NEW returns a matrix x with all its elements set to zero.
M.PUT	MATX PUTM	(0) {8, 9} Puts the matrix x as it is into the indexed matrix, starting with the target element $i\,j$. Cf. M.GET.
M.R⇄R	MATX ▲ R⇄R	(0) {1, 2} Swaps row IP(x) and row IP(y) of the indexed matrix.
M.SIMQ		*Submenu* of MATX, called by SIM_EQ.
M.SQR?	TEST ▲ M.SQR?	(0) Returns true if x is a square matrix.
M.WRAP	WRAP with M.EDIT displayed	(0) Controls the index pointers (see *Section 2* of the *OM*). Cf. M.GROW.
m:	U→ m:	*Submenu*. See p. 119.
m→au	U→ x: m→au	(1) {2}; {1} → {2} Convert distances. See pp. 123ff.
m→chĭ	… ▼ m→chĭ etc.	
m→cùn		
m→fēn		
m→fm.	… ▲ m → fathom	
m→ft.	… m→ft.	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
m→ftUS	… ▲ m → survey footUS	(1) {2}; {1} → {2} Convert distances. See pp. 123ff.
m→lĭ	… ▼ m→lĭ	
m→l.y.	… m→l.y.	
m→pc	… m→pc	
m→yd.	… m→yd.	
m→yĭn	… ▼ m→yĭn	
m→zhàn	… ▼ m → zhàng	
NAND	BITS NAND	(2) Works in analogy to AND. See p. 18.
NaN?	TEST ▲ NaN?	(0) Returns true if x is *Not a Number*. For complex x, Re(x) or Im(x) must be **±NaN**..
NBinp NBin⊿ NBin⊿ NBin⁻¹	PROB g NBin: NBinp etc.	(1) {2}; {1} → {2} *Negative binomial distribution* with the *number of successes* ***k*** in **X**, the *gross probability* ***p₀*** *of a success in a single draw* in **I**, and *number of failures* ***n*** *until the experiment is stopped* in **J**. See pp. 213ff for more information.
NBin:	PROB g NBin:	*Submenu*. See p. 117.
NEIGHB	INFO NEIGHB	(2) {1} Returns … • $x+1$ for $x<y$; • x for $x=y$; • $x-1$ for $x>y$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		(2) {2} Returns the nearest machine-representable number to x in the direction towards y in the mode set. For • … $x < y$, 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	X.FN NEXTP	(1) {1, 10}; {2} → {1} Returns the next prime number greater than |IP(x)|. See also PRIME? on p. 56.
nmi.→km	U→ x: nmi.→m	(1) {2}; {1} → {2} Convert distances and torques. See pp. 123ff.
Nm→lbft	U→ ▼ Nm → lbf·ft	
NOP	P.FN P.FN2 NOP	'Empty' step (for historical reasons only).
NOR	BITS NOR	(2) Works in analogy to AND. See p. 18.
$Norml_p$	PROB Norml: $Norml_p$ etc.	(1) {2}; {1} → {2} *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. 213ff for more. $Norml^{-1}$ returns x for a given probability p in X, with μ in I and σ in J.
Norml (lower-tail symbol)		
Norml (upper-tail symbol)		
$Norml^{-1}$		
Norml:	PROB Norml:	*Submenu*. See p. 117.
NOT	BITS NOT	(1) {10} Inverts x bit-wise as on *HP-16C*. (1) {1, 2} → {1} Returns 1 for $x = 0$, and 0 for $x \neq 0$.
NXTFIT	STAT ▲ g ASSESS NXTFIT	(0) Displays the next fit model allowed for the accumulated data points. Cf. BESTF.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
nΣ	Σ n	(−1) { } → {1} Recalls the number of accumulated data points.
N→lbf	U→ F&p: N→lbf	(1) {2}; {1} → {2} Converts forces. See pp. 123ff.
ODD?	TEST ODD?	(0) Checks if *x* is integer (see INT?) and odd.
OFF	OFF	(0) Turns your *WP 43S* off. In *PEM*, inserts a step to turn it off under program control.
OR	BITS OR	(2) Works in analogy to AND, see p. 18.
OrthoF	STAT ▼ OrthoF	(0) Selects the linear orthogonal fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 222ff for more.
ORTHOG	X.FN Orthog	*Submenu* for orthogonal polynomials, see p. 120.
oz→g	U→ m: oz→g	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
ParabF	STAT ▼ ParabF	(0) Selects the parabolic fit model. Relevant for COV, CORR, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 222ff for more.
PARTS	PARTS	*Menu*. See p. 117.
PAUSE	P.FN PAUSE ***n***	(0) Within a routine running, refreshes the display and pauses program execution for ***n*** ticks (s. TICKS), with 0 ≤ ***n*** ≤ 99. The pause will terminate early when you press a key.
Pa→atm	U→ F&p: ▼ Pa→atm	(1) {2}; {1} → {2} Convert pressures. See pp. 123ff.
Pa→bar	… Pa→bar	
Pa→iHg	… Pa → in.Hg	
Pa→mmH	… Pa → mmHg	
Pa→psi	… Pa→psi	
Pa→tor	… Pa → torr	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
pc→m	U→ x: pc→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
PERM	PROB P_{yx}	(2) {1} Returns the number of possible <u>arrangements</u> (a.k.a. *permutations*) of ***x*** *items* taken out of a set of ***y*** *items*. No *item* occurs more than once in an arrangement, and <u>different orders</u> of the same ***x*** *items* <u>are counted</u> separately. Cf. COMB.
		(2) {2, 3} See pp. 211f for the formula.
PGMINT	ADV PGMINT ***labl***	Specifies the address of the expression to be integrated or solved, respectively. See *Section 4* of the *OM*.
PGMSLV	ADV PGMSLV ***labl***	
PIXEL	P.FN P.FN2 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	STAT ▲ g PLOT	(0) Plots the data points stored in the statistics registers. See *Section 2* of the *OM* for more.
P_n	X.FN Orthog P_n	(1) {2}; {1} → {2} *Legendre polynomials*. See pp. 229f for more.
POINT	P.FN P.FN2 POINT	(1) {1, 2} Turns on a square point (3×3 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.
$Poiss_p$	PROB g Poiss: $Poiss_p$ etc.	(1) {2}; {1} → {2} *Poisson distribution* with the number of successes ***g*** in **X** and the *Poisson* parameter ***λ*** in **I**. See pp. 213ff for details. $Poiss^{-1}$ returns the maximum number of successes ***m*** for a given probability ***p*** in **X** and ***λ*** in **I**.
$Poiss_{\Delta}$		
$Poiss_{\Delta}$		
$Poiss^{-1}$		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Poiss:	PROB g Poiss:	*Submenu*. See p. 117.
PopLR	P.FN PopLR	(0) Pops the local *registers* allocated to the *current routine* (see the *OM*, *Section 3*) without returning to the calling routine. See LOCR and RTN.
PowerF	STAT ▼ PowerF	(0) Selects the power curve fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ (see pp. 222ff for more).
PRCL	P.FN PRCL	(0) Copies the *current program* (from *FM* or *RAM*) and appends it to *RAM*, where it can then be edited (see the *OM*). PRCL allows for duplicating programs in *RAM*. Will only work with enough space at destination. Recall a library routine from *FM*, edit it, and PSTO – this way you can modify this part of the *FM* library (see PSTO).
PRIME?	TEST PRIME?	(0) {1, 2, 10} Checks if $\|IP(x)\|$ is a prime. Returns true for prime and false for composite. For $x > 3.3 \times 10^{24}$, true means 'probably prime' (see p. 164 for more).
PRINT	PRINT	*Menus*. See p. 117.
PROB	PROB	
PROG		*Submenu* of global labels defined when calling XEQ etc. See *Section* 3 of the *OM*.
PROGS	CAT. PROGS	*Submenu* of global labels defined at execution time. See pp. 109f.
pr→dB	U→ ▲ power ratio → dB	(1) {2}; {1} → {2} Convert ratios or pressures. See pp. 123ff.
psi→Pa	U→ F&p: psi→Pa	

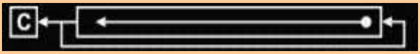
Item	Keystrokes	Remarks (see pp. 12ff for general information)
PSTO	P.FN PSTO	(0) Copies the *current program* (see the *OM)* from *RAM* and appends it to the *FM* library. Cf. PRCL. This program must include at least one LBL statement with a global label (preferably at its beginning). If a program with 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.
pt.→mm	U→ x: point → mm	(1) {2}; {1} → {2} Converts print heights. See pp. 123ff.
PUTK	P.FN PUTK *r*	(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	P.FN	*Menu*. See p. 118.
P.FN2	P.FN P.FN2	*Submenu*. See p. 118.
P:	U→ P:	*Submenu*. See p. 123.
qt.→l	U→ x: qt.→l	(1) {2}; {1} → {2} Converts volumes. See pp. 123ff.
RAD	MODE RAD	(0) Sets the *ADM* to *radians*.
RAD→	L→ RAD→	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
RAM	CAT. PROGS RAM	*Submenu* of global labels defined at execution time. See pp. 109f.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RANGE	DISP ▲ RANGE	(0) {1, 2} Limits the range of displayable *real numbers* to $\pm 10^{\pm n}$ with n = IP(x), $99 \le n \le 6145$. 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?	DISP ▲ RANGE? INFO ▲ RANGE?	(-1) { } → {1} Returns the current range setting.
RANI#	PROB ▲ RANI#	(−1) { } → {1} Returns an integer random number n with $\mathrm{IP}[\min(x,y)] \le n \le \mathrm{IP}[\max(x,y)]$. After executing RANI#, the stack looks like [n, x, y, …], so you can drop or roll down n and call RANI# again to get another random integer with the same parameters. You can use RANI# e.g. for throwing dices.
RAN#	PROB ▲ RAN#	(−1) { } → {2} Returns a random number between 0. and 1. See also SEED.
RBR	RBR	Calls the *register browser* – see the *OM*, *Sect. 5*. You may call RBR also in *PEM* but it is not programmable. Within RBR, *real* and *complex numbers* 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 *reals* and ALL **0**, RBR display will turn to SCI or ENG at 33 digits in worst case. Extended display precision may be observed for *complex numbers* as well.
RCL	RCL *r*	(−1) Recalls the content of a *register* or variable and pushes it on the *stack*.
RCLCFG	RCL Config *r*	(0) Recalls a calculator *configuration* stored by STOCFG (see the *OM*, *Sections 2* and *6*). Cf. also RESET on p. 60.

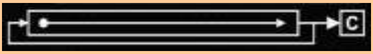
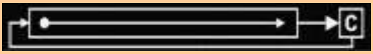
Item	Keystrokes	Remarks (see pp. 12ff for general information)
RCLEL	MATX RCLEL RCL ...EL OLD with M.EDIT displayed	(−1) { } → {2, 3} ((OLD works)) Recalls a copy of the current element a_{ij} of the indexed matrix. Cf. STOEL.
RCLIJ	MATX ▲ RCLIJ RCL ...IJ	(−2) { } → {1} Recalls the current values of 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	RCL 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+	RCL + *r*	(1) Recalls a content of a *register* or variable, executes the operation specified, and puts the result on the *stack* like a *monadic* function. [16] **Example:** RCL−12 replaces ***x*** with ***x − r12***.
RCL−	RCL − *r*	
RCL×	RCL × *r*	
RCL/	RCL / *r*	
RCL↑	RCL Max *r* RCL ▲ *r*	(1) {1, 2, 4, 5, 6, 10} Replaces ***x*** with the maximum of ***r*** and ***x***.[16]
RCL↓	RCL Min *r* RCL ▼ *r*	(1) {1, 2, 4, 5, 6, 10} Replaces ***x*** with the minimum of ***r*** and ***x***.[16]
RDP	DISP RDP *n*	(1) {2, 3, 4, 5, 8*, 9*} Rounds ***x*** to ***n*** decimal places (0 ≤ ***n*** ≤ 99, think of FIX format), taking the RM setting into account. See RM and compare RSD. 123.456 789 012 RDP **5** returns 123.456 79.
Re	CPX Re PARTS Re	(1) {2, 3} → {2}; {9} → {8} Returns the *real* part of ***x***. Cf. IM.

[16] 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)
REALS	CAT. VARS REALS	*Submenu* of *real* variables defined at execution time*.* See pp. 109f.
REAL?	TEST ▲ REAL?	(0) Checks if *x* is a *real* number or matrix.
RECV	I/O RECV	(0) Prepares your *WP 43S* for receiving data via serial I/O. See SEND and *Sect. 3* in the *OM* for more.
RESET	CLR RESET	Executes CLALL and resets your *WP 43S* to *startup default configuration*, 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, it also sets RANGE to **6145**, resets the random number generator, sets the *flags* ASLIFT, DECIM**.**, DENANY, MULT×, PROPFR, TDM24, and clears all other *system flags* (see pp. 97ff). See said commands and *system flags* for more.
RE→CX	CC (works in *run mode* only)	(2) {2} → {3} Composes a *complex number* out of two *reals* or integers *x* and *y*, setting ℂ and taking either… • (for ∟) the *real* part from **Y** and *imaginary* part from **X**, or • (for ⊙) *magnitude* from **Y** and *phase* from **X**.
	CPX RE→CX	(2) {8} → {9} Works in analogy for two *real* matrices *x* and *y*.
Re⇄Im	CPX Re⇄Im	(1) {3, 9*} Swaps *real* and *imaginary* parts of *complex* objects.
RJ	BITS ▲ RJ	{10} 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**. **Example:** $10\,1100_2$ RJ results in $y = 1011_2$ and $x = 2$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RL	BITS ▲ RL ***n***	(1) {10}  Works like ***n*** consecutive RLs on *HP-16C*, similar to RLn there (0 ≤ ***n*** ≤ 63). RL **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RLC	BITS ▲ RLC ***n***	(1) {10}  Works like ***n*** consecutive RLCs on *HP-16C*, similar to RLCn there (0 ≤ ***n*** ≤ 64). RLC **0** acts as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RM	MODE RM ***n***	(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: 0: round half even: ½E 0.5 rounds to next even number (default, used in science). 1: round half up: ½↑ 0.5 rounds up ('business-man's rounding' [17]). 2: round half down: ½↓ 0.5 rounds down. 3: round up: ←0→ rounds away from 0. 4: round down: →0← rounds towards 0. 5: ceiling: ⌈x⌉ rounds towards +∞. 6: floor: ⌊x⌋ rounds towards –∞. The abbreviations printed on grey background are used in STATUS for indicating the respective rounding modes (see *Section 5* of the *OM*).
RMD	RMD INTS RMD	(2) {1, 2, 10} Returns the remainder of a division. Equals RMD on *HP-16C* but works for *reals* as well. See the *OM*, *Section 2*, for examples. Cf. MOD.

[17] 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)
RM?	INFO RM?	(−1) { } → {1} Returns the floating point rounding mode set. See RM for more.
RNORM	MATX ▲ RNORM	(1) {8, 9} → {2} 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	STAT ▼ RootF	(0) Selects the root fit model; relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 222ff for more.
ROUND	DISP ROUND	(1) {2, 3, 4, 5, 6, 8*, 9*} Rounds ***x*** using the current display format; i.e. all digits not currently displayed are lost for further calculations, just the rounded value survives. Works like RND on *HP-42S.*
ROUNDI	DISP ROUNDI	(1) {8*}; {2} → {1} Rounds ***x*** to next integer. ½ rounds to 1. Cf. IP.
RR	BITS ▲ RR ***n***	(1) {10}  Works like ***n*** consecutive RRs on *HP-16C*, similar to RRn there (0 ≤ ***n*** ≤ 63). RR **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RRC	BITS ▲ RRC ***n***	(1) {10}  Works like ***n*** consecutive RRCs on *HP-16C*, similar to RRCn there (0 ≤ ***n*** ≤ 64). RRC **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RSD	DISP RSD ***n***	(1) {2, 3, 4, 5, 8*, 9*} Rounds ***x*** to ***n*** significant digits (1 ≤ ***n*** ≤ 34), taking the RM setting into account. Think of SCI. Cf. RM and RDP. 123.456 789 012 RSD **7** returns 123.456 8 (seven digits survive).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RSUM	MATX ▲ RSUM	(1) {8, 9} Calculates the row sum of the matrix ***x***, returning an ***m*** × 1 matrix filled with the row sums of the ***m*** × ***n*** input matrix.
RTN	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 ↟ 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 ↟ is lit.
RTN+1	P.FN P.FN2 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.
R-CLR	P.FN R-CLR	(0) {2} Interprets ***x*** in the form `sss.nn`. Clears ***nn*** *registers* starting with address ***sss***. **Example:** For ***x*** = 34.567 , R-CLR will clear **R34** through **R89**. **ATTENTION:** For ***nn* = 0** , clearing will cover the maximum available: • For ***sss*** ∈ [0; 99], it will stop at **R99**. • For ***sss*** ∈ [100; 111] , it will stop at **K**. • For ***sss*** ≥ 112, it will stop at the highest currently allocated local *register*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R-COPY	P.FN R-COPY	(0) {2} Interprets ***x*** in the form sss.nnddd. Takes ***nn*** *registers* starting with address ***sss*** and copies their contents to ***ddd*** etc. **Example:** For ***x*** = 7.0304567 , ***r07, r08, r09*** will be copied into **R45, R46, R47**, respectively. For ***x*** < 0 , R-COPY will take ***nn*** *registers* from *FM* instead, starting with *register* number **|*sss*|** . Destination will be in *RAM* always. **ATTENTION:** For ***nn* = 0** , copying will cover the maximum available as explained with R-CLR. Then ***x*** must be negative.
R-SORT	P.FN R-SORT	(0) {2} Interprets ***x*** in the form sss.nn. Sorts the contents of ***nn*** *registers* starting with address ***sss***. **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. **ATTENTION:** For ***nn* = 0** , sorting will cover the maximum available as explained with R-CLR.
R-SWAP	P.FN R-SWAP	(0) {2} Works like R-COPY but swaps the contents of source and destination *registers*.
R→D	L→ R→D	(1) {1, 2, 4} → {4} Converts angles as described on p. 132.
R↑	R↑	Rotates the *stack* contents one level up or down, respectively. See *Section 1* of the *OM* for details.
R↓	R↓	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
s	STAT 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. 222ff for the formula.
SAVE	SAVE	(0) Saves user program space, *registers*, variables and system state to a file in the *FAT* system, and returns Saved. Recall your backup using the different flavors of LOAD (see there).
SB	BITS SB ***n***	(1) {10} Sets the specified bit in x .
SCI	DISP SCI ***n***	(0) Sets scientific display format (see *Section 2* of the *OM*); $0 \leq n \leq 15$ is the number of decimals of the mantissa.
scw→kg	U→ m: short cwt → kg	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
SDIGS?	INFO SDIGS?	(−1) { } → {1} Returns the number of significant digits set by SETSIG. Also returned by STATUS.
SDL	P.FN P.FN2 SDL ***n*** etc. or	(1) {1, 2} Shifts digits left (right) by ***n*** decimal positions, equivalent to multiplying (dividing) x by 10^n. SDR for long integers ends with zero. Compare SL and SR for binary integers.
SDR	DISP SDL ***n*** etc.	
SEED	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	I/O SEND	(0) Sends all *RAM* data to the device connected via serial I/O. See RECV and the *OM*, *Section 3* for more.

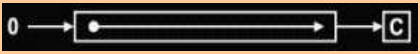
Item	Keystrokes	Remarks (see pp. 12ff for general information)
SETCHN	DISP ▲ CHINA	(0) Sets regional format preferences. [18]
SETDAT	CLK ▲ SETDAT	(0) Sets the date for the real-time clock.[19]
SETEUR	DISP ▲ EUROPE	(0) Set regional format preferences.[18]
SETIND	DISP ▲ INDIA	
SETJPN	DISP ▲ JAPAN	
SETSIG	MODE SETSIG	(0) {1} Sets the number of significant digits (1 ... 34) for rounding after each operation. SETSIG 0 sets maximum precision.
SETTIM	CLK ▲ SETTIM	(0) Sets the time for the real-time clock.[19]
SETUK	DISP ▲ UK etc.	(0) Set regional format preferences.[18]
SETUSA		
SF	FLAGS SF *n* MODE SF *n*	(0) Sets the *flag* specified.
SHOW	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 *complex number*, either part of it will take one display row if one row cannot take both parts. For a *real number*, *time*, or *date* up to 34, 35, or 15 digits will be displayed, respectively. For a *long integer*, 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*).

[18] See *Section 2* of the *OM* about localization of numeric output.

[19] Works on the calculator only – the simulator takes this information from the PC clock.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SIGN	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 $\mathrm{signum}(x)$. (1) {3} Returns the unit vector of the *complex number* x (cf. UNITV). Maintained for backward compatibility only.
SIGNMT	BITS ▼ SIGNMT INTS ▲ SIGNMT	(0) Sets sign-and-mantissa mode for operations on *short integers*. See the *OM*, *Sect. 2.*
SIM_EQ	MATX SIM EQ *n*	(0) Solves a system of n 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** (takes x as tagged, else assumes the current *ADM*; see *Section 2* of the *OM* for details)..
sinc	TRI sinc	(1) {2, 3, 4, 8*, 9*} Returns $\sin(x)/x$ … 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*.
sincπ	TRI sincπ	(1) {2, 3, 4, 8*, 9*} Returns $\sin(\pi x)/\pi x$ … (see sinc)
sinh	TRI sinh EXP sinh	(1) {2, 3, 8*, 9*} Returns the hyperbolic sine of x.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SKIP	P.FN P.FN2 SKIP *n*	(0) Skips ***n*** program steps forwards (0 ≤ ***n*** ≤ 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	BITS ▲ SL *n*	(1) {10}  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	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***, …], and tests the result. The following holds for *real* parameters: • 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 *complex* root in **X** and the second in **Y** (the *complex conjugate* of the first). In a routine, the step after SLVQ will be skipped. In either case (also for *complex* parameters), SLVQ returns ***r*** in **Z**. Higher *stack registers* are kept unchanged. **L** will contain equation parameter ***c***.
s_m	STAT s_m	(−2) { } → {2} Takes the statistical data accumulated and pushes the *standard errors* (i.e. *std. deviations* 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*).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
s_{mi}	STAT ▲ g PLOT s_{mi}	(0) Returns the measured precision of the measuring instrument investigated according to the method shown in *Sect. 2* of the *OM*. Note the basis must be ≥ 30 data points or an error will be thrown.
s_{mw}	STAT s_{mw}	(−1) { } → {2} Returns the *standard error* for weighted data, i.e. the *standard deviation* of the mean $\bar{x}_w$.
SNAP	SNAP	(0) Stores a snapshot of the screen (a.k.a. screen-shot) in a `BMP` file on the calculator's *USB* 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	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*** 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 *stack registers* with ***x*** before calling the routine specified.
Solver	EQN Solver	*Submenu* for solving a given equation. See the *OM, Section 4,* for more.
SPEC?	TEST ▲ SPEC?	(0) True if ***x*** is 'special' (i.e. ±∞ or NaN).
SR	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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SSIZE?	INFO SSIZE?	(−1) { } → {1} Returns the number of *stack registers* currently allocated, 4 or 8.
STAT	STAT	*Menu*. See p. 118.
STATUS	STATUS	Displays number of free bytes in *RAM,* flags set, local registers allocated, and system variables. See the OM, *Sect. 5* for more.
	FLAGS STATUS	
STK	STK	*Menu*. See p. 118.
STO	STO ***r***	(0) Stores ***x*** into destination.[20]
STOCFG	STO Config ***r***	(0) Stores the current calculator *configuration* in a variable or *register* for later use as described in *Sections 2* and *6* of the *OM*. RCLCFG recalls such data. Cf. also RESET on p. 60.[20]
STOEL	MATX STOEL	(1) {1, 2, 3} Stores a copy of ***x*** into the indexed matrix at the current element, a_{ij}. Cf. RCLEL.[20]
	STO ...EL	
STOIJ	MATX ▲ STOIJ	(1) {1} Sets the index pointers to IP(***x***) (column number) and IP(***y***) (row number). Cf. RCLIJ.[20]
	STO ...IJ	
STOP	R/S	(0) Stops program execution. May be inserted in programs to wait for input, for example.
STOS	STO Stack ***r***	(0) Stores the entire *stack* in a set of 4 or 8 *registers*, starting at the destination address specified. Cf. RCLS.[20]
STO+	STO + *r*	(0) Executes the specified operation on ***r*** and stores the result (e.g. ***r*** − ***x***) at the address specified.[21]
STO−	STO − *r*	
STO×	STO × *r*	
STO/	STO / *r*	

[20] **L** will not be loaded.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
sto→kg	U→ m: stone→kg	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
STO↑	STO Max *r*	(0) {1, 2, 4, 5, 6, 10} Stores the maximum (or minimum) of ***r*** and ***x*** in the address specified.[21]
	STO ▲ *r*	
STO↓	STO Min *r*	
	STO ▼ *r*	
STRI?	TEST STRI?	(0) True if ***x*** is a *text string* (cf. STR? in *HP-42S*).
STRING	CATALOG VARS STRING	*Submenu* of *text string* variables defined at execution time. See pp. 109f.
SUM	STAT SUM	(−2) { } → {2} Recalls the linear sums *Σy* and *Σx*. Useful in basic 2D vector algebra. Output labeled in analogy to s.
s_w	STAT s_w	(−1) { } → {2} Calculates the *standard deviation* for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See pp. 213ff for the formula.
s_{xy}	STAT ▲ s_{xy}	(−1) { } → {2} Calculates the *sample covariance* for the two data sets {***x***} and {***y***} entered via Σ+, depending on the curve fit model selected. See pp. 222ff for the formula and COV for the *population covariance*.
SYSTEM	MODE SYSTEM	(0) Returns to *DMCP*. Works on the calculator only (not on the simulator). See *App F*.
SYS.FL		*Submenu* of *system flags* accessible by CF, FC?, FF, FS?, SF, etc.

[21] 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(a)	STAT ▲ s(a)	(−2) { } → {2} Pushes the standard errors s(a_1) (in Y), and s(a_0) (in X) for the parameters of the line fitted through the data points accumulated in the statistical summation *registers* on the *stack*. Works for LINF only. See pp. 229f for more.
S.INTS	CAT. VARS S.INTS	*Submenu* of *short integer* variables defined at execution time. See pp. 109f.
s.t→kg	U→ m: ▲ short ton → t	(1) {2}; {1} → {2} Convert masses and times. See pp. 123ff.
s→year	U→ s→year	
tan	TRI tan	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the tangent of the angle in **X** (takes ***x*** as tagged, else assumes the current *ADM*; see *Sect. 2* of the *OM* for details). Returns "*Not a Number*" for ***x*** = ±90° or equivalents if SPCRES is set.
tanh	EXP tanh TRI tanh	(1) {2, 3, 8*, 9*} Returns the hyperbolic tangent of ***x***.
TDISP	CLK ▲ TDISP *n*	(0) Sets time display format. TDISP **0** allows for the full string from *hours* to *milli-seconds* (trailing decimal zeros will not be shown), TDISP **1** or **2** truncates it to *hours* **:** *minutes* only, TDISP **3** to *hours* **:** *minutes* **:** *seconds*, and **4** ≤ ***n*** ≤ **6** displays also ***n*** − 3 digits for decimal fractions of *seconds*. **Example:** With TDISP **5**, a time string might look like 157:26:38.49
TEST	TEST	*Menu*. See p. 118.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
TICKS	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	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	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	CAT. VARS f TIMES	*Submenu* of *time* variables defined at execution time. See pp. 109f.
T_n	X.FN Orthog T_n	(2) {2}; {1} → {2} *Chebyshev polynomials of first kind*. See pp. 229f for details.
TONE	I/O TONE ***n***	(0) Sounds a tone according to ***n*** (= 1 … 9) .
ton→kg	U→ m: ▲ ton→kg	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
TOP?	TEST 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	U→ F&p: torr → Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 123ff.
$t_p(x)$ $t_{\triangle}(x)$ $t_{\triangle}(x)$ $t^{-1}(p)$	PROB t: $t_p(x)$ etc.	(1) {2}; {1} → {2} *Student's t distribution*. The degrees of freedom are stored in **I**. $t_{\triangle}$ (x) equals Q(t) on *HP-21S*. See *Sect. 2* of the *OM* for an application example and pp. 213ff for more mathematical details.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
TRANS	CAT. FCNS TRANS	Works like $[M]^T$ on p. 89. Maintained for backward compatibility only.
TRI	TRI	*Menu*. See p. 118.
trz→g	U→ m: tr.oz→g	(1) {2}; {1} → {2} Converts masses. See pp. 123ff.
TVM	FIN TVM	*Submenu* for the *Time Value of Money*. See *Section 5* of the *OM*.
t:	PROB t:	*Submenu*. See p. 117.
t⇆	STK t⇆ *r*	Swaps ***t*** and ***r***, in analogy to ***x***⇆
ULP?	INFO ULP?	(1) {1, 2} Returns 1 times the smallest power of ten which can be added to ***x*** or subtracted from ***x*** to actually change the (internal) value of ***x*** in your *WP 43S* in the mode set. Thus, 1 is returned for integers. Indicated in STATUS.
U_n	X.FN Orthog U_n	(2) {2}; {1} → {2} *Chebyshev polynomials of second kind*. See pp. 229f for details.
UNDO	↶	Recalls the *stack*, summation *registers*, and *system flags* as they were before the last operation executed – unless that operation was one which is printed red or orange in this *IOI*.
UNITV	MATX UNITV	(1) {8, 9} Returns the unit vector for the matrix ***x***. Each element of the matrix is adjusted so its overall Euclidean norm becomes 1 (see ENORM); for a vector, its *magnitude* will become 1.
	g CPX UNITV	(1) {3} Returns a *complex number* with *magnitude* |***r***| = 1 in direction of ***x***.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
UNSIGN	BITS ▼ UNSIGN INTS ▲ UNSIGN	(0) Sets unsigned mode for mode for operations on *short integers*. Cf. UNSGN on *HP-16C*. See *Section 2* of the *OM*.
U→	U→	*Menu*. See p. 118.
VAR		*Submenu* of variables defined at execution time when calling STO, RCL, etc.
VARMNU	P.FN VARMNU ***labl***	Creates a variable *menu* using the MVAR instructions following the global label specified. See the *OM*, *Section 3.*
VARS	CAT. VARS	*Submenu* of variables defined at execution time. See pp. 109f.
VERS?	INFO VERS?	(0) Shows firmware version and build number until next keystroke (see the *OM*, *Section 2*).
VIEW	VIEW ***r***	(0) Shows ***r*** until the next key is pressed. **Example:** If a variable called **Test12** contains **−123.45**, VIEW α **Test12** ENTER↑ will display Test12 = -123.45
V:	U→ f V:	*Submenu*. See p. 123.
V∡	∡	(2) {8} → {4} Returns the angle between two 2D or 3D vectors $\vartheta = arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{\lvert\vec{v}_1\rvert\lvert\vec{v}_2\rvert}\right)$
WDAY	CLK WDAY	(1) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real number* in a corresponding format) and returns the name of the respective day and a corresponding integer (Monday = 1).[22]

[22] 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)
Weibl_p Weibl▲ Weibl▲ Weibl^{-1}	PROB f Weibl: Weibl_p etc.	(1) {2}; {1} → {2} *Weibull distribution* with its shape parameter ***b*** in **I** and its characteristic lifetime ***T*** in **J**. See pp. 213ff for details. Weibl^{-1} returns the survival time t_s for a given probability ***p*** in **X**, with ***b*** in **I** and ***T*** in **J**.
Weibl:	PROB f Weibl:	*Submenu*. See p. 117.
WHO?	INFO WHO?	(0) Displays credits to the brave men who made this project work (temporary message).
Wh→J	U→ E: Wh→J	(1) {2}; {1} → {2} Converts energies. See pp. 123ff.
W_m W_p W^{-1}	X.FN ▲ W_m etc.	(1) {2, 3}; {1} → {2} W_p returns the principal branch of *Lambert's* W for given $x \geq -1/e$. W_m returns its negative branch (works for $x \in \mathbb{R}$ only). W^{-1} returns x for a given W_p (≥ -1). See pp. 247ff for more.
WSIZE	BITS ▼ WSIZE *n* INTS ▲ WSIZE *n*	(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 *short integer* values in **L** and in the *stack registers*. All other memory content stays as is (see *App. B* on pp. 166f). 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?	INFO WSIZE?	(−1) { } → {1} Recalls the *word* size set.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
W→hp$_E$ W→hp$_M$ W→hp$_{UK}$	U→ P: W→hp$_E$ etc.	(1) {2}; {1} → {2} Convert powers. See pp. 123ff.
$\hat{x}$	STAT ▲ $\hat{x}$	(1) {2}; {1} → {2} Returns a forecast ***x*** for a given ***y*** (in **X**) according to the curve fit model chosen. See p. 47 for more.
$\bar{x}$	STAT $\bar{x}$	(−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.
x^2	x^2	(1) {1, 2, 3, 8*, 9*, 10} Return the square or cube of ***x***, respectively.
x^3	EXP x^3	
XEQ	XEQ ***labl***	(0) Executes the function or routine with the label specified. – In *PEM*, XEQ inserts a call to the subroutine with the label specified.
$\bar{x}_G$	STAT $\bar{x}_G$	(−2) { } → {2} Calculates the *geometric means* of the ***y***- and ***x***-data accumulated and and pushes them on the *stack*. See pp. 222ff for the formula. Output format will be similar to the one of x̄. See also ε, ε$_m$, and ε$_P$.
$\bar{x}_H$	STAT ▲ $\bar{x}_H$	(−2) { } → {2} Calculates the *harmonic means* of the ***y***- and ***x***-data accumulated and pushes them on the *stack*.
xIN	XEQ ***type***	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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x_{max} / x_{min}	STAT ▲ x_{max} etc.	(−2) { } → {2} Returns the maximum (or minimum) of the ***y***- and ***x***-data accumulated and pushes them on the *stack*. **ATTENTION**: These extrema will not be updated after Σ− (see there).
XNOR	BITS XNOR	(2) Work in analogy to AND. See p. 18.
XOR	BITS XOR	
xOUT	XEQ ***way***	Cleans and reverts the settings of xIN, taking care of a proper return including the correct setting of ***l*** and the *stack*. Typically, ***way*** = `xOUT NORMAL`. Generally, xOUT shall be the last command of an *XROM routine*.
$\bar{x}_{RMS}$	STAT ▲ $\bar{x}_{RMS}$	(−2) { } → {2} Calculates the *quadratic means* of the ***y***- and ***x***-data accumulated and pushes them on the *stack*. Quadratic means are often used in electrical engineering for alternating currents.
$\bar{x}_w$	STAT $\bar{x}_w$	(−1) { } → {2} Returns the *arithmetic mean* for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See pp. 222ff for the formula. See also s_w and s_{mw}.
$\sqrt[x]{y}$	EXP $\sqrt[x]{y}$	(2) Returns the x^{th} root of ***y***. For ***y*** < 0 and CPXRES set, it may return *complex numbers*. See the *DT* tables in *Section 2* of the *OM* for more.
X.FN	X.FN	*Menu*. See p. 120.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x!	x!	(1) {1, 10} Returns the *factorial* $n!$. Note this is only defined for positive integers. *20!* is the biggest factorial < 2^{64}. *450!* is maximum allowed for *long integers*.
		(1) {2, 3} Returns $\Gamma(x + 1)$. *2 123.549 956 662 463 236 31* is the maximum **x** for *real numbers*.
x:	U→ x:	*Submenu*. See p. 123.
x→DATE	CLK x→DATE	(1) {2} → {6} Interprets the *real number* **x** as a date coded in the date format selected (Y.MD, D.MY, or M.DY) and converts it to a proper *date*.
x→α	α.FN x→α	(1) {1, 2, 10} → {7} Interprets the integer part of **x** as a character code and converts it to the respective character **x**, similar to XTOA in the *HP-42S*.
x⇄	STK x⇄ r	Swaps **x** and **r**, in analogy to **x⇆y**. Will be listed like x⇄ J, x⇄ .12, x⇄ →12, etc. in programs.
x⇄y	x⇄y	Swaps the contents of *stack registers* **X** and **Y**.
x = ?	TEST x= ? r etc.	(0) Compare **x** with **r**. See x< ? for more.
x ≠ ?		
x ≈ ?	TEST ▲ x≈ ? r	(0) {2, 3, 4, 5, 8, 9} Will be true if the rounded values of **x** and **r** are equal (see ROUND). See x< ? for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x < ? x ≤ ? x ≥ ? x > ?	TEST x< ? r etc.	(0) {1, 2, 4, 5, 6, 7, 10} Compare *x* with *r*. *Text strings* are compared according to their sorting order. **Example:** TEST x< ? K compares *x* with *k*, and will be listed as x< ? K in a routine. It will return true if *x* < *k* at execution time. See examples in *Section 1* of the *OM* for more.
x = +0? x = -0?	TEST ▲ x=+0? etc.	(0) {1, 2, 10} These two tests are for comparing *short integers* in modes 1COMPL and SIGNMT, and for *real* or *complex numbers* if SPCRES is set. Then e.g. 0./(–7) will display –0.
ŷ	STAT ▲ ŷ	(1) {2}; {1} → {2} Returns a forecast *y* (in **X**) for a given *x* according to the curve fit model chosen. See p. 47 for more.
yd.→m	U→ x: yd.→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
YEAR	CLK YEAR	(1) {2, 6} → {1} Assumes *x* containing a *date* in the format selected (or a *real number* in corresponding format) and extracts the year.
year→s yin→m	U→ year→s U→ x: ▼ yin→m	(1) {2}; {1} → {2} Convert times and distances. See pp. 123ff.
y^x	y^x	(2) Raises *y* to the power of *x*. For *y* < 0 and CPXRES set, it may return *complex numbers*. See the *DT* tables in *Sect. 2* of the *OM* for more.
Y.MD	CLK ▲ Y.MD	(0) Sets the format yyyy-mm-dd for *dates*.
y⇆	STK y⇆ r	Swaps *y* and *r*, in analogy to *x*⇆.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
zhàn→m	U→ x: ▼ zhàng→m	(1) {2}; {1} → {2} Converts distances. See pp. 123ff.
z⇄	STK z⇄ *r*	Swaps ***z*** and ***r***, in analogy to ***x***⇆.
αINTL	CAT. CHARS αINTL	*Submenu*. See pp. 109ff.
	g +	*Menu* in *AIM* (see p. 120).
αLENG?	INFO αLENG? *r* α.FN αLENG? *r*	(−1) { } → {1} Returns the number of characters found in the *text string* ***r***, similar to ALENG in *HP-42S*. [23]
αMATH	CAT. CHARS αMATH	*Submenu*. See pp. 109ff.
	g −	*Menu* in *AIM*. See p. 121.
αPOS?	INFO αPOS? *r* α.FN αPOS? *r*	(−1) { } → {1} Looks in the *text string* ***r*** 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.[23] The ***target*** may be an individual character code or an *text string*. αPOS saves a copy of the ***target*** in **L**. It works similar to POSA in *HP-42S*.
αRL	α.FN αRL *r*	(0) Rotates the *text string* ***r*** by ***x*** characters like AROT in *HP-42S*, but with $x \geq 0$. αRL **0** executes as NOP, but loads **L**. [23]
αRR	α.FN αRR *r*	(0) Works like αRL but rotates to the right.
αSL	α.FN αSL *r*	(0) Shifts the ***x*** leftmost characters out of the *text string* ***r***, like ASHF in *HP-42S*. This allows for deleting the first ***x*** characters in the string. αSL **0** executes as NOP, but loads **L**. [23]

[23] 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)
αSR	α.FN αSR *r*	(0) Works like αSL but for the x rightmost characters out of r, deleting the last x characters in the string.[23]
α.FN	α.FN	*Menu*. See p. 121.
A…Ω	CAT. CHARS A…Ω	*Submenu* of Greek letters, see pp. 109ff.
α•	CAT. CHARS α•	*Submenu*. See pp. 109ff.
	g .	*Menu* in *AIM*. See p. 121.
α→x	α.FN α→x *r*	(−1) { } → {10_{16}} Pushes the character code of the leftmost character in the *text string* r on the *stack* and removes this character from the string, similar to ATOX in *HP-42S*.[23]
β(x,y)	X.FN ▲ β(x,y)	(2) {2, 3}; {1} → {2} Returns *Euler's Beta function* $\mathrm{B}(x,y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}$ with $Re(x) > 0$ and $Re(y) > 0$. Called β here to avoid ambiguity. See $\Gamma(x)$ below.
Γ_{xy}	X.FN ▲ Γ_{xy}	(2) {2}; {1} → {2} Returns the *upper / lower incomplete Gamma function*. See pp. 247ff for more.
γ_{xy}	X.FN ▲ γ_{xy}	
Γ(x)	PROB ▲ Γ(x)	(1) {2, 3}; {1} → {2} Returns $\Gamma(x)$. Note x! calls $\Gamma(x+1)$. See also LNΓ.
δx	CATALOG PROGS δx	Predefined global label for f'(x) and f"(x) – see *Section 4* of the *OM*.
Δ%	Δ%	(1) {2}; {1} → {2} Returns $100\,\frac{x-y}{y}$ leaving y unchanged. Use it also for calculating markups or margins as explained in the *OM*, *Sect. 2*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ε	f STAT ε	(−2) { } → {2} Calculates the *scattering factors* ε_y and ε_x for *log-normally* distributed sample data and pushes them on the *stack*. ε works for the *geometric mean* $\bar{x}_g$ in analogy to the *standard deviation* s for the *arithmetic mean* $\bar{x}$ but multiplicative instead of additive. See pp. 213ff for more information.
ε_m	STAT ε_m	(−2) { } → {2} Works like ε above but returns the *scattering factors* of the two *geometric means* (in analogy to the standard error for *arithmetic means*).
ε_p	STAT ε_p	(−2) { } → {2} Works like ε but returns the *scattering factors* of the two populations.
ζ(x)	X.FN ▲ ζ(x)	(1) {2, 3} Returns *Riemann's Zeta*. See p. 251 for more.
π	π	(−1) { } → {2} Recalls π.
Π_n	ADV Π_n ***labl***	Computes a product using the routine specified. See *Section 4* of the *OM* for more. **ATTENTION:** Π_n fills all *stack registers* with ***x*** before calling the routine specified.
Σ	Σ	*Menu*. See p. 122.
σ	STAT σ	(−2) { } → {2} Works like s but returns the *standard deviations* of the two *populations* instead. See pp. 213ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\Sigma\,{}^1/x$	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 *registers* of your *WP 43S* (see *App. B* on pp. 154ff.).
$\Sigma\,{}^1/x^2$		
$\Sigma\,{}^1/y$		
$\Sigma\,{}^1/y^2$		
Σln^2x	Σ Σln^2x 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. For space reasons, two sums are abbreviated: Σlnxy denotes $\sum ln(x) ln(y)$ and Σlny/x denotes $\sum \frac{ln(y)}{x}$.
Σln^2y		
Σlnx		
Σlnxy		
Σlny		
Σlny/x	Σ Σlny/x	
Σ_n	ADV Σ_n ***labl***	Computes a sum using the routine specified. See *Section 4* of the *OM* for more. **ATTENTION:** Σ fills all *stack registers* with ***x*** before calling the routine specified.
σ_w	STAT σ_w	(−1) { } → {2} Works like s_w but returns the *standard deviation* of the *population* instead. See pp. 213ff.
Σx	Σ Σx etc.	(−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see $\Sigma\,{}^1/x$ above for more).
Σx^2		
$\Sigma x^2 lny$	Σ $\Sigma x^2 lny$ etc.	
$\Sigma x^2 y$		
$\Sigma x^2/y$	Σ ▲ $\Sigma x^2/y$	
Σx^3	Σ ▲ Σx^3 etc.	
Σx^4		
Σxlny	Σ Σxlny	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Σxy	Σ Σxy	
Σx/y	Σ ▲ Σx/y	
Σy	Σ Σy etc.	
$Σy^2$		
Σylnx	Σ Σylnx	
Σ+ [24]	STAT Σ+	{8} → {2} If **X** contains an ***n*** × 2 matrix then Σ+ 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 **L**. See the *OM*, *Sect. 2*. {1, 2} Adds one 2D data point to the statistical sums. Note both **X** and **Y** must contain *DT* 1 or 2.
Σ− [24]	STAT Σ-	{1, 2} Works like Σ+ but subtracts one 2D data point. Updates neither x_{max} nor x_{min}.
$χ^2_p(x)$	PROB $χ^2$: $χ^2_p(x)$ etc.	(1) {2}; {1} → {2} *Chi-square distribution* (with its degrees of freedom given in **I**). $χ^2_▲(x)$ equals $Q(χ^2)$ on *HP-21S*. See *Section 2* of the *OM* for an application example and pp. 213ff for more.
$χ^2_◢(x)$		
$χ^2_◣(x)$		
$(χ^2)^{-1}$		
$χ^2$:	PROB $χ^2$:	*Submenu*. See p. 117.
$(-1)^x$	X.FN ▲ $(-1)^x$	(1) {1, 2, 3, 8*, 9*, 10} If x is non-integer, returns $cos(\pi x)$.

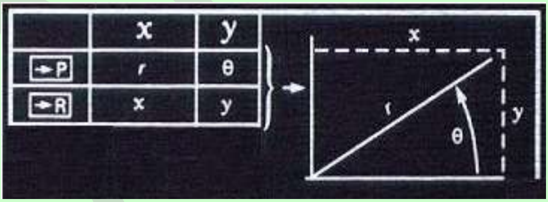
[24] Σ+ and Σ− return *temporary information* as shown in *Section 2* of the *OM* and disable *ASL*. Both commands may also be used for 2D vector adding and subtracting (see the command SUM and the corresponding example in *Section 2* of the *OM*)..

Item	Keystrokes	Remarks (see pp. 12ff for general information)
[M]^T	MATX [M]^T	(1) {8, 9} Returns the transpose of the matrix ***x***. The transpose is another matrix with rows changed by columns. If ***A*** is an ***n***×***m*** matrix and a_{ij} is an element of it then A^T will be an ***m***×***n*** matrix ***B*** with $b_{ij} = a_{ji}$. The transpose is done in-situ and does not require any additional memory.
[M]$^{-1}$	MATX [M]$^{-1}$	(0) {8, 9} Takes the square matrix in **X** and inverts it in-situ.
+	+	(2) Returns $y + x$ for compatible objects. [25]
+/-	+/- (for closed x)	(1) 'Unary minus', returns $x \times (-1)$.[25]
±∞?	INFO g ±∞?	(0) {2} → {1} Tests ***x*** for infinity. Returns true 1 for $x = +\infty$, true -1 for $x = -\infty$, and false 0 else.
–	–	(2) Returns $y - x$ for compatible numeric objects. [25]
×	×	(2) Returns $y \times x$ for compatible numeric objects.[25]
×MOD	INTS ×MOD	(3) {1, 2, 10} Returns $(z \times y) \bmod x$ for $x > 1, y > 0, z > 0$. [26]
/	/	(2) Like × , but returns $y \times x^{-1}$.[25] Returns $y \operatorname{div} x$ if both y and x are of *DT* 1 or 10; cf. IDIV.
^MOD	INTS ^MOD	(3) {1, 10} Returns $(z^y) \bmod x$ for $x > 1, y > 0, z > 0$. [26]

[25] See the combination tables in the *OM, Section 2*, for details and compatibility.

[26] See MOD.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→		Reserved symbol for indirect addressing.
→DATE	CLK →DATE	(3) {1, 2} → {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 *date* in ***x***. Thus inverts DATE→.
→DEG	L→ →DEG	(1) {1, 2, 4} → {4} Convert angles as described on pp. 132f.
→D.MS	L→ →D.MS	
→GRAD	L→ →GRAD	
→HR	CATALOG FCNS →HR	(1) {5} → {2} Operates on *times* like →REAL below. Maintained for backward compatibility only.
→H.MS	h.ms (for closed *x*)	(1) {1, 2, 5} → {5} Converts ***x*** to a sexagesimal *time* – cf. p. 104.
→INT	# ***base*** (for closed *x*)	(1) {1, 2, 10} → {10} Converts the integer part of ***x*** to a *short integer* of the base specified. Conversion to decimal may be abbreviated by # D, to hexadecimal by # H. Note # I converts to a *long integer*. Cf. p. 104.
→MULπ	L→ →MULπ	(1) {1, 2, 4} → {4} Converts angles as described on pp. 132f.
→POL	→P	{2}; {1} → {2} Assumes **X** and **Y** containing 2D *Cartesian* coordinates of a point or components of a vector (***x*** , ***y***). Converts them to the respective polar coordinates or components (***r***, ϑ). Inverted by →REC, see there. – For switching the display format of *complex numbers*, see POLAR.
→RAD	L→ →RAD	(1) {1, 2, 4} → {4} Converts angles as described on pp. 132f

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→REAL	.d (for closed *x*)	(1) {1, 2, 4, 5, 6, 10} → {2} Converts ***x*** to a *real number*. Any object (e.g. a *time*) tagged sexagesimal will be converted in a decimal number. For *dates*, the date format chosen is taken into account. Numbers shown as fractions will be displayed as decimal numbers (cf. FRACT and PROPFR). To return the *real* part of a *complex number*, take RE. To cut a *complex number* into its parts, use CC or CX→RE.
→REC	R←	{2}; {1, 4} → {2} 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 display format of *complex numbers*, see POLAR. 
⇅	STK ⇅_ _ _ _	Shuffles the contents of the *stack registers* **X**, **Y**, **Z**, and **T** at execution time. **Examples:** ⇅xxyz works like ENTER↑ (but enables *ASL*!), ⇅yxzt works like x⇆y, ⇅yztx works like R↓ in a 4-level *stack*, ⇅txyz works like R↑ in a 4-level *stack*, but also ⇅yytt or ⇅zzzx is possible. **ATTENTION:** This is a very powerful command although it does not look it. Note it will affect the bottom four *stack registers* only; there is no connection to **A** … **D**, regardless of *stack* size. Playing with ⇅, you may lose some *stack* contents and make a mess of the *stack* easily.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
|M|	MATX |M|	(1) {8} → {2}; {9} → {3} Requires a square matrix in **X** and returns its determinant. The original matrix is saved in **L**.
|x|	|x| or PARTS |x| or CPX |x|	(1) {1, 2, 4, 10} Returns the absolute (unsigned) value of x.
		(1) {3} → {2} Returns the *magnitude* $\sqrt{\mathrm{Re}(x)^2 + \mathrm{Im}(x)^2}$ in **X**.
		(1) {8*}; {9*} → {8} Returns a *real* matrix with the absolute values or *magnitudes* of all input matrix elements. Cf. ENORM.
||	X.FN ▲ ||	(2) {2, 3}; {1} → {2} Returns $\left(\frac{1}{x} + \frac{1}{y}\right)^{-1}$; useful in electrical engineering especially. Returns 0. for $x \times y = 0$.
%	FIN %	(1) {2}; {1} → {2} Returns $\frac{xy}{100}$, leaving y unchanged.
%MRR	FIN %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	FIN %T	(1) {2}; {1} → {2} Returns $100\ {}^{x}/_{y}$, interpreted as % of total. Leaves y unchanged.
%Σ	FIN %Σ	(1) {2}; {1} → {2} Returns $100\ {}^{x}/_{\sum x}$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
%+MG	FIN %+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.
√x	√x EXP √x	(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.
∫	ADV ∫fdx ∫ *var* (listed in programs as ∫fd 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 ↓Lim and ↑Lim, accuracy by ACC. ∫ returns the (approximated) integral in **X** and an upper limit of its uncertainty in **Y**.[27] **ATTENTION:** ∫ fills all *stack registers* with x before calling the routine specified in PGMINT.
	EQN ∫f ∫	Integrates the current equation.[27]
∫f	EQN ∫f	*Submenus*. See pp. 114f.
∫fdx	ADV ∫fdx	
∡	∡ or CPX ∡ or PARTS ∡	(1) {2} → {4} Returns 180° (or equivalent) for x < 0 and 0 else.
		(1) {3} → {4} Returns the *phase* or argument $arg(x) = arctan\left(\frac{Im(x)}{Re(x)}\right)$. Cf. |x|.
		(1) {9*} → {8} Returns a matrix with the *phases* of all input matrix elements. Cf. |x|.

[27] See *Section 4* of the *OM* for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
∡→	L→	*Menu* of angular conversions. See p. 122.
🖶ADV	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	PRINT 🖶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	PRINT 🖶DLAY *n*	(0) Sets a delay of ***n*** ticks (see TICKS) to be used with each linefeed on the printer.
🖶LCD	PRINT 🖶LCD	(0) Sends the contents of the entire *LCD* to the printer, so you get a hardcopy of the screen.
🖶MODE	PRINT 🖶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	PRINT 🖶PROG	(0) Prints the listing of the *current program* (see *Section 3* of the *OM*), one row per step.
🖶r	PRINT 🖶r *r*	(0) Prints the content of the *register* 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	PRINT 🖶REGS	(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. **ATTENTION for *nn* = 0 :** • For $sss \in [0; 99]$, printing will stop at **R99**. • For $sss \in [100; 111]$, printing stops at **K**. For $sss \geq 112$, printing stops at the highest allocated local *register*.
🖶STK	PRINT 🖶STK	(0) Prints the entire *stack* contents. Each of the 4 or 8 *registers* prints in a separate row starting with its label.
🖶TAB	PRINT 🖶TAB ***n***	(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	PRINT 🖶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	PRINT 🖶WIDTH	(−1) Returns the number of print columns that x would take in the print mode set. See 🖶MODE. 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.
🖶x	PRINT 🖶x	(0) Prints x without a heading label. See 🖶ADV.
🖶Σ	PRINT 🖶Σ	(0) Prints the summation *registers*. Each *register* prints in one row starting with its label.
🖶#	PRINT 🖶# ***n***	(0) Sends a single *byte*, without translation, to the printer (e.g. a control code). ***n*** > 127 can only be specified indirectly. 🖶MODE setting will not be honored. See 🖶ADV.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
#B	BITS #B	(1) {10} Counts the bits set in **x** (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 *items* listed above. As required for them, also the *names* of variables and *system flags* 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* with their remarks 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 *register* **A**
ACC	ADV ∫fdx ACC	Reserved *real* variable for the accuracy of integration (see *Section 4* of the *OM*).
ADM		Reserved integer variable for the *ADM*: 0: DEG, 1: D.MS, 2: RAD, 3: MULπ, 4: GRAD.
ALLENG	A	
ALPHA		*System flags – see next chapter.*
ALP.IN		

Name	Keystrokes	Remarks (see pp. 12ff for general information)
ASLIFT		
AUTOFF		*System flags – see next chapter.*
AUTXEQ		
B		Reserved variables for *registers* **B** and **C**.
C		
CARRY	C	
CPXj		*System flags* – see next chapter.
CPXRES	I (imaginary)	
D		Reserved variable for *register* **D**.
DECIM.		*System flags* – see next chapter.
DENANY		
DENFIX		
DENMAX		Reserved integer variable for the maximum denominator, filled by the command DENMAX.
DMY		*System flags* – see next chapter.
FRACT		
FV	FIN TVM FV	Reserved real variable for the future value of your investment or loan in *TVM*.[28]
GRAMOD		Reserved integer variable determining how the AGRAPH image is displayed: [29] 0: It is merged (OR-ed) with the existing display. 1: It overwrites all pixels in that part of the screen 2: Duplicate "on" pixels get turned "off". 3: It is XOR-ed with the existing display.

[28] See *Section 5* of the *OM.*

[29] Working like *flags* 34 and 35 in *HP-42S.*

Name	Keystrokes	Remarks (see pp. 12ff for general information)
GROW		*System flag* – see next chapter.
I		Reserved variable for *register* **I**.
IGN1ER		*System flags – see next chapter.*
INTING		
ISM		Reserved integer variable for the *integer sign mode*: −1: sign and mantissa mode, 0: unsigned, 1: 1's complement, 2: 2's complement.
i%/a	FIN TVM i%/a	Reserved real variable for the annual interest rate of your investment or loan in *TVM*. [28]
J		
K		Reserved variables for *registers* **J**, **K**, and **L**.
L		
LEAD.0	L	*System flags – see next chapter.*
LOWBAT		
Mat_A	MATX SIM EQ Mat A	
Mat_B	MATX SIM EQ Mat B	Reserved variables for the matrices for solving systems of linear equations (see *Section 2* of the *OM*).
Mat_X	MATX SIM EQ Mat X	
MDY		*System flag* – see next chapter.
MULT×		*System flag* – see next chapter.
NPER	FIN TVM nPER	Reserved variable for the total number of • payment periods for your loan or • compounding periods for your investment.
NUM.IN		*System flags – see next chapter.*
OVERFL	B (big)	

Name	Keystrokes	Remarks (see pp. 12ff for general information)
PER/a	FIN TVM per/a	Reserved variable for the annual number of • payments for your loan or • compounding periods of your investment.
PMT	FIN TVM PMT	Reserved variable for the payment per period for your investment or loan in *TVM*. [28]
POLAR	X	
PRINT		*System flags* – see next chapter.
PROPFR		
PRTACT		
PV	FIN TVM PV	Reserved variable for the present value of your investment or loan in *TVM*. [28]
QUIET		*System flag* – see next chapter.
REALDF		Reserved integer variable for *real number* display format: 0: ALL, 1: FIX, 2: SCI, 3: ENG.
RUNIO		
RUNTIM		
SLOW		*System flags – see next chapter.*
SOLVING		
SPCRES	D (danger)	
SSIZE8		
STATS		
T		
X		Reserved variables for the statistical data matrix (see *Section 2* of the *OM*) and the *stack registers* **T** … **Z**
Y		
Z		

Name	Keystrokes	Remarks (see pp. 12ff for general information)
TDM24		*System flags* – see next chapter.
TRACE	T	
USB		
USER	USER	*System flag* toggled by USER – see next chapter
VMDISP		System flags – see next chapter.
YMD		
αCAP		
↑Lim	ADV ∫fdx ↑Lim etc.	Reserved *real* variables for the upper and lower limit of integration (s. the *OM, Sect. 4*).
↓Lim		
#DEC		Reserved integer variable for the number of decimals in *real number* formatting (actually the parameter specified in last call of 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 their order in the *status bar*. 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.

Purpose	*SBI*	*Flag*[30]	Remarks
Time display	Time string	TDM24 (set)	Set for international 24h time display. Shows four digits for the year in the date display in the *status bar*. 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 *status bar* 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 *flag* and clear the two others.
Complex results	ℂ / ℝ	CPXRES	Set for allowing *complex* results also for *real* input (e.g. $\sqrt{-1}$). Else an error will be thrown in such cases.
Complex letter	—	CPXj	Set for the letter $\boldsymbol{j}$ representing the *imaginary* number $\boldsymbol{i}$, clear for $\boldsymbol{i}$.
Polar notation	∟ / ⊙	POLAR	Set for polar display of *complex numbers*, clear for rectangular.
Fraction display	—	FRACT	Set for fraction display, clear for decimal display (a b/c sets FRACT, .d clears it). See next entries.
Fraction kind 1	—	PROPFR (set)	Set for *proper fractions*, clear for *improper fractions* (a b/c toggles PROPFR: coming from decimal display, it sets it). PROPFR set allows only *proper fractions* in display (e.g. 1 $^2/_3$ instead of $^5/_3$); any *reals* (with $\lvert x \rvert < 10^6$) will be displayed according to the settings of DENANY, DENFIX, and DENMAX as *proper fractions*.

[30] Grey, orange, and light blue colors are used as in previous chapter.

Purpose	*SBI*	*Flag*[30]	Remarks
Fraction kind 2	see the *OM*	DENANY (set)	Set if <u>any</u> denominator up to DENMAX may appear (DENMAX is indicated in the *status bar*).[31] 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[32]	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.[33]
Carry	c or o/c	CARRY	Reflects the status of the carry bit.
Overflow	o or o/c	OVERFL	Reflects the status of the overflow bit.
Leading zeros	—	LEAD.0	Set for leading zeros turned on in *short integers* of bases 2, 4, 8, and 16. Like *flag* 3 of *HP-16C.*
Alpha	A or α	ALPHA	Set for *AIM*, else clear.
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	U	USER	Set if your *WP 43S* is in user mode.
USB power	USB	USB	Set if connected to a USB power source

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

[32] DENFIX is evaluated only if DENANY is clear.

[33] 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).

Purpose	*SBI*	*Flag*[30]	Remarks
Low battery		LOWBAT	Set if battery voltage is low (< 2.5 V, 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. LOWBAT). 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 8, clear for 4 *stack registers*. Note *register* 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	—	MULT× (set)	Set for multiplication symbol ×, clear for ·.
Overflow from ALL	—	ALLENG	Set if *real numbers* 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). `grow` or `wrap` will appear in the status bar as long as a matrix is open for editing. See J+ in the *IOI* and *Section 2* of the *OM*; cf. also GROW in the *HP-42S OM*, 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 *Sect. 2* of the *OM*). Works like *flag* 44 of *HP-42S*
Automatic execution	—	AUTXEQ	Like *flag* 11 of *HP-42S.*

Purpose	*SBI*	*Flag*[30]	Remarks
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 *Integrator* is running.
Solving	—	SOLVING	Set while the *Solver* is computing a root.
Variable menu	—	VMDISP	Set while the variable menu is displayed.

Nonprogrammable Commands and Keys

The commands printed on violet or black in the *IOI* 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.

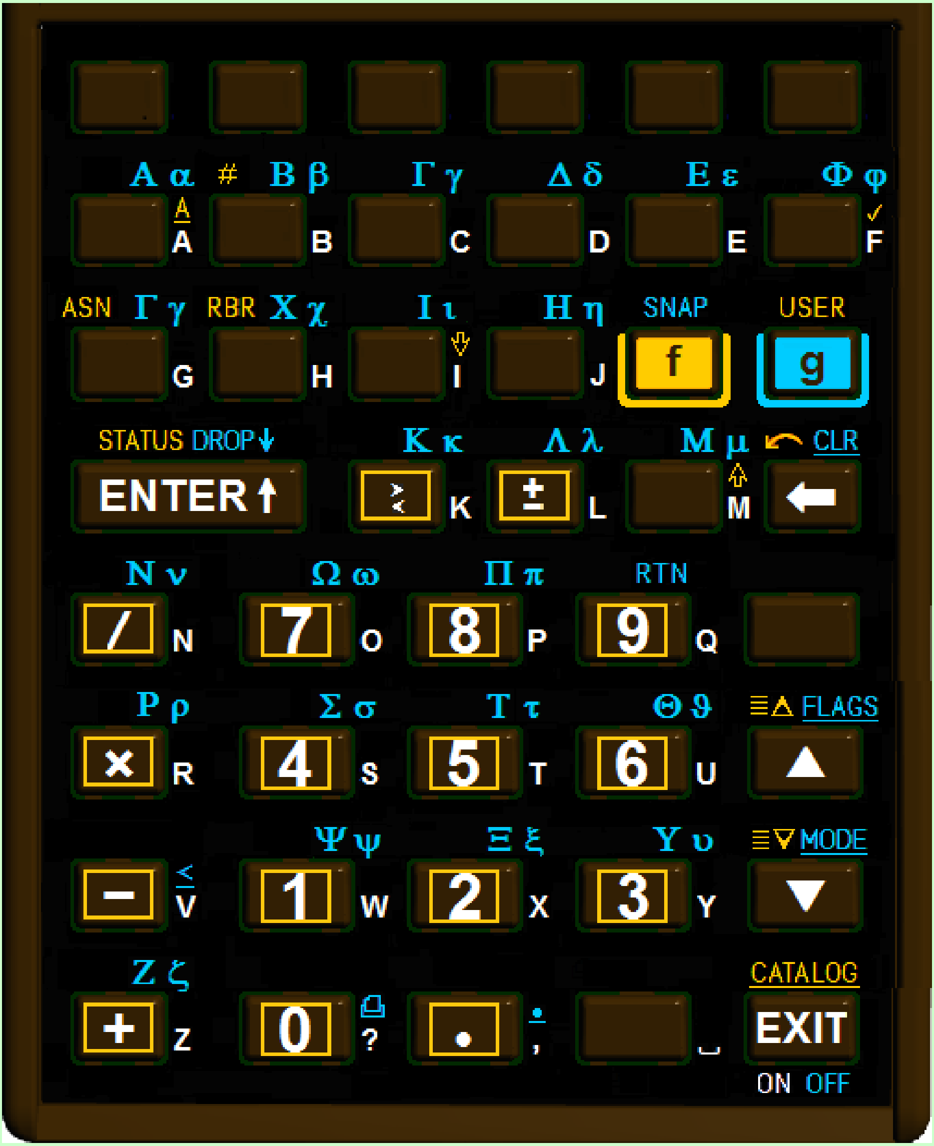
The *browsers* RBR and STATUS as well as the *application* TIMER use some keys for particular control purposes (e.g. STO, RCL, ., and numeric keys – cf. the *OM, Section 5*). Also RBR, STATUS, as well as the applications TIMER and TVM cannot be programmed.

All *catalog* and *menu* calls themselves and the operations called by EXIT, P/R, α, ↶, ≡Δ, ▲, ≡∇, and ▼ 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. 108ff) for more about this topic.

Command Parameter Input and Closing It

The following table shows what will happen when particular keys are pressed <u>while command parameter input is not finished yet</u> (see pp. 104ff 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 *global GP register* or *user flag*.
[A] … [Z] [g] [A] … [g] [O] [f] [0] … [f] [9]	entering a label, a *system flag* or variable *name*	Appends the corresponding Latin or Greek letter or digit to the label, *system flag* or variable *name* pending. Use [▼] and [▲] to switch cases for letters. See the virtual keyboard on p. 103 (cf. the *OM*, *Section 2*).
[ENTER↑]	arbitrary parameter input pending	If there is no input yet, assumes the default, if applicable. Else closes pending input, interprets it as a *register* or *user flag* address, a *system flag* or variable *name*, a label, or alike, and executes the command (cf. the *OM*, *Section 1*).
[←]	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 *local registers* or *local user flags*.
[EXIT]	arbitrary parameter input pending	If there is an open *menu*, 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. 112f). 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 p. 102 for command parameter input instead). The table lists the respective keys on the keyboard top left to bottom right:

Keystrokes	in mode(s)	Meaning
[A] … [Z] [g][A] … [g][O]	A, α	Appends the corresponding Latin or Greek letter to the *text string* ***x***. Use [▼] and [▲] to switch cases. See previous page and cf. *Section 2* of the *OM*.
[#]	A, α	Appends # to the *text string* ***x***.
[#] ***base***	¬ (A, α)	Closes input of a *short integer*
[d.ms]	¬ (A, α)	sexagesimal *angle*
[.d]	¬ (A, α)	*date*
[h.ms]	¬ (A, α)	sexagesimal *time* in **X**.[34]
[f][x^2] (✓)	A, α	Appends a checkmark ✓ to the *text string* ***x***.
[CC]	¬ (A, α)	Closes input of the first part (i.e. *real* part or *magnitude*) of a *complex number* in **X** and waits for input of its second part (i.e. *imaginary* part or *phase*, see the *OM, Section 2* and the *Key Response Table*). Any additional [CC] in input will be ignored.
[f][R↓] (⇩)	A, α	Makes the next character a subscript, if applicable.

[34] After [#], any additional [#] in input will be ignored. Sexagesimal *angles* shall be entered like `d.mmssff` (more than 1 digit is allowed for *degrees*). Input of *times* shall be like `h.mmssf` (more than 1 digit is allowed for *hours* and fractions of *seconds*). See *Section 2* of the *OM* for examples and *App. B* for numeric limitations.

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: • If this input is alphanumeric (i.e. if it contains at least one non-numeric character except .), takes it as a *text string*. • Else (i.e. if this input is purely numeric) if it contains one CC, takes it as a *complex number*. • Else if it contains two ., takes it as a *fraction*. • Else if it contains one . or one E, takes it as a *real number*. • Else (i.e. if it contains neither a CC nor a . nor an E), tests it for #: ○ If it contains one # and a valid base trailing it then takes it as a *short integer* of said base; ○ else looks up if previous entry was a *short integer*: if true then takes the new input as another *short integer* of the same base; else takes the new input as a *long integer*. Then checks the new input (according to the condition met) as outlined in footnote 34 and interprets it. Finally, unless an error had to be thrown, copies ***x*** into **Y**.
f x≷y	A, α	Appends ≷ to the *text string* ***x***.
+/–	¬ (A, α)	Changes the sign of the number entered. Affects the exponent if pressed after E.
f +/–	A, α	Appends ± to the *text string* ***x***.
E	¬ (A, α)	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.
f /	A, α	Appends / to the *text string* **x**.
f ×	A, α	If MULT× is set then appends ×, else · to the *string* **x**.
f −, f +	A, α	Appends the respective sign the *text string* **x**.
INTS A … F BITS ▲ A … F	¬ (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 **x** is closed (see the description of ENTER↑ above).
0 … 9	¬ (A, α)	Standard numeric input, appending the corresponding digit to **x**. Note you can enter … • up to 34 digits plus a sign in the mantissa[35] and up to four digits plus a sign in the exponent for a *real number* or any part of a *complex number*, • an arbitrary number of digits plus a sign for a *long integer*, • up to 64 bits for a *short integer*, or • up to 16 digits for the nominator and up to 4 digits for the denominator of a fraction.
0 (?)	A, α	Appends ? to the *text string* **x**.
f 0 … f 9	A, α	Appends the respective digit to the *text string* **x**.
g 0 (🖶)	A, α	Appends 🖶 to the *text string* **x**.
▲	α	Turns to upper case for the letters following.
▼	A	Turns to lower case for the letters following.

[35] You can enter even more digits but it makes no sense.

Keystrokes	in mode(s)	Meaning
[.]	¬ (A, α)	For sexagesimal angular input, separates *degrees* from *minutes*, *seconds*, and *hundredths of seconds*, so input format is ddddd.mmsshh [d.ms] (cf. p. 104 and *Section 2* of the *OM*). For *time* input, separates *hours* from *minutes*, *seconds*, and fractions of *seconds*, so input format is hhhhh.mmssfffff [h.ms] (cf. p. 104 and *Section 2* of the *OM*). Else inserts a radix mark as selected.
Second [.]	¬ (A, α)	A 2nd [.] in input signals a fraction. See the *OM*, *Section 2* for examples. The 2nd [.] 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 2nd dot while editing the input.
[.]	A, α	Appends , to the *text string* **x**.
[f] [.]	A, α	Appends . to the *text string* **x**.
[R/S]	, program waiting for input	Closes pending input and starts its checks and interpretation like [ENTER↑] above. Resumes program execution.
	A, α	Appends a blank space to the *text string* **x**.
[EXIT]	arbitrary input pending	If there is an open *menu*, closes it. Else closes pending input and starts its checks and interpretation like [ENTER↑] above.

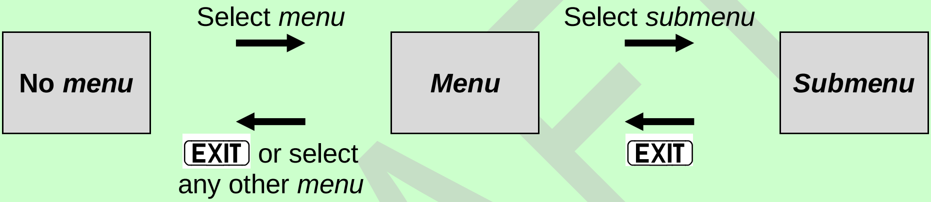
There are many more characters you can enter via the three alpha *menus* or [CATALOG] CHARS (see pp. 109 and 120ff). Call FBR for browsing the entire character sets provided.

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 15 *catalogs*:

- CATALOG'CHARS'αINTL ,
- CATALOG'FCNS (by far the largest *catalog* at startup),
- CATALOG'MENUS ,
- the two *submenus* of CATALOG'PROGS
- 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. 112f).

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 ***cataloged***. Individual *cataloged items* may be accessed quickly in a way demonstrated on pp. 112f.

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:

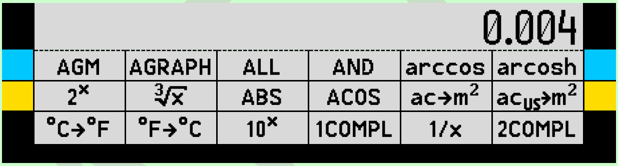
	▭	▭	▭	▭	▭	▭	Remarks
CATALOG:	FCNS		CHARS	PROGS	VARS	MENUS	top branches
FCNS:	°C→°F	°F→°C	10^x	1COMPL	$^1/_x$	2^x	catalog of all the some 740 functions provided
	2COMPL	$\sqrt[3]{x}$	ABS	ACOS	ac→m^2	ac$_{US}$→m^2	
	AGM	AGRAPH	ALL	AND	arccos	arcosh	
	…						
	…	…	#B				
CHARS:	αINTL	A…Ω		αMATH	Myα	α•	character branches
αINTL:	A	À	Á	Â	Ã	…	catalog of all the 99 international Latin letters provided, see p. 120
	…						
	…	…	Ž				
αMATH:	<	≤	=	≈	…		mathematical operators and symbols, see p. 121
A…Ω:	Α	Β	Γ	…			Greek letters, see p. 121
α•:	!	;	…				punctuation marks, see p. 122
PROGS:	RAM					FLASH	global labels currently defined
RAM:	…						both branches are empty at startup; they will be filled with your creations
FLASH:	…						

							Remarks
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. All 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:	Binom:	BITS	Cauch:	*menus* & *submenus* currently defined (shown here at startup, but also this *submenu* will grow with your creations) – see above and below for fix menu contents
	CHARS	CLK	CLR	CONST	CPX	CPXS	
	DATES	DISP	EQN	EXP	Expon:	EXPT	
	...						
	...	∡→					
A:	...						*(sub-) menus* provided unless mentioned above already, see pp. 114ff for predefined contents – here your creations will be inserted as new entries
Binom:	...						
BITS:	...						
...							
...							

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 and CHARS) since all functions 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, 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.[36] EXIT will just leave the open branch without doing anything.

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

[36] 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.

items provided (*system flags* and variable *names*) on pp. 93ff.

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 ▲ and ▼ as explained in previous chapter. In CONST and major parts of CATALOG (FCNS, MENUS, CHARS αINTL, 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	CATALOG FCNS
	Return	**Your *WP 43S* displays the first *view* in this *catalog*;[37] *AIM* is on.**

AGM	AGRAPH	ALL	AND	arccos	arcosh
2^x	$\sqrt[3]{x}$	ABS	ACOS	ac→m²	ac_{US}→m²
°C→°F	°F→°C	10^x	1COMPL	1/x	2COMPL

2	User input	First character of the *item* desired (e.g. F)
	Return	**Your *WP 43S* displays a *view* starting with the first *item* starting with this character[38]**

---

[37] … 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.

[38] 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

	e.g.

FLOOR	fm.→m	FP	$F_p(x)$	FP?	fr→dB
fēn→m	FF	FIB	FILL	FIX	FLASH?
FB	FBR	FC?	FC?C	FC?F	FC?S

3 User input	Second character of the *item* desired (e.g. S)
Return	**Your *WP 43S* displays a *view* starting with the first *item* starting with the string you specified** e.g.

f''(x)	GAP	GaussF	GCD	g_d	g_d^{-1}
fz_{UK}→m^3	fz_{US}→m^3	$F_{\triangle}(x)$	$F_{\triangle}(x)$	$F^{-1}(p)$	f'(x)
FS?	FS?C	FS?F	FS?S	ft.→m	ft_{US}→m

4 User input	Press the corresponding *softkey* e.g. for FS?S
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

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.

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 αINTL (or in *AIM* by f A) + 3 keystrokes maximum.

Further Menus and Their Contents

In the table starting 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	∫fdx	advanced operations, see *Sect. 4* of the *OM*
	PGMSLV		f"(x)			PGMINT	
∫fdx			ACC	↓Lim	↑Lim	∫	
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	
	A	B	C	D	E	F	
	SL	RL	RLC	RRC	RR	SR	
	LJ					RJ	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	

Menu	▭	▭	▭	▭	▭	▭	Remarks
CLK	DATE	→DATE	DATE→	WDAY	TIME	x→DATE	date and time functions (first view) and settings (second view)
	J→D	D→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. 133ff
CPX	dot	cross	UNITV	Re	conj	Re⇄Im	special *complex* functions
	CX→RE	RE→CX	sign	Im	|x|	∡	
DISP	FIX	SCI	ENG	ALL	ROUNDI	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	
EQN	NEW	EDIT	f''	f'	∫f	Solver	equations (see the *OM, Sect. 4*)
	DELETE						
Solver							show the *names* of all variables of the current equation and more
∫f							
f'							
f''							
EQ.EDI	←	( )	^	:	=	→	Equation Editor
EXP	x^3	$\sqrt[x]{y}$	$\log_x y$	lb x	2^x	$\sqrt{x}$	exponential, logarithmic, and hyperbolic functions
	$\sqrt[3]{x}$			ln 1+x	e^x-1		
	sinh	arsinh	cosh	arcosh	tanh	artanh	
FIN	%	%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	

Menu							Remarks
FLAGS	SF	FS?	FF	STATUS	FC?	CF	
	FS?S	FS?C	FS?F	FC?F	FC?S	FC?C	
						CLFall	
INFO	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?	
INTS	A	B	C	D	E	F	digits for *short integers* with bases > 10, integer operations
	IDIV	RMD	MOD	×MOD	FLOOR	LCM	
	DBL /	DBLR	DBL ×	^MOD	CEIL	GCD	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
I/O	LOAD	LOADP	LOADR	LOADSS	LOADV	LOADΣ	data exchange and signalling
	BEEP	TONE			RECV	SEND	
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⇄R	
	EIGVAL					EIGVEC	
M.EDIT	←	↑	OLD	GOTO	↓	→	Matrix Editor as in *HP-42S*
	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						

Menu							Remarks
MyMenu							will pop up out of *AIM* [39]
Myα							will pop up in *AIM* [39]
PARTS	IP	FP	MANT	EXPT	sign	DECOMP	some overlaps with *HP-42S* CONVERT
					|x|	∡	
					Re	Im	
PRINT	⎙x	⎙r	⎙Σ	⎙ADV	⎙LCD	⎙PROG	similar to the PRINT commands of the *HP-42S*
	⎙STK	⎙REGS	⎙USER	⎙TAB	⎙#	⎙CHAR	
				⎙WIDTH	⎙DLAY	⎙MODE	
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◢		Binom^{-1}	
Cauch:	Cauch_p		Cauch◣	Cauch◢		Cauch^{-1}	
Expon:	Expon_p		Expon◣	Expon◢		Expon^{-1}	
F:	$F_p(x)$		F◣(x)	F◢(x)		$F^{-1}(p)$	
Geom:	Geom_p		Geom◣	Geom◢		Geom^{-1}	
Hyper:	Hyper_p		Hyper◣	Hyper◢		Hyper^{-1}	
LgNrm:	LgNrm_p		LgNrm◣	LgNrm◢		LgNrm^{-1}	
Logis:	Logis_p		Logis◣	Logis◢		Logis^{-1}	
NBin:	NBin_p		NBin◣	NBin◢		NBin^{-1}	
Norml:	Norml_p		Norml◣	Norml◢		Norml^{-1}	
Poiss:	Poiss_p		Poiss◣	Poiss◢		Poiss^{-1}	
t:	$t_p(x)$		t◣(x)	t◢(x)		$t^{-1}(p)$	
Weibl:	Weibl_p		Weibl◣	Weibl◢		Weibl^{-1}	
χ^2:	$\chi^2_p(x)$		χ^2◣(x)	χ^2◢(x)		$(\chi^2)^{-1}$	

[39] ... as long as no other *menu* is called (see *Section 6* of the *OM*).

Menu							Remarks
P.FN	INPUT	END	ERR	TICKS	PAUSE	P.FN2	additional programming functions (avoided a *multi-view menu* here).
	PSTO	PRCL	VARMNU	MVAR	CNST	PUTK	
	R-CLR	R-COPY	R-SORT	R-SWAP	LocR	PopLR	
P.FN2	MENU	KEYG	KEYX	CLMENU	EXITall	RTN+1	
	SDL	SDR	MSG	NOP			
	BACK	CASE	SKIP	AGRAPH	PIXEL	POINT	
STAT	Σ+	$\bar{x}$	s	σ	s_m	SUM	for sample statistics.
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}		
	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m		
	L.R.	r	s_{xy}	cov	$\hat{x}$	$\hat{y}$	for curve fitting and 2d sample statistics.
	s(a)	$\bar{x}_H$					
	ASSESS	$\bar{x}_{RMS}$	x_{max}	x_{min}		PLOT	
	LinF	ExpF	LogF	PowerF		OrthoF	for choosing the curve fit model(s)
	ParabF	HypF	RootF				
	GaussF	CauchF		BestF			
STK	x⇄	y⇄	z⇄	t⇄	⇄	DROPy	*stack* related operations.
TEST	x< ?	x≤ ?	x= ?	x≠ ?	x≥ ?	x> ?	binary tests.
	INT?	EVEN?	ODD?	PRIME?	LEAP?	FP?	
	ENTRY?	KEY?	LBL?	STRI?	CONVG?	TOP?	
	x=+0?	x=-0?	x≈ ?	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	

Menu							Remarks
U→	E:	P:	year→s	F&p:	m:	x:	unit conversions (see pp. 123ff).
	°C→°F	°F→°C	s→year		V:	A:	
	power ratio → dB	dB → power ratio	Nm → lbf·ft	lbf·ft → Nm	field ratio → dB	dB → field ratio	
A:	acre → ha	ha → acre	ha→m^2	m^2→ha	acre$_{US}$ → ha	ha → acre$_{US}$	units of area
	mŭ→m^2	m^2→mŭ					
E:	cal→J	J→cal	Btu→J	J→Btu	Wh→J	J→Wh	units of energy
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→g	g→oz	units of mass
	stone → kg	kg → stone	short cwt→kg	kg → sh.cwt	tr.oz → g	g → tr.oz	
	ton→kg	kg→ton	short ton → kg	kg → short ton	carat → g	g → carat	
	liăng → kg	kg → liăng			jīn→kg	kg→jīn	
P:	hp$_E$→W	W→hp$_E$	hp$_{UK}$→W	W→hp$_{UK}$	hp$_M$→W	W→hp$_M$	units of power
V:	gal$_{UK}$→l	l→gal$_{UK}$	qt.→l	l→qt.	gal$_{US}$→l	l→gal$_{US}$	units of volume
	floz$_{UK}$ → ml	ml → floz$_{UK}$	barrel → m^3	m^3 → barrel	floz$_{US}$ → ml	ml → floz$_{US}$	
x:	au→m	m→au	l.y.→m	m→l.y.	pc→m	m→pc	units of length
	mi.→km	km→mi.	nmi→km	km→nmi	ft.→m	m→ft.	
	in.→mm	mm→in.			yd.→m	m→yd.	
	lĭ→m	m→lĭ	yĭn→m	m→yĭn	zhàng → m	m → zhàng	
	chĭ→m	m→chĭ	cùn→m	m→cùn	fēn→m	m→fēn	
	fathom → m	m → fathom	point → mm	mm → point	survey foot$_{US}$ → m	m → survey foot$_{US}$	

Menu							Remarks
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}	$I\Gamma_p$	$I\Gamma_q$	
	$J_y(x)$	lnβ	lnΓ	max	min	NEXTP	
	W_m	W_p	W^{-1}	β(x,y)	γ_{xy}	Γ_{xy}	
	ζ(x)	$(-1)^x$				||	
Orthog	H_n	L_m	$L_{m\alpha}$	P_n	T_n	U_n	orthogonal polynomials
	H_{np}						
αINTL	A a	À à	Á á	Â â	Ã ã	Ä ä	[α] *catalog* of all Latin letters provided.[40] 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 ă	Ą ą	B b	You can reach each of these 99 letters by 3 keystrokes maximum since it can be accessed alphabetically (cf. pp. 112f).
	C c	Ç ç	Ć ć	Č č	D d	Ð ð	
	Ď ď	Đ đ	E e	È è	É é	Ê ê	
	Ë ë	Ē ē	Ĕ ĕ	Ė ė	Ę ę	Ě ě	
	F f	G g	Ğ ğ	H h	I i	Ì ì	
	Í í	Î î	Ï ï	Ī ī	Ĭ ĭ	Į į	
	İ ı	J j	K k	L l	Ĺ ĺ	Ľ ľ	
	Ł ł	M m	N n	Ñ ñ	Ń ń	Ň ň	
	O o	Ò ò	Ó ó	Ô ô	Õ õ	Ö ö	
	Ø ø	Ō ō	Ŏ ŏ	Œ œ	P p	Q q	
	R r	Ŕ ŕ	Ř ř	S s	Ś ś	Ş ş	
	Š š	ß	T t	Ţ ţ	Ť ť	U u	
	Ù ù	Ú ú	Û û	Ü ü	Ũ ũ	Ū ū	
	Ŭ ŭ	Ů ů	Ų ų	V v	W w	Ŵ ŵ	
	X x	Y y	Ý ý	Ŷ ŷ	Ÿ ÿ	Z z	
	Ź ź	Ż ż	Ž ž				

[40] 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 herein by 3 keystrokes maximum.
	{	[	(	)	]	}	
	× / · [41]	÷ / :	∫	$_{\infty}$	$^{\infty}$	∞	
	¬	∧	∨	≠	|	&	
	∡	∟	⊥	∛	√	ˣ√	
	x̄	ȳ	x̂	ŷ	x̄	ȳ	
	:=	≙	≘	E	ℂ	ℝ	
	⊙	⊙	⊕				
	±	^	^{T}	$^{-1}$	ℏ		

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

A…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. [42] 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	
	ς	Τ τ	Υ υ	Φ φ	Χ χ	Ψ ψ	
	Ω ω	ά	έ	ή	ί	ΐ	
	Ϊ ϊ	ό	ύ	Ϋ ϋ	ΰ	ώ	

[41] With startup default settings, the multiplication dot is found here and the multiplication cross is called via f × in *AIM*. If MULT× is clear, however, this dot is called via f × in *AIM* and the multiplication cross via αMATH. The symbols : and ÷ will swap, too.

[42] The Greek alphabet (sic!) goes **a**lpha, **b**eta, **g**amma, **d**elta, **e**-psilon, **z**ēta, **ē**ta, **th**ēta, **i**ōta, **k**appa, **l**ambda, **m**y, **n**y, **x**i, **o**-mikron, **p**i, **r**hō, **s**igma, **t**au, **y**-psilon, **ph**i, **ch**i, **ps**i, **ō**-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.
	¡	¿	§	ø	~	\	
	$	€	%	&	£	¥	
	←	↑	↕	↓	→	⤒	
	«	»	≷	🕓	•	*	
	⌛	❗	U	🔋	@	*	
	„	"	…	_			

Menu							Remarks
Σ	n	Σx	Σx^2	Σxy	Σy^2	Σy	all the sums necessary for the statistics in STAT.
		Σlnx	$\Sigma \ln^2 x$	Σlnxy	$\Sigma \ln^2 y$	Σlny	
	$\Sigma x^2 y$	Σxlny	$\Sigma x^2 \ln y$	Σlny/x		Σylnx	
	$\Sigma x^2/y$	$\Sigma\, ^1/x$	$\Sigma\, ^1/x^2$	Σx/y	$\Sigma\, ^1/y^2$	$\Sigma\, ^1/y$	
	Σx^3	Σx^4					

Menu							Remarks
∡→	→DEG	→RAD	→GRAD		→D.MS	→MULπ	angular conversions, cf. pp. 132f.
	DEG→	RAD→	GRAD→		D.MS→	MULπ→	
	D→R	R→D		D→D.MS	D.MS→D		

The following *menus* will pop up after particular commands; their *sub-menus* VAR and SYS.FL will show only writeable *items* for write operations but all for read. These *menus* will appear after calling…

… STO and RCL	→	VAR	X	Y	Z	T	see the *OM, Section 1*
	Config	Stack			Max	Min	
	…EL	…IJ					

… x≷, y≷, etc.	→	VAR	X	Y	Z	T	see the *OM, Section 1*

… x < ?, etc.	→	VAR	X	Y	Z	T	see the *OM, Section 1*
	0.	1.					

… SF, CF, etc.	→	SYS.FL	X	Y	Z	T	see the *OM, Section 1*

… XEQ, GTO, etc.	→	PROG	X	Y	Z	T	see the *OM*, *Section 3*

… DELITM				PROGS	VARS	MENUS	see the *OM*, *Section* 6

Unit Conversions

Your *WP 43S* features 14 angular conversions provided in ∡→ (cf. p. 122) and 112 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.

Cf. pp. 119f for more details of the structure.

The seven fundamental *SI* units are *kelvin*, *meter*, *kilogram*, *second*, *ampere*, *mol*, and *candela* (the latter measuring *luminous intensity*[43]). Beyond these and products or powers of them, knowledge of the symbols and names of some *SI derived units* is helpful:

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

Quantity	Unit	Symbol and formula
Temperature	*degree Celsius*	$\boldsymbol{\vartheta}[°\mathrm{C}] = \boldsymbol{T}[\mathrm{K}] - 273.15$
Force	*newton*	$1\ \mathrm{N} = 1\ \mathrm{kg}\ {}^{\mathrm{m}}/_{\mathrm{s}^2}$
Pressure	*pascal*	$1\ \mathrm{Pa} = 1\ {}^{\mathrm{N}}/_{\mathrm{m}^2} = 1\ {}^{\mathrm{kg}}/_{\mathrm{m\ s}^2}$
Energy	*joule*	$1\ \mathrm{J} = 1\ \mathrm{N\ m} = 1\ \mathrm{kg}\ {}^{\mathrm{m}^2}/_{\mathrm{s}^2}$
Power	*watt*	$1\ \mathrm{W} = 1\ {}^{\mathrm{J}}/_{\mathrm{s}}$
Electric potential, voltage	*volt*	$1\ \mathrm{V} = 1\ {}^{\mathrm{W}}/_{\mathrm{A}} = 1\ {}^{J}/_{A\ s}$
Charge	*coulomb*	$1\ \mathrm{C} = 1\ \mathrm{A\ s}$
Capacitance	*farad*	$1\ \mathrm{F} = 1\ {}^{\mathrm{C}}/_{\mathrm{V}} = 1\ {}^{\mathrm{A\ s}}/_{\mathrm{V}}$
Resistance	*ohm*	$1\ \Omega = 1\ {}^{\mathrm{V}}/_{\mathrm{A}}$
Conductance	*siemens*	$1\ \mathrm{S} = 1\ {}^{\mathrm{A}}/_{\mathrm{V}}$
Magnetic flux	*weber*	$1\ \mathrm{Wb} = 1\ \mathrm{V\ s}$
Magnetic flux density	*tesla*	$1\ \mathrm{T} = 1\ {}^{\mathrm{Wb}}/_{\mathrm{m}^2} = 1\ {}^{\mathrm{V\ s}}/_{\mathrm{m}^2}$
Inductance	*henry*	$1\ \mathrm{H} = 1\ {}^{\mathrm{Wb}}/_{\mathrm{A}} = 1\ {}^{\mathrm{V\ s}}/_{\mathrm{A}}$
Frequency	*hertz*	$1\ \mathrm{Hz} = {}^{1}/_{\mathrm{s}}$
Absorbed dose	*gray*	$1\ \mathrm{Gy} = 1\ {}^{\mathrm{J}}/_{\mathrm{kg}}$
Plane angle	*radian*	$1\ \mathrm{rad} = 1\ {}^{\mathrm{m}}/_{\mathrm{m}}$
Solid angle	*steradian*	$1\ \mathrm{sr} = 1\ {}^{\mathrm{m}^2}/_{\mathrm{m}^2}$
Luminous flux	*lumen*	$1\ \mathrm{lm} = 1\ \mathrm{cd\ sr}$
Luminance, illuminance	*lux*	$1\ \mathrm{lx} = 1\ {}^{\mathrm{cd}}/_{\mathrm{m}^2}$

For talking about inputs and results, knowing symbols and names of *SI* prefixes is beneficial as well. These cover 36 orders of magnitude:

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

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

All the conversions featured in U→ and ∡→ are explained in alphabetical order below.[44] Numeric values are either exact (printed on white background in the table) or rounded to six significant digits (for your orientation only – your *WP 43S* uses more precise values wherever applicable). Commas are printed as radix marks for better visibility.

[44] Note that these conversions often begin or end with *SI* base units, mainly m and kg – this eases further calculations and conversions. Else do not forget to apply the necessary factors of 10.

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 *Shìzhì* (市制) valid for mainland China; note that units carrying the same names but used in Hongkong and Macao may differ significantly, as well as those used in other states in this region (e.g. Singapore, Malaysia, Taiwan, Vietnam) with major Chinese parts of population.

Also note that *British Imperial* units differentiate between dry and liquid volumes. Do not ask why. For *SI*, a *cubic meter* stays a *cubic meter*, be it evacuated or filled with gas, water, or solid iron.

Softkey	Calculation	Remarks	Branch
°C → °F	× 1,8 + 32	ϑ [°C] = (T [K] + T_0) and τ [°F] = (t [°R] + T_0×1,8)	U→
°F → °C	− 32) / 1,8		
acre → ha	× 4,04686	This *acre* is based on the *'internat. foot'*, see below	U→ f A:
acre$_{US}$ → ha	× 4,04687	This *acre* is based on the *U.S. survey foot*, see below	
atm → Pa	× 101 325	*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*	U→ E:
cal → J	× 4,186 8	*Calory*	
carat → g	× 0,2		U→ m:
chǐ → m	/ 3	*'Chinese foot'* 1 市尺 := 10 *cùn*	U→ x:
cǜn → 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*	U→ ▼
dB → power ratio	$10^{R_{dB}/10}$		
fathom → m	× 1,828 8	1 *fathom* := 2 *yards* := 6 *feet*	U→ x:
fēn → m	/ 300	1 市分 := 0,1 *cùn*	
field ratio → dB	$20\ lg(a_1/a_2)$	Also known as *amplitude ratio*	U→ ▼
floz$_{UK}$ → ml	× 28,413 1	*Fluid ounce*	U→ f V:
floz$_{US}$ → ml	× 29,573 5		

Softkey	Calculation	Remarks	Branch
ft.→m	× 0,304 8	The so-called '*international foot*' of 1959 1 *foot* := 12 *inches*	U→ x:
g→carat	/ 0,2	*Gramm*; $1\text{ g} = \frac{1}{1000}\text{ kg}$	U→ m:
g→oz	/ 28,349 5		
g→tr.oz	/ 31,103 5		
gal_{UK}→l	× 4,546 09	*Gallon*; 1 (*Imperial*) *gallon* := 4 *quarts*	U→ f V:
gal_{US}→l	× 3,785 42		
ha→m^2	× 10 000	1 a [*are*] := 100 m^2, 1 ha [*hectare*] := 10 000 m^2, 1 km^2 = 100 ha = 10^6 m^2	U→ f A:
ha→acre	/ 4,04686		
ha→acre_{US}	/ 4,04687		
hp_E→W	× 746	*Electric horsepower*	U→ P:
hp_M→W	× 735,499	So-called '*metric*' *horsepower* (equivalent to PS in German)	
hp_{UK}→W	× 745,700	*British Imperial horsepower*	
in.→mm	× 25,4	1 *inch* := 1000 *mil*	U→ x:
in.Hg→Pa	× 3 386,39	*Inch of mercury*	U→ F&p:
jīn→kg	× 0,5	'*Chin. pound*'; 1 市斤 := 10 liǎng	U→ m:
J→Btu	/ 1 055,06	*Joule*	U→ E:
J→cal	/ 4,186 8		
J→Wh	/ 3 600		

Softkey	Calculation	Remarks	Branch
kg→cwt	/ 50,802 4		
kg→jīn	/ 0,5		
kg→lb.	/ 0,453 592		U→ m:
kg→liǎng	/ 0,05		
kg→sh.cwt	/ 45,359 2	*Kilogram*	
kg→short ton	/ 907,185	1 t [(*metric*) *ton*] := 1000 kg	U→ m:
kg→stone	/ 6,350 29		
kg→ton	/ 1 016,05		
km→mi.	/ 1,609 344	*Kilometer*	U→ x:
km→nmi	/ 1,852	1 km = 1000 m	
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:
liǎng→kg	× 0,05	*'Chin. ounce'*; 1 市两 := 0,1 jīn	
lǐ→m	× 500	*'Chin. mile'* 1 市里 := 15 yǐn	U→ x:
l.y.→m	× 9,460 73×10 ^{15}	*Light year*	
l→gal$_{UK}$	/ 4,546 09		
l→gal$_{US}$	/ 3,785 42	1 *liter* := 1 dm^3 = 10^{-3} m^3	U→ f V:
l→qt.	/ 1,136 52		
m^2→ha	/ 10 000	*Square meter*	U→ f A:
m^2→mǔ	× 0,001 5		

Softkey	Calculation	Remarks	Branch
m³→barrel	/ 0,158 987	*Cubic meter*	U→ f V:
mi.→km	× 1,609 344	1 *mile* := 1 760 *yards*	U→ x:
ml→floz$_{UK}$	/ 28,413 1	1 ml [*milliliter*] = 1 cm^3	U→ f V:
ml→floz$_{US}$	/ 29,573 5		
mmHg→Pa	× 133,322	See Pa below.	U→ F&p:
mm→in.	/ 25,4	*Millimeter*; $1\ \text{mm} = \frac{1}{1000}\ \text{m}$	U→ x:
mm→point	/ 0,352 778		
mŭ→m²	/ 0,001 5	平方亩 (píng fāng mŭ), in particular for pieces of land	U→ f A:
m→au	/ 1,495 98×10^{11}		
m→chĭ	× 3		
m→cùn	× 30		
m→fathom	/ 1,828 8		
m→fēn	× 300		
m→ft.	/ 0,304 8		
m→lĭ	/ 500		
m→l.y.	/ 9,460 73×10^{15}	*Meter*	U→ x:
m→pc	/ 3,085 68×10^{16}		
m→survey foot$_{US}$	/ 0,304 801		
m→yd.	/ 0,914 4		
m→yĭn	× 0,03		
m→zhàng	× 0,3		
nmi→km	× 1,852	*Nautical mile*	

Softkey	Calculation	Remarks	Branch
Nm→lbf·ft	/ 1,355 82	*Newton meter*	U→ ▼
N→lbf	/ 4,448 22	*Newton*	U→ F&p:
oz→g	× 28,349 5	*Ounce*	U→ m:
Pa→atm	/ 101 325	*Pascal*; 1 hPa = 1 mbar 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→bar	/ 100 000		
Pa→in.Hg	/ 3 386,39		
Pa→mmHg	/ 133,322		
Pa→psi	/ 6 894,76		
Pa→torr	/ 133,322		
pc→m	× 3,085 68×10^{16}	*Parsec*	U→ x:
point→mm	× 0,352 778	1 *(typogr.) point* := 1/72 *inch*	
power ratio→dB	$10\ lg(p_1/p_2)$		U→ ▼
psi→Pa	× 6 894,76	*Pound per square inch*	U→ F&p:
qt.→l	× 1,136 52	1 *(Imperial) quart* := 40 *(Imp.) fluid ounces*	U→ f V:
short cwt →kg	× 45,359 2	1 *short hundredweight* := 100 lbs	U→ m:
short ton →kg	× 907,185	1 *short ton* := 2000 lbs	
stone→kg	× 6,350 29		
survey foot$_{US}$→m	× 0,304 801	1 U.S. *survey foot* := $\frac{1200}{3937}$ *m*	U→ x:
s→year	/ 31 556 952		U→

Softkey	Calculation	Remarks	Branch
ton→kg	× 1 016,05	1 *Imperial ton* := 200 cwt	U→ m:
torr→Pa	× 133,322	See Pa above.	U→ F&p:
tr.oz→g	31,103 5	*Troy ounce*	U→ m:
Wh→J	× 3 600	*Watt-hour*	U→ E:
W→hp_E	/ 746	*Watt*	U→ P:
W→hp_M	/ 735,499		
W→hp_{UK}	/ 745,700		
yd.→m	× 0,914 4	1 *yard* := 3 *feet*	U→ x:
yǐn→m	/ 0,03	1 市引 := 10 *zhàng*	
year→s	× 31 556 952	= 365,242 5 × 24 × 60^2	U→
zhàng→m	/ 0,3	1 市丈 := 10 *chǐ*	U→ x:

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

- $1\ \mathit{ångström} \equiv 1\ \text{Å} = 10^{-10}\ \text{m}$ convenient unit for atomic diameters and wavelengths,
- $1\ \mathit{barn} \equiv 1\ \text{b} = 10^{-28}\ \text{m}^2$ for cross sections in nuclear and particle physics (note the name),
- $1\ \mathit{tex} = 10^{-6}\ {}^{\text{kg}}/_{\text{m}}$ for yarn density in textile industry (one of the few units where *deci* is frequently seen), and
- $1\ \mathit{muggeseggele} \approx 2 \ldots 0.5\ \mu\text{m}$ for Swabian precision mechanics.[45]

[45] See https://en.wikipedia.org/wiki/Muggeseggele for more about this. (I admit I did not expect an article about this pretty local unit there at all. Personally, I had the privilege of experiencing the *muggeseggele* in its habitat at the premises of a Swabian manufacturer of precision components for textile machines – founded in 1852 and one of several hidden champions located in south-western Germany – and its affiliates worldwide. It was definitely fun listening to e.g. Portuguese or Indian staff talking about *muggeseggele* – you cannot translate it. In daily life on the shop floor, 1 *muggeseggele*

<table>
<tr><th>L→ …</th><th colspan="3">Remarks (see also the OM, Section 2)</th></tr>
<tr><td>DEG→</td><td colspan="3" rowspan="2">Takes an integer or real[46] x as an angular input in decimal or sexagesimal degrees, resp., and converts it to the current ADM.</td></tr>
<tr><td>D.MS→</td></tr>
<tr><td>D.MS→D</td><td colspan="3">Takes an integer or real[46] x as an angular input in sexagesimal degrees (formatted <code>ddddd.mmsshh</code>) and converts it to an angle in decimal degrees (corresponding to the old command H.MS→H).</td></tr>
<tr><td>D→R</td><td rowspan="2">Takes an integer or real[46] x as an angular input in decimal degrees and converts it to …</td><td colspan="2">… radians.[47]</td></tr>
<tr><td>D→D.MS</td><td colspan="2">… sexagesimal degrees (corresponding to the old command H→H.MS).</td></tr>
<tr><td>GRAD→</td><td rowspan="3">Takes an integer or real[46] x as an angular input in …</td><td>… grades/gon …</td><td rowspan="3">… and converts it to the current ADM.</td></tr>
<tr><td>MULπ→</td><td>… multiples of π …</td></tr>
<tr><td>RAD→</td><td>… radians[47] …</td></tr>
<tr><td>R→D</td><td colspan="3">Takes an integer or real[46] x as an angular input in radians and converts it to decimal degrees.[47]</td></tr>
</table>

meant significantly less than 0.01 mm but could still be reached manually by a well-trained technician tightening the proper bolt with the appropriate manual torque. Any quantity deviating from 1 *muggeseggele* was never observed in the wild by me.)

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.

[46] 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.

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

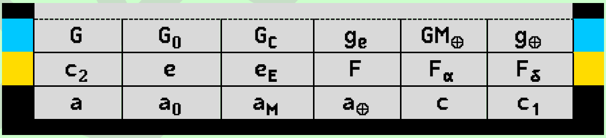
Real angles given in *radians* cannot represent full circles (or multiples of them, as well as simple fractions of π like π/2, π/3, π/4, etc.) exactly but with an accuracy of 34 digits ‘only’. If you want to avoid rounding errors caused by that but must keep π in play, *multiples of π* may be a better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

L→ …	**Remarks** (see also the *OM*, *Section 2*)
→DEG →D.MS →GRAD →MULπ →RAD	Takes an integer (*DT* 1) or *real* (*DT* 2) x as an angular input in the current *ADM* and converts it to *decimal degrees*, *sexagesimal degrees*, *grades/gon*, *multiples of* π, or *radians*, respectively.[47] If x is a tagged *real* (*DT* 4), on the other hand, this information is used in conversion (e.g. if $x = \mathbf{1.5}\,\boldsymbol{\pi}$ then →GRAD will return **300**^{g} regardless of current *ADM*). If x is neither of *DT* 1, 2, nor 4, error 24 will be thrown ('illegal input data type for this operation').

Angular output is tagged always.

Constants

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



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 white background if they are exactly defined – the darker the background, the less precisely the particular value is known.[48] We

[48] 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 today, agreed on by the national standards institutes worldwide (e.g. by *NIST* and *PTB*). Note that the fundamental constants (printed **bold** in the table) 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 use commas as radix marks for better visibility in this chapter and multiplication dots for space reasons. Formulas are printed where applicable.

Name	Numeric value and *rel. SD*	Remarks
a (0)[49]	365,242 5 d (*per definition*)	*Gregorian* year
a_0	$5{,}291\,772\,109\,03 \cdot 10^{-11}$ m *($1{,}5 \cdot 10^{-10}$)*	*Bohr* radius $a_0 = {}^{\alpha}/_{4\pi\, R_\infty}$
a_{Moon}	$3{,}844 \cdot 10^{8}$ m *($1 \cdot 10^{-3}$)*	Semi-major axis of the Moon's orbit around the earth ≈ 1,3 *light seconds*.
$a_\oplus$	$1{,}495\,979 \cdot 10^{11}$ m *($1 \cdot 10^{-6}$)*	Semi-major axis of the Earth's orbit around the sun. Within the uncertainty stated here, it equals 1 *astronomic unit* ≈ 499 *light seconds* ≈ 8 *light minutes*.
c	**$2{,}997\,924\,58 \cdot 10^{8}$ $^{\mathbf{m}}/_{\mathbf{s}}$**	**Speed of light in vacuum** $\approx 300\,000\,\frac{km}{s} = 300\,\frac{km}{ms} = 300\,\frac{m}{\mu s} = 30\,\frac{cm}{ns} = 0{,}3\,\frac{mm}{ps}\ \ldots$

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, and *https://physics.nist.gov/cuu/Uncertainty/index.html*). 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 any 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.

[49] The numbers in parentheses in this column one are to support determination of parameters for CNST – see the *IOI*.

Name	Numeric value and *rel. SD*	Remarks
c_1 (5)	3,741 771 85…·10^{-16} $\mathrm{W\,m^2}$	First radiation constant $c_1 = 2\pi\, h\, c^2$
c_2	0,014 387 768 77… $\mathrm{m{\cdot}K}$	Second radiation constant $c_2 = {}^{hc}/_{k}$
e	**1,602 176 634·10^{-19} A s**	**Elementary charge** $e = \frac{2}{K_J R_K} = \Phi_0 G_0$
e_E	2,718 281 828 459 045 2…	*Euler's e.*
F	96 485,332 12… ${}^{\mathrm{A\,s}}/_{\mathrm{mol}}$	*Faraday* constant $F = e\, N_A$
F_α (10)	2,502 907 875 095 892 8...	*Feigenbaum's* α and δ
F_δ	4,669 201 609 102 990 6...	
G	6,674 30·10^{-11} ${}^{\mathrm{m^3}}/_{\mathrm{kg\,s^2}}$ *(2,2·10^{-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…·10^{-5} / Ω	Conductance quantum $G_0 = {}^{2e^2}/_{h} = {}^{2}/_{R_K} = eK_j$
G_C	0,915 965 594 177 219 0…	*Catalan's* constant
g_e (15)	−2,002 319 304 362 56 *(1,7·10^{-13})*	*Landé's* electron g-factor
$GM_\oplus$	3,986 004 418·10^{14} ${}^{\mathrm{m^3}}/_{\mathrm{s^2}}$ *(2,0·10^{-9})*	*Newtonian* constant of gravitation times the Earth's mass with its atmosphere included (according to *WGS84* [50])
$g_\oplus$	9,806 65 ${}^{m}/_{s^2}$ *(per def.)*	Standard earth acceleration (note the actual values vary from 9,780 4 ${}^{m}/_{s^2}$ at equator to 9,832 2 ${}^{m}/_{s^2}$ at the poles)

[50] See *http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html*

Name	Numeric value and *rel. SD*	Remarks
h	**$6{,}626\ 070\ 15 \cdot 10^{-34}$ J s**	***Planck* constant**
ħ	$1{,}054\ 571\ 817\ldots \cdot 10^{-34}$ J s	Reduced *Planck* constant $\hbar = h/2\pi$
k (20)	**$1{,}380\ 649 \cdot 10^{-23}$ J/K**	***Boltzmann* constant** $k = R/N_A$
K_J	$4{,}835\ 978\ 48\ldots \cdot 10^{14}$ Hz/V	*Josephson* constant $K_j = 2e/h$
l_{PL}	$1{,}616\ 255 \cdot 10^{-35}$ m *($1{,}1 \cdot 10^{-5}$)*	*Planck* length $l_{PL} = t_{PL} c$
m_e	$9{,}109\ 383\ 701\ 5 \cdot 10^{-31}$ kg *($3{,}0 \cdot 10^{-10}$)*	Electron mass ≙ 511,00 keV
M_{Moon}	$7{,}349 \cdot 10^{22}$ kg *($5 \cdot 10^{-4}$)*	Mass of the Moon
m_n (25)	$1{,}674\ 927\ 498\ 04 \cdot 10^{-27}$ kg *($5{,}7 \cdot 10^{-10}$)*	Neutron mass ≙ 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 ≙ 938,27 MeV
m_{PL}	$2{,}176\ 435 \cdot 10^{-8}$ kg *($1{,}1 \cdot 10^{-5}$)*	*Planck* mass $m_{PL} = \sqrt{\hbar c / G} \approx 22\ \mu g$
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 ≈ 931,49 MeV

Name	Numeric value and *rel. SD*	Remarks
m_μ	$1{,}883\,531\,627 \cdot 10^{-28}$ kg *($2{,}2 \cdot 10^{-8}$)*	Muon mass ≙ 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**	***Avogadro's* number**
NaN	*Not a Number*	See p. 163 and the corresponding entry in *Section 5* of the *OM.*
p_0	101 325 Pa *(per defin.)*	Standard atmospheric pressure
R	8,314 462 618… $^J/_{mol\,K}$	Molar gas constant
r_e	$2{,}817\,940\,326\,2 \cdot 10^{-15}$ m *($4{,}5 \cdot 10^{-10}$)*	Classical electron radius $r_e = \alpha^2 a_0$
R_K (40)	25 812,807 45… Ω	*Von Klitzing* constant $R_K = {}^h/_{e^2}$
R_{Moon}	$1{,}737\,530 \cdot 10^{6}$ m *($5 \cdot 10^{-7}$)*	Mean radius of the Moon
R_∞	10 973 731,568 160 / m *($1{,}9 \cdot 10^{-12}$)*	*Rydberg* constant $R_\infty = \frac{\alpha^2 m_e c}{2h}$
$R_\odot$	$6{,}96 \cdot 10^{8}$ m *($5 \cdot 10^{-3}$)*	Mean radius of the sun
$R_\oplus$	$6{,}371\,010 \cdot 10^{6}$ m *($5 \cdot 10^{-7}$)*	Mean radius of the Earth

Name	Numeric value and *rel. SD*	Remarks	
Sa (45)	6,378 137 0·10^{6} m *(p. def.)*	Semi-major axis	… according to *WGS84* (see footnote 50 on p. 135).
Sb	6,356 752 314 2·10^{6} m *(1,6·10^{-11})*	Semi-minor axis	
Se²	6,694 379 990 14·10^{-3} *(1,5·10^{-12})*	First eccentricity squared	
Se'²	6,739 496 742 28·10^{-3} *(1,5·10^{-12})*	Second eccentricity squared	
Sf⁻¹	298,257 223 563 *(per def.)*	Flattening parameter	
T₀ (50)	273,15 K *(per definition)*	= 0°C, standard temperature	
Tp	1,416 785·10^{32} K *(1,1·10^{-5})*	*Planck* temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_P c^2}{k} = \frac{E_P}{k}$	
tPL	5,391 245·10^{-44} s *(1,1·10^{-5})*	*Planck* time $t_{PL} = {}^{l_{PL}}/_{c}$	
Vm	0,022 413 969 5… $^{m^3}/_{mol}$	Molar volume of an ideal gas at standard conditions $V_m = \frac{RT_0}{p_0} \approx 22{,}4\ {}^{l}/\text{mol}$	
Z₀	376,730 313 668 Ω *(1,5·10^{-10})*	Characteristic impedance of vacuum	
α (55)	7,297 352 569 3·10^{-3} *(1,5·10^{-10})*	Fine-structure constant $\alpha = \frac{e^2}{2\varepsilon_0 h c} \approx \frac{1}{137}$	
γ	6,674 30·10^{-11} $^{m^3}/_{kg\ s^2}$ *(2,2·10^{-5})*	*Newtonian* constant of gravitation; also known as G from other authors. See **GM⊕** below for a more precise value.	

Name	Numeric value and *rel. SD*	Remarks
γ_{EM}	0,577 215 664 901 532 9…	*Euler-Mascheroni* constant
γ_p	2,675 221 874 4·10^{8} $\mathrm{Hz}/_{\mathrm{T}}$ *(4,2·10^{-10})*	Proton gyromagnetic ratio $\gamma_p = 4\pi \, {}^{\mu_p}/_{h}$
$\Delta\nu_{Cs}$	**9 192 631 770 Hz**	**Hyperfine transition frequency of ^{133}Cs**
ε_0 (60)	8,854 187 812 8·10^{-12} $\frac{\mathrm{As}}{\mathrm{Vm}}$ *(1,5·10^{-10})*	Vacuum electric permittivity $\varepsilon_0 = {}^{1}/_{\mu_0 c^2}$ (note the so-called *Coulomb's* constant is just ${}^{1}/_{4\pi\varepsilon_0}$)
λ_C	2,426 310 238 67·10^{-12} m *(3,0·10^{-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_{Cn}	1,319 590 905 81·10^{-15} m *(5,7·10^{-10})*	
λ_{Cp}	1,321 409 855 39·10^{-15} m *(3,1·10^{-10})*	
μ_0	1,256 637 062 12·10^{-6} $\frac{\mathrm{Vs}}{\mathrm{Am}}$ *(1,5·10^{-10})*	Vacuum magnetic permeability
μ_B (65)	9,274 010 078 3·10^{-24} $\mathrm{J}/_{\mathrm{T}}$ *(3,0·10^{-10})*	*Bohr* magneton $\mu_B = {}^{e\hbar}/_{2m_e}$
μ_e	−9,284 764 704 3·10^{-24} $\mathrm{J}/_{\mathrm{T}}$ *(3,0·10^{-10})*	Electron magnetic moment
μ_e/μ_B	−1,001 159 652 181 28 *(1,7·10^{-13})*	Ratio of electron magnetic moment to *Bohr's* magneton
μ_n	−9,662 365 1·10^{-27} $\mathrm{J}/_{\mathrm{T}}$ *(2,4·10^{-7})*	Neutron magnetic moment

Name		Numeric value and *rel. SD*	Remarks
μ_p		$1{,}410\,606\,797\,36 \cdot 10^{-26}\ \mathrm{J}/\mathrm{T}$ *(4,2·10⁻¹⁰)*	Proton magnetic moment
μ_u	(70)	$5{,}050\,783\,746\,1 \cdot 10^{-27}\ \mathrm{J}/\mathrm{T}$ *(3,1·10⁻¹⁰)*	Nuclear magneton $\mu_u = e\hbar/2m_p$
μ_μ		$-4{,}490\,448\,30 \cdot 10^{-26}\ \mathrm{J}/\mathrm{T}$ *(2,2·10⁻⁸)*	Muon magnetic moment
σ_B		$5{,}670\,374\,41\ldots \cdot 10^{-8}\ \frac{\mathrm{W}}{\mathrm{m^2 K^4}}$	*Stefan-Boltzmann* constant $\sigma_B = \frac{2\pi^5 k^4}{15 h^3 c^2}$
Φ		1,618 033 988 749 894 8…	Golden ratio $\Phi = \frac{1}{2}(1+\sqrt{5})$
Φ_0		$2{,}067\,833\,848\ldots \cdot 10^{-15}\ \mathrm{V\ s}$	Magnetic flux quantum $\Phi_0 = \frac{h}{2e}$
ω	(75)	$7{,}292\,115 \cdot 10^{-5}\ \mathrm{rad}/\mathrm{s}$ *(2·10⁻⁸)*	Angular velocity of the Earth according to *WGS84* (see footnote 50 on p. 135).
$-\infty$		$-\infty$	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.
∞		∞	

Each of 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, thus not stored in CONST, follow below for your convenience:

Numeric value	Remarks
$11{,}2 \cdot 10^{3}\ \mathrm{m}/\mathrm{s}$	Escape speed from Earth
$331\ \mathrm{m}/\mathrm{s}$	Speed of sound in air at standard conditions

Numeric value	Remarks
$1{,}217 \, {}^{\mathrm{kg}}/_{\mathrm{m}^3}$	Air density at standard conditions (= $1{,}217 \, {}^{\mathrm{Mt}}/_{\mathrm{km}^3}$)
$28{,}97 \cdot 10^{-3} \, {}^{\mathrm{kg}}/_{\mathrm{mol}}$	Molar mass of air
$1370 \, {}^{\mathrm{W}}/_{\mathrm{m}^2}$	Solar constant (average incident power at the mean distance of Earth to the Sun outside the atmosphere)
$1{,}62 \, {}^{\mathrm{m}}/_{\mathrm{s}^2}$	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	$\sim 10^{-15}\ \mathrm{m}$
Radius of an atom [51]	$\sim 10^{-10}\ \mathrm{m}$
Radius of our home galaxy	$\sim 10^{5}\ l.y. \approx 10^{21}\ \mathrm{m}$
Radius of the observable universe [52]	$\sim 45 \cdot 10^{9}\ l.y. \approx 4{,}3 \cdot 10^{26}\ \mathrm{m}$
Age of the universe	$13{,}8 \cdot 10^{9}\ \mathrm{a} = 1{,}21 \cdot 10^{14}\ \mathrm{h} = 4{,}35 \cdot 10^{17}\ \mathrm{s}$
Hubble 'constant'	$\sim 70 \frac{\mathrm{km/s}}{\mathrm{Mpc}} \approx {}^{1}/_{4{,}4 \cdot 10^{17}\mathrm{s}}$

[51] 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).

[52] 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.

Baryonic mass[53] in observable universe	$\sim 10^{53}$ kg
Number of stars in our home galaxy [54]	$\sim 2 \cdot 10^{11}$
Number of neuron connections in human brain	$\sim 10^{14}$
Number of stars in observable universe	$\sim 10^{23}$
Number of atoms in the Sun [55]	$\sim 10^{57}$
Number of atoms in observable universe	$\sim 10^{80}$

Note these quantities are all <u>far</u> 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.

[53] 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.

[54] Assume our home galaxy fills a huge disk-like 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 57.)

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

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↑. Your input will be checked – if the operation specified exists, …
   a. then it will be checked for required trailing parameters (see 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- RCL× RCL/ RCL↑ RCL↓ STO STOCFG STOS STO+ STO- STO× STO/ STO↑ STO↓ t≷ VIEW x≷ x=? x≠? x≈? x<? x≤? x≥? x>? y≷ z≷ αLENG? αPOS? αRL αRR αSL α→x 🖶r	*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*
CNST	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		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 display precision	
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	*Item*	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 *DT*	Operation(s)	Output *DT*	Output *registers*
1	1/x $\sqrt[3]{x}$ AGM ALL cos ENG erf erfc e^x FIX f' f'' gd gd^{-1} $J_y(x)$ LN LNβ LNΓ LOG_{10} LOG_2 $LOG_x y$ MANT POISS... SCI sin tan W_m W_p W^{-1} $\sqrt[x]{y}$ β(x,y) Γ_{xy} γ_{xy} Γ(x) Δ% →REAL % %MRR %T %Σ %+MG $\sqrt{x}$ || as well as all unit conversions and all orthogonal polynomials	2 [56]	**X**
	→POL →REC	2	**X, Y**
	SLVQ	2 or 3	**X, Y, Z**
	f'(x) f''(x) SOLVE	2	**X, Y, Z, T**
	all angular conversions	4	**X**
	→H.MS	5	**X**
	J→D →DATE	6	**X**
	x→α	7	**X**
	M.GET M.NEW	8 or 9	**X**
	→INT	10	**X**

[56] 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| ∡	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 *DT*	Operation(s)	Output *DT*	Output *registers*
8	M.DIM?	1	**X, Y**
	DET DOT ENORM |M|	2	**X**
	Σ+	2	**X, Y**, statistic *registers*
	V∡	4	**X**
9	M.DIM?	1	**X, Y**
	ENORM	2	**X**
	DET DOT |M|	3	**X**
	ABS IM RE ROUNDI |x|	8	**X**
10	SIGN	1	**X**
	e^x LN $LOG_x y$ →REAL	2	**X**
	x→α	7	**X**

[57]

[57] Solution of footnote 54: 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 in a collision.

For estimating the consequences of a galactic collision more realistically, take the orientation of both galaxies relative to each other into account when they collide. And star density is not uniform but highest in the galactic center and decreases significantly from there outwards. Furthermore, there use to be giant amounts of free gas (although very thin) and dust between the stars.

Photographs of colliding galaxies are available in the net.

APPENDIX A: HARDWARE

Dimensions: wedge-shaped 77 mm × 144 mm × 13 mm or 8 mm (see picture overleaf).

Mass: 180 g incl. battery.

LCD: monochrome high contrast (14:1) transflective 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. 154ff), 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.

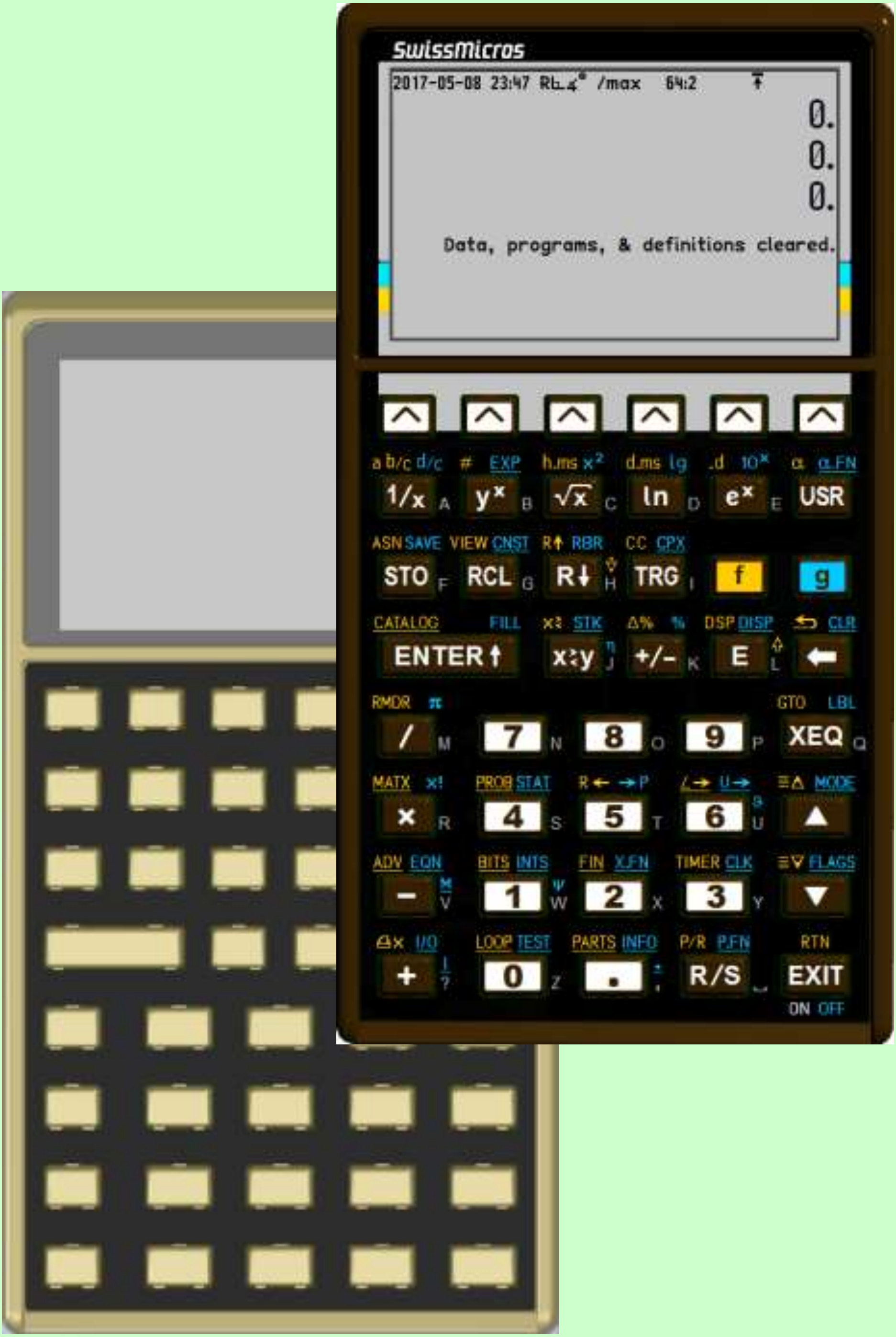
Buzzer: piezo-electric with 4 kHz resonance frequency, tunable from 1 Hz up to 20 kHz in steps of 1 Hz.

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. 208). 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.


SwissMicros
2017-05-08 23:47 RL₄° /max 64:2
0.
0.
0.
Data, programs, & definitions cleared.
a b/c d/c # EXP h.ms x² d.ms lg .d 10ˣ α α.FN
1/x A yˣ B √x C ln D eˣ E USR
ASN SAVE VIEW CNST R↑ RBR CC CPX
STO F RCL G R↓ H TRG I f g
CATALOG FILL x⇄ STK Δ% % DSP DISP CLR
ENTER↑ x⇄y J +/− K E L ←
RMDR π GTO LBL
/ M 7 N 8 O 9 P XEQ Q
MATX x! PROB STAT R← →P U→
× R 4 S 5 T 6 U ▲
ADV EQN BITS INTS FIN X.FN TIMER CLK FLAGS
− V 1 W 2 X 3 Y ▼
I/O LOOP TEST PARTS INFO P/R P.FN RTN
+ 0 Z . R/S EXIT
ON OFF

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 *FM* for the *WP 43S* program and its constant data (incl. fonts, *menus,* messages, etc.) – this memory is in the *CPU.*
0x10 00 00 00 – 0x10 00 7F FF	32 KiB system *RAM* for the headers[58] of the 112 global *registers* and for other global *WP 43S* variables like expandable *submenus* (cf. pp. 109ff).
0x20 00 00 00 – 0x20 01 FF FF	96 KiB user *RAM* for programs, *registers*, *flags*, *menus*, and variables. STATUS and MEM? show the free space there.
0x90 00 00 00 – 0x90 1F CF FF	2036 KiB *QSPI (= Quad Serial Peripheral Interface) FM* for the operating system
0x90 1F D0 00 – 0x90 1F DF FF	4 KiB *QSPI* user *FM*
0x90 1F E0 00 – 0x90 1F FF FF	8 KiB *QSPI FM*
0x90 20 00 00 – 0x90 7F FF FF	6144 KiB *FM* for the *FAT (= File Allocation Table)* system, where backups and snapshots are stored. This part of the memory of your *WP 43S* is visible on your computer and may accessed by it as long as 'activate *USB* disk' is selected.

Of these $9\,^{1}/_{8}$ MB, we printed the sections writable by the *WP 43S* system only (some 3 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.

[58] 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,[59]
- 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 register[60], the *internal DT* (see overleaf), and some data information.[61]

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.

---

[59] We need far less than 112 *system flags* so far.

[60] I.e. to the first memory block number of the *register* content. This allows access to 65 535 *blocks* × 4 *bytes / block* = 262 140 *bytes* = 256 kB − 4 *bytes*. A special value is reserved for the NULL pointer.

[61] 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,[61] 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* [62]	≥ 4 + 4 =	8 - 420
2	1	ℝ *Real number* [63] (*real34*)		16
3	2	ℂ *Complex number*, always stored in rectangular notation (*complex34*)	2 × 16 =	32
4	1	*Angle* [64]		16
5	3	*Time* [65]		16
6	4	*Date* [66]		16
7	5	*Alphanumeric string* (a.k.a. *text string*)	4 + ***j*** + 2 ***k*** + 1[67]	≤ 396

---

[62] 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.

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

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

[65] Internally, a *time* or time interval is stored as a *real number* of *seconds*, just with a specific header. Decimal times are displayed in *hours*. One day corresponds to 86 400 s. This format allows for expressing intervals of some 10^{12} *years* with *femtoseconds* precision.

[66] 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. Cf. also previous footnote.

[67] The length of a *text string* in *blocks* is ***L*** = (***j*** + 2 ***k*** + 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*.

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

***DT* number and meaning**			**Size** [*bytes*]
8	6	*Real* matrix (of ***n*** rows and ***m*** columns)[68]	$4 + n \times m \times 16$
9	7	*Complex* matrix (of ***n*** rows and ***m*** columns)[68]	$4 + n \times m \times 32$
10	8	*Short integer*[69]	8
11	9	*Configuration* (as stored by STOCFG)	$4 + M$[70]
12	5	*Label* (or *name* – up to 7 characters)[67]	$1 + j + 2k =$ 2 - 15
13		Constant[71]	$4 + 0 =$ 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)[72]	$4 + 32 =$ 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. 172f	See pp. 172f
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)	$4 + M$

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

[68] 2 *bytes* for ***n***, 2 *bytes* for ***m***, then follow the individual matrix elements row by row, taking either 2 or 4 *bytes* each.

[69] This *DT* is for computer science problems.

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

[71] A pointer to the constant is sufficient here, regardless of its precision.

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

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. 54 and 84f).

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 × 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 *stack* and the *summation registers*, if allocated), all user-defined variables, 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. 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 RESET events, according to your requests:

- LOADSS recovers the saved system state of your *WP 43S* (incl. its *configuration*, cf. p. 45).[73]
- LOADΣ recalls just the *summation registers*,
- LOADR all the numbered *registers*,
- LOADV all user-defined variables,
- LOADP the programs, and
- LOAD recalls the entire data set as stored by SAVE at once (incl. the lettered registers).

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.[74] Results should show a relative error of less than $\pm 1 \times 10^{-33}$ per plain operation. Some random examples:

- ***n*** [1/x] **2*n*** [×] **2** [−] returns exactly 0. for all primes < 500 – except **7** [1/x] [1][4] [×] **2** [−] **2** [/] returning 5×10^{-34}, and for 67 and 83 the respective results are -5×10^{-34};
- **7** [sin] [arcsin] **7** [−] **7** [/] returns 1.4×10^{-34};

---

[73] STOCFG stores the WP 43S *configuration* in a *register*/variable, i.e. in *RAM* instead.

[74] 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 expression: $729^{33.5}/3^{201} - 1$. Your *WP 43S* will return precisely 0.

- 7 [ln] [e^x] 7 [−] 7 [/] = 2.9×10^{-34} while 7 [e^x] [ln] 7 [−] returns plain 0;
- 7 [$\sqrt{x}$] [x^2] 7 [−] 7 [/] = −2.9×10^{-34} while 7 [x^2] [$\sqrt{x}$] 7 [−] returns plain 0;
- 7 [ENTER↑] [$\sqrt[x]{y}$] 7 [y^x] 7 [−] 7 [/] returns −4.3×10^{-34};
- but SLVQ shows an accuracy of 2×10^{-33} (cf. the *OM*, *Sect. 4*).

Overall accuracy should be < ±3×10^{-33} except at particularly 'nasty' locations (e.g. **0.5** [cos] [arccos] **0.5** [−] **0.5** [/] returns 3.6×10^{-31} due to the cosine of a small angle). Check your algorithms carefully to avoid such locations. Note that any rounding errors due to calculating with a finite number of digits accumulate.[75] In almost every case, precision of your results will exceed anything known in the real world nevertheless.

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*).

---

[75] There is a quasi-standard established to find out about processors, firmware, and accuracy of calculators – compute $arcsin\{arccos[arctan(tan\{cos[sin(9°)]\})]\}$ (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(π/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 5·10^{-29}
  for $arcsin\{real(arccos\{real[arctan(tan\{cos[sin(9°)]\})]\})\}$.

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

- A comparable test (from 1976) is **29** *sin cos tan* [$\sqrt{x}$] [ln] [e^x] [x^2] *arctan arccos arcsin*. Your *WP 43s* returns 29 – 3,159 9·10^{-28}.

- Far simpler works another test discussed in the internet: Enter **1,000 000 1** and press [x^2] just 27 times. Your *WP 43s* will then return

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

  Although this is the most precise result known for a pocket calculator so far (the *WP 34S* with DBLON and *Free42* concur) only its first 25 digits 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 10^9 digits of the entire result (you get 896 decimals after 7 presses of [x^2] already).

Please take this information into account when assessing small deviations or many decimals returned by your *WP 34S*, in particular after long calculations or iterations. It will return 34 digits always but not all of them are guaranteed being true – there should be sufficient space left, however, that the important digits of your results are true.

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.
- The *significand*[77] 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 carries 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 `t` 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

[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 .

[77] *Significand*, not *mantissa* – the *significand* does not contain a radix mark.

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 $-6176 + 34 = -6142$, *DT* 2 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:

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 1.999 999 999 999 999 999 999 999 999 999 999 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*
9.999 999 999 999 999 999 999 999 999 999 999 ×10^6144 (= 9 999 … 999 999 ×10^6110)	77 ff be 7f 9f e7 f9 fe 7f 9f e7 f9 fe 7f 9f e7	9 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7	0111 0111 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**.**999 999 999 999 999 999 999 999 999 999 999×10^6144 will be only unveiled by SHOW (see p. 66). The greatest legal *significand* on your *WP 43S* is 9 999 999 999 999 999 999 999 999 999 999 999 = $10^{34} - 1$.

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
−∞	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' (∞, −∞, 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 −∞ may be also legal numeric inputs on your *WP 43S*.**

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.

Remember not every 34-digit number displayed will be true to 34 digits – cf. footnote 75 on p. 160. And errors accumulate as explained in footnote 48 on p. 134.

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

Limitations

Maximum numeric input for *data type (DT)* …

- **1:** $\pm 10^{1001}$ (if your patience will suffice for the input) or $\pm 2^{3326}$ (if you are less patient)
- **2:** $\pm 9.999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,999 \times 10^{\mathit{RANGE}-1}$ (absolute maximum for ***RANGE*** is 6145)
- **3:** $\pm 9.999\ldots \times 10^{\mathit{RANGE}-1}$ for either part (cf. *DT* 2)
- **4:** $\pm 9.999\ldots \times 10^{\mathit{RANGE}-1}$ in arbitrary *ADM,* so the actual absolute maximum is $9.999\ldots \times 10^{\mathit{RANGE}-1}\,\pi$
- **5:** ±9.999 999 999 999 999:59:59.9… $hours \approx \pm 1.14 \times 10^{12} years$ exceeding the age of the universe.
- **6:** 99 999 999 999.1231 .d in Y.MD; though the input maxima are smaller for the other date display modes: 31.129 999 .d in D.MY and merely 12.319 999 .d in M.DY. Minimum input is −4 712.010 1 .d in Y.MD (or −1.014 712 .d in D.MY/M.DY, cf. p. 156).
- **8:** $\pm 9.999\ldots \times 10^{\mathit{RANGE}-1}$ for each matrix element (cf. *DT* 2)
- **9:** $\pm 9.999\ldots \times 10^{\mathit{RANGE}-1}$ for either part of each element (cf. *DT* 3).

- **10:** For arbitrary word size ***n*** and unsigned mode, $\boldsymbol{x}_{max} = 2^{\boldsymbol{n}} - 1$ (i.e. ~1.8×10^{19} for ***n*** = 64); for signed modes, $|\boldsymbol{x}| < 2^{\boldsymbol{n}-1}$.

Internal limits:

- PRIME? works like the other binary tests described in the *OM*. Above 3 317 044 064 679 887 385 961 981 (≈ 3.3×10^{24}), however, NEXTP cannot find primes anymore but 'probable primes' only (cf. p. 56). If you want to apply NEXTP or PRIME? above this limit, consult appropriate reference literature.[78]
- Trigonometric functions actually operate between $+\boldsymbol{\pi}$ and $-\boldsymbol{\pi}$ for *radians* exclusively. The necessary modulo $2\boldsymbol{\pi}$ reduction ensures that these functions return correct 34-digit results for the entire legal range of angles. *ADM*s other than *radians* may be easier to deal with.
- Any intermediate real result within $(-10^{-6176},\ 10^{-6176})$ will be assessed as 0. (cf. previous chapter). Any intermediate real result exceeding $(-10^{6145},\ 10^{6145})$ will be assessed as $-\infty$ or ∞, respectively, and an overflow error will be thrown if SPCRES is clear. This holds for matrix elements and complex components as well.

Maximum numeric output for *DT* …

- **1:** $\pm 10^{1001}$ with full 1001-digit precision (though you can <u>see</u> and read up to 296 digits only of such numbers – cf. SHOW, p. 66).
- **2, 3, 8, and 9:** The maxima are as specified for input above. Any real result exceeding $\left(-10^{\boldsymbol{RANGE}},\ 10^{RANGE}\right)$ will be displayed as -'∞' or '∞', respectively, and any one within $(-10^{-RANGE},\ 10^{-RANGE})$ will be displayed as '0.' as long as ***RANGE*** is within the absolute limits (as stated above for intermediate results); else any real result

[78] See https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test for a start (also available in other languages). Whatever is a prime will be confirmed by PRIME? – 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.

assessed as 0. will be displayed as such, and any one assessed as $-\infty$ or ∞ will be displayed as `-∞` or `∞` if SPCRES is set else an overflow error will be thrown. This holds for matrix elements and complex components as well.

- **4:** For angular conversions, the limits 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. Time intervals smaller than the display limit set by TDISP (< 1 ms for TDISP 0) are shown in a format like SCI 4 (or ENG 4 with ALLENG set) but with a trailing 's', e.g. `-1.234×10⁻⁹s` (more digits can be shown by RBR or SHOW).
- **6:** The date limits are as specified for input in Y.MD above. Large dates may be displayed as well in D.MY or M.DY, however; negative dates may be displayed like 07.07.−0753.
- **10:** The maxima are as specified for input above.

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 (e.g. in RBR).

What Happens when Changing Word Size?

WSIZE affects *short integer* contents of all 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 with empty bits 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. 163).[79]

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

Input x	Operation(s)	Output for ℝ lit
-1.	artanh	-∞
0 or 0.	ln, lg, lb x	-∞
0.	1/x	∞
1.	artanh	∞
0 or 0.	Γ(x)	NaN
$\mathrm{Re}(x) < 1$	arcosh	NaN
$\lvert\mathrm{Re}(x)\rvert > 1$	arccos, arcsin, artanh	NaN
±90° or equivalents in other *ADM*	tan	NaN

[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.

And the following monadic functions operate also on infinities:

Input *x*	Operation(s)	Output for ℝ lit
−∞	x^3, $\sqrt[3]{x}$	−∞
	arctan	−90.° or equivalents
	tanh	−1.
	1/x, e^x, 10^x, 2^x, sinc	0.
	x^2, arsinh	∞
−∞	arcosh	NaN
$-\infty \leq x < 0$	ln, lg, lb x	NaN
∞	1/x, sinc	0.
	tanh	1.
	arctan	90.° or equivalents
	ln, e^x, x^2, $\sqrt{x}$, lg, 10^x, lb x, x^3, $\sqrt[3]{x}$, sinh, cosh, arsinh, arcosh	∞
−∞ 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]
−∞	$x \neq \infty$		−∞ [80]
−∞	∞	+	NaN [80]
∞	$x \neq \infty$	−	∞ [81]
−∞	$x \neq -\infty$		−∞ [81]
−∞	−∞	−	NaN
∞	∞		

[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 ℝ lit
∞	$x > 0$	[×]	∞ [80]
−∞	$x < 0$		
∞	$x < 0$	[×]	−∞ [80]
−∞	$x > 0$		
0 or 0.	−∞ or ∞	[×]	NaN [80]
$0 < y \leq \infty$	0.	[/]	∞
$-\infty \leq y < 0$			−∞
−∞ 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 $x > 0$		−∞
−∞	even $x > 0$	[y^x]	∞
		[$\sqrt[x]{y}$]	NaN
$y \neq 0$	−∞	[y^x]	0.
	∞		∞
0.	$-\infty \leq x < 0$	[y^x]	∞
	$0 < x \leq \infty$		0.
	$-\infty < x < 0$	[$\sqrt[x]{y}$]	∞
	$0 \leq x < \infty$		0.
$0 \leq y \leq \infty$	−∞ or ∞	[$\sqrt[x]{y}$]	1.
∞	$-\infty \leq x < 0$	[y^x]	0.
	$0 < x \leq \infty$		∞
	$-\infty < x < 0$	[$\sqrt[x]{y}$]	0.
	$0 \leq x < \infty$		∞
0.	$0 < x < \infty$	$\log_x y$	−∞

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 ±∞ or close to it (or returning a value at the edge) are listed here:

Input[82] Re(*x*)	Im(*x*)	r(*x*)	φ(*x*)	Op.	Output for ℂ lit
−∞	0	∞	180°	x^2	∞ ∡ 360° = ∞+i×0.
0.	∞	∞	90°		∞ ∡ 180° = −∞+i×0.
−∞	—	—		$\sqrt{x}$	∞ ∡ 90° = 0.+i×∞
−∞	0	∞	180°		
−∞	—	—		x^3	−∞
−∞	0	∞	180°		−∞+i×0.
−∞	—	—		$\sqrt[3]{x}$	−∞
−∞	0	∞	180°		−∞+i×0. [83]
-10^{999}	0	10^{999}	180°		1.×10^{333} ∡ 60° = 0.5×10^{333}+i×0.866 025 404×10^{333} $= 5 \times 10^{332} \left(1 + i \times \sqrt{3}\right)$
−∞	0	∞	180°	e^x	0.+i×0.
-10^{999}	10^{999}	10^{999}	135°		0.+i×0.
−∞	∞	∞	135°		NaN+i×NaN
0.	∞	∞	90°		NaN+i×NaN
∞	∞	∞	45°		NaN+i×NaN

[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)	φ(x)	Op.	Output for ℂ lit
∞	0	∞	0°		∞+i×0.
∞	−∞	∞	−45°		NaN+i×NaN
0.	−∞	∞	−90°	e^x	
−∞	−∞	∞	−135°		NaN+i×NaN
−10^999	−10^999	10^999			0.+i×0.
−∞	—	—			∞+i×3.141 592 65… $= \infty + i\pi$
−∞	0	∞	180°		
−∞	∞	∞	135°		∞+i×2.356 194 49… $= \infty + i\,{}^{3\pi}/_4$
0.	∞	∞	90°		∞+i×1.570 796 32… $= \infty + i\,{}^{\pi}/_2$
∞	∞	∞	45°		∞+i×0.785 398 16… $= \infty + i\,{}^{\pi}/_4$
∞	—	—		ln	∞
∞	0	∞	0°		∞+i×0.
∞	−∞	∞	−45°		∞−i×0.785 398 16… $= \infty - i\,{}^{\pi}/_4$
0.	−∞	∞	−90°		∞−i×1.570 796 32… $= \infty - i\,{}^{\pi}/_2$
−∞	−∞	∞	−135°		∞−i×2.356 194 49… $= \infty - i\,{}^{3\pi}/_4$
0.	0.	0.	0.		−∞+i×0.
0.	—	—			−∞

Computation of lg and 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 ±∞ 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↑, FILL, GTO, IP, ISG, LN, MOD, NEXTP, RTN, R↓, STO, TAN, x^2, XEQ, x≷y, y^X, +, √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 $U+0080_{16}$ only; any character above requires 2 *bytes* (cf. *DT* 7 as described on p. 156).

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. 159ff for the representation).

$8\ 07\ 06\ 05\ 04\ 03\ 02\ 01_{16}$ 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.xxx

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 five basic flag commands **CF**, **FC?**, **FF**, **FS?**, and **SF**,
- the four basic *Boolean* operations **AND**, **OR**, **XOR**, and **NOT**,
- **RCL**, **RCL+**, **RCL-**, **RCL×**, **RCL/**, **STO**, **STO+**, **STO-**, **STO×**, and **STO/**,
- most commands of STK: **DROP**, **ENTER↑**, **FILL**, **R↑**, **R↓**, **x⇄**, and **x⇄y**,
- all commands of LOOP: **DEC**, **DSE**, **DSL**, **DSZ**, **INC**, **ISE**, **ISG**, and **ISZ**,
- angular conversions: **D→R**, **R→D**, **→DEG**, **→D.MS**, **→GRAD**, **→MULπ**, and **→RAD**,
- basic programming commands: **GTO**, **INPUT**, **LBL**, **PAUSE**, **ROUND**, **RTN**, **STOP**, **VIEW**, and **XEQ**,
- thirteen basic transcendental functions: 10^x, 2^x, $\sqrt[3]{x}$, e^x, **LN**, LOG_{10}, LOG_2, $LOG_x y$, x^2, x^3, $\sqrt[x]{y}$, y^x, and $\sqrt{x}$,
- **1/x**, **CEIL** and **FLOOR**, **CLX**, **COMB** and **PERM**, **FP** and **IP**, **GCD** and **LCM**, **IDIV**, **max**, **min**, **MOD** and **RMD**, **NEIGHB**, **NEXTP**, **x!**, **π**, **+/-**, and **|x|**,
- almost all commands 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, $\bar{x}$, $\hat{x}$, $\hat{y}$, Σ+, Σ−, σ, →POL, →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):

	EC	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 $x \le 0$ (unless CPXRES is set), by 0^0, $x/0$, $0/0$, $\Gamma(0)$, $\tan(\pm 90°)$ and equivalents, by $\mathrm{artanh}(x)$ for $\lvert\mathrm{Re}(x)\rvert \ge 1$, by $\mathrm{arcosh}(x)$ for $\mathrm{Re}(x) < 1$, etc.[84]
Bad time or date input	2	{2, 5, 6} Invalid date format or incorrect *date* or *time* in input, like month > 12, day > 31, or *time* exceeding the age of the universe. Will be thrown as soon as input is closed.
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 *FM* to regain space.
Flash memory is write protected	19	There was an attempt to edit or delete program steps in *FM*. See PRCL and PSTO to circumvent.

[84] Note that e.g. tan(90°) 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
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 *DTs* (e.g. for a *Boolean* 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 "*operation is undefined in this mode*"). But this error may also appear in attempts to calculate with a *configuration*.
Invalid or corrupted data	18	Thrown when there is a checksum error either in *FM* or as part of a serial download. Also thrown if a *FM* 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} • A matrix isn't square although it should be. • Matrix sizes aren't miscible.
No backup data found	35	Futile attempt to LOAD something from *FM*.

	EC	Explanations, countermeasures and examples
`No root found`	20	{2} The *Solver* 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 *real*-number operation in *AIM*. Cf. “*illegal input data type for this operation*”.
`Out of range`	8	{1, 2, 3, 10} • A number exceeds the valid range. This can be caused by specifying more decimals than 34, *word size* > 64, short integers ≥ 2^{64}, invalid *dates* or *times*, denominators > 9 999, etc. • A *register* or *flag* address exceeds the valid range of currently allocated *registers* or *flags*. May also happen in indirect addressing or when calling nonexistent local addresses. • An R-operation (like R-COPY) attempts accessing invalid *register* addresses.
`Output would exceed 196 characters`	33	{7} Maximum length of *alphanumeric strings*.

	EC	Explanations, countermeasures and examples
`Overflow at +∞`	4	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a number >0 by 0. • Divergent sum or product or integral. • Positive overflow (see p. 158).
`Overflow at -∞`	5	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a number <0 by 0. • Divergent sum or product or integral. • Negative overflow (see p. 158).
`Please enter a NEW name`	26	Attempt to define a new variable or user *menu* with a *name* 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. 154ff for the space required by different *DTs*). May happen also in program execution due to dynamic allocations (see *Section 3* of the *OM*).
`Singular matrix`	22	{8, 9} • 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 *registers* that would overlap the *stack* (see *Section 1* of the *OM*). Will happen with SSIZE8 set and STOS 93, for example.
`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.
`This variable is write protected`	37	Self-explanatory.
`Too few data points for this statistic`	15	{2} A statistical calculation was attempted with too few data, e.g. mean or *standard deviation* for < 2 points, regression for < 2 or 3 points, s_{mi} for < 30 points.
`Undefined op-code`	3	An instruction with an undefined operation code occurred. Should never happen – but who knows?
`Undefined source variable`	36	Try to recall a variable which is not defined yet.
`Word size is too small`	14	{10} User input or *register* content is too great to be handled by the *word* size currently set.
	25	Left unused for *WP 34S* compatibility
	27	Left unused since used earlier in this project

If SPCRES is set, errors 4 and 5 will not occur at all, and error 1 will happen less frequently, since ±∞ and NaN are legal results then (cf. the corresponding entries in CONST on pp. 133ff and the tables on pp. 163ff). E.g., [0] [ln] will return –∞ then.

Each error message will be displayed in **Z** numeric row and is *temporary information* (see *Section 2* of the *OM*). So [←] 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.


WP 34S
RPN SCIENTIFIC
34S 3.02310
PAULI, WALTE
A B C D CPX
STO RCL R↓ f g h
ENTER↑ x≷y +/− EEX ←
XEQ 7 8 9 ÷
▲ 4 5 6 ×
▼ 1 2 3 −
EXIT 0 . R/S +

HEWLETT PACKARD
42S
RPN SCIENTIFIC
x: 812.6000
FIX SCI ENG ALL
Σ+ 1/x √x LOG LN XEQ
STO RCL R↓ SIN COS TAN
ENTER x≷y +/− E ←
▲ 7 8 9 ÷
▼ 4 5 6 ×
1 2 3 −
EXIT 0 • R/S +
ON

FAdCbAd9 .h
A B C D E F 7 8 9 ÷
GSB GTO HEX DEC OCT BIN 4 5 6 ×
R/S SST R↓ x≷y BSP ENTER 1 2 3 −
ON f g STO RCL 0 • CHS +
HEWLETT · PACKARD

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 view. 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. 187, one for the *HP-21S* on p. 189, and another one for the *WP 34S* on p. 191. Functions newly introduced with *WP* calculators are compiled on pp. 195ff.

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*.

HP-42S	***WP 43S***	**Remarks**
ACOSH	arcosh	In EXP
ADV	⎙ADV	In PRINT
AIP	Dispensable	You can merge text and numeric data easily using [+] as described in *Section 2* of the *OM*.
ALENG	αLENG	In α.FN
ALLΣ	Dispensable	Your *WP 43S* runs in ALLΣ mode always. The summation *registers* do not overlap with *GP registers*.
ALPHA	α	See the description of *AIM* in *Sect. 2* of the *OM*.

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

HP-42S	*WP 43S*	**Remarks**
CLV	See remark	Variables are cleared as specified in *Section 6* of the *OM.*
COMPLEX	CC	You can also enter *complex numbers* directly using CC as explained in *Section 2* of the *OM*.
CONVERT	L→ & PARTS	
CUSTOM	n/a	You can create as many *menus* as memory will hold – not only one CUSTOM *menu*. See *Section 6* of the *OM*.
DECM	Dispensable	Any input featuring a . or an E is interpreted as a *real* (decimal) number.
DEL	n/a	Not featured. Too dangerous, in our opinion.
DELAY	🖶DLAY	In PRINT
DELR	M.DELR	In MATX
DET	|M|	
DIM	M.DIM	
DIM?	M.DIM?	
EDIT	M.EDI	
EDITN	M.EDIN	
FCSTX	x̂	In STAT
FCSTY	ŷ	
FNRM	ENORM	In MATX. Euclid is older than Frobenius.
GAMMA	Γ(x)	In PROB
GETKEY	KEY?	In P.FN
GETM	M.GET	In MATX
GROW	M.GROW	
HEXM	Dispensable	Press # H for converting any closed integer number or integer part in x to hexadecimal.
H.MS+	Dispensable	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 MATX
INTEG	∫	In ADV
INVRT	$[M]^{-1}$	In MATX
KEYASN	Dispensable	Not needed since no CUSTOM *menu* is featured (see CUSTOM).
LASTx	RCL L	
LBL		Press LBL.
LCLBL	Dispensable	Obsolete since no CUSTOM *menu* is featured (see CUSTOM). Nevertheless, your *WP 43S* provides local labels (see *Section 3* of the *OM*).
LINΣ	Dispensable	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 *startup default* here.
MAT?	MATR?	In TEST
MEAN	$\bar{x}$	In STAT
MOD		Press MOD.
MODES	MODE	
N!	x!	
NEWMAT	M.NEW	In MATX
NORM	n/a	Not featured.
OCTM	Dispensable	Press # 8 for converting any closed integer number or integer part in x to octal.
OLD	RCLEL	In MATX
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	αPOS	In α.FN
PRA	🖶r K	In PRINT

HP-42S	***WP 43S***	**Remarks**
PRGM	P/R	
PRLCD	🖶LCD	In PRINT
PROFF	CF T	
PROMPT	Dispensable	Use VIEW , STOP instead.
PRON	SF T	
PRP	🖶PROG	In PRINT
PRSTK	🖶STK	In PRINT
PRUSR	🖶USER	In PRINT
PRV	🖶r	In PRINT
PRX	🖶x	In PRINT
PRΣ	🖶Σ	In PRINT
PUTM	M.PUT	In MATX
PWRF	PowerF	In STAT
RAN	RAN#	In PROB
RND	ROUND	In PARTS
RNRM	RNORM	In MATX
ROTXY	RL, RLC, RR, and RRC	In BITS
RTN	RTN	Press RTN.
SDEV	s	In STAT
SIZE	Dispensable	There are 100 global *GP registers* always.
SLOPE	L.R.	In STAT
SOLVE	SLV	In ADV
SQRT	√x	
SST	≡V (▼)	Shortcut works if no *multi-view menu* is open.
STR?	STRI?	In TEST
TOP.FCN	Dispensable	Obsolete – no top functions are overwritten.
TRACE	SF T	

HP-42S	*WP 43S*	**Remarks**
TRANS	[M]^T	In MATX
UVEC	UNITV	In MATX and CPX
VARMENU	VARMNU	Truncated to 6 characters to fit the *menu* space.
VIEW		Press VIEW.
WMEAN	$\bar{x}_w$	In STAT
WRAP	M.WRAP	In MATX
XTOA	x→α	The conversion is done in **X**.
X<0?, X<Y?	x< ?	In TEST
X≤0?, X≤Y?	x≤ ?	
X=0?, X=Y?	x= ?	
X≠0?, X≠Y?	x≠ ?	
X≥0?, X≥Y?	x≥ ?	
X>0?, X>Y?	x> ?	
YINT	L.R.	In STAT
y^x		Press y^x.
ΣREG	Dispensable	There are 100 global *GP registers* always. Statistical registers are separate.
ΣREG?		
→DEC	→INT 10	Press # D ... for closed input.
→HR		Press .d
→H.MS		Press h.ms
→OCT	→INT 8	Press # 8
→POL		Press →P.
→REC		Press R←.
%CH	Δ%	Press Δ%.
÷	/	Cf. *ISO 80000-2*: "The symbol ÷ should not be used."

Corresponding Operations on *HP-16C*

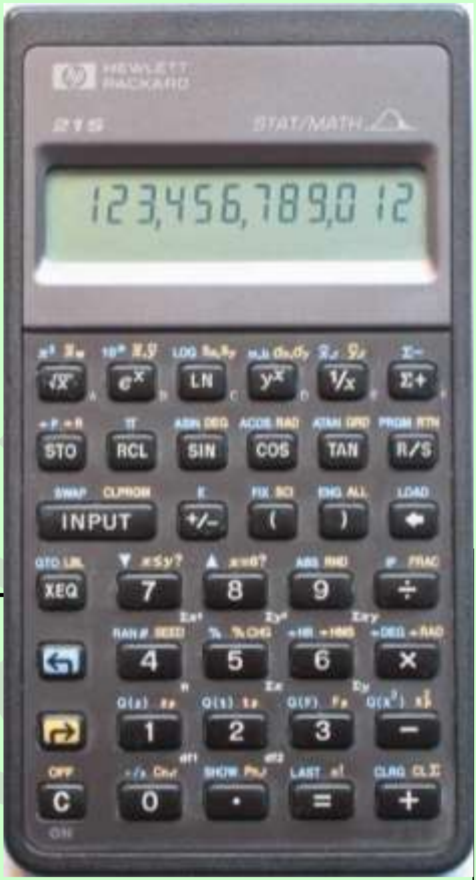
The table for the functions of the *HP-16C* is sorted following its keyboard layout, starting top left. As for the *HP-42S*, only functions carrying different names on both calculators are listed.

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

HP-16C	*WP 43S*	**Remarks**
CLEAR PRGM	CLP	In CLR. Note here is also CLPALL.
CLEAR REG	CLREGS	In CLR
CLEAR PREFIX	Dispensable	See *Section 2* of the *OM*.
WINDOW	Dispensable	64 *bits* can be displayed in one row.
SET COMPL 1S	1COMPL	In MODE and BITS. Note here is also SIGNMT.
SET COMPL 2S	2COMPL	
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≤ ?	In TEST. Note far more tests are covered here.
x<0	x< ?	
x>y , x>0	x> ?	
FLOAT	FIX	In DISP
MEM	STATUS	In FLAGS
CHS	+/_	
◀, ▶	Dispensable	64 *bits* can be displayed in one row.
LSTX	RCL L	
x≠y , x≠0	x≠ ?	In TEST. 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.

HP-21S	*WP 43S*	Remarks
$\bar{x}_w$	$\bar{x}_w$	
$\bar{x},\bar{y}$	$\bar{x}$	
s_x,s_y	s	
m,b	L.R.	In STAT
σ_x,σ_y	σ	
$\hat{x}$,r	r, $\hat{x}$	
$\hat{y}$,r	r, $\hat{y}$	
PRGM	P/R	
SWAP	x⇄y	
CLPRGM	CLP	In CLR
INPUT	Dispensable	This functionality is contained in ENTER. Also your *WP 43S* features a command called INPUT but this works in programs.
(,)	Dispensable	You can forget these keys in *RPN.*
LOAD	n/a	Loads predefined programs in the *HP-21S*. Also your *WP 43S* features a command called LOAD but this recalls data from backup.

HP-21S	*WP 43S*	**Remarks**
ABS	|x|	In PARTS
RND	ROUND	
FRAC	FP	
÷	/	Cf. *ISO 80000-2*: "The symbol ÷ should not be used."
SEED	SEED	In PROB
%CHG	Δ%	
Q(z)	Norml_e	In *submenus* of PROB. Note your *WP 43S* features the *normal distribution* for arbitrary *μ* and *σ* instead of the standardized one (*μ*=0, *σ*=1). And the implementation of the inverse distribution functions deviates on both calculators: your *WP 43S* calculates with the probability *P* while the *HP-21S* calculates with the error probability *Q* = 1 – *P* as input (cf. the *OM*, *Section 2*). Labeling these functions on the *HP-21S* using the letter *p* may add some confusion.
z_p	Norml^{-1}	
Q(t)	$t_\Delta(x)$	
t_p	$t^{-1}(p)$	
Q(F)	$F_\Delta(x)$	
F_p	$F^{-1}(p)$	
$Q(\chi^2)$	$\chi^2{}_\Delta(x)$	
$\chi^2{}_p$	$(\chi^2)^{-1}$	
Cn,r	COMB	In PROB
Pn,r	PERM	
LAST	RCL L	
=	Dispensable	You can forget this key in *RPN.*
n!	x!	
CLRG	CLREGS	In CLR

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: your *WP 43S* allows for *softkeys* – the *WP 34S* can only carry four *hotkeys* instead. Also dealing with matrices is greatly eased by the relatively 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	∡	
Binom	Binom◭	
Binom$_u$	Binom◮	
Binom	Binom◭	
Cauch$_u$	Cauch◮	
CLα	**0** STO K	Check the *OM* for the conditions when this register is used.
CONST	CNST	For keyboard space reasons.
CONV	U→	
DBLOFF	Dispensable	Your *WP 43S* features *34-digit DTs* per default – it does neither need nor feature any *double precision* mode.
DBLON		
Expon	Expon◭	
Expon$_u$	Expon◮	
F(x)	F◭(x)	
F$_u$(x)	F◮(x)	
dRCL	Dispensable	Cf. DBLOFF.

WP 34S	*WP 43S*	**Remarks**
gCLR, gDIM, gDIM?, gFLP, gPIX?, gPLOT, gSET	n/a	The *LCD* 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* plus POINT and PLOT.
Geom	Geom◭	
Geom$_u$	Geom◮	
GTOα	Dispensable	Use GTO with an appropriate parameter instead.
H.MS+, H.MS–	Dispensable	Your *WP 43S* features a dedicated *DT* for *times*, so + and − suffice for adding or subtracting sexagesimal times, respectively.
INTM?	Dispensable	Your *WP 43S* features dedicated *DTs* for integers – it does neither need nor feature an integer mode.
iRCL	Dispensable	Your *WP 43S* features various *data types*.
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$_{n\alpha}$	L$_{m\alpha}$	Renamed in consequence to L$_m$.
Logis	Logis◭	
Logis$_u$	Logis◮	
MROW+×, MROW×	Dispensable	Obsolete matrix commands.
MROW⇆	M.R⇄R	
M+×	Dispensable	Obsolete matrix command.
M^{-1}	[M]$^{-1}$	

WP 34S	*WP 43S*	**Remarks**
M-ALL, M-COL, M-DIAG, M-ROW	Dispensable	Obsolete matrix commands.
M×	Dispensable	Your *WP 43S* features two dedicated *DTs* for matrices. Thus you can multiply matrices using [×] and copy matrices like any other objects.
M.COPY		
M.IJ, M.REG	Dispensable	Obsolete matrix commands.
nBITS	#B	
nCOL, nROW	Dispensable	Obsolete matrix commands.
Norml	Norml△	
Norml$_u$	Norml△	
Poiss	Poiss△	
Poiss$_u$	Poiss△	
REALM?	Dispensable	Your *WP 43S* features a dedicated *DT* for *reals* – it does not need a *real* mode.
REGS, REGS?	Dispensable	The number of global *GP registers* is fixed to 100 on your *WP 43S.*
SENDA, SENDP, SENDR, SENDΣ	SEND	SEND combines all those four commands of the *WP 34S*.
SEPOFF, SEPON	GAP	
SHOW	RBR	
sRCL	Dispensable	Cf. DBLOFF.
TRANSP	[M]^T	
TSOFF	GAP 0	
TSON	GAP 3	
t(x)	t△(x)	
t$_u$(x)	t△(x)	

WP 34S	*WP 43S*	Remarks
VIEWα, VWα+	Dispensable	Use VIEW instead; *alphanumeric strings* are just another *DT*. Combine text and numeric data easily using + as shown in the OM, *Section 2.*
Weibl	Weibl$_{\blacktriangle}$	
Weibl$_u$	Weibl$_{\blacktriangle}$	
XEQα	Dispensable	Use XEQ with an appropriate parameter instead.
XTAL?	Dispensable	A quartz crystal is installed by default.
YDOFF, YDON	Dispensable	Your *WP 43S* displays ***y*** whenever possible and wanted. See DSTACK.
αDATE, αDAY	Dispensable	You can combine text and numeric data easily using + as shown in *Section 2* of the *OM.*
αGTO	Dispensable	Use GTO w/ an appropriate parameter instead.
αIP, αMONTH	Dispensable	Cf. αDATE.
αRCL, αRC#	Dispensable	Your *WP 43S* features various *data types* and 'knows' which type is in the *register* specified. Appending texts is done by +.
αSTO	Dispensable	Simply press STO instead (any *register* can take a *text string*).
αTIME	Dispensable	Cf. αDATE.
αXEQ	Dispensable	Use XEQ with an appropriate parameter instead.
β	β(x,y)	
Γ	Γ(x)	
ΔDAYS	Dispensable	Simply subtract two *dates*.
ζ	ζ(x)	
Φ(x) …	Dispensable	Use NORML… with $\mu=0$ and $\sigma=1$ instead.
$\chi^2(x)$	$\chi^2_{\blacktriangle}(x)$	
$\chi^2_u(x)$	$\chi^2_{\blacktriangle}(x)$	
→H	→HR	
🖶PLOT	n/a	See gCLR.

WP 34S	*WP 43S*	Remarks
🖶$^{C}r_{XY}$	Dispensable	Use 🖶r instead.
🖶α, 🖶α+, 🖶+α	Dispensable	Combine text and numeric data easily using [+] as shown in *Section 2* of the *OM.* Then use 🖶r.
🖶?	Dispensable	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**	—	—
$Binom_p$ $Binom_{⊿}$ (of *Binomial distribution*)	●	●	new
B_n B_n^* CEIL FLOOR	●	—	new
$Cauch_p$ $Cauch_{⊿}$ $Cauch_{⊿}$ $Cauch^{-1}$	●	●	new
CauchF GaussF HypF ParabF RootF	**new**	—	—

Command	***WP 43S***	*WP 31S*	*WP 34S*
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
$Expon_P$ $Expon_{\triangle}$ $Expon_{\triangle}$ $Expon^{-1}$	●	●	new
EXPT MANT	●	—	new
FBR	**new**	—	—
FC?F FC?S FF FS?F FS?S	●	—	new
FIB	●	—	new
FILL	●	●	new
FLASH? FP?	●	—	new
$F_P(x)$ $F_{\triangle}(x)$ (of *F distribution*)	●	●	new
f' f''	**new**	—	—
f'(x) f''(x)	**extended**	—	new
GAP	**extended**	●	new

Command	*WP 43S*	*WP 31S*	*WP 34S*
GCD LCM	●	●	new
g_d g_d^{-1}	●	—	new
Geom_p Geom_{Δ} Geom_{Δ} Geom^{-1}	●	—	new
H_n H_{nP} L_m $L_{m\alpha}$ P_n T_n U_n	●	—	new
Hyper_p Hyper_{Δ} Hyper_{Δ} Hyper^{-1}	**new**	—	—
IDIV	●	—	new
IDIVR IM RE	**new**	—	—
INT? ISM? I_{xyz} $I\Gamma_p$ $I\Gamma_q$	●	—	new
$J_y(x)$ J/G?	**new**	—	—
J/G	**extended**	●	new
KEY? KTYP? LBL? LEAP?	●	—	new
LgNrm_p LgNrm_{Δ} LgNrm_{Δ} LgNrm^{-1}	●	—	new
LNβ LNΓ LOADP LOADR LOADSS LOADΣ LocR LocR? LOG_2 $\mathrm{LOG}_x y$	●	—	new
LOAD SAVE	●	●	new
Logis_p Logis_{Δ} Logis_{Δ} Logis^{-1}	●	●	new
max min MIRROR	●	—	new
MOD	●	●	new
MULπ MULπ→	**new**	—	—
M.LU M.SQR? NAND NaN? NEIGHB NOR	●	—	new
NBin_p NBin_{Δ} NBin_{Δ} NBin^{-1}	**new**	—	—
NEXTP PRIME?	**extended**	●	new
Norml_p Norml_{Δ} Norml_{Δ} Norml^{-1}	●	●	new
nΣ (callable by name)	●	●	new
OrthoF PLOT POINT	**new**	—	—
PAUSE	●	—	extended
Poiss_p Poiss_{Δ} Poiss_{Δ} Poiss^{-1}	●	●	new
PopLR PRCL PSTO PUTK	●	—	new

Command	***WP 43S***	*WP 31S*	*WP 34S*
RANGE RANGE? RANI#	**new**	—	—
RBR	●	●	new
RCLCFG STOCFG	**extended**	—	new
RCLS STOS	●	—	new
RCL↑ RCL↓ STO↑ STO↓	●	—	new
RDP RECV SEND	●	—	new
Re⇄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**	—	—
TICKS	●	—	new
TIMER	●	—	(●)
TOP? ULP?	●	—	new

Command	*WP 43S*	*WP 31S*	*WP 34S*
$t_p(x)$ $t_\Delta(x)$ (of *t distribution*)	●	●	new
t≷ y≷ z≷ ≷	●	—	new
(UNDO)	●	new	—
V∡	**new**	—	—
VERS? WDAY WHO?	●	●	new
$Weibl_p$ $Weibl_\Delta$ $Weibl_\Delta$ $Weibl^{-1}$	●	●	new
W_m W_p W^{-1} WSIZE? $\bar{x}_G$ XNOR	●	—	new
$\bar{x}_H$ x_{max} x_{min} $\bar{x}_{RMS}$ x→DATE	**new**	—	—
x< ? x≤ ? x= ? x≠ ? x≥ ? x> ?	**extended**	—	extended
x= +0? x= -0? x≈ ?	●	—	new
y^x	**extended**	●	●
Y.MD	●	●	new
αLENG?	**extended**	—	●
αPOS?	**extended**	—	—
αRL αRR αSL αSR	●	—	extended
β(x,y) Γ_{xy} γ_{xy} ε ε_m ε_p ζ(x) Π_n Σ_n σ_w	●	—	new
Σ1/x $\Sigma 1/x^2$ Σ1/y $\Sigma 1/y^2$ Σlny/x $\Sigma x^2/y$ Σx^3 Σx^4 Σx/y	**new**	—	—
Σln^2x Σln^2y Σlnx Σlnxy Σlny Σx Σx^2 Σx^2y Σxlny Σxy Σy Σylnx Σy^2 (callable by names)	●	●	new
$\chi^2_p(x)$ $\chi^2_\Delta(x)$ (of *chi-square distribution*)	●	●	new
$(-1)^x$ ×MOD ^MOD	●	—	new
±∞?	**new**	—	—
→DEG →RAD	●	●	new
→D.MS →MULπ	**new**	—	—
→GRAD	●	—	new
→INT →REAL	**new**	—	—

Command	***WP 43S***	*WP 31S*	*WP 34S*
🖶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	*HP-21S Owner's Manual*, especially pp. 63 – 105.[85]
Manipulating bits and *short integers*	*HP-16C Owner's Handbook*[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
Programming	*HP-42S Owner's Manual* [87] *HP-42S Programming Examples & Techniques* [91]
Root finding and numeric integration	*HP-34C Owner's Handbook & Programming Guide* [88] *HP-15C Owner's Handbook* [89] *HP-15C Advanced Functions Handbook* [90] *HP-42S Programming Examples & Techniques* [91]
Accuracy of numeric calculations	*HP-15C Advanced Functions Handbook,* pp. 172 – 211.[90]
Financial calculations	*HP-17BII+ User's Guide* [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 101 on p. 222 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 (see https://en.wikipedia.org/wiki/Floating-point_arithmetic as a starter, also about floating point numbers in general).

[87] Download from http://www.hp41.net/forum/fileshp41net/manuel-hp42s-us.pdf

[88] 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

[89] Download a reprint from http://h10032.www1.hp.com/ctg/Manual/c03030589.pdf

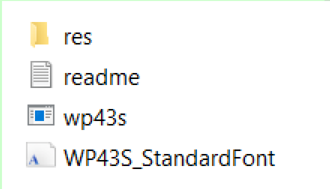
[90] Download a reprint from http://h10032.www1.hp.com/ctg/Manual/c03308725.pdf

[91] Download from https://literature.hpcalc.org/community/hp42s-prog-en.pdf

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

APPENDIX E: EMULATING A *WP 43S* ON YOUR COMPUTER

Under Windows, you can employ the *WP 43S* simulator, featuring exactly the same functionality as the calculator. You find it under https://gitlab.com/Over_score/wp43s/-/releases. You will be asked to open a zipped folder. Extract it to any suitable location on your *PC*. Open its root directory and you'll find this:

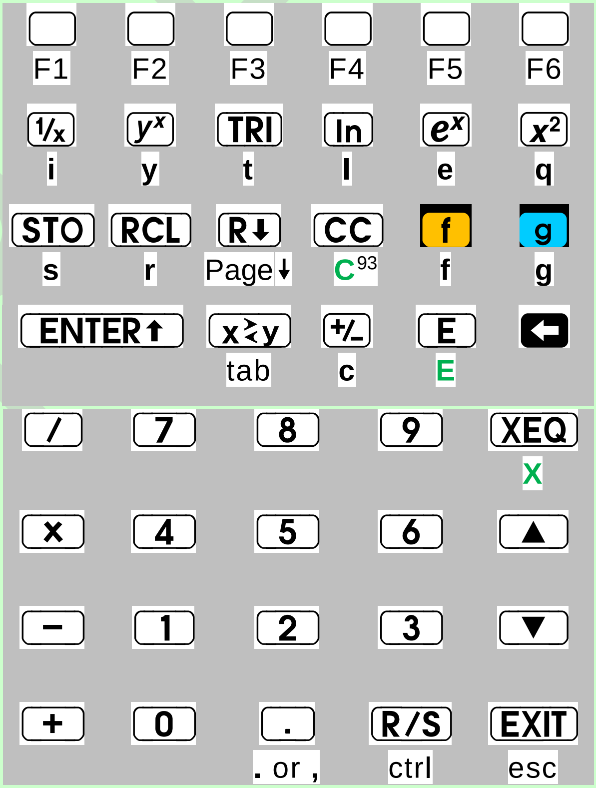


Start `wp43s` and the simulator window will open, looking like one of the two pictures following though larger.

Operate the simulator with the mouse. The ten digits as well as [.], [ENTER↑], [+], [−], [×], and [/] may also be entered via the numeric keypad of your computer directly, [▲] and [▼] via the cursor keys, and [⬅] via [←Backspace].

Further computer keyboard short-cuts to simulator keys are present-ed on next page.

(A vertical screen size of ≥ 980 px is required for the portrait window; else the landscape window overleaf will open needing 1000 × 568 px. This second picture shows an older calcu-lator keyboard.)



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

[93] Capitals are printed green here for better differentiation.

Pressing …

… **h** copies the entire simulator screen image to the clipboard.

… **x** copies the full content of **X** thereto.[94]

… **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 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*).

---

[94] This is the way to output also 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 then. This way is explained in next chapter.

The chapter thereafter (beginning on p. 207) 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*

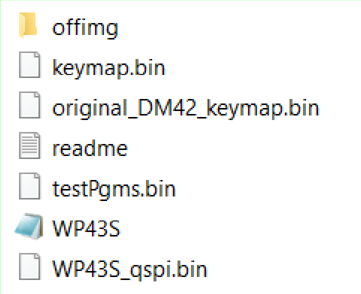
1. Start your computer. Take a *DM42* and turn it on; then press

   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 fetch the firmware you find under https://gitlab.com/Over_score/wp43s/-/releases. You will be asked to open a zipped folder. Extract it to any suitable

location on your computer. Open its root directory and you'll find this:



3. Copy WP43S.pgm and WP43S_qspi.bin onto 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.)

4. Press SETUP 5 2 4 3 WP43S.pgm ENTER↑ ENTER↑. Wait some 15 s for flashing completed. Then press EXIT EXIT 1 EXIT. Then, your *WP 43S* is up and waiting for your commands.

As long as you rely on a converted *DM42*, WP43S_layout.svg may ease your life. Print it, cut, and apply the overlay (see p. 208); the picture indicates the keys for which stickers are needed.

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 will return to the bottom left key.

To retrieve the original *DM42* keyboard layout (cf. p. 150), 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.[95]

3. Connect your computer to your calculator *Micro USB* socket using a suitable data cable.[96] The flash disk of your calculator should show up as an external mass storage volume on your computer after you pressed …
   6 Activate *USB* Disk.

4. Start the internet browser on your computer and go to https://gitlab.com/Over_score/wp43s/-/releases. You will be asked to open a zipped folder (cf. p. 206). Extract it to any suitable location on your computer; open its root directory and copy WP43S.pgm and WP43S_qspi.bin to the root directory of the *DM42* flash disk.[97]

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

---

[95] 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. 151).

[96] Ensure it connects properly to the calculator – cut the insulation back a bit if necessary.

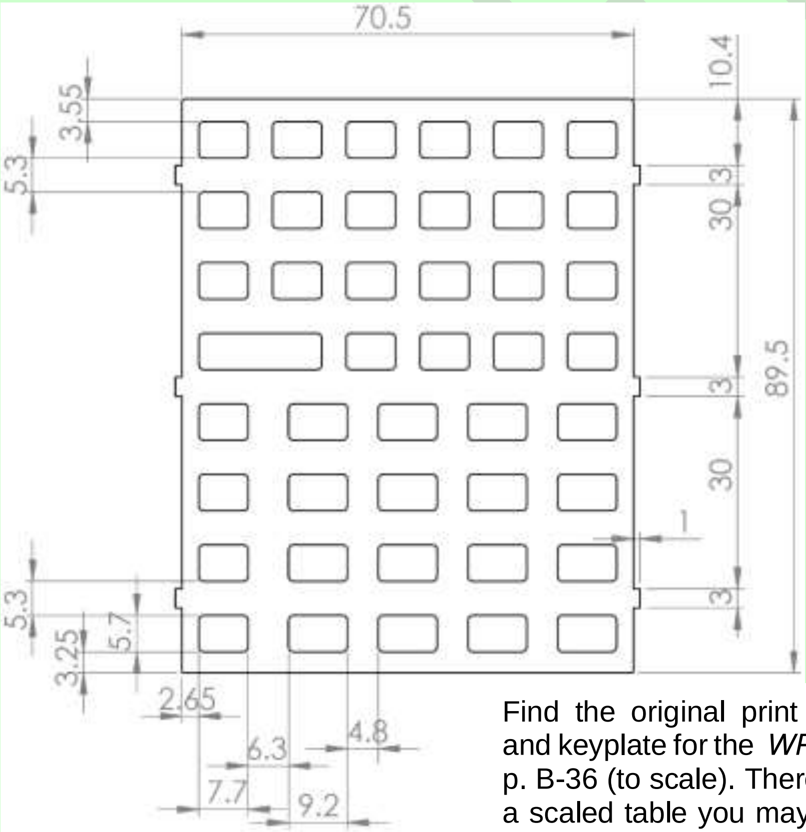
[97] Option: Copy keymap.bin to the flash disk if you have not done so far. This will reassign the keys to match the *WP 43S* layout also when leaving *WP 43S*. Else EXIT/ON remains bottom left.

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-36 (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, and the battery voltage is ok then 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. 153 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.

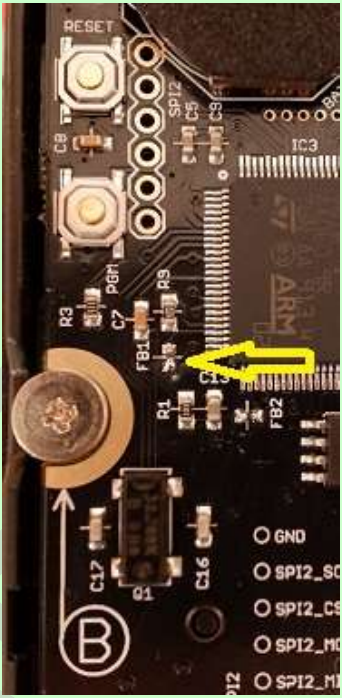
   This shall reset your *DM42* and put it in bootloader mode.[98]

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

---

98 If this method should not work, however, this may point to a real hardware problem. Read also the other trouble cases in this appendix. If nothing applies, 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 **g** **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. 251 for $\zeta(x)$.

Name	Remarks (see pp. 12ff for general information)
COMB, PERM	For $y \geq x \geq 0$ and $x, y \in \mathbb{N}$, $C_{y,x} = \binom{y}{x} = \frac{y!}{x! \cdot (y-x)!}$ is the number of *combinations* and $P_{y,x} = \frac{y!}{(y-x)!} = x! \cdot C_{y,x}$ the number of *permutations* of x and y as explained in the *IOP* (see pp. 25 and 55, respectively). Note $C_{y,0} = 1$, $C_{y,1} = y$, and $C_{y,2} = \frac{1}{2} y(y-1)$. $C_{y,x}$ applies to the *binomial distribution* (see p. 213): In a *Galton box*[99] (a.k.a. *bean machine*) 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). 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 *complex* domain.
FIB	For integers, FIB returns the *Fibonacci* number f_n with $n = x$. The *Fibonacci* 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 *long integers*, f_{4791} is the maximum on your *WP 43S.* For non-integers, FIB returns the extended Fibonacci number $$F_x = \frac{1}{\sqrt{5}} [\Phi^x - \Phi^{-x} cos(x\,\pi)]$$ for an arbitrary *real* or *complex number* x, with $\Phi = \frac{1+\sqrt{5}}{2}$ denoting the *golden ratio*.

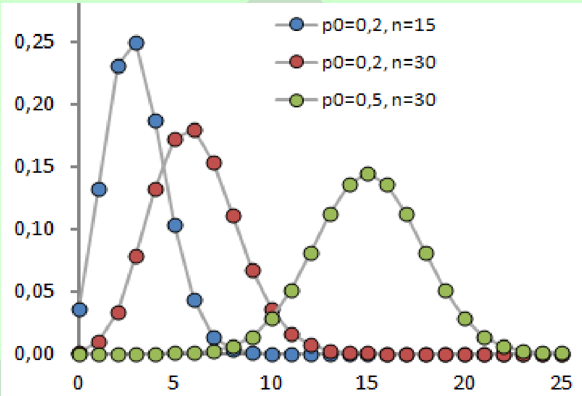
[99] Translator's note: This is called «Planche de Galton» in French, "Galtonbrett" in German, and "macchina di Galton" in Italian. Note the subtle differences in naming. Galton invented his box in 1889.

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

$$p_B(g;n;\ p_0) = \binom{n}{g}\ p_0^g\ (1-p_0)^{n-g} = C_{n,g}\ p_0^g\ (1-p_0)^{n-g}$$

(see COMB on p. 212 for the explanation of the notation).

BINOM$_{\Delta}$ 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:

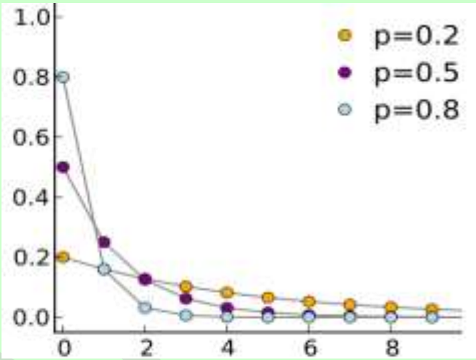
What is the probability for finding no fault in a sample of 15 items drawn from a batch of 300 wherein you expect 3% defective items overall? This will tell you:

.03 STO I **15** STO J **0** PROB g Binom: Binom$_{\Delta}$

… returning 0.633 – so the odds are almost two out of three that you will not detect any defect in your sample! [100]

Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm .

Geom: *Geometric distribution*:



GEOM$_P$ returns

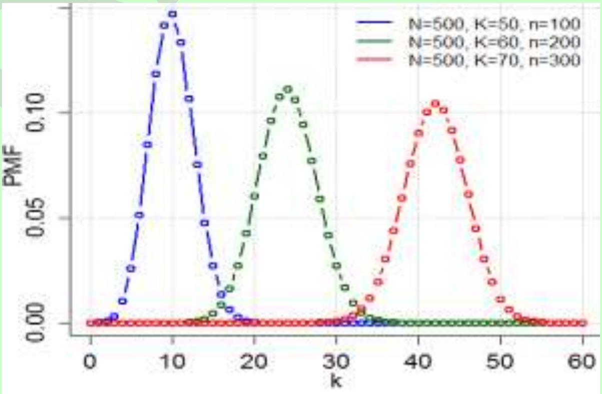
$p_{Ge}(n) = p_0 (1 - p_0)^n$

GEOM$_\Sigma$ returns

$F_{Ge}(m) = 1 - (1 - p_0)^{m+1}$, being the probability for a first success after $\boldsymbol{m} = \boldsymbol{x}$ *Bernoulli* experiments. The probability $\boldsymbol{p_0}$ 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* $\boldsymbol{g}$ in **X**, *gross probability of a success* $\boldsymbol{p_0}$ in **I**, *sample size* $\boldsymbol{n}$ in **J**, and batch size $\boldsymbol{n_0}$ in **K** (in the diagram here, $\boldsymbol{g}$ = k, $\boldsymbol{p_0}$ = K / N, and $\boldsymbol{n_0}$ = N).



[100] The exact result for said boundary conditions is 0.626, calculated via the *hypergeometric distribution*. These results show nicely that two significant digits are a typical accuracy of theoretical statistical statements to compare with real data – frequently the (often simplified) statistical model used matches reality no better than that.

HYPER$_P$ 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. 212 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 (***n₀*** > 10) and for small sample sizes (<10% of ***n₀***). 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* ***p₀*** in **I**, and ***n*** in **J**.

NBIN$_P$ returns $p_{NB}(f;\ n;\ p_0) = \binom{n-1}{f-1}\ p_0^f\ (1-p_0)^{n-f} =$

$C_{n-1;f-1}\ p_0^f\ (1-p_0)^{n-f}$ (s. COMB on p. 212 and cf. BINOM).

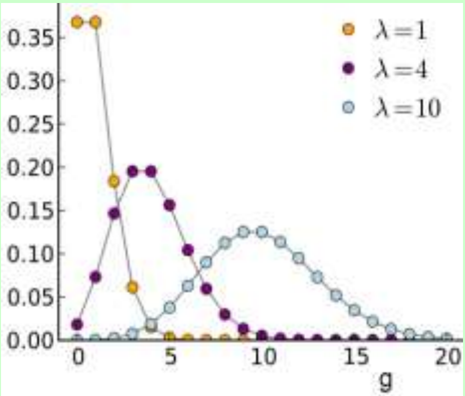
Start reading here for more:
http://en.wikipedia.org/wiki/Negative_binomial_distribution .

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

POISS$_P$ 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**.

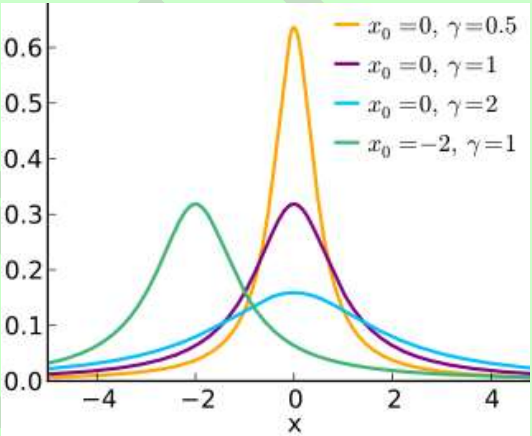


The *Poisson distribution* provides the mathematically simplest model for industrial sampling tests – use $\lambda = np_0$ with the gross error probability $\boldsymbol{p_0}$ and the sample size $\boldsymbol{n}$ (cf. BINOM). For the example introduced with BINOM above, POISS returns 0.638.

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

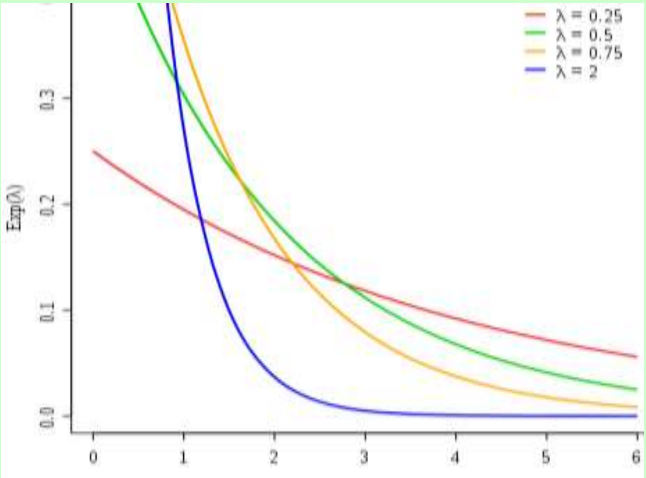
CAUCH^{-1} returns $F_{Ca}^{-1}(p) = x_0 + \gamma \tan\left[\pi \cdot \left(p - \frac{1}{2} \right) \right]$.

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 $\mathbf{I}$.

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

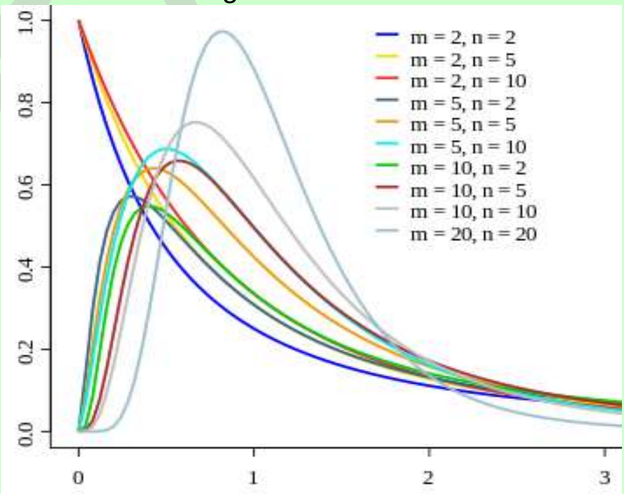
EXPON returns $F_{Ex}(x) = 1 - e^{-\lambda x}$.



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 $\mathbf{I}$ and $\mathbf{J}$. 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***.

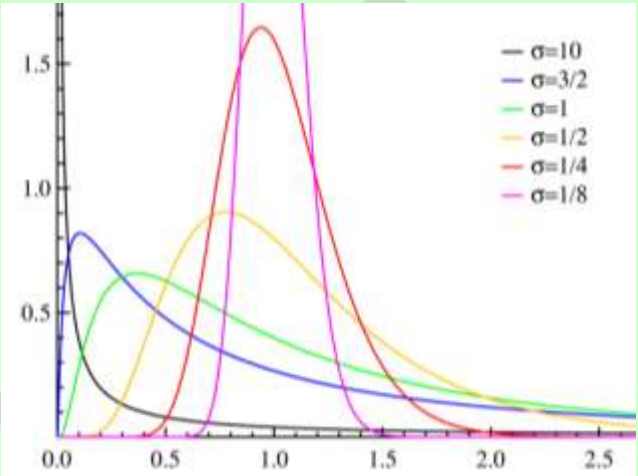


Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3665.htm

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).

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

LGNRM$_{\blacktriangle}$ returns $F_{Ln}(x) = \Phi\left(\frac{ln(x)-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized normal CDF* as presented on p. 219.



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**.

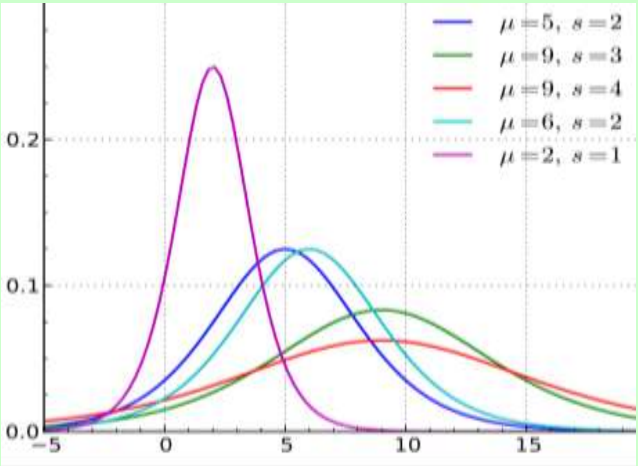
Substituting $\xi = \frac{x-\mu}{s}$,

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

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

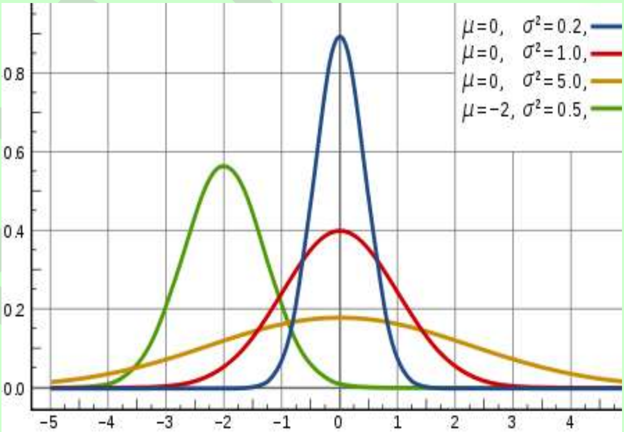
LOGIS^{-1} returns $F_{Lg}^{-1}(p) = \mu + s\,ln\left(\frac{p}{1-p}\right)$.

Start reading here for more:



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

Norml: *Normal distribution* with a *mean* $\boldsymbol{\mu}$ given in **I** and a *standard deviation* $\boldsymbol{\sigma}$ in **J**. The red curve (for $\boldsymbol{\mu}=0$ and $\boldsymbol{\sigma}=1$) represents the *standardized normal distribution* φ.



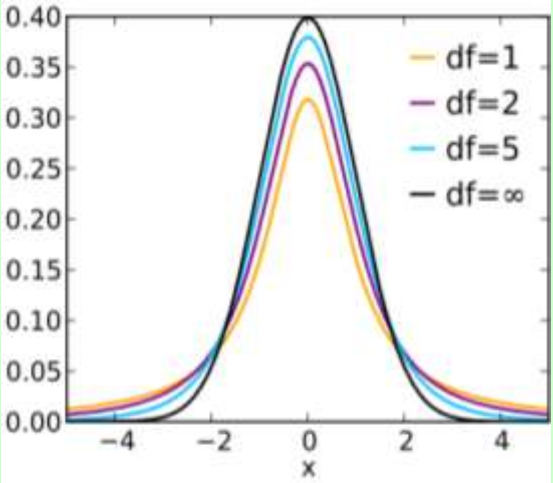
NORML$_P$ returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \varphi\left(\frac{x-\mu}{\sigma}\right)$ and

NORML$_\Delta$ 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. 247).

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 $t(\mathbf{x})$ shrink and it approaches the *PDF* of the *standardized normal distribution* (compare the red curve at NORML on p. 219).



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**.

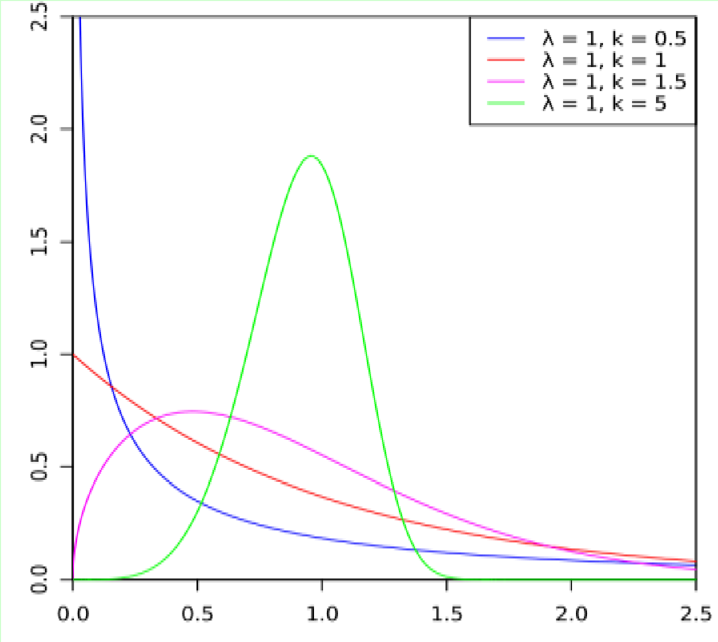
WEIBL$_P$ returns $f_W(t) = \frac{b}{T} \cdot \left(\frac{t}{T}\right)^{b-1} e^{-\left(t/T\right)^b}$ for $t \geq 0$, else 0. This is a very flexible function – see the curves plotted overleaf.

WEIBL$_{\Sigma}$ returns $F_W(t) = 1 - e^{-\left(t/T\right)^b}$

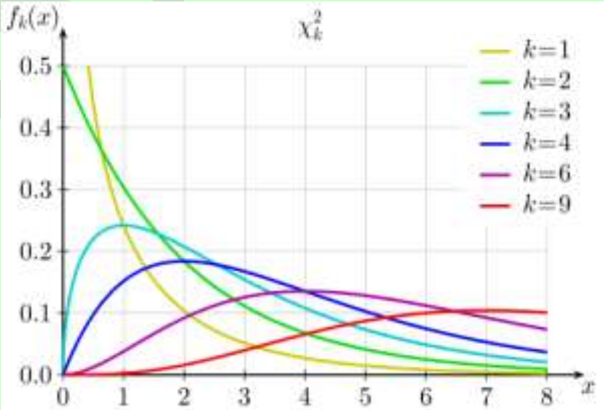
This distribution is widely used e.g. for analyzing tool and product lifetimes.

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 .



χ²(x): *Chi-square distribution* with its *degrees of freedom* given in **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 $\overline{x}$ 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 $\overline{x}_g$ 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.[101] Be assured not everything is *normal* (*Gaussian*) in real world.[102] 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.

---

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

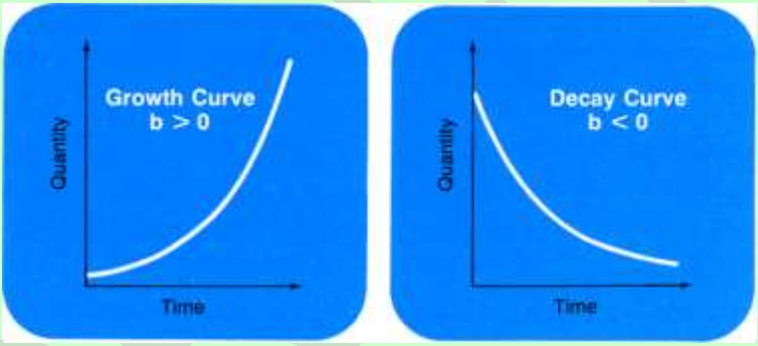
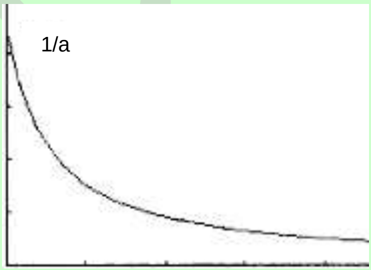
As mentioned in the main text, we recommend you look deeply into statistics textbooks to ensure you fully understand what you do with the functions provided in your *WP 43S*. The real world shows lots of sad examples where people full of good will caused large damages by applying tools they did not know 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).

[102] 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 <u>STAT</u>:

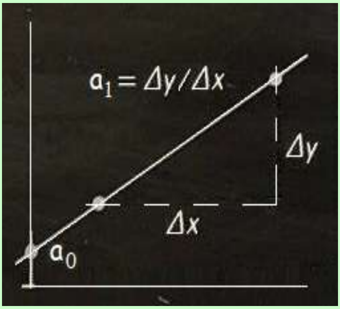
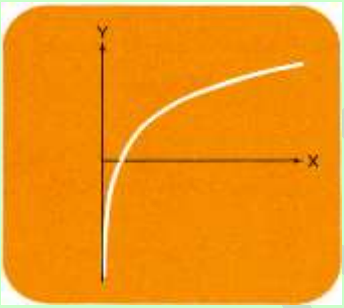
Name	Remarks (see pp. 12ff for general information)
CauchF	Selects a *Cauchy* (a.k.a. *Lorentz*, *Breit-Wigner*) peak fit model $$R(x) = \frac{1}{a_0\,(x+a_1)^2 + a_2}$$ for least squares regression.[103] See p. 216 for shapes of such peaks.
CORR (r)	For any set of data points (x_i, y_i), the *coefficient of correlation* is $r = {}^{s_{xy}}/_{s_x\, s_y}$. See s_{XY} and s below. For an arbitrary fit model $R(\boldsymbol{x})$, $r^2 = 1 - \frac{\sum\left[R(x_i) - y_i\right]^2}{\sum\left(\bar{y} - y_i\right)^2}$ is its *coefficient of determination* indicating the fraction of the variation of the dependent data $\boldsymbol{y}$ determined by the variation of the independent data $\boldsymbol{x}$. For $r^2 = 1$, $\boldsymbol{y}$ is fully determined by $\boldsymbol{x}$; for $r^2 = 0$, $\boldsymbol{y}$ is completely independent of $\boldsymbol{x}$; and e.g. $r^2 = 0.85$ means 85% of the variation of $\boldsymbol{y}$ is due to $\boldsymbol{x}$. Note BESTF picks the fit model showing the maximum $\boldsymbol{r}^2$ 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) *significant* at a 99% *confidence level* 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 *degrees of freedom* $n-2$ (see p. 220).

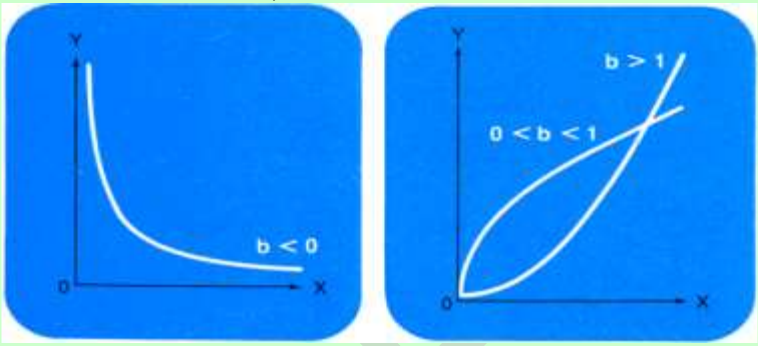
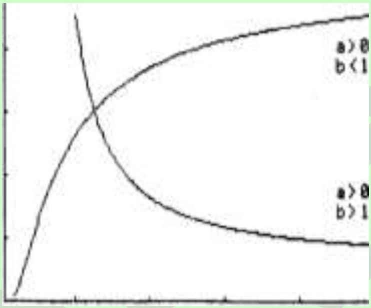
[103] Note that least squares regression is best for data point errors in vertical direction $\boldsymbol{y}$ being significantly greater than the errors in horizontal direction $\boldsymbol{x}$. See pp. 232ff for the formulas and more about the curve fit models provided.

<table>
<tr><th>Name</th><th>Remarks (see pp. 12ff for general information)</th></tr>
<tr><td>COV,
s_{XY}</td><td>For any set of data points (x_i, y_i), the population covariance is
$$COV_{xy} = \frac{1}{n^2}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$$
and the sample covariance is
$$s_{xy} = \frac{1}{n(n-1)}\left(n\sum x_i y_i - \sum x_i \sum y_i\right).$$</td></tr>
<tr><td>ExpF</td><td>Selects the exponential curve fit model $R(x) = a_0 e^{a_1 x}$ for least squares regression. [103] Generally, this will be a good choice if the measured data follow the shape of one of the two curves pictured here (think of e.g. human population growth or nuclear decay).[104]

</td></tr>
<tr><td>GaussF</td><td>Selects a Gauss peak fit model $R(x) = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$ for least squares regression. [103] See p. 219 for the shapes of such peaks.</td></tr>
<tr><td>HypF</td><td>

Selects the hyperbolic fit model $R(x) = {}^{1}/_{(a_0 + a_1 x)}$ for least squares regression. [103]</td></tr>
</table>

[104] 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	 Selects the linear fit model $R(x) = a_0 + a_1 x$ for least squares regression.[103] Generally, this will be a good choice if the measured data follow a straight line, raising or falling (but compare ORTHOF below).
LogF	 Selects the logarithmic curve fit model $R(x) = a_0 + a_1 ln(x)$ for least squares regression.[103] Generally, this will be a good choice if the measured data follow a curve looking like drawn at left.
L.R.	Uses the fit model selected and computes the two or three parameters of the regression for the data accumulated. For all curve fit models provided on your *WP 43S*, a regression parameter is (statistically) *significant* at a 99% *confidence level* if $\left\| a_i \right\| / s(a_i) > t^{-1}_{n-2}(0.995)$, with the right side being the inverse of the *t distribution* for the *degrees of freedom* $n-2$ (cf. p. 220).
OrthoF	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***. The sum of squared distances of the data points to the fit line will be minimized. This model is called *orthogonal regression*. See pp. 229ff for more and the *OM* for application examples.
ParabF	Selects a parabolic fit model $R(x) = a_0 + a_1 x + a_2 x^2$ for least squares regression.[103]

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.[103] 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*).[104] 
RootF	 Selects the root curve fit model $R(x) = a\, b^{1/x} = a_0\, a_1^{1/x}$ for a least squares regression.[103]
s, s_m	The *sample standard deviation* (*SD*) is the positive square root of the *sample variance* $$s_x^2 = \frac{1}{n(n-1)}\left[n\sum x_i^2 - \left(\sum x_i\right)^2\right] = \frac{1}{n-1}\left(\sum x_i^2 - n\,\bar{x}^2\right)$$ And the *standard error* (i.e. the *SD* of the *mean* x̄) is $s_m = {}^{s}/_{\sqrt{n}}$
s_W , s_{mW}	The *sample SD* for <u>weighted</u> data (where the weight $\boldsymbol{y_i}$ of each data point $\boldsymbol{x_i}$ was entered via Σ+) is $$s_w = \sqrt{\frac{\sum y_i \sum y_i x_i^2 - \left(\sum y_i x_i\right)^2}{\sum y_i \left(\sum y_i - 1\right)}}$$

Name	Remarks (see pp. 12ff for general information)
	And the corresponding *standard error* (the *SD* of the *mean* $\bar{x}_w$) 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.
$\hat{x}$	See next chapter.
$\bar{x}$	The *arithmetic mean* is calculated as $\bar{x} = \frac{1}{n} \sum x_i$
$\bar{x}_G$	The *geometric mean* is calculated as $$\bar{x}_G = \sqrt[n]{\prod x_i} = e^{\left[\frac{1}{n} \sum ln(x_i)\right]}$$
$\bar{x}_H$	The *harmonic mean* is calculated as $\bar{x}_H = n \Big/ \sum \frac{1}{x_i}$
$\bar{x}_{RMS}$	The *quadratic mean* is calculated as $\bar{x}_{RMS} = \sqrt{\frac{1}{n} \sum x_i^2}$
$\bar{x}_w$	The *arithmetic mean* for <u>weighted</u> data (where the weight $\boldsymbol{y_i}$ of each data point $\boldsymbol{x_i}$ was entered via Σ+) is $$\bar{x}_w = \sum x_i y_i \Big/ \sum y_i$$
$\hat{y}$	See next chapter.

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

Curve Fit 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 23 sums listed on pp. 83ff and increments the number of data points ***n***. Some of these 23 sums may turn non-numeric for non-positive entries – just do not care.

2. The subsequent evaluation will depend on the fit model you select (cf. pp. 224ff):

   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 forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 + a_1\,x$; for a given $\boldsymbol{y}$ it is $\hat{x} = {(y - a_0)}/{a_1}$. 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

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

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. 229 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 $a_1 = a_{1,tEXP}$. The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 e^{a_1 x}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \frac{1}{a_1} \ln({y}/{a_0})$.

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. 229 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 $a_1 = a_{1,tPOW}$. The forecast for a given ***x*** is $\hat{y} = a_0\, x^{a_1}$; for a given ***y*** it is $\hat{x} = \sqrt[a_1]{y/a_0}$.

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. 229 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 $a_1 = a_{1,tLOG}$. The forecast for a given ***x*** is $\hat{y} = a_0 + a_1\, ln(x)$; for a given ***y*** it is $\hat{x} = \exp\left({y - a_0}/{a_1} \right)$.

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

$$a_0 = \frac{\sum x_i^2 \cdot \sum \frac{1}{y_i} - \sum x_i \cdot \sum \frac{x_i}{y_i}}{n\sum x_i^2 - (\sum x_i)^2} \quad \text{and} \quad a_1 = \frac{n\sum \frac{x_i}{y_i} - \sum x_i \cdot \sum \frac{1}{y_i}}{n\sum x_i^2 - (\sum x_i)^2}$$

$$r_{HYP}^2 = \frac{a_0 \sum \frac{1}{y_i} + a_1 \sum \frac{x_i}{y_i} - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}{\sum \frac{1}{y_i^2} - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}$$

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = {}^{1}\big/_{(a_{0,HYP} + a_{1,HYP}\, x)}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \left(\frac{1}{y} - a_{0,HYP}\right)/a_{1,HYP}$.

f. **If you choose ROOTF** then the parameters of the least squares regression curve 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 $a_{0,\sqrt{}} = e^B$ and $a_{1,\sqrt{}} = e^C$.

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

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_{0,\sqrt{}}\, a_{1,\sqrt{}}^{1/x}$; for a given $\boldsymbol{y}$ it is $\hat{x} = {}^{\lg(a_{1,\sqrt{}})}\big/_{[\lg(y) - \lg(a_{0,\sqrt{}})]}$.

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

$$A = n\sum x_i^2 - \left(\sum x_i\right)^2, \qquad B = n\sum x_i^2 y_i - \sum x_i^2 \cdot \sum y_i\,,$$

$$C = n\sum x_i^3 - \sum x_i^2 \cdot \sum x_i\,, \qquad D = n\sum x_i y_i - \sum x_i \cdot \sum y_i\,,$$

$$E = n\sum x_i^4 - \left(\sum x_i^2\right)^2$$

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

$$a_2 = \frac{A\,B - C\,D}{A\,E - C^2}\,, \qquad a_1 = \frac{D - a_2 C}{A}\,,$$

$$\text{and} \quad a_0 = \frac{1}{n}\left(\sum y_i - a_2\sum x_i^2 - a_1\sum x_i\right).$$

$$\text{And} \quad r_{PAR}^2 = \frac{a_0\sum y_i + a_1\sum x_i y_i + a_2\sum x_i^2 y_i - \frac{1}{n}\left(\sum y_i\right)^2}{\sum y_i^2 - \frac{1}{n}\left(\sum y_i\right)^2}$$

The forecast for a given ***x*** is $\hat{y} = a_0 + a_1 x + a_2 x^2$; for a given ***y*** it

$$\text{is} \quad \hat{x}_{1,2} = \frac{1}{2a_2}\left(-a_1 \pm \sqrt{a_1^2 - 4a_2(a_0 - y)}\right).$$

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

$$B = n\sum x_i^2 \ln(y_i) - \sum x_i^2 \cdot \sum \ln(y_i),$$

$$D = n\sum x_i \ln(y_i) - \sum x_i \cdot \sum \ln(y_i)\,,$$

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

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

$$a_2 = \frac{1}{F}\,, \quad a_1 = -\frac{G}{2}\,a_2 \quad \text{and} \quad a_0 = e^{\,H-\,F\,a_1^2}\,.$$

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

The forecast for a given ***x*** is $\hat{y} = a_0\, e^{(x-a_1)^2/a_2}$; for a given ***y*** it is $\hat{x} = a_1 \pm \sqrt{a_2 \ln\left({}^{y}/_{a_0}\right)}$

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

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

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

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

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

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

$$a_0 = F\,, \quad a_1 = \frac{G}{2a_0}\,, \quad \text{and} \quad a_2 = H - F\,a_1^2.$$

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

The forecast for a given ***x*** is $\hat{y} = {}^{1}/_{[a_0(x+a_1)^2 + a_2]}$; the forecast for a given ***y*** is

$$\hat{x} = -a_1 \pm \sqrt{\frac{1}{a_0}\left(\frac{1}{y} - a_2\right)}$$

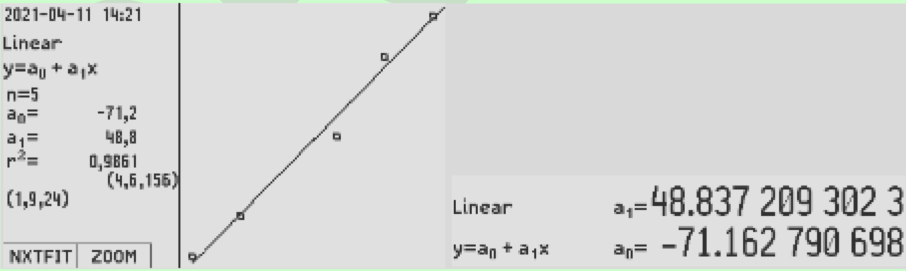
j. **If you choose BESTF** then the correlation coefficient will be computed with your data for model a and with the transformed data for models b through i, if allowed (cf. the *IOI*). The model delivering the greatest r^2 value will be selected.

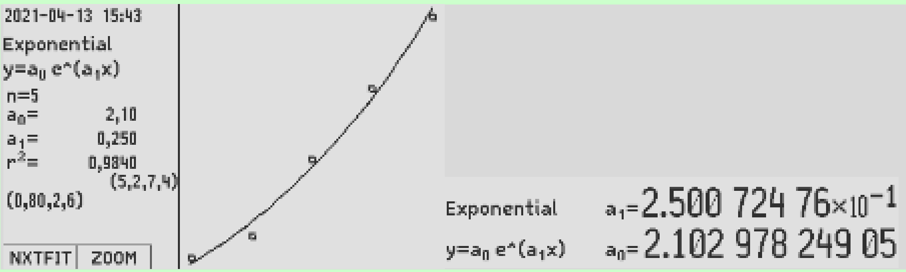
k. **If you choose ORTHOF** then the least squares regression line parameters a_0 and a_1 will be computed following the formulas:

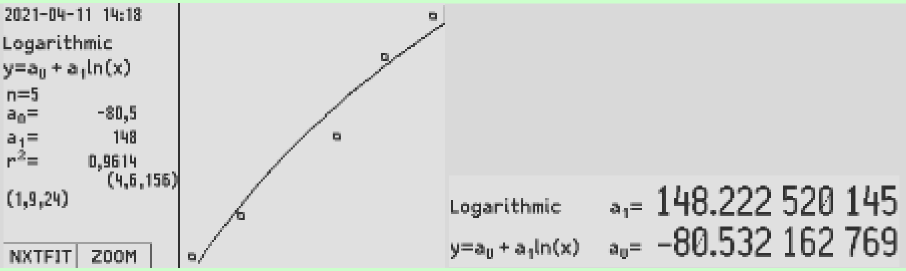
$$a_1 = \frac{1}{2s_{xy}}\left[s_y^2 - s_x^2 + \sqrt{\left(s_y^2 - s_x^2\right)^2 + 4s_{xy}^2}\right] \quad and \quad a_0 = \bar{y} - a_1\bar{x}$$

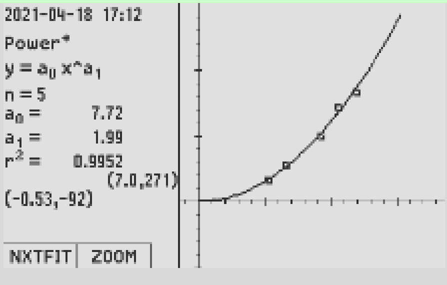
The other formulas for ORTHOF can be taken from model **a** (i.e. from LINF).

The outputs of ASSESS (left) and L.R. are formatted like this, allowing for up to 11 significant digits displayed for each model parameter:

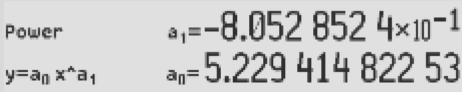


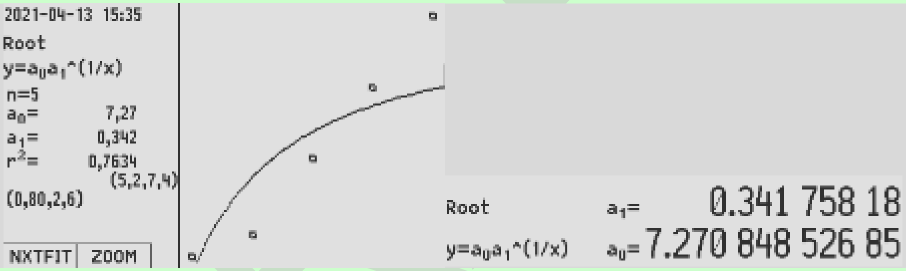


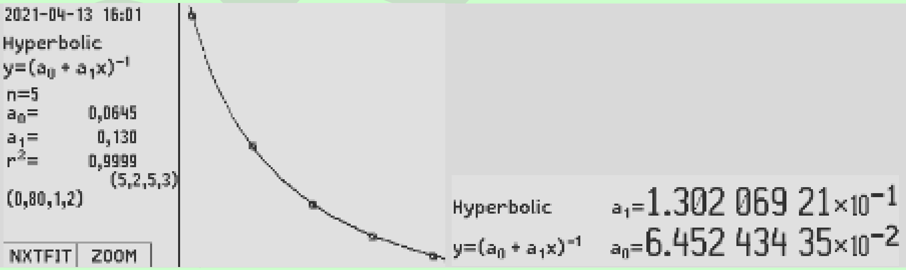


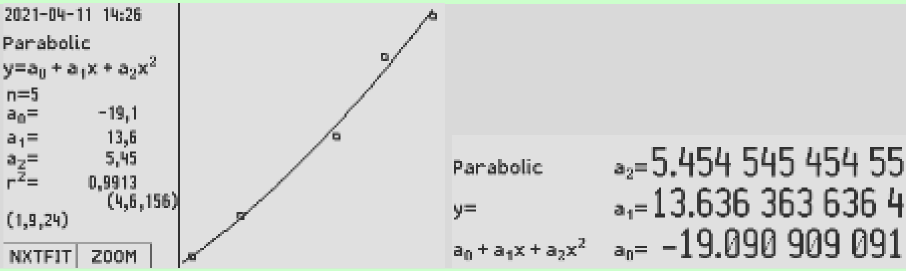


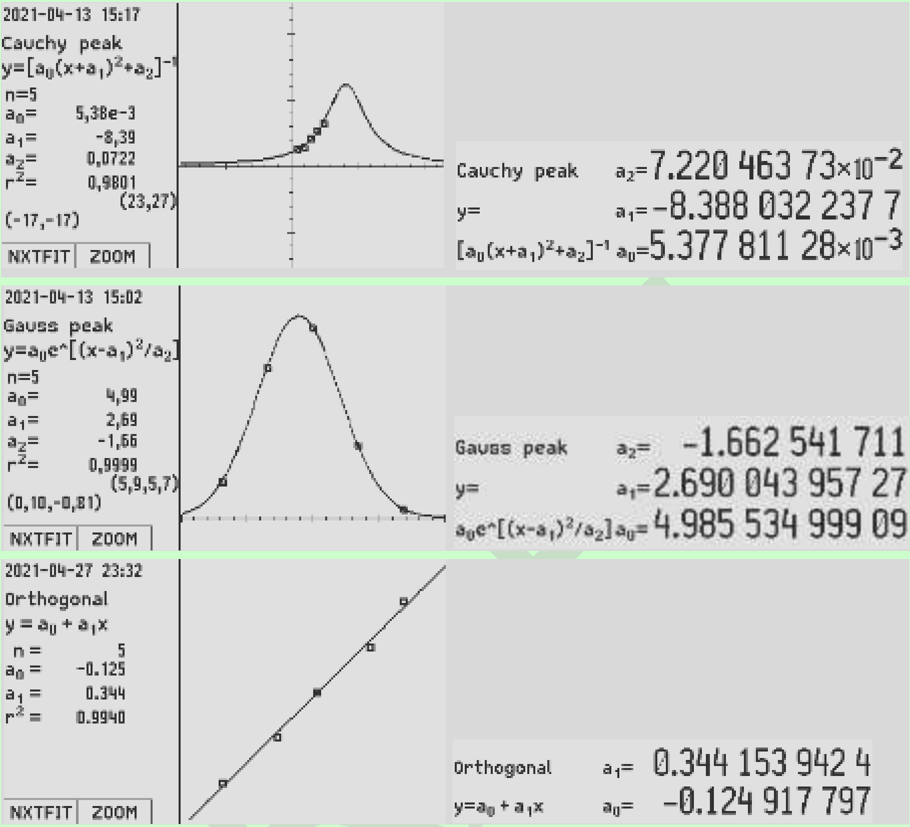
Note the values in parentheses bottom left in the ASSESS screen are the max. and min. of the plot area in format (***x***,***y***). If zoomed and zero is included, the respective axis will be drawn (see also Gauss and Cauchy).











Error Propagation in Calculations

Experimental data always go with errors, caused by e.g. the uncertainties of the measuring method, the instrument used, and/or environmental variations (cf. footnote 48 on p. 133). Even under controlled environmental and measuring conditions, random errors remain. These errors must be taken into account for a proper estimation of the uncertainties 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 $\boldsymbol{R}$ depends on several experimental parameters $\boldsymbol{x_1}$ through $\boldsymbol{x_n}$. Each such parameter $\boldsymbol{x_i}$ has an uncertainty or 'error' $\boldsymbol{\Delta x_i}$. Now, if $R = f(x_1, \ldots, x_n)$ then its total '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, the differential terms under this 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

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

So the terms under the square root have become simple differences which are determined most easily using 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 $R_2 = f(x_1, x_2 + \Delta x_2, \ldots, x_n)$ 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

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

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

In this chapter, we have replaced a formula requiring a lot of symbolic mathematics and relatively few numeric calculations by an equivalent formula requiring no symbolic mathematics but many numeric calculations. Programmed, your *WP 43S* will do all these repetitive numeric calculations for you automatically.

For very quick first estimation of ***ΔR***, here is a rule of thumb:

> **A result can never have more significant figures than the least accurate of its constituents.**

Solving Differential Equations

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

First, we are going to solve one-dimensional problems of the kind

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

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 $\Delta t = t_{i+1} - t_i$:

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

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

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

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

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

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

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. 239, 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

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

with an <u>arbitrary</u> *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 2D 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:

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

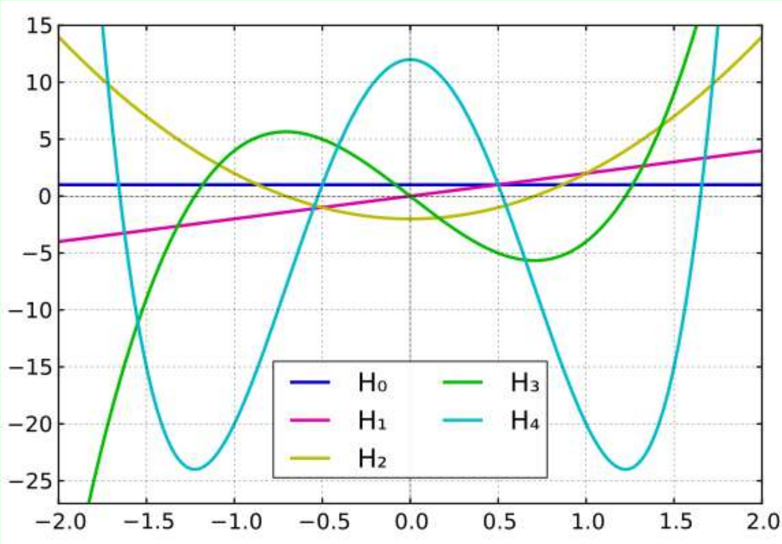
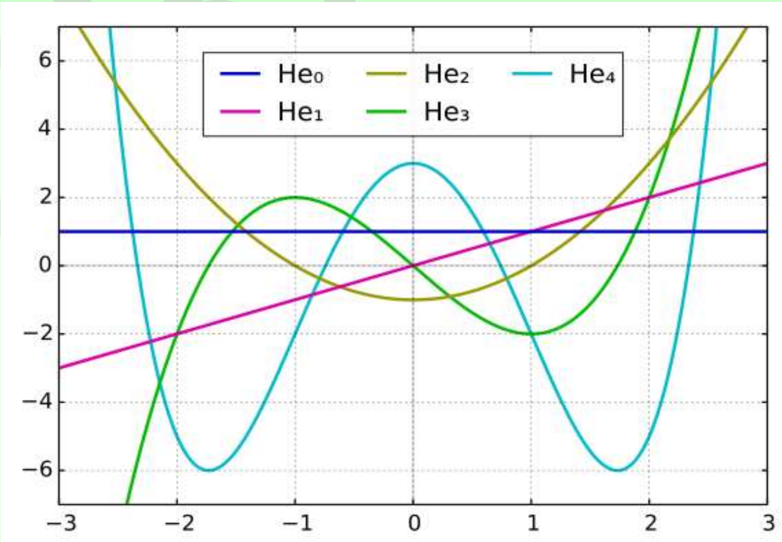
$$\left(\frac{dx}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dx}{dt}\right)_{i-\frac{1}{2}} + K_x\Delta t \qquad \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dy}{dt}\right)_{i-\frac{1}{2}} + K_y\Delta t$$

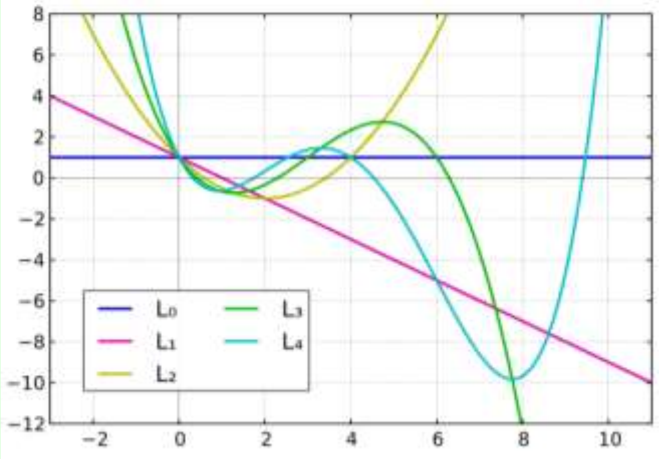
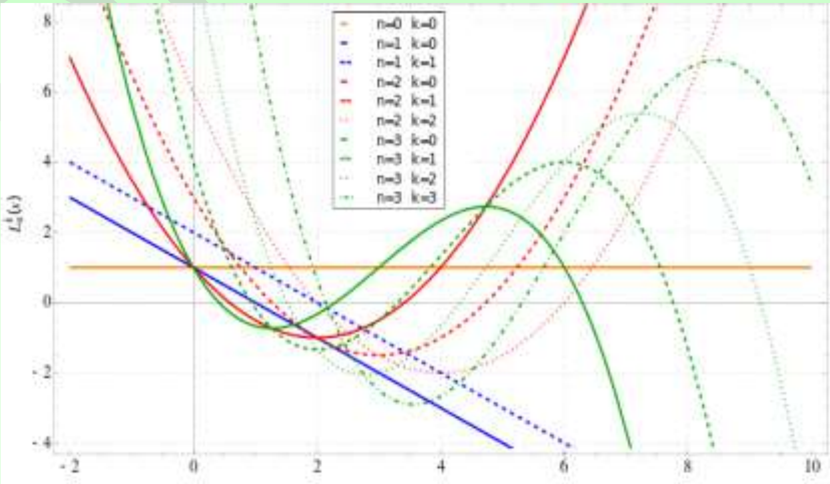
$$x_{i+1} \approx x_i + \left(\frac{dx}{dt}\right)_{i+1/2}\Delta t \qquad y_{i+1} \approx y_i + \left(\frac{dy}{dt}\right)_{i+1/2}\Delta t$$

Orthogonal Polynomials

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

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

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

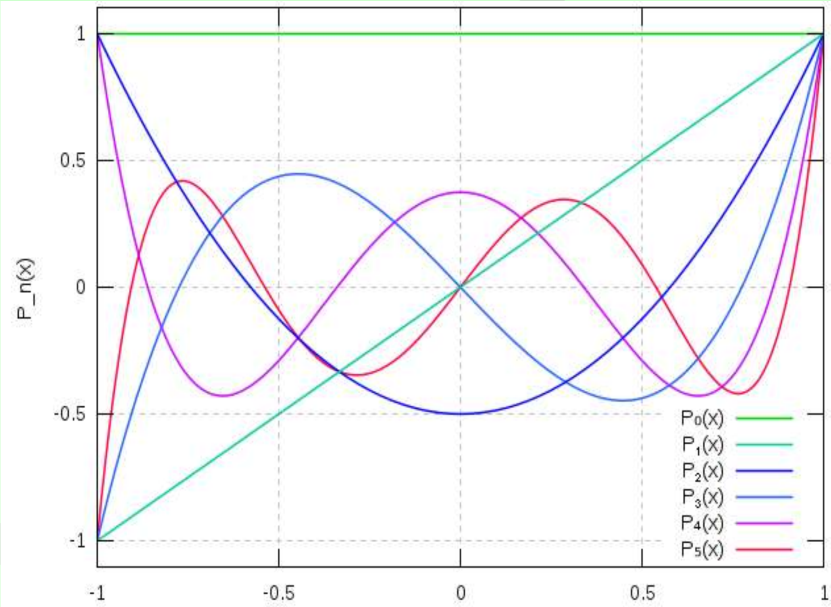
Name	Remarks (see pp. 12ff for general information)
L_m	***Laguerre** polynomials* (compare $L_{n\alpha}$ below): $L_n(x) = \frac{e^x}{n!} \cdot \frac{d^n}{dx^n}\left(x^n e^{-x}\right) = L_n^{(0)}(x)$ with ***n*** in $\mathbf{Y}$, solving the differential equation $x \cdot f''(x) + (1-x) \cdot f'(x)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

<table>
<tr><th>Name</th><th>Remarks (see pp. 12ff for general information)</th></tr>
<tr><td>P_n</td><td>

Legendre** polynomials*: $P_n(x) = \frac{1}{2^n n!} \cdot \frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]$ with ***n in $\mathbf{Y}$, solving the differential equation

$$\frac{d}{dx}\left[\left(1-x^2\right)\cdot\frac{d}{dx}f(x)\right]+n(n+1)f(x)=0\cdot$$

See the first six polynomials plotted here:



</td></tr>
<tr><td>T_n</td><td>

Chebyshev (a.k.a. *Čebyšev, Tschebyschow, Tschebyscheff*) *polynomials of **first** kind*

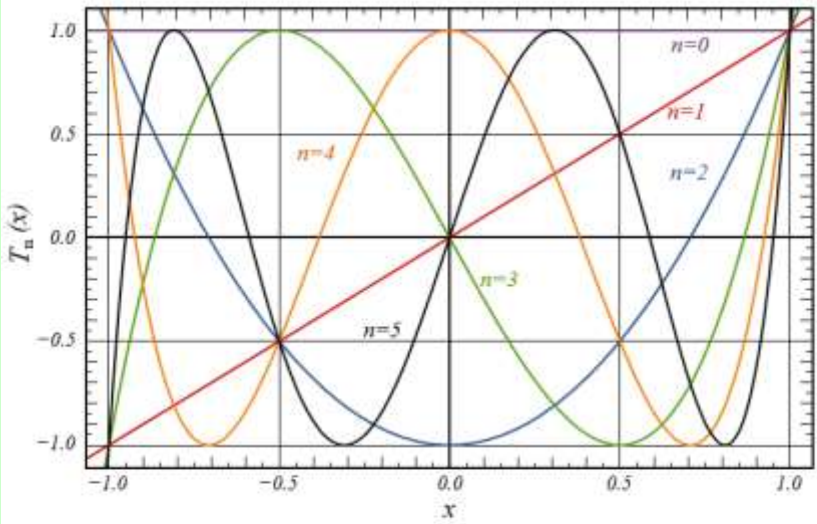
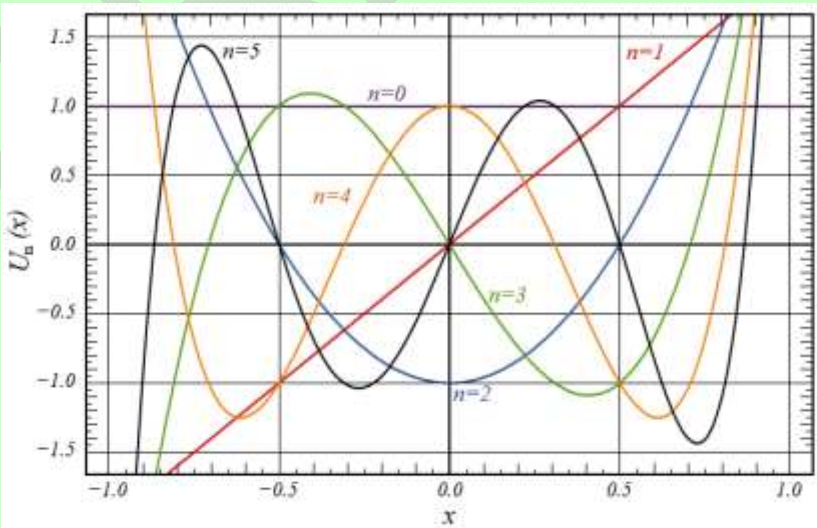
$$T_n(x) = \begin{cases} \cos(n \arccos(x)) & \text{for } -1 \le x \le 1 \\ \cosh(n \operatorname{arcosh}(x)) & \text{for } x > 1 \\ (-1)^n \cosh(n \operatorname{arcosh}(-x)) & \text{for } x < -1 \end{cases}$$ with ***n*** in $\mathbf{Y}$, solving

the differential equation

$$f''(x) - \frac{x}{1-x^2} f'(x) + \frac{n^2}{1-x^2} f(x) = 0$$

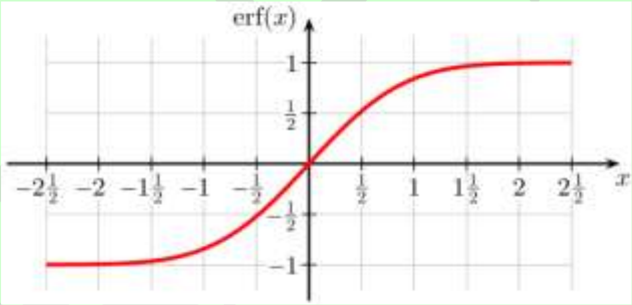
The plot overleaf shows $T_0(x)$ ... $T_5(x)$.

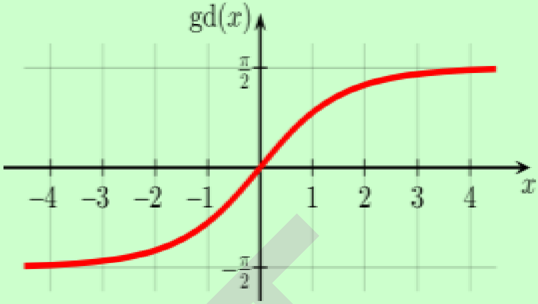
</td></tr>
</table>

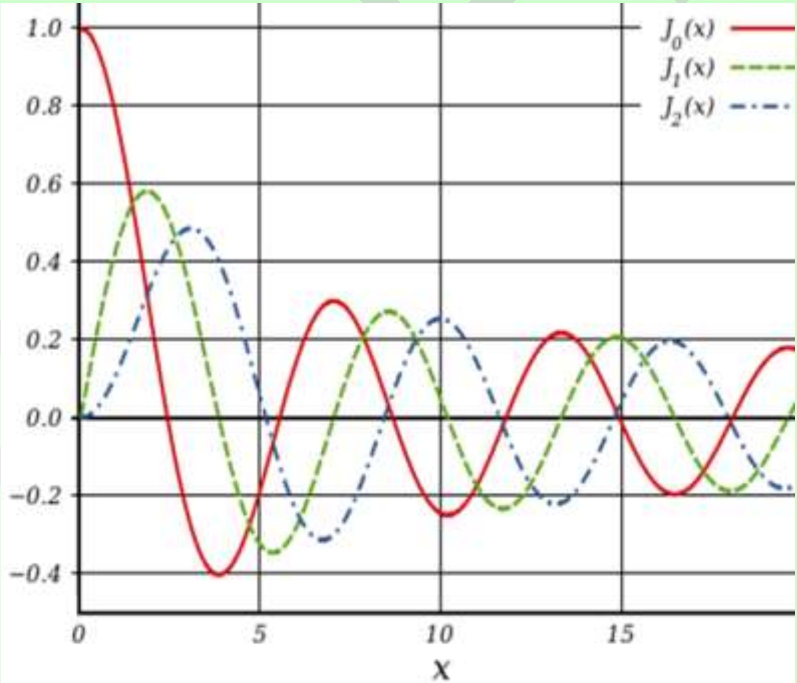
Name	Remarks (see pp. 12ff for general information)
	
U_n	***Chebyshev** polynomials of **second** kind* $U_n(\boldsymbol{x})$ with ***n*** in $\mathbf{Y}$, solving the differential equation $f''(x) - \frac{3x}{1-x^2} f'(x) + \frac{n(n+2)}{1-x^2} f(x) = 0$ The plot below shows $U_0(x)$ … $U_5(x)$: 

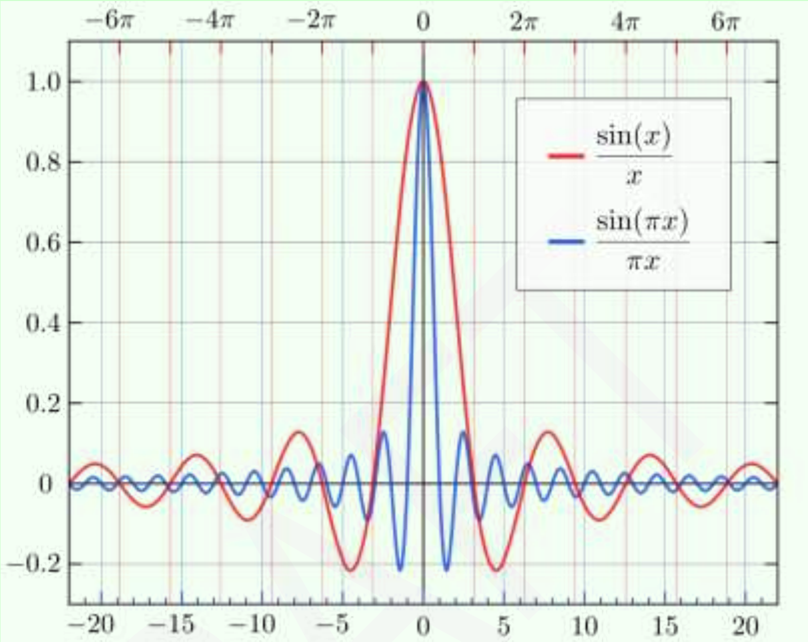
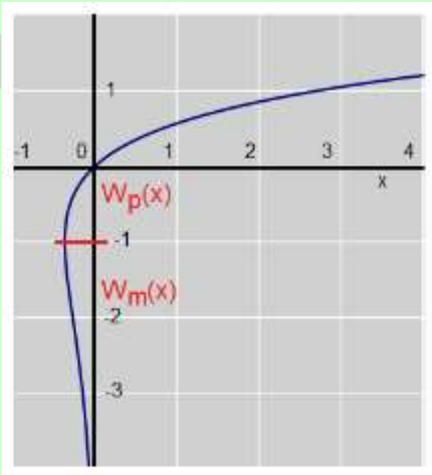
Even More Mathematical Functions

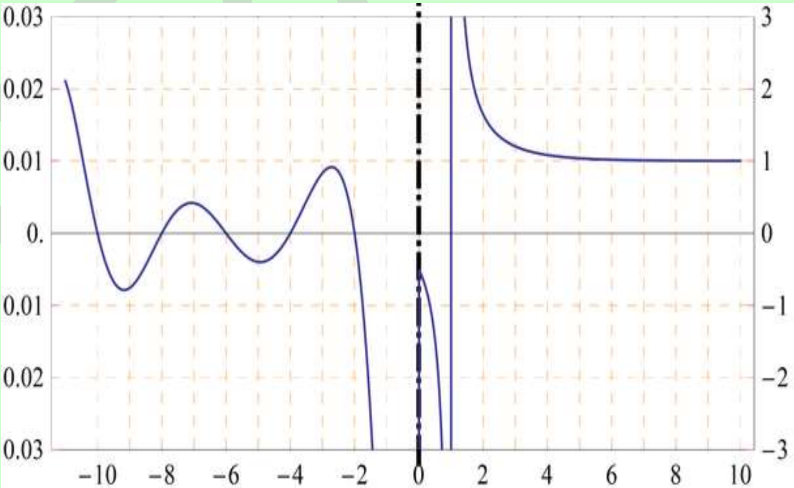
All the following functions are found in X.FN. Some of them are for pure mathematics only but were useful at some stages of the *WP 34S* or *WP 43S* projects, so we made them accessible for the public.

Name	Remarks (see pp. 12ff for general information)
AGM	Returns the *arithmetic-geometric mean*. Find more about it here: http://mathworld.wolfram.com/Arithmetic-GeometricMean.html .
erf	Returns the *error function* $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau$.  Note that $erf\left(x/\sqrt{2}\right) = 2\,\Phi(x) - 1$ with $\Phi(x)$ representing the *standardized normal CDF* as described on p. 219. Beyond statistics, the *error function* may be helpful in heat conduction and diffusion problems, for instance.
erfc	This command returns the *complementary error function* $erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-\tau^2} d\tau$. This function is related to the *error probability* of the *standardized normal distribution*.

Name	Remarks (see pp. 12ff for general information)
g_d , g_d^{-1}	Returns the *Gudermannian function* $g_d(x) = \int_0^x \frac{d\xi}{cosh(\xi)}$ linking hyperbolic and trigonometric functions. See the plot for its *real* values. The *inverse of this function* is $g_d^{-1}(x) = \int_0^x \frac{d\xi}{cos(\xi)}$.  Start reading here for more: http://en.wikipedia.org/wiki/Gudermannian_function .
I_{xyz}	Returns the *regularized (incomplete) Beta function* $\beta_x(x,y,z) / B(y,z)$ with $\beta_x(x,y,z) = \int_0^x \tau^{y-1} (1-\tau)^{z-1} \, d\tau$ being the *incomplete Beta function* and $B(y,z)$ being *Euler's Beta function* (see p. 82 and https://en.wikipedia.org/wiki/Beta_function).
$I\Gamma_p$	Returns the *regularized Gamma function* $P(x,y) = \frac{\gamma(x,y)}{\Gamma(x)}$ See γ_{XY} below for $\gamma(x,y)$ and p. 82 for $\Gamma(x)$. See here for more: https://en.wikipedia.org/wiki/Incomplete_gamma_function
$I\Gamma_q$	Returns the *regularized Gamma function* $Q(x,y) = \frac{\Gamma_u(x,y)}{\Gamma(x)}$ See Γ_{XY} below for $\Gamma_u(x,y)$ and p. 82 for $\Gamma(x)$.

Name	Remarks (see pp. 12ff for general information)
$J_y(x)$	Generally, the *Bessel functions* solve the differential equation $x^2 f''(x) + x f'(x) + (x^2 - \nu^2) f(x) = 0$ with $\nu \in \mathbb{C}$. $J_y(x)$ returns the *Bessel function of first kind* and order $y = \nu$. For arbitrary ν, this is $J_\nu(\boldsymbol{x}) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\ \Gamma(m+\nu+1)} \left(\frac{\boldsymbol{x}}{2}\right)^{2m+\nu}$ For integer ν, this is also $J_\nu(\boldsymbol{x}) = \frac{1}{\pi} \int_0^\pi cos[\nu\, t - \boldsymbol{x}\, sin(t)]\, dt$  Start reading here for more information: http://en.wikipedia.org/wiki/Bessel_function .

Name	Remarks (see pp. 12ff for general information)
sinc, sincπ	
W_p, W_m	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 *real* values of both branches.  Start reading here for more information: http://en.wikipedia.org/wiki/Lambert_W_function . Learn more here: http://mathworld.wolfram.com/LambertW-Function.html .

Name	Remarks (see pp. 12ff for general information)
γ_{XY}	Returns the *lower incomplete Gamma function* $\gamma(x,y)=\int_0^y t^{x-1}e^{-t}dt$. Required for $I\Gamma_p$ above. See here for more: https://en.wikipedia.org/wiki/Incomplete_gamma_function
Γ_{XY}	Returns the *upper incomplete Gamma function* $\Gamma_u(x,y)=\int_y^\infty t^{x-1}e^{-t}dt$. Required for $I\Gamma_q$ above.
ζ(x)	Returns *Riemann's Zeta* for *real* arguments, with $\zeta(x)=\sum_{n=1}^{\infty}\frac{1}{n^x}$ for $x > 1$, and its analytical continuation for $x < 1$: $\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x)\cdot$ Note the different vertical scales for negative and positive $\boldsymbol{x}$ in the plot below. And $\sqrt{6\,\zeta(2)}\;=\;\pi$.  Look here for more: http://mathworld.wolfram.com/RiemannZetaFunction.html .

Note that the *incomplete gamma* and *error functions* as well *as Laguerre*, *Legendre*, and *Bessel functions* were provided 1976/77 already on the *Commodore M55* pocket calculator (*The Mathematician*, featuring 55 keys). At the same time, the *Commodore S61* (*The Statistician*, with 60 keys) featured the *binomial*, *Poisson*, *hypergeometric*, *Gaussian*, *t*, *chi-square*, and *F distributions* as well as some ‘test statistics’. Taking into account that no *HP* pocket calculator so far features orthogonal polynomials nor the *hypergeometric distribution*, that was a very impressive early pair.

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. 213ff) is a competent source. And *Mathworld* (quoted on pp. 247ff) or *WolframAlpha* may contain more details than you ever want to know. Further references are found at these sites easily.

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'
  RCL× .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 FACT_2 will allocate a local register ($\mathbf{R.00_2}$) and store **3** therein. After decrementing ***x***, FACT will call itself again.

Then FACT_3 will allocate a local register ($\mathbf{R.00_3}$) and store **2** therein. After decrementing ***x***, FACT will call itself once more.

Then FACT_4 will return to FACT_3 with $x = 1$. This ***x*** will be multiplied by $\boldsymbol{r.00_3}$ there, returning to FACT_2 with $x = 2$. This ***x*** will be multiplied by $\boldsymbol{r.00_2}$ there, returning to FACT with $x = 6$, where it will be multiplied by ***r.00*** and will finally become 24.

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*:

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 *Museum of HP Calculators*** (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 *WP 34S*. Passed to *Jake Schwartz*, *Eric Smith*, and *Richard Ottosen* for first information.
0.2	3.10.15	Update based on *Jake's* feedback and further thoughts, distributed to *Eric*, *Jake*, *Marcus*, and *Pauli*.
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 *Key Response Table*. Passed to *Michael* for information.
0.4	28.3.16	Renamed LOGS to EXP and EEX to E. Added hardware information from 2nd 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 ŷ, 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 ≡▲, SST by ≡▼, and UNDO by ↩; added some *alpha input mode* reminders on the keyboard. Added AGRAPH, CLLCD, EQ.xxx, <u>HYP</u>, J/G, M.GOTO, ORTHOF, PIXEL, POINT, TDISP, and ⎙USER. Moved the background considerations out of *ReM App. D*. Introduced **K** as *alpha register* for alphanumeric constants in programs. Removed *fraction data type*. Extended *items* from 6 to 7 characters to match *HP-42S*. Specified *data types* more precisely in *ReM App. D*. Reduced the

	Date	Release notes
	2.4.18	maximum number of *local registers* from 888 to 100. Deleted JG1582 and JG1752. Renamed two commands for TVM. Replaced the heading apostrophe for *menu* names. Put SUMS in STAT. Renamed the trigonometric and hyperbolic functions according to mathematical standards, and 🖶CHR to 🖶CHAR. Redistributed the chapter about constants. Modified STATUS display. Refined the unit conversions to ensure *SI* on one side. Specified 0 SEED. Expanded *ReM App. A*. 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	Changed keyboard layout: introduced TRG containing trigonometric functions, removed HYP into EXP and [π] to g-shifted [/], swapped some shifted labels. Refined the chapters about register arithmetic, *Command Parameter Input*, *Alphanumeric Input*, *Matrix Calculations*, and *Orthogonal Polynomials*. Introduced CLCVAR and more vintage examples. Rearranged *temporary information* on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE.
	20.9.18	Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9	3.1.19	Removed *angle data type*. Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded *App. B*. 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 [EXIT] and [1/x] to match *HP-42S*. Added a chapter about curve fitting. Modified functionalities of [ENTER↑] and [↶]. Expanded *App. K*. Renamed DOUBLE to →DP. Added →SP and conversions of *quarts*. Rearranged X.FN. Replaced [USR] by [UM]. Changed keyboard moving [UM], [√x], and [TRI]. Moved 🖶 to [f] [R/S]. Added XIN and XOUT. Added a chapter in *App. E* and information about infinite integers Extended the domain of GCD and LCM. Refined and corrected.
0.10	3.3.19	Returned *angle data type* and αSR. 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 CPX, MATX, and α•. Added a summary of matrix functions. Removed the [ON]-key combinations. Modified MEM?. Rewrote the angular conversions. Renamed infinite and finite integers to *long* and *short integers*. Added a chapter about ±∞ and NaN. Modified RBR and the menu for STO and RCL. Removed 🖶 from the keyboard. Renamed X_u to X_e for the distributions.

	Date	Release notes
0.11	8.5.19	Changed keyboard making [CC] primary and user mode shifted, removing x^2, x≷, and DSP, adding |x|, DROP, and SHOW, and moving some shifted labels. Modified BITS, CLREGS, CNST, CPX, DISP, EXP, INTS, MODE, PARTS, SHOW, STAT, U→, αMATH, the division matrix, *data type* conversions, and the *Quick Reference Guide*. Added conversions of *barrels*, *carats*, and *fathoms*. Deleted DSP. – Separated predefined variables. Refined *Sect. 6*. Added $\bar{x}_H$, $\bar{x}_{RMS}$, nine statistical sums and five curve fit models. Split STAT in STAT and SUMS; renamed RMDR to RMD, L_n to L_m, $L_{n\alpha}$ to $L_{m\alpha}$, Π to Π_n, Σ to Σ_n, and some constants to avoid search ambiguities. Refined *App. J*, *Sect. 3 and 4*, →INT, CLR, and the functions of [▲] and [▼]. Put SUMS instead of RMD on the keyboard, moved ADV, BITS, CATALOG, EQN, FILL, INTS, MATX, MODE, PROB, RTN, SHOW, STAT, and α.FN. Rearranged A…Ω and *Sect. 2* of the *OM*.
0.12	16.10.19	Rearranged the appendices of the *ReM* from *App. D* on. Expanded *App. A* of the *OM* and *App. K*. Deleted the standardized normal distribution Φ and rearranged PROB. Updated CNST following *CODATA 2018*. Renamed the angular conversions. Changed the composing and cutting functionality of [CC]. Refined exiting *short integer* input. Expanded *App. D*. Specified maximum size of *long integers*. Changed keyboard adding ∡, moving CPX, FIN, RBR, R↑, and SHOW, removing %. Renamed VANGLE to V∡. Modified CPX, MATX, TRI, and X.FN. Rearranged *Section 1* of the *OM*. Added some internal *data types* to *App. B*; reduced the range of *long integer* 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 Σ+. Refined the *Addressing Tables*. Added a *data type* matrix for IDIVR. Refined the *Special Results* in *App. B*.
0.13	30.11.19	Expanded the alpha keyboard and *App. I*. Modified CPX, INTS, MODE, PROB, STK, TEST, α•, SHOW, and STATUS. Refined the sorting order of *items*, ALL, CX→RE, MEM?, RE→CX, RBR, RM, SLVQ, and U→. Started filling *App. F* and G. Refined *App. 2*. Added a *long integer* example, CPXR?, LZ?, Δv_{Cs}, conversions of *hectares*, and a proposal for system status information.
0.14	7.3.20	Introduced *system flags* for status information. Split I/O. Added CATALOG'SYS.FL, PRINT, PROG, RANI#, VAR, auxiliary constants, some predefined variables, and an index in *App. I*. Changed keyboard swapping MODE and FLAGS, U→ and ∡→, moving CPX, FILL, RBR, R↑, USER, α.FN, αINTL, √x, and 🖶, displaying PRINT, RMD, STATUS, x^2, and ':', and removing [c/d], 🖶x, →SP, and →DP. Renamed DISP to DSP and SUMS to Σ, changed [↶] to [↶]. Refined the addressing tables and catalog access, [a b/c], ADV, BATT?, BITS, CATALOG'CHARS and 'MENUS, CLALL, CLFALL, CPX, EXP,

	Date	Release notes
		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?, MULT×, 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, R↑, 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, ∡, 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 *IOI*. Refined I/O and the descriptions of LOAD, LOADSS, RESET, and UNDO. Marked the not-undoable *items* in the *IOI*. Renamed the constants according to the *OM* and kicked them out of the *IOI.* 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 SINCπ and more vintage pictures. Expanded *App. G*. Refined the power matrix. Corrected.
0.17	16.3.21	Empty menus show up. Refined ∛x, AUTOFF, BATT?, BITS, CLFALL, CLP, CLREGS, DSZ, e^x, FP, GTO., IP, ISZ, J/G, LOAD, LOCR, LOCR?, LN, MEM?, M.EDI, M.EDIN, NaN?, RCLEL, RDP, RSD, SAVE, SDL, SDR, TDISP, WSIZE, x=+0?, x=−0?, $\sqrt[x]{y}$, y^x, Σ+, Σ−, ^MOD, √x, 🖶r, labels in U→, the 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, area, and mass to U→. Modified many conversions. Deleted M.OLD, added DELITM, J/G?, and 🖶x. Moved 'no' from [7] to [/]. Corrected.

	Date	Release notes
0.18	18.4.21	Refined the chapter *Times*, the commands EXPON…, M.GET, M.PUT, M.R⇄R, NBIN…, SAVE, STAT, and TDISP, *Sect. 3* of the *OM*, *App. B, E, F, H, and I*. Removed error 27, CATALOG'DIGITS, 'REGIST, and 'SYS.FL. Added ASSESS, CENTRL, NXTFIT, PLOT, and s_{mi}. Brought *wrap* and *grow* in the status bar. Corrected.
0.19	29.4.21	Release. – Introduced STATS. Refined *App. E* and *F*. 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 [f] or [g], 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 **R*nn***, [RCL] ***nn*** recalls a copy of the contents of **R*nn*** into **X**, and [x≷] ***nn*** swaps ***x*** and the contents of **R*nn***.

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: Any number you enter without using [.], [E], [CC], or [#] is taken as a long integer of base 10.

Real numbers: Any number you enter using [.] 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×4.56 in rectangular mode (CF POLAR and press **1.23** [CC] **4.56** [+/-][ENTER↑]) or its magnitude and phase like -7.89 ∡ 120° 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 also 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 read 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 read as a *date* in the format chosen (yyyy.mmdd for Y.MD or dd.mmyyyy for D.MY or mm.ddyyyy for M.DY).

Matrices: see pp. Q-7f.

Short integers: Any pure 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 [α]. 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 [+], [−], [×], [.], 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 firmware 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 current 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 (you can use ▲ or ▼ unless
≡▽	moves it to next step a *multi-view menu* is displayed).
⬅	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.

```
   ...
   1.05203
   STO 'Count'
LBL 01
   ...
   ISG 'Count'
   GTO 01
   BEEP
   ...
```

Looping

ISE, ISG, ISZ, DSE, DSL, and DSZ (found in <u>LOOP</u>) 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.

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 ≡▽ (or ▼ if no *multi-view menu* is displayed). Press ≡△ (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	

CLΣ Clears all statistical data.
CLP Clears (deletes) the *current program*.
CF ***n*** Clears *flag* ***n***.
CLMENU Clears the *programmable menu*.
CLCVAR Clears all variables used in the *current program*.
CLX Clears *stack register* **X**.

CLREGS Clears all *registers* (except the *stack* and statistical data).
CLPALL Clears (deletes) all programs in *RAM*.
CLFALL Clears all numbered user *flags*.
CLLCD Clears the *LCD* above and to the right of pixel x, y.
CLSTK Clears the entire *stack* (i.e. fills all its *registers* with zero).

CLALL Clears almost everything but the modes set.
RESET Resets the *WP 43S* to *startup default configuration*.

DELITM VARS ... ▭ Deletes the user variable selected.

DELITM MENUS ... ▭ Deletes the user *menu* selected.

DELITM PROGS ... ▭ 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*×*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*.

If you want to invert or transpose the matrix *x*, press $[M]^{-1}$ or $[M]^{T}$. Press |M| for computing its determinant.

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 $(A)\vec{X} = \vec{B}$:

1. 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**.
2. Press Mat A; fill the matrix; press EXIT.
3. Press Mat B; fill the matrix; press EXIT.
4. Press Mat X to compute the solution matrix.

PROBABILITY

PROB						
	RAN#	SEED	RANI#			Γ(x)
		NBin:	Geom:	Hyper:	Binom:	Poiss:
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:
	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:
Binom:	Binom_p		Binom_{Δ}	Binom_{Δ}		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 IP(x) and IP(y).

Γ(x) Returns the *Gamma function* value of x.

These 14 continuous and discrete distributions (*d.*) are provided:

Binom:	*Binomial d.*	($i = p_0$ = gross prob. 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:	*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_{Δ}** the *cumulated d. function*, **Norml_{Δ}** 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** ($0 \le x \le 1$).

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 ↶ (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		BestF		
	ParabF	HypF	RootF			
	LinF	ExpF	LogF	PowerF		OrthoF
	ASSESS	$\bar{x}_{RMS}$	x_{max}	x_{min}		PLOT
	s(a)	$\bar{x}_H$				
	L.R.	r	s_{xy}	cov	$\hat{x}$	$\hat{y}$
STAT	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	
	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.

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.
s(a)	Returns the standard errors of a_0 and a_1 (for LINF and ORTHOF).
ASSESS	Plots the data points together with the fit model curve(s) selected.
LinF	Linear fit model: $y = a_0 + a_1 x$.
ExpF	Exponential fit model: $ln(y) = ln(a_0) + a_1 x$ 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 $y = a_0\, x^{a_1}$.
BestF	Blindly selects the model returning the best correlation coefficient.
ParabF	Parabolic fit model: $y = a_0 + a_1 x + a_2 x^2$.
HypF	Hyperbolic fit model: $y = 1/(a_0 + a_1 x)$.
RootF	Root fit model: $y = a_0\, a_1^{1/x}$.
GaussF	*Gauss* peak fit model: $y = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$.
CauchF	*Cauchy* peak fit model: $y = 1/[a_0\,(x + a_1)^2 + a_2]$.
PLOT	Plots the data points to analyse the measuring system under investigation; with ≥ 30 points, its standard error can be measured.
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.

ADV is for programmed summing, multiplying, solving, integrating, & deriving.

Interactive Operations on Equations

EQN

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	∫fdx

SLVQ solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input *stack* [***c, b, a***, …]. 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(x₁, x₂, ...)* 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 $f(x_1, x_2, ...) = 0$ 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
INTS	DBL /	DBLR	DBL ×	^MOD	CEIL	GCD
	IDIV	RMD	MOD	×MOD	FLOOR	LCM
	A	B	C	D	E	F

A ... F — Digits for input of short integers of bases >10.

IDIV ℝ ℤ — Integer divide – works also for real numbers (ℝ) and long integers (ℤ).

RMD, MOD ℝ ℤ — Remainder and modulo.

×MOD ℝ ℤ — Returns $(z \cdot y) \bmod x$.

^MOD ℝ ℤ — Returns $(z^y) \bmod x$.

FLOOR ℝ — Returns the greatest integer $\leq x$.

CEIL ℝ — Returns the smallest integer $\geq x$.

LCM ℤ — Returns the least common multiple of x and y.

GCD ℤ — Returns the greatest common divisor of x and y.

DBL /, DBLR, DBL× — Double *word* length commands for division, remainder, and multiplication.

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*.

BITS

1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE
LJ					RJ
SL	RL	RLC	RRC	RR	SR
A	B	C	D	E	F
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).

OPERATIONS ON ALPHANUMERIC STRINGS

Connect strings by pressing [+]. Then ***x*** will be appended to the string ***y***. With numeric data in **X**, their current display format is taken into account.

α.FN

FBR					
				αLENG?	αPOS?
x→α	αRL	αRR	αSL	αSR	α→x

x→α ***s*** Converts a code ***x*** to the corresponding character and appends it to the string in ***s***.
αRL, αRR ***s*** Rotates the string in ***s*** by ***x*** characters to the left (or right).
αSL, αSR ***s*** Deletes the first (or last) ***x*** characters of the string in ***s***.
α→x ***s*** Pushes the code of the first character in ***s*** on the stack.

αLENG? ***s*** Pushes the length of the string in ***s*** on the stack.
αPOS? ***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. Note in this section are no update marks since I presume you will read entirely what you are interested in.

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], [←], [+], [−], [×], [/], [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], [x^2], [$\sqrt{x}$], [e^x], [ln], [10^x], [lg], [sin], [cos], and [tan] are the most popular – twelve candidates for seven key tops left. Either [x^2] or [$\sqrt{x}$] shall be primary (we chose [x^2] since a shifted [x^2] is of little use); and we can ditch [10^x] and [lg] when we have [e^x] and [ln] 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 [$\sqrt{x}$], [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 $\sqrt{x}$ 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. 114ff):

ADV	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx
BITS	AND	OR	XOR	NOT	MASKL	MASKR
CLK	DATE	→DATE	DATE→	WDAY	TIME	x→DATE
CLR	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX
CPX	dot	cross	UNITV	Re	conj	Re≷Im
DISP	FIX	SCI	ENG	ALL	ROUNDI	ROUND
EQN	NEW	EDIT	f''	f'	∫f	Solver
EXP	x^3	$\sqrt[x]{y}$	$\log_x y$	lb x	2^x	$\sqrt{x}$
FIN	%	%MRR	%T	%Σ	%+MG	TVM
FLAGS	SF	FS?	FF	STATUS	FC?	CF
INFO	SSIZE?	MEM?	RM?	ISM?	WSIZE?	KTYP?
INTS	A	B	C	D	E	F
I / O	LOAD	LOADP	LOADR	LOADSS	LOADV	LOADΣ
LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG
MATX	NEW	$[M]^{-1}$	|M|	$[M]^T$	SIM EQ	EDIT
MODE	SF	DEG	RAD	GRAD	MULπ	CF
PARTS	IP	FP	MANT	EXPT	sign	DECOMP
PRINT	🖶x	🖶r	🖶Σ	🖶ADV	🖶LCD	🖶PROG
PROB	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:
P.FN	INPUT	END	ERR	TICKS	PAUSE	P.FN2
STAT	Σ+	$\bar{x}$	s	σ	s_m	SUM
STK	x≷	y≷	z≷	t≷	≷	DROPy
TEST	x< ?	x≤ ?	x= ?	x≠ ?	x≥ ?	x> ?
TRI	sin	arcsin	cos	arccos	tan	arctan
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 (√x and √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 *Formatting* of Reals

Just some examples to show how display works for a plain and handy number like −30 405.004 302 071 8:

	SCI	ENG	FIX
0	-3.×10^4	-30.×10^3	-30 405.
1	-3.0×10^4	-30.×10^3	-30 405.0
2	-3.04×10^4	-30.4×10^3	-30 405.00
3	-3.041×10^4	-30.41×10^3	-30 405.004
4	-3.040 5×10^4	-30.405×10^3	-30 405.004 3
5	-3.040 50×10^4	-30.405 0×10^3	-30 405.004 30
6	-3.040 500×10^4	-30.405 00×10^3	-30 405.004 302
7	-3.040 500 4×10^4	-30.405 004×10^3	-30 405.004 302 1
8	-3.040 500 43×10^4	-30.405 004 3×10^3	-30 405.004 302 07
9	-3.040 500 430×10^4	-30.405 004 30×10^3	-30 405.004 302 072
10	-3.040 500 430 2×10^4	-30.405 004 302×10^3	-30 405.004 302 071 8
11	-3.040 500 430 21×10^4	-30.40 … 2 1×10^3	-30 405.004 302 071 80
12	-3.040 500 430 207×10^4	-30.40 … 2 07×10^3	-30 405.004 302 071 800
13	-3.040 500 430 207 2×10^4	-30.40 … 2 072×10^3	-30 405.004 302 071 800 0
14	-3.040 500 430 207 18×10^4	-30.40 … 071 8×10^3	-30 405.004 302 071 800 00
15	-3.040 500 430 207 180×10^4	-30.40 … 071 80×10^3	-30 405.004 302 071 800 000

For positive numbers, a blank space shall be displayed instead of the minus.

Now, let us look at the same for −3.040 500 430 207 18:

	SCI	ENG	FIX
0	-3.×10^{-4}	-300.×10^{-6}	-3.×10^{-4}
1	-3.0×10^{-4}	-300.×10^{-6}	-3.0×10^{-4}
2	-3.04×10^{-4}	-304.×10^{-6}	-3.04×10^{-4}
3	-3.041×10^{-4}	-304.1×10^{-6}	-3.041×10^{-4}
4	-3.040 5×10^{-4}	-304.05×10^{-6}	-0.000 3
5	-3.040 50×10^{-4}	-304.050×10^{-6}	-0.000 30
6	-3.040 500×10^{-4}	-304.050 0×10^{-6}	-0.000 304
7	-3.040 500 4×10^{-4}	-304.050 04×10^{-6}	-0.000 304 1
8	-3.040 500 43×10^{-4}	-304.050 043×10^{-6}	-0.000 304 05
9	-3.040 500 430×10^{-4}	-304.050 043 0×10^{-6}	-0.000 304 050
10	-3.040 500 430 2×10^{-4}	-304.050 043 02×10^{-6}	-0.000 304 050 0
11	-3.040 500 430 21×10^{-4}	-304.050 043 021×10^{-6}	-0.000 304 050 04
12	-3.040 500 430 207×10^{-4}	-304.050 043 020 7×10^{-6}	-0.000 304 050 043
13	-3.040 500 430 207 2×10^{-4}	-304.05 … 0 72×10^{-6}	-0.000 304 050 043 0
14	-3.040 500 430 207 18×10^{-4}	-304.05 … 0 718×10^{-6}	-0.000 304 050 043 02
15	-3.040 500 430 207 180×10^{-4}	-304.05 … 0 718 0×10^{-6}	-0.000 304 050 043 021

For small parameters (0 … 11), the displays in both tables are identical to the ones of the *HP-42S.*

Layouting on pp. B-18f.

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 ±180° 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 2***π*** 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 2***π*** 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 often provided catching traditional *names*.

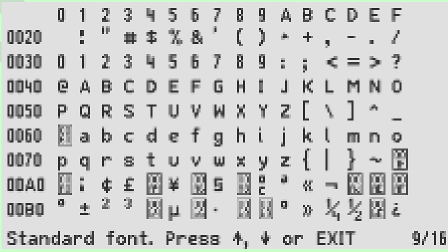
Calculation Internals

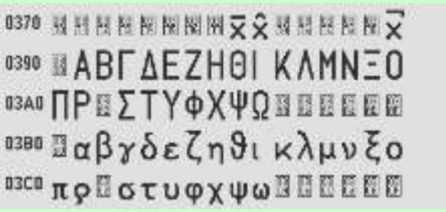
For $y > 0$, general powers in ℝ and ℂ are calculated as $y^x = e^{x\,ln(y)}$ and general roots as $\sqrt[x]{y} = e^{\frac{ln(y)}{x}}$ for all values of x except the integers 2 and 3. For odd ('integer') roots of $y < 0$, $\sqrt[x]{y} = -e^{\frac{ln(-y)}{x}}$; here, 'integer' includes *DT* 1 and 10 and reals with zero fractional part. Some special results in App. B can be deduced from these calculation paths.

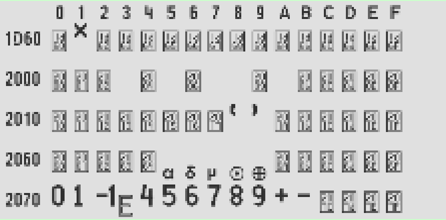
Note that *Free42* may calculate even powers differently (as a series of multiplications using repeated squaring), never employing more than 34 digits (see also p. B-24).

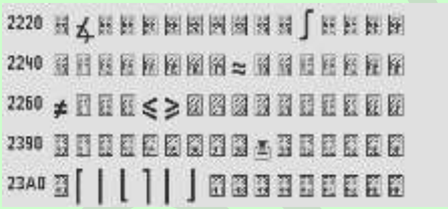
Character Sets

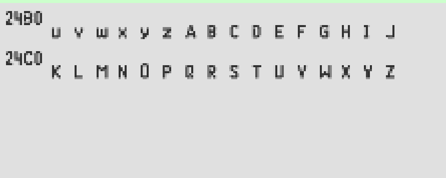
The browser FBR displays the characters of both fonts provided as designed and implemented for the *WP 43S*, sorted according to their hexadecimal codes (most of them following *Unicode*). Find them printed here to scale:











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.

	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

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-9ff).

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 also 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.

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, Bhāsa 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, Oʻzbek tili, polskì, 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:

$-1.602\ 22{\times}10^{-19}$ C	$-1{,}602\ 22{\cdot}10^{-19}$ C
$-1.602\ 22{\times}10^{-19}$ As	$-1{,}602\ 22{\cdot}10^{-19}$ 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. 163), 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:

Re(z)	Im(z)	r(z)	φ(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***; ***φ***) in polar notation will return $\left(\sqrt[n]{r}\,;\,{}^{\varphi}/_{n}\right)$. Hence, e.g. $\sqrt[3]{(\infty\,;\,180°)} = (\infty\,;60°)$, corresponding to the Cartesian point $\infty \times \left(1 + i\sqrt{3}\right)$ which the calculator will display as $\infty + i \times \infty$, converted following the calculation rules – but mathematically wrong; and $\sqrt[6]{(\infty\,;\,180°)} = (\infty\,;30°)$, corresponding to the Cartesian point $\infty \times \left(\sqrt{3} + i\right)$, would return $\infty + i \times \infty$ as well.

Integer powers of (***r***; ***φ***), on the other hand, will return $(r^n\,;\,\varphi \times n)$. E.g. $\infty + i \times \infty = (\infty\,;45°)$ squared shall return $(\infty\,;90°) = 0 + i \times \infty$. Naïve pedestrian's approach: $(\infty + i \times \infty)(\infty + i \times \infty) = \infty - \infty + 2\,i \times \infty$ $= 0 + i \times \infty$. 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-6f), the screen could take inputs of up to 23 digits, a sign, and an 8-px radix mark:

-4.23456789012345678901 23 ,

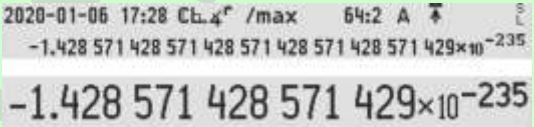
occupying 15 + 23 × 16 + 8 = 391 px. Numeric output would allow for the same 23 digits. Without digit group separators, however, this would hardly be readable. With 3-digit separators (*startup default*), 20 digits are displayable in one row instead:

-4.234 567 890 123 456 789 0 ,

taking 15 + 20 × 16 + 7 × 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

$-4.234\ 567\ 890\ 123\ 456\times10^{-925}$,

taking 395 px (= 15 + 16 × 16 + 5 × 8 + 15 + 16 + 4 × 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:



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: $-5.234\ 567\ 8\times10^{-92}$

$y=a_0+a_1\ln(x)$ a_0: $-1.234\ 567\ 890\ 12$.

Complex numbers in Cartesian notation require 1 + 15 + 12 + 15 + 1 = 44 px for +j× 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×4.234 567 89 ,

since 44 + 15 + 2 × (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 × (4 × 16 + 8 + 15 + 16 + 4 × 13) + 44 + 1 = 370 px, but another digit would need 2 × 16 px at least):

$-4.234\times10^{-925}+i\times4.234\times10^{-925}$.

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 × (5 × 16 + 3 × 8 + 16 + 4 × 13) + 37 + 1 = 397 px:

$-4{,}234\ 5\cdot10^{-925}+i\cdot4{,}234\ 5\cdot10^{-925}$.

Alternatively, 13 px wide narrow digits allow for 4 decimals even with multiplication crosses, while 5 decimals are viable with multiplication dots:

$-6.234\ 5\times10^{-925}+i\times6.234\ 5\times10^{-925}$,

$-6.234\ 56\cdot10^{-925}+i\cdot6.234\ 56\cdot10^{-925}$.

With SHOW, any *complex number* can be displayed with 34 digits precision in two rows:

2020-01-06 17:15 C⊾∡r /max 64:2 A ↑ SL

$-8.403\ 361\ 344\ 537\ 815\ 126\ 050\ 420\ 168\ 067\ 229\times10^{-126}$

$-i\times5.882\ 352\ 941\ 176\ 470\ 588\ 235\ 294\ 117\ 647\ 059\times10^{-158}$

$-1.\times10^{-33}$

$-8.403\times10^{-126}-i\times5.882\times10^{-158}$.

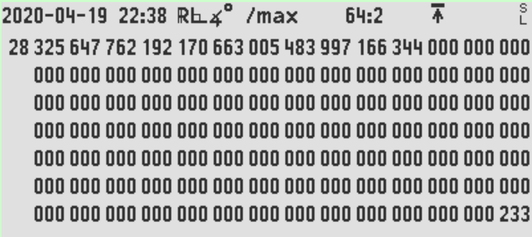
Complex numbers in **polar notation** need 4 + 16 + 4 = 24 px for ∡ plus 16 px for the angular unit in addition to the space for two signed reals. Both magnitude and angle may require a 15 px sign. 7 decimals in FIX occupy 40 + 2 × (15 + 8 × 16 + 3 × 8) = 374 px, so we can display them this way:

$-4.234\ 567\ 8\ \measuredangle-0.234\ 567\ 8\pi$.

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° or its equivalents, e.g. –π to +π 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*:

$-4.234\ 5\times10^{-925}\ \measuredangle-120.234\ 5°$.

For *radians* or *multiples of* π, however, 5 decimals are displayable always at least:



$-4.234\,56\times10^{-925}\ ∡-0.234\,56\pi$.

Digits in **fractions** are 13 px wide like in exponents. Thus, a 4-digit numerator and denominator take 4 × 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 × 16 + 3 × 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 × 16 + 6 × 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:

$-123\,456\,789\,012\,345\,678\,901\,234\,567\,890\,123\,456\times10^{21}$.

With SHOW, one *long integer* may take up to 7 rows meaning up to 296 digits (see e.g. the prime number 2.8…×10^{295} shown below with all its digits). Only the most significant 294 digits will be displayed of an integer ≥ 10^{297}, however, with ellipses added at its end. You can get all of its digits with the simulator only so far (cf. *App. E* on p. 204).

For unsigned ***short integers***, up to 21 *bits* may be shown in the usual large digits in binary representation:

$0\,1100\,0010\,1101\,0110\,0000_2$.

Base 3 (with narrow blanks every three digits) allows for displaying 20 digits and a sign:

$-22\ 211\ 200\ 201\ 120\ 001\ 212_3$.

In base 4 (with narrow blanks every two digits), 19 digits representing 38 *bits* are displayable:

$3\ 21\ 23\ 30\ 22\ 11\ 21\ 20\ 32\ 12_4$.

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:

$0\ 1110\ 1100\ 0010\ 1101\ 0110\ 0000_2$.

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:

$1110\ 1100\ 0101\ 1101\ 0110\ 1110\ 1100\ 0010\ 1101\ 0110\ 0000_2$.

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:

$1110\ 1100\ 0010\ 1101\ 0110\ 1110\ 1100\ 0010\ 1101\ 0110\ 1110\ 1100\ 0010\ 1101\ 0110\ 0000_2$.

Summing up, for given base and word size, the following fonts will do for *short integers*:

<table>
<tr><th rowspan="2">Base ▼</th><th colspan="4">Allowable size of digits for display</th></tr>
<tr><th>large</th><th>narrow</th><th>small</th><th>special</th></tr>
<tr><td>2</td><td>21 bits</td><td>25 bits</td><td>44 bits</td><td>64 bits</td></tr>
<tr><td>3</td><td>31 bits</td><td>38 bits</td><td rowspan="6">64 bits</td><td rowspan="6"></td></tr>
<tr><td>4</td><td>38 bits</td><td>44 bits</td></tr>
<tr><td>5</td><td>46 bits</td><td>55 bits</td></tr>
<tr><td>6</td><td>51 bits</td><td>62 bits</td></tr>
<tr><td>7</td><td>56 bits</td><td rowspan="2">64 bits</td></tr>
<tr><td>8</td><td>57 bits</td></tr>
</table>

Base ▼	Allowable size of digits for display			
	large	narrow	small	special
> 8	64 *bits*			

One row of four arbitrary **real matrix elements** (with absolute values $< 10^{100}$) takes 399 px in small font, SCI 3:

$$[-6{,}609\cdot10^{-19}\ \ -6{,}609\cdot10^{-19}\ \ -1{,}609\cdot10^{-19}\ \ -1{,}609\cdot10^{-19}]$$

using multiplication dots. Else you will lose one decimal. A slightly different notation allows for SCI 4:

$$[-6.609\,2\mathrm{E}^{-19}\ \ -6.609\,2\mathrm{E}^{-19}\ \ -1.609\,2\mathrm{E}^{-19}\ \ -1.609\,2\mathrm{E}^{-19}]$$

Matrices with more than four columns will need ellipses added on one or both sides:

$$[\ldots\ -6{,}609\,2\cdot10^{-19}\ \ \ -6{,}609\,2\cdot10^{-19}\ \ \ -1{,}609\,2\cdot10^{-19}\ \ldots] .$$

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 × 20 + 4 = 104 px) can be put in the space taken by 3 standard numeric rows (3 × 32 + 2 × 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

$$[\ldots\ -6.609\,226+i\cdot6.609\,226\ \ \ \ -1.609\,226+i\cdot1.609\,226\ \ldots]$$

while displaying complex matrix elements featuring large exponents may become inconvenient very soon, regardless of the symbols used:

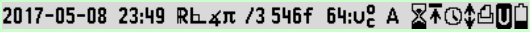
$$[\ldots\ -6.60\mathrm{E}^{-199}+i\cdot6.60\mathrm{E}^{-199}\ \ -1.60\mathrm{E}^{-199}+i\cdot1.60\mathrm{E}^{-199}\ \ldots] .$$

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:

$$[\ldots\ -6{,}609+i\cdot6{,}609\ \ \ -6{,}609+i\cdot1{,}609\ \ \ -1{,}609+i\cdot1{,}609\ \ldots] .$$

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



containing 45 characters.

Putting the alphabet in a row allows for

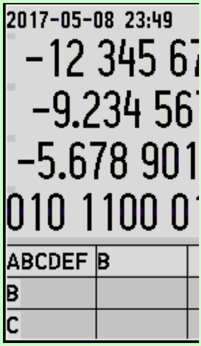
i.e. 41 characters.

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.

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-6f). 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-6) plus 4 px separating it from the next such output row. Thus, for 4 rows we need 4 × 32 + 3 × 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 × (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. 66 and B-13.

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 × 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 *WYSIWYG* 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-32), 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 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 ^); 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 √x ( ), 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 xy^{23}. For easier handling, this will be reverted when editing such an equation again.

Formatting of Reals

Just some examples to show how display works for a plain and handy number like −30 405.004 302 071 8:

	SCI	ENG	FIX
0	$-3.\times10^{4}$	$-30.\times10^{3}$	-30 405.
1	-3.0×10^{4}	$-30.\times10^{3}$	-30 405.0
2	-3.04×10^{4}	-30.4×10^{3}	-30 405.00

3	-3.041×10^{4}	-30.41×10^{3}	-30 405.004
4	-3.040 5×10^{4}	-30.405×10^{3}	-30 405.004 3
5	-3.040 50×10^{4}	-30.405 0×10^{3}	-30 405.004 30
6	-3.040 500×10^{4}	-30.405 00×10^{3}	-30 405.004 302
7	-3.040 500 4×10^{4}	-30.405 004×10^{3}	-30 405.004 302 1
8	-3.040 500 43×10^{4}	-30.405 004 3×10^{3}	-30 405.004 302 07
9	-3.040 500 430×10^{4}	-30.405 004 30×10^{3}	-30 405.004 302 072
10	-3.040 500 430 2×10^{4}	-30.405 004 302×10^{3}	-30 405.004 302 071 8
11	-3.040 500 430 21×10^{4}	-30.40 … 2 1×10^{3}	-30 405.004 302 071 80
12	-3.040 500 430 207×10^{4}	-30.40 … 2 07×10^{3}	-30 405.004 302 071 800
13	-3.040 500 430 207 2×10^{4}	-30.40 … 2 072×10^{3}	-30 405.004 302 071 800 0
14	-3.040 500 430 207 18×10^{4}	-30.40 … 071 8×10^{3}	-30 405.004 302 071 800 00
15	-3.040 500 430 207 180×10^{4}	-30.40 … 071 80×10^{3}	-30 405.004 302 071 800 000

For positive numbers, a blank space shall be displayed instead of the minus.

Now, let us look at the same for −3.040 500 430 207 18:

	SCI	ENG	FIX
0	-3.×10^{-4}	-300.×10^{-6}	-3.×10^{-4}
1	-3.0×10^{-4}	-300.×10^{-6}	-3.0×10^{-4}
2	-3.04×10^{-4}	-304.×10^{-6}	-3.04×10^{-4}
3	-3.041×10^{-4}	-304.1×10^{-6}	-3.041×10^{-4}
4	-3.040 5×10^{-4}	-304.05×10^{-6}	-0.000 3
5	-3.040 50×10^{-4}	-304.050×10^{-6}	-0.000 30
6	-3.040 500×10^{-4}	-304.050 0×10^{-6}	-0.000 304
7	-3.040 500 4×10^{-4}	-304.050 04×10^{-6}	-0.000 304 1
8	-3.040 500 43×10^{-4}	-304.050 043×10^{-6}	-0.000 304 05
9	-3.040 500 430×10^{-4}	-304.050 043 0×10^{-6}	-0.000 304 050

10	-3.040 500 430 2$\times 10^{-4}$	-304.050 043 02$\times 10^{-6}$	-0.000 304 050 0
11	-3.040 500 430 21$\times 10^{-4}$	-304.050 043 021$\times 10^{-6}$	-0.000 304 050 04
12	-3.040 500 430 207$\times 10^{-4}$	-304.050 043 020 7$\times 10^{-6}$	-0.000 304 050 043
13	-3.040 500 430 207 2$\times 10^{-4}$	-304.05 … 0 72$\times 10^{-6}$	-0.000 304 050 043 0
14	-3.040 500 430 207 18$\times 10^{-4}$	-304.05 … 0 718$\times 10^{-6}$	-0.000 304 050 043 02
15	-3.040 500 430 207 180$\times 10^{-4}$	-304.05 … 0 718 0$\times 10^{-6}$	-0.000 304 050 043 021

For small parameters (0 … 11), the displays in both tables are identical to the ones of the *HP-42S*.

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 2012-11 (the picture shows my last draft before said post, wishfully assuming the good old slanted keys of the Seventies). Instead of resurrecting an old calculator or copying an existing layout, we wanted to create a new, optimized pocket calculator based on our experience.

Even after years, the order and location of the four basic arithmetic operators on the keyboard may become subject to repeated vivid discussions. All the early *HP* pocket calculators (from the world's first scientific pocket calculator, the *HP-35* of 1972, up to the *HP-41C*, *CV*, and *CX*) presented them beneath [ENTER↑] in the order [−], [+], [×], [÷] (top down); though actually no one knows the reason for this old sorting order 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 [+], [−], [×], [÷] (bottom up). The poll of 2012-11 resulted in a majority for placing the operators beneath [ENTER↑] for ergonomic reasons.

Everything after the first layout of 2012-11 was and is just patient refinement and careful tuning of the basic idea, just row two of the keys changed significantly due to increasing support of complex numbers, and the shifted functions wandered around. The layout 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, the lack of qualified manpower for coding *WP 43S* was the lasting bottleneck in our project since 2012. It was overcome not earlier than 2021.

Menus

The community does not like deep *menus*: it prefers the *HP-32SII* to *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 employed 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. 141f). So we can conclude that the *real number* range supported (cf. p. 158) suffices <u>by far</u> for solving what has to be solved. Saving display space gave reason for RANGE.

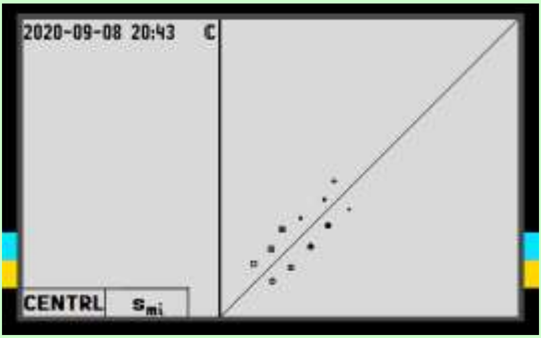
For sake of consistency, maximum numbers within different *DTs* should match. I.e. the maximum absolute value allowed for a *long integer* should be approximately equal to the respective values for a *real* and the parts of a *complex number*.

Note the number range determines the precision required for calculating accurately (see p. B-24).

Plotting

This chapter contains background considerations about PLOT, CENTRL, and s_{mi} (see the third industrial statistics application example in *Sect. 2* of the *OM*). PLOT and CENTRL require the statistical data (e.g. max. 100 experimental data points, i.e. 100 pairs of x and y values) stored point by point in a matrix, not just summed up as in earlier *RPN* calculators.

The data points collected shall be displayed in a quadratic diagram. Both axes shall be equal and reach from minimum value measured to maximum value measured (plus a little extension, see below). 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 still (max. 3 × 2 labels). The *status bar* will be partially overwritten. 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. Data points can then be plotted using POINT (containing 3×3 px); positioning them properly in the diagram, however, requires some background calculations best performed by a program (see below). 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). Settings 1 (= 0, 0) and maybe 3 (= 1, 0) herein should suffice for our plotting (cf. **GRAMOD** on p. 94).

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.

The necessary *softkey* functions could be as follow:

- PLOT shall set up the plot screen and plot the data points all at once:

  Let x_{min} and x_{max} be the minimum and maximum values for x, and y_{min} and y_{max} be the same for y. Then the diagram shall extend in either direction from $d_{min} = min(x_{min};\ y_{min}) - \delta$ to $d_{max} = max(x_{max};\ y_{max}) + \delta$ with $\delta = [max(x_{max};\ y_{max}) - min(x_{min};\ y_{min})] \times 0.05$. All the data points $(x_i;\ y_i)$ shall then be projected into the screen area between $(d_{min}; d_{min}) = (161; 1)$ and $(d_{max}; d_{max}) = (400; 240)$ and plotted there.

- CENTRL shall fit the center line for checking deviations from the 45° line:

  Select ORTHOF and let L.R. fit the line through the data points; then plot it in the diagram.

- s_{mi} shall calculate the minimum experimental standard deviation of the measuring instrument (see also p. B-25) from the variances of the data and push it on the stack, displayed at left end of **X**:

$$s_{mi}^2 = s_x^2\ s_y^2\ \frac{1 - r^2}{s_x^2 + r^2\ s_y^2}$$

  s_{mi} shall work only if more than 29 data points are accumulated.

It may be beneficial to define a general origin for graphics at a location deviating from pixel (1; 1) (i.e. the bottom left corner of the *LCD*) – the point (161; 1) 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 the quadratic drawing area then, using the commands AGRAPH, CENTRL, CLLCD, CLOSE, PIXEL, PLOT, and POINT. Though this may come in further future…

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* following the representation as explained on pp. 159ff) deviates from the truth by more than 3×10^{-35}. This looks very small though such errors accumulate during longer calculations (cf. footnote 74).

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 74). 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-5f).

Luckily, real-world problems are usually many orders of magnitude less precisely defined than the internal precision of the *WP 43S*. Compare also the precisions in the set of physical and astronomical constants provided (cf. pp. 133ff).

Prefixes

Prefixes **f** and **g** 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.

Quick Measurement System Analysis

All technical processes scatter. Measuring, although fundamental for our knowledge of the world, is a technical process as well. Measuring constant objects repeatedly will reveal the scattering of the measuring process and its width. With a high number of repeated measurements of one constant object, and combining every two subsequent measurements to one data point (x, y), the result will be a circular cloud of points. Within a given diagram, the diameter of this cloud will be the smaller the better the measuring system is.

Investigating two different objects this way will result in two such clouds, etc.

Instead of measuring just a few different objects many times, one can as well measure many different objects just two times. 30 objects are minimum required to achieve a reliable result for the uncertainty of the measuring system by determining the width of the ant track (see the *OM*, *Sect. 2*). Further analyses of the system under investigation are possible looking at the diagram.

If you sample these ≥ 30 objects from a steady state production process, you can determine not only the precision of the measuring system used but also the standard deviation of this production process. In such a case the ratio $\frac{L-W}{W}$ shall be greater than a fixed limit to warrant this measuring system controlling this process sensibly. With samples uniformly distributed over a certain range the ant track will be sausage-shaped (not elliptic!).

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. **s**, **SAVE**, **SB**, **SCI**, **SCIOVR**, **scw→kg**, ..., **SLVQ**, s_m, s_{mw}, **SOLVE**, ...

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	∛ 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	ª 00aa	$_A$ 24b6	$_a$ 2090	$_a$ 249c	
	À 00c0	à 00e0	Á 00c1	á 00e1	Â 00c2	â 00e2
	Ã 00c3	ã 00e3	Ä 00c4	ä 00e4	Å 00c5	å 00e5
	Æ 00c6	æ 00e6	Ā 0100	ā 0101	Ă 0102	ă 0103
					Ą 0104	ą 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	Č 010c	č 010d	ℂ 2102	∁ 2201
D 0044	d 0064	$_D$ 24b9	$_d$ 249f	Ð 00d0	ð 00f0	
			Ď 010e	ď 010f	Đ 0110	đ 0111

E 0045	e 0065	Ⓔ 24ba	ₑ 2091	⒠ 24a0	È 00c8	è 00e8
	É 00c9	é 00e9	Ê 00ca	ê 00ea	Ë 00cb	ë 00eb
	Ē 0112	ē 0113	Ĕ 0114	ĕ 0115	Ė 0116	ė 0117
	Ę 0118	ę 0119	Ě 011a	ě 011b	E 2073	
F 0046	f 0066	⒡ 24a1	Ⓕ 24bb			
G 0047	g 0067	⒢ 24a2	Ⓖ 24bc	Ğ 011e	ğ 011f	
H 0048	h 0068	ℎ 210e	⒣ 24a3	Ⓗ 24bd	ₕ 2095	
					ħ 0127	ℏ 210f
I 0049	i 0069	Ⓘ 24be	⒤ 24a4	Ì 00cc	ì 00ec	
	Í 00cd	í 00ed	Î 00ce	î 00ee	Ï 00cf	ï 00ef
	Ī 012a	ī 012b	Ĭ 012c	ĭ 012d	Į 012e	į 012f
					İ 0130	ı 0131
J 004a	j 006a	Ⓙ 24bf	⒥ 24a5			
K 004b	k 006b	Ⓚ 24c0	⒦ 24a6	ₖ 2096		
L 004c	l 006c	Ⓛ 24c1	⒧ 24a7	ₗ 2097	Ĺ 0139	ĺ 013a
			Ľ 013d	ľ 013e	Ł 0141	ł 0142
M 004d	m 006d	Ⓜ 24c2	⒨ 24a8	ₘ 2098		
N 004e	n 006e	Ⓝ 24c3	⒩ 24a9	ₙ 2099	Ñ 00d1	ñ 00f1
	Ń 0143	ń 0144	Ň 0147	ň 0148	ℕ 2115	
O 004f	o 006f	º 00ba	© 00a9	Ⓞ 24c4	⒪ 24aa	ₒ 2092
	Ò 00d2	ò 00f2	Ó 00d3	ó 00f3	Ô 00d4	ô 00f4
	Õ 00d5	õ 00f5	Ö 00d6	ö 00f6	Ø 00d8	ø 00f8
	Ō 014c	ō 014d	Ŏ 014e	ŏ 014f	Œ 0152	œ 0153
P 0050	p 0070	Ⓟ 24c5	⒫ 24ab	ₚ 209a		
Q 0051	q 0071	Ⓠ 24c6	⒬ 24ac	ℚ 211a		
R 0052	r 0072	⒭ 24ad	Ⓡ 24c7	Ŕ 0154	ŕ 0155	
				Ř 0158	ř 0159	ℝ 211d
S 0053	s 0073	Ⓢ 24c8	⒮ 24ae	ₛ 209b	Ś 015a	ś 015b
	Ş 015e	ş 015f	Š 0160	š 0161	ß 00df	

T 0054	t 0074	ᵀ 24af	ᵀ 22a4	ₜ 24c9	ₜ 209c	
			Ţ 0162	ţ 0163	Ť 0164	ť 0165
U 0055	u 0075	ᵤ 24ca	ᵤ 24b0	ᵤ 1d64	Ù 00d9	ù 00f9
	Ú 00da	ú 00fa	Û 00db	û 00fb	Ü 00dc	ü 00fc
	Ũ 0168	ũ 0169	Ū 016a	ū 016b	Ŭ 016c	ŭ 016d
			Ů 016e	ů 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	
			x̂ 0379	x̄ 0378	x̄ 037f	∜ 221c
Y 0059	y 0079	y 24ce	y 24b4	Ý 00dd	ý 00fd	
	Ŷ 0176	ŷ 0177	Ÿ 0178	ÿ 00ff	ȳ 0233	Ȳ 0232
Z 005a	z 007a	z 24cf	z 24b5	Ź 0179	ź 017a	Ż 017b
			ż 017c	Ž 017d	ž 017e	ℤ 2124
Α 0391	α 03b1	α 2065	ά 03ac			
Β 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	∏ 220f	π 03c0				
Ρ 03a1	ρ 03c1					
Σ 03a3	σ 03c3	ς 03c2				
Τ 03a4	τ 03c4					
Υ 03a5	υ 03c5	ύ 03cd	Ϋ 03ab	ϋ 03cb	ΰ 03b0	
Φ 03a6	φ 03c6					
Χ 03a7	χ 03c7					
Ψ 03a8	ψ 03c8					
Ω 03a9	ω 03c9	ώ 03ce				
(0028	) 0029					
[005b	⎡ 23a1	⎢ 23a2	⎣ 23a3			
	] 005d	⎤ 23a4	⎥ 23a5	⎦ 23a6		
{ 007b	} 007d					
2430	2431	2432	2433			
+ 002b	⁺ 207a	₊ 208a	± 00b1			
- 002d	⁻ 207b	⁻¹ 2072	₋ 208b	∓ 2213		
× 00d7	· 00b7	∙ 2219	∘ 2218	* 002a	* 208f	
/ 002f	\ 005c					
^ 005e						
, 002c	2429					
. 002e	2428	… 2026				
! 0021	¡ 00a1					
? 003f	¿ 00bf					
: 003a	∶ 2236	÷ 00f7				
; 003b						
' 0027	‘ 2018	’ 2019	‚ 201a	‛ 201b	[105]	

[105] Please 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.

" 0022	“ 201c	” 201d	„ 201e	‟ 201f	« 00ab	» 00bb
@ 0040						
_ 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		
≪ 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	∬ 222c	∭ 222d	∮ 222e	∯ 222f	∰ 2230	
⊙ 2299	⊙ 229a	⊙ 2068				
⊕ 2295	⊕ 2069					
∟ 221f	⊥ 22a5					
∠ 2220	∡ 2221	∢ 2222				
⌈ 2308	⌉ 2309					
⌊ 230a	⌋ 230b					
2399	⌛ 231b	231a	242a	242c	242f	USB 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. 39 as well as M.EDI and M.EDIN on p. 50.

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 Σ−).

It turned out that undoing ENTER and EXIT can be ambiguous. Comparison with *WP 31S* was not very helpful since the functionality of EXIT there deviates from EXIT on the *HP-42S* and *WP 43S*.

				1		
		1	1	2	1	1
	1_	1	2_	2	1	123_
WP 31S	**1**	**ENTER↑**	**2**	**ENTER↑**	**UNDO**	**123**

				1		1
		1	1	2	1	1
	1_	1	2_	2	1	123_
WP 43S	**1**	ENTER↑	**2**	ENTER↑	UNDO	**123**

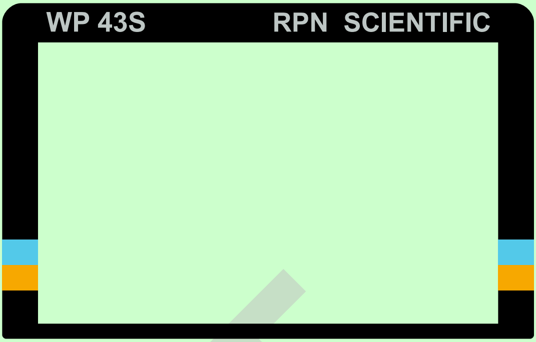
			1	1		1
	1_	1	2_	2	1	123_
WP 43S	**1**	EXIT	**2**	EXIT	UNDO	**123**

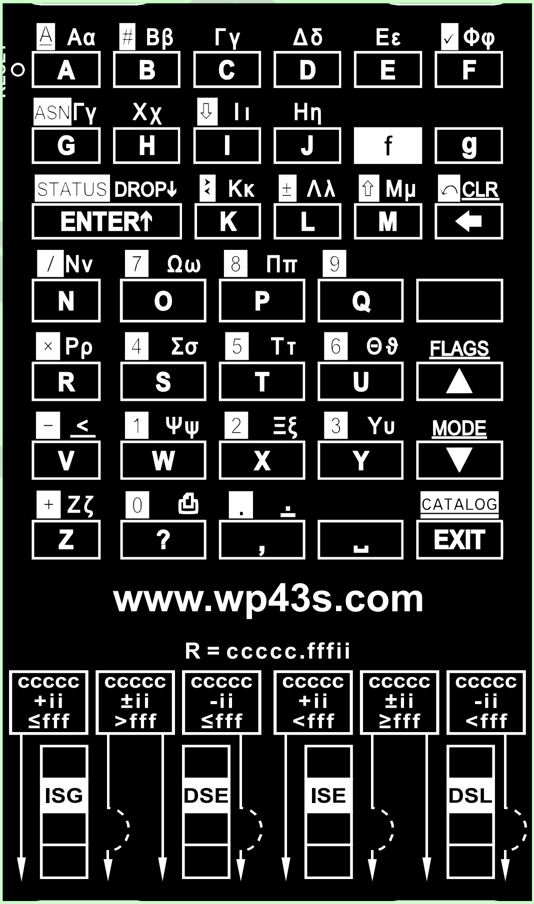
For Your Convenience

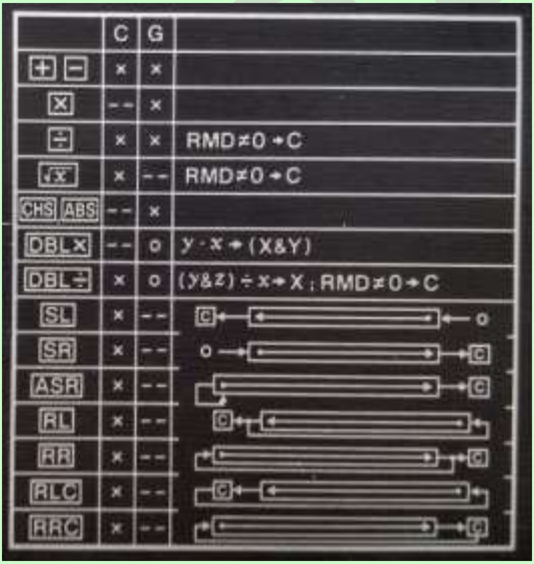
Please find here the display frame of your *WP 43S* as we designed it (printed to scale), and below its virtual key-board in alpha input mode (*AIM*) plus a branching helper.

Overleaf, the original keys and keyplate of your *WP 43S* are printed to scale. You 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 shall be found on its rear for your reference.

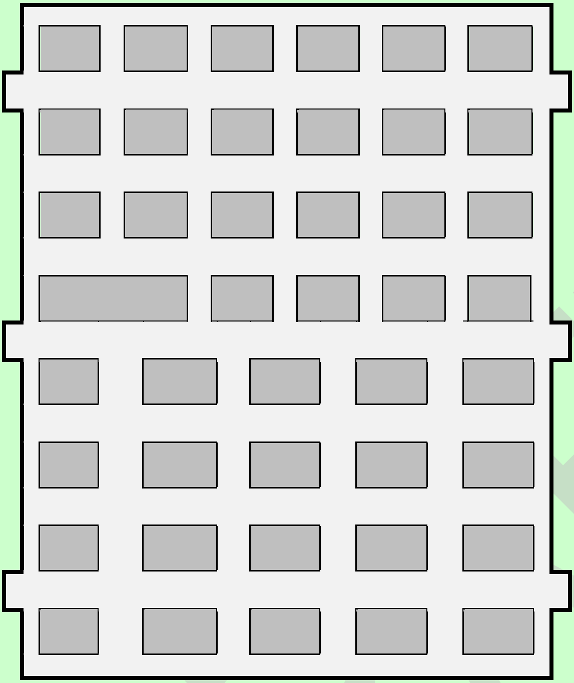






	C	G	
+ −	×	×	
×	--	×	
÷	×	×	RMD≠0→C
√x	×	--	RMD≠0→C
CHS ABS	--	×	
DBL×	--	o	y·x→(X&Y)
DBL÷	×	o	(Y&Z)÷x→X; RMD≠0→C
SL	×	--	
SR	×	--	
ASR	×	--	
RL	×	--	
RR	×	--	
RLC	×	--	
RRC	×	--	

Here is a simple template for a quick overlay (almost to scale – see point 3 below):



1. Print it 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: The tabs (a.k.a. tongues) as drawn are too wide and too long, cut smaller.