WP 43S REFERENCE MANUAL

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

DRAFT

  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. 151 and bottom of p. 152 were kindly supplied by *SwissMicros* as well as the drawing on p. 200, the picture on p. 198 by *Martin Lorang*. The plots in *Appendix H* are based on material found in *Wikipedia*. The other pictures, diagrams, and graphics were created by the author.

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

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

Printed in the USA

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

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

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

Statement of Corporate Objectives
Hewlett-Packard

TABLE OF CONTENTS

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 and MENUS 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. 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 of this font 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 commas as well, of course). Although that point is less visible than a comma, 'comma people' seem to be more

tolerant against points used as radix marks than vice versa (based on the number of complaints read).

All this holds unless stated otherwise locally.

The following abbreviations are used throughout this manual:

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

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

BCD = binary coded decimal.

CDF = cumulated distribution function (see *Section 2* of the *OM*).

DP = double precision.

FM = flash memory (a special kind of *RAM*, see *Sect. 3* of the *OM*).

HP = *Hewlett-Packard.*

IOI = *Index of Items* (see pp. 12ff).

LCD = liquid crystal display.

PDF = probability density function (see *Section 2* of the *OM*).

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 operations.

RPN = reverse Polish notation (see *Section 1* of the *OM*).

SP = single precision.

SRS = subroutine return stack (see *App. B* on pp. 155ff).

TVM = *Time Value of Money* – a preprogrammed application for dealing with investments and loans, featured by all financial *HP* calculators since 1972 (see *Sect. 5* of the *OM*).

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 below with their *names* (as they are printed in routines) in column 1 and the keystrokes necessary to call them. Most *items* shall be picked from *menus* (see pp. 113ff).[1]

> There is an important difference between the *names* of *items* and their labels as printed on the keyboard or displayed in *menus*: **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**, featuring less or more characters than its *name* (see some unit conversions, for example).[2]

On your *WP 43S*, sorting (e.g. of *names*) works in the following order:[3]

␣ 0…9 10 11…16 Aa…Zz Αα…Ωω ( ) [ ] { }
+ - × / \ ^ ± , ; . : ! ¡ ? ¿ : ÷ ; ' " « » · • * @ _ ~
→ ← ↑ ⤒ ↓ ≷ ⇕ ¬ ∧ ∨ & | ‖
« < ≤ ≡ := = ≃ ≈ ≘ ≙ ≠ ≥ > » % $ € ¢ £ ¥
√ ∞ ∫ ⊙ ⊕ ⊾ ∠ ∡ ⊥ 🕓 ⌛ 🖶 🔋 # 🅤 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 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 calculators as mentioned in the index below, their manuals may contain helpful additional information.

---

[1] For commands stored in *menus*, we list the keys calling the respective *menu*, the *prefix* of the respective *menu* row (if applicable), and the command as shown therein. We are confident you will find the corresponding *softkey*. *Items* stored in CNST are listed with their *names* only, however, since they are sorted alphabetically and will be explained in detail in a separate chapter below.

[2] These labels are not required to be unique.

[3] Characters printed on grey background are inaccessible for users for the time being. The entire sorting table is printed in an appendix.

Operations working with the accumulated statistical data are marked light blue. Those operating also on complex parameters are marked light yellow. Operations asking you for confirmation are printed red.

All operations may be entered in *PEM* as well unless marked violet or stated otherwise – many functions contained in P.FN and TEST will make most sense in *PEM*.

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

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

(1) is for *monadic functions*,

(2) for *dyadic functions*, and

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

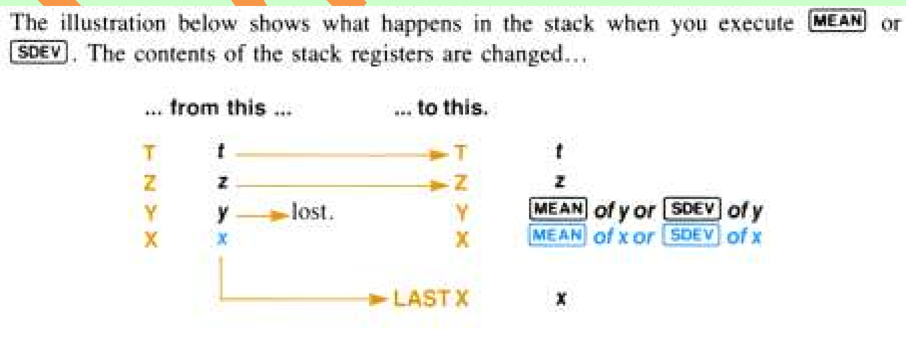
(−1) stands for functions pushing one object on the *stack* and

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

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

---

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


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

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

Operation or function ***parameters*** will be taken from the lowest *stack register*(s) unless mentioned explicitly in second column 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.

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 → ST.X and it will take its parameter from **X** instead – like a good old traditional *RPN* command.

Automatic stack lift is enabled after each command – except CLX, ENTER↑, Σ+, and Σ− (cf. *Section 1* of the *OM*); numeric input immediately following one of these four operations will overwrite x instead of pushing it on the *stack* as usual.[5]

Some 300 functions featured in your *WP 43S* are new compared to *HP's RPN* pocket calculators.[6] Operations carrying familiar *names* but deviating in their functionality from previous *HP RPN* calculators or the *WP 34S* are marked light red.

The *data types* a particular function operates on are listed in { } under "remarks" if there are restrictions – cf. *App. B* on pp. 155ff. Most bit and integer functions operate on *short integers* only (*data type* 10). The other functions typically work with more kinds of objects. Functions stating *data types* 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 *data types* differing from their input, the output types are listed as well.[7]

---

[5] Some reasoning why *automatic stack lift* is disabled for these four:

a) CLX is for clearing **X** to make room for a corrected value. This value shall overwrite x – an extra zero on the stack would make no sense.

b) ENTER↑ is a *stack* lift manually initiated by the user. An additional *automatic stack lift* immediately after this command would make 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.

[6] We did not compare the *RPL* calculators of the last three decades or the *HP Prime*. They are exceeding the realm of shirt pocket calculators.

[7] This applies for °C→°F, for instance: For *SP* (*DP*) real input, output will stay *SP* (*DP*) real. For integer input, however, output will be *SP* real.

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

Below, the functions checked already are highlighted green, those which didn't work (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 so far isn't highlighted at all. This applies to the respective *data types*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
°C→°F °F→°C	g U→ f °C→°F etc.	(1) {2, 11}; {1} → {2} Convert temperatures. See pp. 135ff.
10^x	g 10^x	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} Returns 10^x, the inverse of $\lg(x)$.
1COMPL	f BITS ▼ 1COMPL g INTS ▲ ...	(0) Sets 1's complement mode for operations on *short integers*. Indicated in the status bar. See *Sect. 2* of the *OM*.
½	g CNST ½	(−1) { } → {2} Trivial but helpful constant for iterations.
$^1/_x$	$^1/_x$	(1) {2, 3, 8*, 9*, 11, 12}; {1} → {2} Inverts the number x or all elements of the matrix x.
2COMPL	f BITS ▼ 2COMPL g INTS ▲ ...	(0) Sets 2's complement mode for operations on *short integers*. Indicated in the status bar. See *Sect. 2* of the *OM*.
2^x	g EXP 2^x	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} Returns 2^x.
$\sqrt[3]{x}$	g EXP f $\sqrt[3]{x}$	(1) {1, 2, 3, 8*, 9*, 10, 11, 12}; ({1} → {2}) Returns the cube root of x. Roots of non-cube *long integers* will return reals.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
a	g CNST a	(−1) { } → {2} *Gregorian* year in *days* and *Bohr* radius in *meter*.
a_0	g CNST a_0	
ABS	f CAT. FCNS ABS	Points to $\lvert x \rvert$ on p. 100. Maintained for backward compatibility only.
ACOS	f CATALOG FCNS ACOS	Points to arccos on p. 18. Maintained for backward compatibility only.
ac→m²	g U→ f A: acre → m²	(1) {2, 11}; {1} → {2} Convert areas. See pp. 135ff.
ac_{US}→m²	g U→ f A: $acre_{US}$ → m²	
ADV	f ADV	*Menu*. See p. 119.
AGM	g X.FN AGM	(2) {2, 3, 11, 12}; {1} → {2} Returns the *arithmetic-geometric mean* of x and y. Will throw an error for x or y being negative. See p. 234 for more.
AGRAPH	g P.FN P.FN2 g 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 column. $1 \le x \le 400$ and $1 \le y \le 232$ are valid (but see *App. K*). So one row (8 px high) starting in column 1 may need up to 400 characters to specify – the more blank space is found therein the less characters may be required for describing it entirely. Cf. *HP-42S Owner's Manual*, pp. 135 – 140, and *HP-42S Programming Examples and Techniques*, pp. 214 – 223.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ALL	g DISP ALL *n*	(0) Sets the numeric display format to show all decimals of real or complex *SP* or *DP* numbers whenever displayable (trailing decimal zeros will not be shown). ALL 00 works like ALL in *HP-42S* almost. For $x \geq 10^{16}$ (or earlier for complex numbers), however, display will switch to SCI or ENG … with the maximum number of necessary decimals displayable using the large font (see SCIOVR and ENGOVR). The same will happen if $x < 10^{-n}$ <u>and</u> 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. 64.
a_{Moon}	g CNST a_{Moon}	(−1) { } → {2} Semi-major axis of the Moon's orbit around the earth in *meter*.
AND	f BITS AND	(2) {10} Works bitwise as in *HP-16C* (see the *OM, Sect. 2*).
		(2) {1, 2, 11} → {1} Works like AND in *HP-28S*, i.e. x and y are interpreted before executing this operation. Zero is 'false'; any other real number is 'true'.
ANGLES	f CAT. VARS ANGLES	*Submenu* of tagged angular variables defined at execution time. See pp. 113f.
arccos	TRI arccos	(1) {3, 8*, 9*, 12}; {1, 2, 11} → {4}; Returns the tagged angle $arccos(x)$.[8]
arcosh	g EXP g arcosh TRI g arcosh	(1) {2, 3, 8*, 9*, 11, 12} Returns $arcosh(x)$.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
arcsin	TRI arcsin	(1) {3, 8*, 9*, 12}; {1, 2, 11} → {4}; Returns the tagged angle $arcsin(x)$.[9]
arctan	TRI arctan	(1) {3, 8*, 9*, 12}; {1, 2, 11} → {4}; Returns the tagged angle $arctan(x)$.[10]
arsinh	g EXP g arsinh TRI g arsinh	(1) {2, 3, 8*, 9*, 11, 12} Returns $arsinh(x)$.
artanh	g EXP g artanh TRI g artanh	(1) {2, 3, 8*, 9*, 11, 12} Returns $artanh(x)$.
ASIN	f CATALOG FCNS ASIN	Points to arcsin above. Maintained for backward compatibility only.
ASR	f BITS f ASR ***n***	(1) {10} Works like ***n*** (≤ 63) consecutive ASR commands in *HP-16C*, corresponding to a division of ***x*** by 2^n. ASR **0** executes as NOP, but loads **L**. See *Sect. 2* of the *OM*.
ASSIGN	f ASN ***item, location***	(0) Assigns an *item* (i.e. a function, a *menu*, a label, or a character) to a specified sequence of keystrokes, corresponding to a specific location on the keyboard or in a *menu*. See *Section 6* of the *OM*.
ATAN	f CATALOG FCNS ATAN	Points to arctan above. Maintained for backward compatibility only.

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

10 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 *flag* **D** 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)
atm→Pa	g U→ F&p: f atm→Pa	(1) {2, 11}; {1} → {2} Convert pressures and distances. See pp. 135ff.
au→m	g U→ x: au→m	
A…Z	f CATALOG CHARS A…Z	*Submenu* of Latin letters. See pp. 113ff.
A:	g U→ f A:	*Submenu*. See p. 135.
$a_{\oplus}$	g CNST $a_{\oplus}$	(−1) { } → {2} Semi-major axis of the Earth's orbit around the sun in *meter*.
BACK	g P.FN P.FN2 g 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 ⭱. Cf. 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	g U→ F&p: bar→Pa	(1) {2, 11}; {1} → {2} Converts pressures. See pp. 135ff.
BATT?	g INFO f BATT?	(−1) { } → {2} Measures the battery voltage in the range between 1.9*V* and 3.4*V* and returns this value.
bbl→m^3	g U→ f V: f barrel → m^3	(1) {2, 11}; {1} → {2} Converts volumes. See pp. 135ff.
BC?	f BITS g BC? ***n***	(−1) {10} Tests if the specified bit in x is clear.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
BEEP	g I/O BEEP	(0) Sounds a sequence of four tones. See also TONE and QUIET.
BeginP	g FIN TVM f Begin	(0) Sets "Begin" mode in *TVM*: payments occur at the beginning of each period. Typical for savings plans and leasing. Cf. ENDP.
BestF	f 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 fit models making no sense for your data (e.g. for physical or technical reasons). The parameter ***n*** carries this information. Each fit model corresponds to a number as listed: • LINF 1 • EXPF 2 • LOGF 4 • POWERF 8 • ROOTF 16 • HYPF 32 • PARABF 64 • CAUCHF 128 • GAUSSF 256 Take the numbers of all models you can exclude and sum them up – the result is ***n***. **Example:** Excluding the three 3-parameter models results in $n = 64 + 128 + 256 = 448$. So call BESTF **448** to look for the best-fitting 2-parameter model..

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		Note ORTHOF is not part of the set of models under investigation. See pp. 204ff for more.
Binom Binom_e Binom_p Binom^{-1}	g PROB g Binom: Binom etc.	(1) {2, 11} *Binomial distribution* with the number of successes ***g*** in **X**, the probability of a success p_0 in **I**, and the sample size ***n*** in **J**. See p. 204 for more. Binom^{-1} returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, p_0 in **I** and ***n*** in **J**.
Binom:	g PROB g Binom:	*Submenu*. See p. 122.
BITS	f BITS	*Menu*. See p. 117.
B_n	g X.FN B_n	(1) {1, 2, 11} B_n and B_n^* return the Bernoulli number for an integer ***n*** > 0 given in **X**, working with different definitions (see both formulas on p. 202).
B_n^*	g X.FN B_n^*	
BS?	f BITS g BS? ***n***	(0) {10} Tests if the specified bit in ***x*** is set.
Btu→J	g U→ E: Btu→J	(1) {2, 11}; {1} → {2} Converts energies. See pp. 135ff.
c c_1 c_2	g CNST c etc.	(−1) { } → {2} Speed of light in vacuum in *meter per second*; first and second radiation constants in *Planck's* Law (see p. 129).
cal→J	g U→ E: cal→J	(1) {2, 11}; {1} → {2} Converts energies. See pp. 135ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CASE	g P.FN P.FN2 g CASE s	(0) Works like SKIP below 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 ... 153 LBL 05 ... 234 LBL 07 ... In execution of this program, *r12* will be checked in step 100: if *r12* ≤ 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 ≤ *r12* ≤ 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	f CATALOG	*Catalog* of everything. See pp. 113ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Cauch $Cauch_e$ $Cauch_p$ $Cauch^{-1}$	g PROB f Cauch: Cauch etc.	(1) {2, 11} *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. 207 for more. $Cauch^{-1}$ returns x for a given probability p in **X**, with x_0 in **I** and γ in **J**.
Cauch:	g PROB f Cauch:	*Submenu*. See p. 122.
CauchF	f STAT ▼ f 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. 213ff for more.
CB	f BITS g CB *n*	(1) {10} Clears the specified bit in x, i.e. sets it to **0**.
CEIL	g INTS g CEIL	(1) {8*}; {1, 2, 11} → {1} Returns the smallest integer ≥ x. Cf. FLOOR.
CF	g FLAGS CF *n* g CLR CF *n*	(0) Clears the *flag* specified, i.e. sets it to **0**.
CHARS	f CATALOG CHARS	*Submenu* of characters. See pp. 113ff.
CLALL	g CLR g CLall	(0) Clears all *registers*, user *flags*, variables, and programs in *RAM*. Modes will stay as they are. Cf. CLCVAR, CLFALL, CLPALL, CLREGS, CLSTK, and RESET.
CLCVAR	g CLR CLCVAR	(0) Clears all variables used in the *current program*, i.e. sets all such real and complex variables to **0.**, all integer ones to **0**, all *time* variables to **0:00:00**, all *date* variables to **January 1st** of year **0**, all character strings to zero length, and all the elements of all matrix variables used to zero.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CLFALL	g FLAGS g CLFall g CLR f CLFall	(0) Clears all global and local user *flags*. Compare CF.
CLK	g CLK	*Menu*. See p. 113.
CLK12	g CLK ▲ f CLK12	(0) Sets 12h time display mode: 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. Cf. CLK24.
CLK24	g CLK ▲ f CLK24	(0) Sets international 24h time display mode. Expands the date display in the *status bar* to four digits for the year. Cf. CLK12.
CLLCD	g CLR f CLLCD	(0) Clears the *LCD* in the rectangular window north and west of the point x, y. I.e. all pixels ≥ x and ≥ y are cleared.
CLMENU	g CLR CLMENU g P.FN P.FN2 ...	(0) Clears all *menu* key definitions for the programmable *menu*. See MENU.
CLP	g CLR CLP	(0) Clears the *current program* in *RAM* or *FM*. Freed memory is returned to the pool of free space.
CLPALL	g CLR f CLPall	(0) Clears all programs in *RAM*. Cf. CLP.
CLR	g CLR	*Menu*. See p. 119.
CLREGS	g CLR f CLREGS	(0) Clears all global and local general purpose *registers* allocated (see also LOCR), i.e. sets all these registers to **0**. The contents of the *stack* and **L** are kept.
CLSTK	g CLR f CLSTK 0 g FILL	Clears all *stack registers* currently allocated (i.e. either **X** … **T** or **X** … **D**). All other *register* contents are kept. Cf. CLREGS.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CLX	g CLR CLX ←	(1) Clears *register* **X**, disabling *automatic stack lift*. The shortcut works for closed x only. Cf. CLREGS.
CLΣ	f STAT g CLΣ g CLR CLΣ	(0) Clears the statistical summation *registers* and releases the memory allocated for them (see p. 158).
CNST	g CNST	*Menu*. See CONST below and pp. 135ff.
COMB	g 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. Cf. PERM. (2) {2, 3, 11, 12} See pp. 202ff for the formula.
CONJ	g CPX conj	(1) {3, 9*, 12} Returns the complex conjugate of x.
CONST	g P.FN f CONST n	(−1) { } → {2} Returns the constant stored at position n in CNST (see pp. 127ff). Allows for indirectly addressing these constants.
CONVG?	g TEST g CONVG? r	(0) {2, 11} Checks for convergence by comparing x and y as determined by the lowest five bits of r. a) The very lowest two 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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		c) The top bit tells how special numbers are handled: 0 = NaN and ±∞ are considered converged, 1 = they are not considered converged. Now, $r = a + 4b + 16c$.
CORR	f STAT ▲ r	(−1) { } → {2} Returns the *coefficient of correlation* for the current statistical data and curve fit model. See pp. 213ff for more.
cos	TRI cos	(1) {2, 3, 8*, 9*, 11, 12}; {1, 4} → {2} Returns the cosine of the angle in **X** (see *Section 2* of the *OM* for details).
cosh	g EXP g cosh TRI g cosh	(1) {2, 3, 8*, 9*, 11, 12} Returns the hyperbolic cosine of x.
COV	f STAT ▲ f COV	(−1) { } → {2} Returns the population covariance for the two data sets entered via Σ+, depending on the curve fit model selected. See s_{XY} for the sample covariance and pp. 204ff for more.
CPX	g CPX	*Menu*. See p. 120.
CPXi CPXj	g CPX g CPXi etc.	(0) Selects either the letter $\boldsymbol{i}$ or $\boldsymbol{j}$ for displaying the imaginary number $\boldsymbol{i}$.
CPXRES	g MODE f CPXRES g FLAGS SF I	(0) Allows for complex results of real number calculations. Indicated in the status bar. Cf. REALRE.
CPXS	f CAT. VARS CPXS	*Submenu* of complex variables defined at execution time. See pp. 113f.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CPX?	g TEST ▲ CPX?	(0) Checks if x is complex. Returns true if $\mathbf{X}$ contains data of type 3, 9, or 12 with nonzero imaginary part.
CROSS	f MATX f cross	(2) {8} Requires two real 2D or 3D vectors in $\mathbf{X}$ and $\mathbf{Y}$ and returns their cross product. Crossing of 2D vectors works as for complex numbers.
	g CPX cross	(2) {3} → {2}; {12} → {11} 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.
ct→kg	g U→ m: f carat→kg	(1) {2, 11}; {1} → {2} Convert masses. See pp. 135ff.
cwt→kg	g U→ m: cwt→kg	
CX→RE	CC (works in run mode only)	(−1) {3} → {2}; {9} → {8}; {12} → {11} Cuts a closed complex object x, putting either • (for ⊾) its real part in $\mathbf{Y}$ and its imaginary part in $\mathbf{X}$ or • (for ⊙) magnitude in $\mathbf{Y}$ and phase in $\mathbf{X}$.
	g CPX f CX→RE	
DATE	g CLK DATE	(−1) { } → {6} Recalls the date from the real-time clock and displays it in the format selected. See D.MY, M.DY, and Y.MD. Furthermore, DATE shows the day of week (see *Sect. 2* of the *OM*).
DATES	f CATALOG VARS f DATES	*Submenu* of *date* variables defined at execution time. See pp. 113f.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DATE→	g CLK DATE→	(−2) {2, 6, 11} → {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	g CLK f DAY	(1) {2, 6, 11} → {1} Assumes x containing a *date* in the format selected (or a real number in corresponding format) and extracts the day.
DBLR	g INTS g DBLR etc.	{10} Double *word* length commands for remainder, multiplication and division.[11] 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/		
DBL?	g TEST ▲ f DBL?	(0) Checks if x contains a double precision real or complex number.
dB→fr	g U→ ▲ dB →field ratio	(1) {2, 11}; {1} → {2} Convert ratios. See pp. 135ff.
dB→pr	g U→ ▲ dB →power ratio	
DEC	f LOOP f DEC *r*	(0) {1, 2, 10, 11} Decrements r by 1. Does not load **L** even for target address **X**.

[11] See the *HP-16C Owner's Handbook*, *Section 4* (pp. 52ff).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DECOMP	f PARTS DECOMP	(−1) {1, 2, 11} → {1} Decomposes x (after converting it to an *improper fraction*, if applicable), returning a *stack* [***denominator(x), numerator(x),*** …]. Reversible by division. **Example:** If **X** contains 2.25 then DECOMP will return $x = 4$ and $y = 9$.
DEG	g MODE DEG	(0) Sets the *ADM* to *decimal degrees*. Indicated in the status bar.
DEG→	f L→ f DEG→	(1) {1, 2, 11} → {4} Converts angles as described on p. 142.
DENANY	g MODE g DENANY	(0) Sets default fraction display format like in *HP-35S* – any denominator up to the value set by DENMAX may appear. Indicated in the status bar. This is the most precise way of displaying a decimal number as a fraction with DENMAX given. **Example:** If DENMAX = 5 then DENANY allows denominators 1, 2, 3, 4, and 5.
DENFAC	g MODE g DENFAC	(0) Sets 'factors of the maximum denominator', i.e. the denominator may be an integer factor of DENMAX only. Indicated in the status bar. **Example:** If DENMAX = 60 then DENFAC will allow for denominators 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60. Now you know why 60 was a holy number in ancient Babylon.
DENFIX	g MODE g DENFIX	(0) Sets fixed denominator format, i.e. the one and only denominator allowed is the value set by DENMAX. Indicated in the status bar.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DENMAX	g MODE g 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. Indicated in the status bar. For $x = 1$, the current DENMAX setting is recalled, replacing x.
DET	f CATALOG FCNS DET	Points to |M| explained on p. 100. Maintained for backward compatibility only.
DIGITS	f CAT. DIGITS	*Submenu* of digits. See pp. 113f.
DISP	g DISP	*Menu*. See p. 120.
DOT	g CPX dot	(2) {3} → {2}; {12} → {11} Returns $Re(x) \cdot Re(y) + Im(x) \cdot Im(y)$
	f MATX f dot	(2) {8} → {2}; {9} → {3} Requires two matrices in x and y and returns their dot (scalar) product. The dot product is defined as the sum of the products of the corresponding elements in both matrices. Note both matrices must be of the same size; else DOT will throw an error. See the *OM*, *Sect. 2.*
DROP	g DROP↓	Drops x … from the stack. See *Section 1* of the *OM* for details.
DROPy	g STK DROPy	Drops y
DSE	f LOOP DSE ***r***	(0) {1, 2, 10, 11} Given `ccccc.fffii` in the source, DSE decrements r by `ii`, skipping next program step if then `ccccc` ≤ `fff`. 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 destination address **X**. Note that neither `fff` nor `ii` can be negative, and DSE is only sensible with `ccccc` > 0.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DSL	f LOOP DSL *r*	(0) {1, 2, 10, 11} Works like DSE but skips if `ccccc` < `fff` .
DSTACK	g DISP ▲ f 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 multi-level returns like SUM will work as described in the *OM* regardless of the number chosen for DSTACK. In any case, command input will be echoed directly below the *status bar*. This command is for old-school calculator users who may feel distracted by a multitude of *stack registers* displayed changing simultaneously.
DSZ	f LOOP DSZ *r*	(0) {1, 2, 10, 11} Decrements r by 1 and skips the next step if $\lvert r \rvert < 1$ thereafter. Does not load **L** even for target address **X**. Cf. *HP-29C*, *HP-67*, *HP-16C*.
D.MS	f d.ms (for closed input)	(0) Sets the *ADM* to *sexagesimal degrees*. Indicated in the status bar.
D.MS→	f L→ f D.MS→	(1) {1, 2, 11} → {4} Convert angles as described on pp. 142f.
D.MS→D	f L→ g D.MS→D	
D.MY	g CLK ▲ D.MY	(0) Sets the format `dd.mm.yyyy` for *dates*.
D→D.MS	f L→ g D→D.MS	(1) {1, 2, 11} → {4} Converts angles as described on p. 142.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
D→J	g CLK f D→J	(1) {2, 6, 11} → {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*[12] according to J/G setting.
D→R	f L→ g D→R	(1) {1, 2, 11} → {4} Converts angles as described on p. 142.
e	g CNST e	(−1) { } → {2} Elementary charge in *coulomb* and *Euler's* e.
e_E	g CNST e_E	
EIGVAL	f MATX ▲ g EIGVAL	(−1) {8, 9} Evaluates the matrix x and pushes a diagonal matrix containing its eigenvalues on the *stack*.
EIGVEC	f MATX ▲ g EIGVEC	(−1) {8, 9} Evaluates the matrix x and pushes a matrix containing its eigenvectors on the *stack*.
END	g P.FN END	(0) Last command in a program and terminal for searching local labels as described in the *OM, Section 3*. Works like RTN in all other aspects.
ENDP	g FIN TVM f End	(0) Sets "End" mode in *TVM*: payments occur at the end of each period. Typical for loans and investments. Cf. BEGINP.
ENG	g DISP ENG *n*	(0) Sets engineer's display format (see *Section 2* of the *OM*).
ENGOVR	g DISP ▲ ENGOVR	(0) Defines that numbers exceeding the range displayable in ALL or FIX will be shown in engineer's format. Cf. SCIOVR.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ENORM	f MATX g ENORM	(1) {8, 9} → {2} Calculates the Euclidean norm of the matrix in **X**. The Euclidean norm is defined as the square root of the sum of squares of all matrix elements. Works like FNRM on *HP-42S*. For a vector, ENORM returns its length. Cf. |x| on p. 100.
ENTER↑	ENTER↑	(−1) Separates two entries in input. Copies x into **Y**, disabling *automatic stack lift*. See p. 110 and the *OM* (*Section 1*) for details.
ENTRY?	g TEST g 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	g EQN	*Menu*. See p. 120.
EQ.DEL	g EQN f DELETE	Deletes an equation. (See *Section 4* of the *OM*.)
EQ.EDI	g EQN EDIT	Opens the *Equation Editor* to edit an existing equation. (See *Section 4* of the *OM*.)
EQ.NEW	g EQN NEW	Opens the *Equation Editor* to enter a new equation. (See *Section 4* of the *OM*.)
erf erfc	g X.FN erf etc.	(1) {2, 11}; {1} → {2} Returns the error function or its complement. See pp. 234ff for more.
ERR	g P.FN ERR *n*	(0) Raises the error specified. The consequences are the same as if the corresponding error really occurred, so e.g. a running routine will be stopped and the message will be thrown. See *App. C* on pp. 169ff for the respective error codes. Cf. MSG.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
EVEN?	g TEST f EVEN?	(0) Checks if x is integer and even.
e^x	e^x	(1) {2, 3, 8*, 9*, 11, 12}; {1, 10} → {2} Returns e^x.
EXITALL	g P.FN P.FN2 EXITall	(0) Exits all *menus*.
EXPT	f PARTS EXPT	(1) {1, 2, 11} → {1} Returns the exponent $\boldsymbol{h}$ of the number $x = m \cdot 10^h$ displayed. Cf. MANT.
EXP	g EXP	*Menu*. See p. 120.
ExpF	f STAT ▼ ExpF	(0) Selects the exponential curve fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 204ff for more.
Expon Expon_e Expon_p Expon^{-1}	g PROB f Expon: Expon etc.	(1) {2, 11} *Exponential distribution* with the rate λ in $\mathbf{J}$. See pp. 204ff for more. Expon^{-1} returns the survival time $\boldsymbol{t_s}$ for a given probability $\boldsymbol{p}$ in $\mathbf{X}$, with λ in $\mathbf{J}$.
Expon:	g PROB f Expon:	*Submenu*. See p. 122.
EXPT	f PARTS EXPT	(1) {1, 2, 11} → {1} Returns the exponent $\boldsymbol{h}$ of the number $x = m \cdot 10^h$ displayed. Cf. MANT.
e^x-1	g EXP f e^x-1	(1) {2, 8*, 11} For $\boldsymbol{x} \approx 0$, this returns a more accurate result for the fractional part than e^x does.
E:	g U→ E:	*Submenu*. See p. 135.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
e/m_e	g CNST e/m_e	(−1) { } → {2} Electron charge to mass ratio in *coulomb per kilogram*.
F	g CNST F	(−1) { } → {2} *Faraday* constant in *coulomb per mol*.
FAST	g MODE f FAST	(0) Sets the processor speed to 'fast'. This is *startup default* and is kept for fresh batteries. Cf. SLOW.
FB	f BITS g FB *n*	(1) {10} Inverts ('flips') the specified bit in ***x***.
FBR	g α.FN g FBR	(0) Font browser. Shows all characters implemented in the 2 fonts designed for your *WP 43S*.
FCNS	f CAT. FCNS	*Submenu* of provided functions. See pp. 113ff.
FC?	g FLAGS FC? *n*	(0) Tests if the specified *flag* is clear.
FC?C FC?F FC?S	g FLAGS f FC?C *n* etc.	(0) Tests if the specified *flag* is clear. Clears, flips, or sets this *flag* after testing, respectively.
$F_e(x)$ $F_p(x)$ F(x) $F^{-1}(p)$	g PROB F: $F_e(x)$ etc.	(1) {2, 11} *Fisher's F distribution*. $F_u(x)$ equals Q(F) on *HP-21S*. The degrees of freedom are specified in **I** and **J**. See pp. 204ff for more.
FF	g FLAGS FF *n*	(0) Flips the *flag* specified.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FIB	g X.FN f FIB	(1) {1} Returns the Fibonacci number (see pp. 202f).
		(1) {2, 3, 11, 12} Returns the extended Fibonacci number.
FILL	g FILL	Copies x to all *stack registers*.
FIN	g FIN	*Menu*. See p. 120.
FIX	g DISP FIX *n*	(0) Sets fixed point display format (see the *OM, Sect. 2*).
FLAGS	g FLAGS	*Menu*. See p. 120.
FLASH	f CATALOG PROGS FLASH	*Submenu* of global labels defined at execution time. See pp. 113f.
FLASH?	g INFO f FLASH?	(−1) { } → {1} Returns the number of free *words* in *FM* (1 *word* = 2 *bytes*).
FLOOR	g INTS g FLOOR	(1) {8*}; {1, 2, 11} → {1} Returns the greatest integer ≤ x. Cf. CEIL.
fm.→m	g U→ x: ▲ fathom → m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
FP	f PARTS FP	(1) {1, 2, 8*, 10, 11} Returns the fractional part of x. Cf. IP.
FP?	g TEST f FP?	(0) Tests x for having a fractional part $\neq 0$.
fr→dB	g U→ ▲ field ratio→dB	(1) {2, 11}; {1} → {2} Converts ratios. See pp. 135ff.
FS?	g FLAGS FS? *n*	(0) Tests if the specified *flag* is set.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FS?C FS?F FS?S	g FLAGS f FS?C *n* etc.	(0) Tests if the specified *flag* is set. Clears, flips, or sets this *flag* after testing, respectively.
ft.→m	g U→ x: f ft.→m	(1) {2, 11}; {1} → {2} Convert distances and volumes, respectively. See pp. 135ff.
ft_{US}→m	g U→ x: ▲ survey $foot_{US}$ → m	
fz_{UK}→m^3	g U→ f V: f $floz_{UK}$ → m^3	
fz_{US}→m^3	g U→ f V: f $floz_{US}$ → m^3	
F_α F_δ	g CNST F_α etc.	(−1) { } → {2} *Feigenbaum's* α and δ.
F:	g PROB F:	*Submenu*. See p. 122.
f'	g EQN f'	*Submenus* for calculating the first or second derivative of a given equation. See the *OM* (*Sect. 4*) for more.
f"	g EQN f"	
f' (x)	f ADV f' (x) ***labl***	{1, 2, 11} → {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)	f ADV f f"(x) ***labl***	
F&p:	g U→ F&p:	*Submenu*. See p. 135.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
G	g CNST G	(−1) { } → {2} *Newtonian* constant of gravitation in $m^3/_{kg\ s^2}$; also called γ by other authors.
G_0	g CNST G_0	(−1) { } → {2} Conductance quantum in *siemens*.
GAP	g DISP ▲ f 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. After GAP **0**, no group separators will be displayed neither in reals nor integers at all. See *Sect. 2* of the *OM*.
GaussF	f STAT ▼ f GaussF	(0) Selects the Gauß peak fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 213ff for more.
G_C	g CNST G_C	(−1) { } → {2} *Catalan's* (mathematical) constant.
GCD	g INTS g GCD	(2) {1; 10} Returns the Greatest Common Divisor of x and y.[13] This will always be positive.
g_d g_d^{-1}	g X.FN f g_d etc.	(1) {2, 3, 11, 12} ; {1} → {2} Returns the Gudermannian function or its inverse. See p. 235 for details.

[13] See also LCM. Remember school?
Translator's notes for French readers: GCD correspond à *PGCD* en français,
Translator's notes for German readers: GCD entspricht *ggT* auf Deutsch.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
g_e	g CNST g_e	(−1) { } → {2} *Landé's* electron g-factor.
Geom $Geom_e$ $Geom_p$ $Geom^{-1}$	g PROB g Geom: Geom etc.	(1) {2, 11} *Geometric distribution*: The *CDF* returns the probability for a first success after ***m*** = ***x*** Bernoulli experiments. The probability p_0 for a success in each such experiment must be specified in **J**. See pp. 204ff for more. $Geom^{-1}$ returns the number of failures ***f*** before 1st success for given probabilities ***p*** in **X**, p_0 in **J**.
Geom:	g PROB f Geom:	*Submenu*. See p. 122.
$gl_{UK}\to m^3$ $gl_{US}\to m^3$	g U→ f V: $gl_{UK}\to m^3$ etc.	(1) {2, 11}; {1} → {2} Convert volumes. See pp. 135ff.
$GM_\oplus$	g CNST $GM_\oplus$	(−1) { } → {2} *Newtonian* constant of gravitation times the Earth's mass with its atmosphere included according to *WGS84*.[14] Displayed in m^3/s^2 .
GRAD	g MODE GRAD	(0) Sets the *ADM* to *grad/gon*.[15] Indicated in the status bar.
GRAD→	f L→ f GRAD→	(1) {2, 11}; {1} → {2} Converts angles as described on pp. 142f.
GTO	f GTO ***labl***	(0) In *PEM*, inserts an unconditional branch to ***labl***. Else positions the program pointer to ***labl***.

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

[15] This angular unit is also known as *gradian* in the English language.

<table>
<tr><th>Item</th><th>Keystrokes</th><th colspan="2">Remarks (see pp. 12ff for general information)</th></tr>
<tr><td rowspan="5">GTO.</td><td>f GTO . n</td><td rowspan="5">(0) Puts the program pointer …</td><td>to step n (specify up to four digits until becoming unambiguous in used program memory).</td></tr>
<tr><td>f GTO . labl</td><td>to the global label specified.</td></tr>
<tr><td>f GTO . ▲</td><td>directly after <u>previous</u> END, going to the top of current program (see Sect. 3 of the OM).</td></tr>
<tr><td>f GTO . ▼</td><td>directly after <u>next</u> END, going to the top of next program.</td></tr>
<tr><td>f GTO . .</td><td>to the end of used program memory in RAM (i.e. to the final END statement).</td></tr>
<tr><td>$g_\oplus$</td><td>g CNST $g_\oplus$</td><td colspan="2">(−1) { } → {2}
Standard earth acceleration in $^m/_{s^2}$.</td></tr>
<tr><td>h</td><td>g CNST h</td><td colspan="2">(−1) { } → {2}
Planck constant in joule-second.</td></tr>
<tr><td>H_n</td><td>g X.FN Orthog H_n</td><td colspan="2" rowspan="2">(2) {2, 11}; {1} → {2}
Hermite polynomials for probability (H_n) and physics (H_{np}). See p. 229 for details.</td></tr>
<tr><td>H_{nP}</td><td>… Orthog f H_{nP}</td></tr>
<tr><td>hp_E→W</td><td rowspan="3">g U→ P: hp_E→W
etc.</td><td colspan="2" rowspan="3">(1) {2, 11}; {1} → {2}
Convert powers. See pp. 135ff.</td></tr>
<tr><td>hp_M→W</td></tr>
<tr><td>hp_{UK}→W</td></tr>
</table>

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Hyper $Hyper_e$ $Hyper_p$ $Hyper^{-1}$	g PROB g Hyper: Hyper etc.	(1) {2, 11} *Hypergeometric distribution* with the number of successes ***g*** in **X**, the probability of a success p_0 in **I**, the sample size ***n*** in **J**, and the batch size n_0 in **K**. See pp. 204ff for the formula. $Hyper^{-1}$ returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, p_0 in **I**, ***n*** in **J**, and n_0 in **K**.
Hyper:	g PROB g Hyper:	*Submenu*. See p. 122.
HypF	f STAT ▼ f HypF	(0) Selects the hyperbolic fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 213ff for more.
ħ	g CNST ħ	(−1) { } → {2} $= h/2\pi$, so-called *Dirac* constant in *joule-second*.
IDIV	g INTS f IDIV	(2) {1, 10}; {2, 11} → {1} Integer division, working like / IP. See the *OM, Sect.* 2, for the *data type* of the quotient.
IDIVR	f CATALOG FCNS IDIVR	{1, 2, 10, 11} Like IDIV but also returns the remainder in **Y**. See the *OM*, *Section* 2, for the resulting *data types* of quotient and remainder.
iHg→Pa	g U→ F&p: f in.Hg →Pa	(1) {2, 11}; {1} → {2} Converts pressures. See pp. 135ff.
Im	g CPX f Im f PARTS g Im	(1) {3} → {2}; {9} → {8}; {12} → {11} Returns the imaginary part of ***x***. Cf. RE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
IMPFRC	g d/c	(1) {2, 11} Allows only *improper fractions* in display (e.g. $^5/_3$ instead of 1 $^2/_3$). Displays any reals (with $\lvert x\rvert < 10^6$) according to the settings by DEN… as *improper fractions*. Cf. PROFRC.
INC	f LOOP f INC ***r***	(0) {1, 2, 10, 11} Increments ***r*** by 1. Does not load **L** even for target address **X**.
INDEX	f MATX f 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 Owner's Manual*, pp. 223ff.
INFO	g INFO	*Menu*. See p. 121.
INPUT	g 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	g INTS	*Menu*. See p. 121.
INT?	g TEST f INT?	(0) Tests ***x*** for being an integer, i.e. having a fractional part equal to zero. Cf. FP?.
INVRT	f CATALOG FCNS INVRT	Works like $[M]^{-1}$ on p. 100. Maintained for backward compatibility only.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
in.→m	g U→ x: g in.→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
IP	f PARTS IP	(1) {1, 8*, 10}; {2, 11} → {1} Returns the integer part of ***x***. Cf. FP.
ISE	f LOOP ISE ***r***	(0) {1, 2, 10, 11} Given `cccccc.fffii` in the source, ISE increments ***r*** by `ii`, skipping next program step if `cccccc` ≥ `fff` then. If ***r*** has no fractional part then `fff` = 0 and `ii` = 0. If `ii` = 0, `cccccc` will be incremented by 1. ISE does not load **L** even for target address **X**. Note that neither `fff` nor `ii` can be negative, but `cccccc` can.
ISG	f LOOP ISG ***r***	(0) {1, 2, 10, 11} Works like ISE but skips if `cccccc` > `fff`.
ISZ	f LOOP ISZ ***r***	(0) {1, 2, 10, 11} Increments ***r*** by 1, skipping next program step if then $\|r\| < 1$. ISZ does not load **L** even for target address **X**. Cf. *HP-29C*, *HP-67*, and *HP-16C*.
I_{xyz}	g X.FN f I_{xyz}	(3) {1, 2, 11} Returns the *regularized Beta function*.
$I\Gamma_p$ $I\Gamma_q$	g X.FN f $I\Gamma_p$ etc.	(2) {1, 2, 11} Returns the *regularized Gamma function* (one of two kinds).
I+	f MATX ▲ I+	(1) Increments or decrements the row index of the indexed matrix. See INDEX and also J+, J−, RCLEL, STOEL, RCLIJ, and STOIJ.
I-	f MATX ▲ I-	
I/O	g I/O	*Menu*. See p. 121.

See p. 235 for more

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$J_y(x)$	g X.FN g $J_y(x)$	(2) {2, 11}; {1} → {2} $J_y(x)$ returns the *Bessel function of first kind* and order y. See p. 236 for details.
J+	f MATX ▲ J+	(1) Increments or decrements the column index of the indexed matrix. If M.GROW is set and the pointers I and J are at the last element of the matrix, executing J+ creates a new row at the end of the matrix. See INDEX and also I+, I−, RCLEL, STOEL, RCLIJ, and STOIJ.
J-	f MATX ▲ J-	
J/G	g CLK ▲ f J/G	(0) {2, 6} Sets the date the Gregorian calendar was introduced in the region you are interested in. See *Dates* in *Section 2* of the *OM.*
J→Btu	g U→ E: J→Btu etc.	(1) {2, 11}; {1} → {2} Convert energies. See pp. 135ff.
J→cal		
J→D	g CLK f J→D	(1) {1} → {6} Takes x as a *Julian day number* [16] and converts it to a common *date* according to J/G (see above) and the date format selected.
J→Wh	g U→ E: J→Wh	(1) {2, 11}; {1} → {2} Converts energies. See pp. 135ff.
k	g CNST k	(−1) { } → {2} *Boltzmann* constant in *joule per kelvin.*
KEY		See KEYG and KEYX below.

[16] Translator's note: *Julian day number* translates to "Julianisches Datum" in German and «jour Julien» in French. See the corresponding articles in *Wikipedia* for more information about these numbers.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
KEYG	g P.FN P.FN2 KEYG ***key#, labl***	Defines 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	g P.FN P.FN2 KEYX ***key#, labl***	
KEY?	g TEST g KEY? ***r***	(0) Tests if a key was pressed while a routine was running or paused. If no key was pressed in that interval, 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 (see the *OM, Sect. 3*; cf. GETKEY on *HP-42S*).
kg→ct	g U→ m: f kg→carat	(1) {2, 11}; {1} → {2} Convert masses. See pp. 135ff.
kg→cwt	g U→ m: kg→cwt etc.	
kg→lb.		
kg→oz		
kg→scw	g U→ m: f kg→sh.cwt	
kg→sto	g U→ m: f kg→stone	
kg→s.t	g U→ m: ▲ kg→short ton	
kg→ton	g U→ m: ▲ kg→ton	
kg→trz	g U→ m: f kg→tr.oz	
K_J	g CNST K_J	(−1) { } → {2} *Josephson* constant in *hertz per volt*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
KTYP?	g INFO KTYP? *r*	(−1) { } → {1} Assumes a key code in the address specified (see 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. May help in user interaction with routines (see the *OM, Section 3*)..
LASTx	RCL L	(−1) See *Sect. 1* of the *OM*. Actually, this command will be recorded as RCL L in routines.
lbf→N	g U→ F&p: lbf→N	(1) {2, 11}; {1} → {2} Converts forces. See pp. 135ff.
LBL	g LBL *labl*	(0) Identifies programs and routines for execution and branching. Read more about labels and specifying them in *Sect. 3* of the *OM*.
LBL?	g TEST g LBL? *labl*	(0) Tests for existence of the label specified, anywhere in program memory. See LBL for more.
lb.→kg	g U→ m: lb.→kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
LCM	g INTS f LCM	(2) {1; 10} Returns the *Least Common Multiple* of x and y.[17] This will always be positive.

[17] See also GCD. Remember school?

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LEAP?	g TEST f LEAP?	(0) {2, 6, 11} 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 LgNrm_e LgNrm_p LgNrm^{-1}	g PROB f LgNrm: LgNrm etc.	(1) {2, 11} *Log-normal distribution* with $\mu = \ln \bar{x}_g$ specified in $\mathbf{I}$ and $\sigma = \ln \varepsilon$ in $\mathbf{J}$. See $\bar{x}_g$ and ε below and pp. 204ff for more. LgNrm^{-1} returns $\boldsymbol{x}$ for a given probability $\boldsymbol{p}$ in $\mathbf{X}$, with $\boldsymbol{\mu}$ in $\mathbf{I}$ and $\boldsymbol{\sigma}$ in $\mathbf{J}$.
LgNrm:	g PROB f LgNrm:	*Submenu*. See p. 122.
LinF	f STAT ▼ LinF	(0) Selects the linear fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 204ff for more.
LJ	f BITS ▲ g 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 $\mathbf{Y}$ and the count of bit-shifts necessary to left justify the *word* in $\mathbf{X}$. **Example** for *word* size 8: $1\ 0110_2$ LJ returns $x = 3$ and $y = 1011\ 0000_2$.
L_m $L_{m\alpha}$	g X.FN Orthog L_m etc.	(2) {2, 11}; {1} → {2} *Laguerre polynomials* and *Laguerre's generalized polynomials*. See pp. 219f for more.
LN	ln	(1) {2, 3, 8*, 9*, 11, 12}; {1, 10} → {2} Returns the natural logarithm of x.

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)
LNβ	g X.FN g lnβ	(2) {2, 3, 11, 12}; {1} → {2} Returns the natural logarithm of *Euler's* Beta function (see p. 90).
LNΓ	g X.FN g lnΓ	(1) {2, 3, 11, 12}; {1} → {2} Returns the natural logarithm of $\Gamma(x)$ (see p. 90). Allows also for calculating really great factorials.
LN(1+x)	g EXP f ln 1+x	(1) {2, 8*, 11} For $x \approx 0$, this returns a more accurate result for the fractional part than $\ln(x)$ does.
LOAD	g I/O LOAD	Restores the entire backup from *FM* and returns `Backup restored`. Thus, LOAD = LOADP + LOADR + LOADSS + LOADΣ.[18]
LOADP	g I/O LOADP	(0) Loads the complete program memory from backup and appends it to the programs already in *RAM*. This will only work if there is enough space – else an error will be thrown.[18]
LOADR	g I/O LOADR	(0) Recovers the numbered general purpose *registers* from backup. Lettered *registers* will not be recalled.[18]
LOADSS	g I/O LOADSS	(0) Recovers the system state from backup.[18]
LOADΣ	g I/O LOADΣ	(0) Recovers the statistical summation *registers* from backup. Throws an error if there are none.[18]
LocR	g P.FN g LocR *n*	(0) Allocates n *local registers* (≤ 100) and 16 *local flags* for the *current routine*. See the *OM, Sect. 3*.

[18] See SAVE on p. 73 and *App. A* on pp. 143f for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LocR?	g INFO f LocR?	(−1) { } → {1} Returns the number of *local registers* currently allocated.
LOG_{10}	g lg	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} ({1} → {2}) Returns the logarithm of x for base 10.
LOG_2	g EXP lb x	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} ({1} → {2}) Returns the logarithm of x for base 2.
LOG_xy	g EXP log_xy	(2) {1, 2, 3, 8*, 9*, 10, 11, 12}; ({1} → {2}) Returns the logarithm of y for the base x .
LogF	f STAT ▼ LogF	(0) Selects the logarithmic curve fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 204ff for more.
Logis	g PROB f Logis: Logis etc.	(1) {2, 11} *Logistic distribution* with μ given in **I** and ***s*** in **J**. See pp. 204ff for details.
$Logis_e$		
$Logis_p$		
$Logis^{-1}$		
Logis:	g PROB f Logis:	*Submenu*. See p. 122.
LOOP	f LOOP	*Menu*. See p. 121.
l_{PL}	g CNST l_{PL}	(−1) { } → {2} *Planck* length in *meter*.
ly→m	g U→ x: ly→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LZOFF	f BITS ▼ f LZOFF	(0) Turns leading zeros in *short integers* off and on like *flag* 3 does in *HP-16C*. Works in bases 2, 4, 8, and 16 only.
	g INTS ▲ f ...	
LZON	f BITS ▼ LZON	
	g INTS ▲ ...	
L.INTS	f CATALOG VARS L.INTS	*Submenu* of *long integer* variables defined at execution time. See pp. 113ff.
L.R.	f STAT ▲ L.R.	(−2) or (-3) { } → {2} Pushes the parameters a_2 (in **Z**), a_1 (in **Y**), and a_0 (in **X**) of the fit curve through the data points accumulated in the statistical summation *registers* on the *stack*, according to the curve fit model selected (see LINF, ORTHOF, EXPF, POWERF, LOGF, HYPF, ROOTF, PARABF, CAUCHF, GAUSSF). For a straight line, a_0 is its y-intercept and a_1 is its slope. See pp. 204ff for more.
m^2→ac	g U→ f A: m^2 → acre	(1) {2, 11}; {1} → {2} Convert areas. See pp. 135ff.
m^2→ac$_{US}$	g U→ f A: m^2 → acre$_{US}$	
m^3→bbl	g U→ f V: f m^3 → barrel	(1) {2, 11}; {1} → {2} Convert volumes. See pp. 135ff.
m^3→fz$_{UK}$	g U→ f V: f m^3 → floz$_{UK}$ etc.	
m^3→fz$_{US}$		
m^3→gl$_{UK}$	g U→ f V: m^3→gl$_{UK}$ etc.	
m^3→gl$_{US}$		
MANT	f PARTS MANT	(1) {2, 11}; {1} → {2} Returns the mantissa m of the number $x = m \cdot 10^h$ displayed. Cf. EXPT.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MASKL	f BITS MASKL *n*	(−1) { } → {10} Work like MASKL and MASKR on *HP-16C*, but with the mask length (or its address) following the command instead of taken from **X**. Thus, the mask is pushed on the *stack*. **Example:** For WSIZE **8**, MASKL **3** returns a mask *word* $1110\ 0000_2$. Use it e.g. for extracting the three most significant bits of an arbitrary *byte* via AND.
MASKR	f BITS MASKR *n*	
MATRS	f CATALOG VARS MATRS	*Submenu* of matrix variables defined at execution time. See pp. 113f.
MATR?	g TEST ▲ MATR?	(0) Checks if *x* is a real or complex matrix.
MATX	f MATX	*Menu*. See p. 121.
Mat_X	f MATX SIM EQ Mat X	(−1) Returns the solution vector of a system of linear equations (see *Section 2* of the *OM*).
max	g X.FN g max	(2) {1, 2, 4, 5, 6, 10, 11} Returns the maximum of *x* and *y*.
m_e	g CNST m_e	(−1) { } → {2} Electron mass in *kilogram*.
MEM?	g INFO MEM?	(−1) { } → {1} Returns the number of free *words* in program memory (1 *word* = 2 *bytes*), also taking into account the local *registers* allocated.
MENU	g P.FN P.FN2 MENU	Displays the programmable *menu*. See the *HP-42S OM, Part 2, Section 10*, p. 146.
MENUS	f CAT. MENUS	*Submenu* of all *menus* defined at execution time. See pp. 113ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
min	g X.FN g min	(2) {1, 2, 4, 5, 6, 10, 11} Returns the minimum of x and y.
MIRROR	f BITS f MIRROR	(1) {10} Reflects the bit pattern in x (e.g. $0001\ 0111_2$ would become $1110\ 1000_2$ for *word* size 8).
mi.→m	g U→ x: f mi.→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
M_{Moon} m_n m_n/m_p	g CNST M_{Moon} etc.	(−1) { } → {2} Mass of the Moon in *kilogram*; neutron mass in *kilogram*; neutron to proton mass ratio.
MOD	g INTS f MOD	(2) {1, 2, 10, 11} Returns $y \bmod x$ (modulo, see *Section 2* of the *OM* for examples). Cf. RMD.
MODE	g MODE	*Menu*. See p. 121.
MONTH	g CLK f MONTH	(1) {2, 6, 11} → {1} Assumes x containing a *date* in the format selected (or a real number in corresponding format) and extracts the month.
m_p m_{PL} m_p/m_e	g CNST m_p etc.	(−1) { } → {2} Proton mass and *Planck* mass in *kilogram*; proton to electron mass ratio.
MSG	g P.FN P.FN2 f MSG	(0) {1, 2, 11} Throws the (*temporary*) *error message* specified by the integer part of x. Cf. ERR. See *App. C* on pp. 169ff for the respective error codes.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
m_u	g CNST m_u	(−1) { } → {2} Atomic mass constant in *kilogram* and its energy equivalent in *joule*.
m_uc^2	g CNST m_uc^2	
MULπ	g MODE MULπ	(0) Sets the *ADM* to *multiples of π*. Indicated in the status bar.
MULπ→	f L→ f MULπ→	(1) {1, 2, 11} → {4} Converts angles as described on pp. 142f.
MULT×	g DISP ▲ MULT×	(0) Select the symbol for multiplication display.
MULT·	g DISP ▲ MULT·	
MVAR	g P.FN P.FN2 f MVAR ***name***	(0) Defines a *menu variable*. Such variables are required for VARMNU. Works in *PEM* only.
MyMenu	f CAT. MENUS MyMenu	User *menu*. See the *OM, Sect. 6.*
Myα	f CAT. CHARS Myα	User *menu* in *AIM*. See the *OM, Sect. 6.*
m_μ	g CNST m_μ	(−1) { } → {2} Muon mass in *kilogram*.
M.DELR	f DELR with M.EDIT displayed	(0) {8, 9} Deletes the current row of elements (where the cursor is in). Will not work if the matrix has only one row.
M.DIM	f MATX f DIM ***name***	(0) {1, 2} Creates a new named matrix or re-dimensions an existing matrix to IP(y) rows and IP(x) columns. See DIM in the *HP-42S Owner's Manual*, p. 217.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
M.DIM?	g INFO g DIM? f MATX ▲ f DIM?	{8, 9} → {1} Returns the dimensions of the matrix x (rows to **Y**, columns to **X**). Note the matrix is saved in **L**. Previous y goes into **Z**, previous z into **T**, etc.
M.DY	g CLK ▲ M.DY	(0) Selects the format `mm/dd/yyyy` for *dates*.
M.EDI	f MATX EDIT	(2) {8, 9} Opens x using the *Matrix Editor* (like MATRIX EDIT in *HP-42S*).[19] See *Section 2* of the *OM*.
M.EDIN	f MATX f EDITN ***name***	(2) Opens a named matrix using the *Matrix Editor* (like MATRIX EDITN in *HP-42S*). See *Section 2* of the *OM*.
M.EDIT		*Submenu* for matrix editing, called by M.EDI or M.EDIN. See p. 121.
M.GET	f MATX g GETM	(0) {1, 2} → {8, 9} Gets a sub-matrix with IP(y) rows and IP(x) columns out of the indexed matrix into **X** (like GETM in *HP-42S*). Cf. M.PUT.
M.GOTO	f GOTO (with M.EDIT displayed)	(0) Asks for target row and column and moves to this matrix element.
M.GROW	f GROW (with M.EDIT displayed)	(0) Allows the indexed matrix to grow automatically (see J+ above and *Section 2* of the *OM*; see also GROW in the *HP-42S Owner's Manual*, p. 213.). Cf. M.WRAP.
M.INSR	f INSR (with M.EDIT displayed)	(0) Inserts a new row of elements containing zero, left of the current cursor position in the matrix.

[19] In the real *HP-42S*, EDIT and EDITN don't actually disable *automatic stack lift*; 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, but it needs to be emulated anyway, since not doing so risks breaking *HP-42S* programs.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
M.LU	f MATX ▲ f M.LU	(1) *WP 34S*: Takes a *descriptor* of a square matrix in **X**. Transforms (*X*) into its LU decomposition in-situ. The value in **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.
M.NEW	f MATX NEW	(2) {1, 2} → {8} Creates a new matrix (like NEW in *HP-42S*). Its number of rows shall be supplied in **Y** and its number of columns in **X**. M.NEW returns a matrix ***x*** with all its elements set to zero.
M.OLD	OLD with M.EDIT displayed	(0) Recalls the old element content (like OLD in *HP-42S*). See *Section 2* of the *OM*.
M.PUT	f MATX g PUTM	(0) {8, 9} Puts the matrix ***x*** as is into the indexed matrix (like PUTM in *HP-42S*). Cf. M.GET.
M.R⇄R	f MATX ▲ f R⇄R	(0) {8, 9} Swaps row ***x*** and row ***y*** of the indexed matrix (like R<>R in *HP-42S*).
M.SIMQ		*Submenu* of MATX, called by SIM_EQ.
M.SQR?	g TEST ▲ f M.SQR?	(0) Returns `true` if ***x*** is a square matrix.
M.WRAP	f WRAP with M.EDIT displayed	(0) Controls the index pointers (see *Section 2* of the *OM*). Cf. M.GROW.
m:	g U→ m:	*Submenu*. See p. 135.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
m→au	g U→ x: m→au	(1) {2, 11}; {1} → {2} Convert distances or heights. See pp. 135ff.
m→fm.	g U→ x: ▲ m → fathom	
m→ft.	g U→ x: f m→ft.	
m→ft_{US}	g U→ x: ▲ m → survey $foot_{US}$	
m→in.	g U→ x: g m→in.	
m→ly	g U→ x: m→ly	
m→mi.	g U→ x: f m→mi.	
m→nmi.	g U→ x: f m→nmi.	
m→pc	g U→ x: m→pc	
m→pt.	g U→ x: g m → point	
m→yd.	g U→ x: g m→yd.	
$m_⊕$ $m_⊙$	g CNST $m_⊙$ etc.	(−1) { } → {2} Masses of the Earth and Sun in *kilogram*.
N_A	g CNST N_A	(−1) { } → {2} *Avogadro's* number in *particles per mol*.
NAND	f BITS f NAND	(2) Works in analogy to AND. See p. 18.
NaN	g CNST NaN	(−1) *Not a Number.*
NaN?	g TEST ▲ f NaN?	(0) Returns true if x is *Not a Number*.
NBin $NBin_e$ $NBin_p$ $NBin^{-1}$	g PROB g NBin: NBin etc.	(1) {2, 11} *Negative binomial distribution* with the total number of failures f in **X**, the probability of a success p_0 in **I**, and the number of draws n in **J**. See pp. 204ff for more information.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
NBin:	g PROB g NBin:	*Submenu*. See p. 122.
NEIGHB	g INFO f NEIGHB	(2) {1} Returns … • $x+1$ for $x<y$; • x for $x=y$; • $x-1$ for $x>y$. (2) {2, 3, 11, 12} 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	g X.FN g NEXTP	(1) {1, 2, 11} → {1} Returns the next prime number greater than x.
nmi.→m	g U→ x: f nmi.→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
NOP	g P.FN P.FN2 f NOP	(0) 'Empty' program step (for historical reasons only).
NOR	f BITS f NOR	(2) Works in analogy to AND. See p. 18.
Norml Norml$_e$ Norml$_p$ Norml^{-1}	g PROB Norml: Norml etc.	(1) {2, 11} *Normal distribution* with an arbitrary mean μ given in **I** and a *standard deviation* σ in **J**. See *Sect. 2* of the *OM* for an application example and pp. 204ff for more. Norml^{-1} returns x for a given probability p in **X**, with μ in **I** and σ in **J**.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Norml:	g PROB Norml:	*Submenu*. See p. 122.
NOT	f BITS NOT	(1) {10} Inverts x bit-wise as on *HP-16C*. (1) {1, 2, 11} → {1} Returns 1 for $x = 0$, and 0 for $x \neq 0$.
nΣ	g SUMS n	(−1) { } → {1} Recalls the number of accumulated data points.
N→lbf	g U→ F&p: N→lbf	(1) {2, 11}; {1} → {2} Converts forces. See pp. 135ff.
ODD?	g TEST f ODD?	(0) Checks if x is integer and odd.
OFF	g OFF	(0) In *PEM*, inserts a step to turn your *WP 43S* off under program control. Else turns your *WP 43S* off.
OR	f BITS OR	(2) Works in analogy to AND. See p. 18.
OrthoF	f STAT ▼ f OrthoF	(0) Selects the linear orthogonal fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 213ff for more.
ORTHOG	g X.FN Orthog	*Submenu*. See p. 125.
oz→kg	g U→ m: oz→kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
P_0	g CNST P_0	(−1) { } → {2} Standard atmospheric pressure in *pascal*.
ParabF	f STAT ▼ f ParabF	(0) Selects the parabolic fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 213ff for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PARTS	f PARTS	*Menu*. See p. 122.
PAUSE	g P.FN PAUSE ***n***	(0) With 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	g U→ F&p: f Pa→atm	(1) {2, 11}; {1} → {2} Convert pressures. See pp. 135ff.
Pa→bar	g U→ F&p: Pa→bar	
Pa→iHg	g U→ F&p: f Pa → in.Hg	
Pa→psi	g U→ F&p: Pa→psi	
Pa→tor	g U→ F&p: f Pa → torr	
pc→m	g U→ x: pc→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
PERM	g 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, 11, 12} See pp. 202ff for the formula.
PGMINT	f ADV f PGMINT ***labl***	Specifies the address of the expression to be integrated or solved, respectively. See *Section 4* of the *OM*.
PGMSLV	f ADV f PGMSLV ***labl***	
PIXEL	g P.FN P.FN2 g PIXEL	(0) Turns on a single pixel (dot) on the screen. The location of the pixel is given by the **X**- and **Y**-*registers*. See AGRAPH on p. 17 for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PLOT	f STAT g PLOT	(0) Plots the ***n*** data points given by the ***n***×2 matrix ***x***. See p. xxviii for more.
P_n	g X.FN Orthog P_n	(1) {2, 11}; {1} → {2} *Legendre polynomials*. See pp. 219f for more.
POINT	g P.FN P.FN2 g POINT	(1) {1, 2, 11} Turns on a square point (3×3 px ■) on the screen. The location of its center is given by the integer parts of the numbers in **X** and **Y**. See AGRAPH on p. 17 for more.
Poiss Poiss_e Poiss_p	g PROB g Poiss: Poiss etc.	(1) {2, 11}; {1} → {2} *Poisson distribution* with the number of successes ***g*** in **X** and the *Poisson* parameter ***λ*** in **I**. See pp. 204ff for details.
Poiss^{-1}	g PROB g Poiss: Poiss^{-1}	(1) {2, 11} Returns the maximum number of successes ***m*** for a given probability ***p*** in **X** and ***λ*** in **I**.
Poiss:	g PROB g Poiss:	*Submenu*. See p. 122.
POLAR	g MODE POLAR g CPX g POLAR	(0) Sets polar format for displaying complex numbers. Indicated in the status bar.
PopLR	g P.FN g PopLR	(0) Pops the local *registers* allocated to the *current routine* (see *Section 3* of the *OM*) without returning to the calling routine. See LOCR and RTN.
PowerF	f STAT ▼ PowerF	(0) Selects the power curve fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ (see pp. 204ff for more).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PRCL	g P.FN f 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?	g TEST f PRIME?	(0) {1, 2, 11} Checks if the absolute value of IP(x) is a prime. The method is believed to work for integers up to 9×10^{18}.
PROB	g PROB	*Menu*. See p. 122.
PROFRC	f a b/c	(1) {2, 11} Allows only *proper fractions* in display. Displays any reals (with $\|x\| < 10^6$) according to the settings by DEN… as *proper fractions*, e.g. 1.25 or $^5/_4$ as 1 $^1/_4$. Cf. IMPFRC.
PROGS	f CAT. PROGS	*Submenu* of global labels defined at execution time. See pp. 113f.
pr→dB	g U→ ▲ power ratio → dB	(1) {2, 11}; {1} → {2} Converts ratios. See pp. 135ff.
psi→Pa	g U→ F&p: psi→Pa	(1) {2, 11}; {1} → {2} Converts pressures. See pp. 135ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PSTO	g P.FN f 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.→m	g U→ x: g point → m	(1) {2, 11}; {1} → {2} Converts print heights. See pp. 135ff.
PUTK	g P.FN f 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	g P.FN	*Menu*. See p. 123.
P.FN2	g P.FN P.FN2	*Submenu*. See p. 123.
P:	g U→ P:	*Submenu*. See p. 135.
QUIET	g I/O f QUIET g MODE f …	(0) Toggles the *flag* to disable or enable the beeper. Indicated in STATUS.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R	g CNST R	(−1) { } → {2} Molar gas constant in *joule per mol and kelvin*.
RAD	g MODE RAD	(0) Sets the *ADM* to *radians*. Indicated in the status bar.
RAD→	f L→ f RAD→	(1) {1, 2, 11} → {4} Converts angles as described on pp. 142f.
RAM	f CATALOG PROGS RAM	*Submenu* of global labels defined at execution time. See pp. 113f.
RAN#	g PROB ▲ RAN#	(−1) { } → {2} Returns a random number between 0 and 1 like RAN does in *HP-42S*. See also SEED.
RBR	f RBR	Calls the *register browser*. See the *OM*, *Sect. 5*. You may call RBR also in *PEM* but it is not programmable. **ATTENTION:** Within RBR, real and complex numbers are generally displayed in the format chosen within the screen space available. Since RBR uses the small font, however, more digits are displayable here than in a numeric row using large font. For real *DP* numbers and ALL 00 set, for example, RBR display will turn to SCI or ENG at 33 instead of 16 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.
RCLCFG	RCL f Config *r*	(0) Recalls a *configuration* stored by STOCFG (see *Sections 2* and *6* of the *OM*).
RCLEL	f MATX g RCLEL RCL g ...EL	(−1) { } → {2, 3} Recalls a copy of the current element a_{ij} of the indexed matrix. Cf. STOEL.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RCLIJ	f MATX ▲ RCLIJ RCL g ...IJ	(−2) { } → {1} Recalls the current values of the matrix index pointers into **X** (= column number) and **Y** (= row number). If the pointers both equal zero, then there is currently no indexed matrix. Cf. STOIJ.
RCLS	RCL f 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.[20] **Example:** RCL−12 replaces x with $x - r12$.
RCL−	RCL − *r*	
RCL×	RCL × *r*	
RCL/	RCL / *r*	
RCL↑	RCL f Max *r* RCL ▲ *r*	(1) {1, 2, 4, 5, 6, 10, 11} Replaces x with the maximum of r and x.[20]
RCL↓	RCL f Min *r* RCL ▼ *r*	(1) {1, 2, 4, 5, 6, 10, 11} Replaces x with the minimum of r and x.[20]
RDP	g DISP f RDP *n*	(1) {2, 3, 5, 8*, 9*, 11, 12} Rounds x to n decimal places ($0 \le n \le 99$, think of FIX format), taking the RM setting into account. See RM and compare RSD.
RDX,	g DISP ▲ RDX, etc.	(0) Select a comma or a point as decimal radix mark.
RDX.		
Re	g CPX Re f PARTS g Re	(1) {3} → {2}; {9} → {8}; {12} → {11} Returns the real part of x. Cf. IM.

[20] Only legal operations according to the matrices in *Section 2* of the *OM* will work. See also the examples given there.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
r_e	g CNST r_e	(−1) { } → {2} Classical electron radius in *meter*.
REALRE	g MODE f REALRE g FLAGS CF I	(0) Allows only real results, no complex ones. Indicated in the status bar. The letter S cannot be shown in *menu views* for space reasons there. Cf. CPXRES.
REALS	f CAT. VARS REALS	*Submenu* of real variables defined at execution time. See pp. 113f.
REAL?	g TEST ▲ REAL?	(0) Checks if x is a real number or matrix.
RECT	g MODE RECT g CPX g RECT	(0) Sets rectangular (Cartesian) format for displaying complex numbers. Indicated in the status bar.
RECV	g I/O f RECV	(0) Prepares your *WP 43S* for receiving data via serial I/O. See SEND and *Sect. 3* in the *OM* for more.
RESET	g CLR g RESET	Executes CLALL and resets all modes to *startup default*, i.e. 24h, 2COMPL, ALL **0**, CPXi, DEG, DENANY, DENMAX **0**, DSTACK **4**, GAP **3**, J/G **1752-01-01**, LinF, LocR **0**, LZOFF, MULT×, PROFRC, RDX., REALRES, RECT, RM **0**, SCIOVR, SSIZE4, TDISP **−1**, WSIZE **64**, and Y.MD. See these individual commands for more.
RE→CX	CC (works in run mode only) g CPX f RE→CX	(2) {2} → {3}; {11} → {12} 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 the imaginary part from **X**, or • (for ⊙) the magnitude from **Y** and the phase from **X**. (2) {8} → {9} Works in analogy for two real matrices x and y.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Re⇄Im	g CPX Re⇄Im	(1) {3, 9*, 12} Swaps real and imaginary parts of complex objects.
RJ	f BITS ▲ f 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 right justify the *word* in **X**. **Example:** 10 1100_2 RJ results in $y = 1011_2$ and $x = 2$.
R_K	g CNST R_K	(−1) { } → {2} *Von Klitzing constant in ohm.*
RL RLC	f BITS ▲ RL *n* etc.	(1) {10} Work like ***n*** consecutive RLs / RLCs on *HP-16C*, similar to RLn / RLCn there. For RL, $0 \leq n \leq 63$. For RLC, $0 \leq n \leq 64$. RL **0** / RLC **0** execute as NOP, but load **L**. See the *OM, Sect. 2,* for more.
RM	g MODE f 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 (39 digits) to packed reals. It will not alter the display nor change the behavior of ROUND. The following seven modes are supported: 0: round half even: ½E 0.5 rounds to next even number (default).[21] 1: round half up: ½↑ 0.5 rounds up ('business-man's rounding' [22]). 2: round half down: ½↓ 0.5 rounds down. 3: round up: ←0→ rounds away from 0.

---

[21] This is the way of rounding used in science.

[22] 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)
		4: round down: →0← rounds towards 0. 5: ceiling: →∞ or ⌈x⌉ rounds towards + ∞. 6: floor: →-∞ or ⌊x⌋ rounds towards – ∞. The abbreviations printed on grey are used in STATUS for indicating the respective rounding modes (see *Section 5* of the *OM*).
RMD	g INTS f RMD	(2) {1, 2, 10, 11} Returns the remainder of a division. Equals RMD on *HP-16C* but works for reals as well. See *Sect. 2* of the *OM* for examples. Cf. MOD.
R_{Moon}	g CNST R_{Moon}	(−1) { } → {2} Mean radius of the Moon in *meter*.
RM?	g INFO RM?	(−1) { } → {1} Returns the floating point rounding mode set. See RM for more.
RNORM	f MATX ▲ f RNORM	(1) {8, 9} Calculates the row norm of the matrix x, i.e. the maximum value (over all rows) of the sums of the absolute values of all elements in a row (like RNRM on *HP-42S*). For a vector, the row norm is the largest absolute value of any of its elements.
RootF	f STAT ▼ f RootF	(0) Selects the root fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 213ff for more.
ROUND	g DISP ROUND f PARTS f …	(1) {2, 3, 4, 5, 6, 8*, 9*, 11, 12} Rounds x using the current display format like RND on *HP-42S*.
ROUNDI	g DISP ROUNDI f PARTS f …	(1) {8*}; {2, 11} → {1}; Rounds x to next integer. ½ rounds to 1.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RR RRC	f BITS ▲ RR *n* etc.	(1) {10} Work like ***n*** consecutive RRs / RRCs on *HP-16C*, similar to RRn / RRCn there. For RR, $0 \le n \le 63$. For RRC, $0 \le n \le 64$. RR **0** / RRC **0** execute as NOP, but load **L**. See the *OM, Sect. 2,* for more.
RSD	g DISP f RSD *n*	(1) {2, 3, 4, 8*, 9*, 11, 12} Rounds ***x*** to ***n*** significant digits ($1 \le n \le 34$), taking the RM setting into account. See RM, cf. RDP.
RSUM	f MATX ▲ f 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	g RTN	(0) In *PEM*, RTN is the logically last command in a routine (see *Section 3* of the *OM*). 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. this 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*, *Sec. 3*). If the program is in *FM*, the pointer is set to step 0000 in *RAM*, and ⍑ is lit.
RTN+1	g P.FN P.FN2 RTN+1	(0) Works like RTN, but moves the program pointer <u>two</u> steps behind the XEQ instruction that called said routine.
R-CLR	g P.FN g R-CLR	(0) {2, 11} 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**.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		ATTENTION: For ***nn* = 0** , clearing will cover the maximum available: • For ***sss*** $\in [0;99]$, it will stop at **R99**. • For ***sss*** $\in [100;111]$, it will stop at **K**. • For ***sss*** ≥ 112, it will stop at the highest currently allocated local *register*.
R-COPY	g P.FN g R-COPY	(0) {2, 11} 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	g P.FN g R-SORT	(0) {2, 11} 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	g P.FN g R-SWAP	(0) {2, 11} Works like R-COPY but swaps the contents of source and destination *registers*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R→D	f L→ g R→D	(1) {1, 2, 11} → {4} Converts angles as described on pp. 142f.
R↑	f R↑	Rotates the *stack* contents one level up or down, respectively. See *Section 1* of the *OM* for details.
R↓	R↓	
R∞	g CNST R∞ etc.	(−1) { } → {2} *Rydberg* constant (see p. 132); mean radii of the Earth and Sun in *meter*.
R⊕		
R⊙		
s	f 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. 204ff for the formula.
Sa	g CNST Sa	(−1) { } → {2} Semi-major axis in *meter* of the Earth model *WGS84*.[23]
SAVE	f SAVE	(0) Saves user program space, *registers* and system state to *FM*, and returns Saved. Recall your backup using the different flavors of LOAD.
SB	f BITS g SB *n*	(1) {10} Sets the specified bit in x .
Sb	g CNST Sb	(−1) { } → {2} Semi-minor axis in *meter* of *WGS84*.[23]
SCI	g DISP SCI *n*	(0) Sets scientific display format (see *Section 2* of the *OM*).

[23] This model is used to define the Earth's surface for surveying and *GPS*. See *http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html*

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SCIOVR	g DISP ▲ SCIOVR	(0) Defines that numbers exceeding the range displayable in ALL or FIX will be shown in scientific format. Cf. ENGOVR, see RESET.
scw→kg	g U→ m: f short cwt → kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
SDIGS?	g INFO f SDIGS?	(−1) { } → {1} Returns the number of significant digits set by SETSIG. Also returned by STATUS.
SDL SDR	g P.FN P.FN2 f SDL n etc.	(1) {2, 11} Shifts digits left (right) by n decimal positions, equivalent to multiplying (dividing) x by 10^n . Cf. SL and SR for binary integers.
Se^2	g CNST Se^2	(−1) { } → {2} First eccentricity squared of the Earth model *WGS84* (see footnote 23 on p. 71).
SEED	g PROB ▲ SEED	(0) {2, 11} Stores a seed for random number generation. If $x = 0$, the seed is taken from the real-time clock.
SEND	g I/O f SEND	(0) Sends all *RAM* data to the device connected via serial I/O. See RECV and *Section 3* in the *OM* for more.
SETCHN	g DISP ▲ g CHINA	(0) Sets regional format preferences (see *Section 2* of the *OM*).
SETDAT	g CLK ▲ SETDAT	(0) Sets the date for the real-time clock (the emulator takes this information from the PC clock).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SETEUR	g DISP ▲ g EUROPE	(0) Set regional format preferences (see *Section 2* of the *OM*).
SETIND	g DISP ▲ g INDIA	
SETJPN	g DISP ▲ g JAPAN	
SETSIG	g MODE ▲ SETSIG	(0) {1} Sets the number of significant digits (0 ... 34) for rounding after each operation. SETSIG 0 sets maximum precision.
SETTIM	g CLK ▲ SETTIM	(0) Sets the time for the real-time clock (the emulator takes this information from the PC clock).
SETUK	g DISP ▲ g UK etc.	(0) Set regional format preferences (see *Section 2* of the *OM*).
SETUSA		
Se'²	g CNST Se'²	(−1) { } → {2} Second eccentricity squared of the Earth model *WGS84* (see footnote 23 on p. 71).
SF	g FLAGS SF *n*	(0) Sets the *flag* specified.
Sf^{-1}	g CNST Sf^{-1}	(−1) { } → {2} Flattening parameter of the Earth model *WGS84* (see footnote 23 on p. p. 71).
SHOW	f SHOW	(0) {1, 2, 3, 4, 11, 12} Shows all digits stored in **X** until next keystroke. Wherever one display row is not sufficient, small font will be employed. For *DP* reals, 34 digits will be shown using two display rows. *SP* and *DP* complex numbers will be displayed with all their digits using as many display rows as necessary. Up to 294 digits of *long integers* will be shown using up to 7 display rows; for *long integers* exceeding 294 digits, the most significant 288 will be shown with a trailing power of 10.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SIGN	f PARTS sign	(1) {8}; {1, 2, 10, 11} → {1} Returns 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to the mathematical function $\mathrm{signum}(x)$.
	g CPX f sign	(1) {3, 12} Returns the unit vector of the complex number x (cf. UNITV). Maintained for backward compatibility only.
SIGNMT	f BITS ▼ SIGNMT g INTS ▲ …	(0) Sets sign-and-mantissa mode for operations on *short integers*. See the *OM, Section 2.* Indicated in the status bar.
SIM_EQ	f 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 *Sect. 2* of the *OM* for more.
sin	TRI sin	(1) {2, 3, 8*, 9*, 11, 12}; {1, 4} → {2} Returns the sine of the angle in **X**.
sinc	g X.FN ▲ sinc	(1) {2, 3, 8*, 9*, 11, 12} Returns $\sin(x)/_x$ for $x \neq 0$ and **1** for $x = 0$. Note input has to be supplied in *radians*.
sinh	g EXP g sinh TRI g sinh	(1) {2, 3, 8*, 9*, 11, 12} Returns the hyperbolic sine of x.
SKIP	g P.FN P.FN2 g SKIP *n*	(0) Skips ***n*** program steps forwards ($0 \leq n \leq 255$). So e.g. SKIP **2** skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. **ATTENTION:** If you edit a section of your routine crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to **manually maintain all those statements individually.**

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SL	f BITS ▲ SL *n*	(1) {10} Works like ***n*** (≤ 63) consecutive SLs on *HP-16C*. SL **0** executes as NOP, but loads **L**. See *Sect. 2* of the *OM* for more.
SLOW	g MODE f SLOW	(0) Sets the processor speed to 'slow', about ½ of 'fast'. This is also automatically set for low battery voltage (see the *OM, Sect. 2*). Cf. FAST.
SLVQ	f ADV SLVQ	{1, 2, 3} → {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. • 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, SLVQ returns ***r*** in **Z**. Higher *stack registers* are kept unchanged. **L** will contain equation parameter ***c***.
s_m	f STAT s_m	(−2) { } → {2} Takes the statistical data accumulated and pushes the *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, Sect. 2*).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SMODE?	g INFO SMODE?	(−1) { } → {1} Returns the *integer sign mode* set for *short integers*, i.e. true 2 for 2's complement, true 1 for 1's complement, false 0 for unsigned, or true −1 for sign & mantissa mode.
s_{mw}	f STAT f s_{mw}	(−1) { } → {2} Returns the *standard error* for weighted data, i.e. the *standard deviation* of the mean $\bar{x}_w$.
SOLVE	f ADV SOLVE ***var***	{2, 3} Solves the equation $f(var) = 0$, with f calculated by the equation specified (in *PEM* by PGMSLV). Two initial estimates of the root must be supplied in **X** and **Y** when calling SOLVE. It returns ***var****root* in **X**, the second last ***var***-value tested in **Y**, then $f(var_{root})$ in **Z**, and 0 in **T**. Additionally, SOLVE acts as 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	g EQN Solver	*Submenu* for solving a given equation. See the *OM, Sect. 4,* for more.
SPEC?	g TEST ▲ f SPEC?	(0) True if x is 'special' (±∞ or NaN).
SR	f BITS ▲ SR ***n***	(1) {10} Works like ***n*** (≤ 63) consecutive SRs on *HP-16C*. SR **0** executes as NOP, but loads **L**. See *Section 2* of the *OM* for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SSIZE4 SSIZE8	g STK f SSIZE4 etc. or g MODE g … etc.	Set the *stack* size to 4 or 8 *registers*, respectively (see *Section 1* of the *OM*). Indicated in STATUS. Note *register* contents will remain unchanged in this operation (as well as if *stack* size is modified by any other operation – e.g. by RCLCFG).
SSIZE?	g INFO SSIZE?	(−1) { } → {1} Returns the number of *stack registers* currently allocated, 4 or 8.
STAT	f STAT	*Menu*. See p. 123.
STATUS	g FLAGS STATUS	*Flag browser*. See *Section 5* of the *OM*.
STK	g STK	*Menu*. See p. 123.
STO	STO *r*	(0) Stores x into destination.
STOCFG	STO f Config *r*	(0) Stores the current *configuration* for later use as described in *Section 2* of the *OM*. RCLCFG recalls such data.
STOEL	f MATX g STOEL STO g ...EL	(1) {1, 2, 3} Stores a copy of x into the indexed matrix at the current element, a_{ij}. Cf. RCLEL.
STOIJ	f MATX ▲ STOIJ STO g ...IJ	(1) {1} Sets the index pointers to IP(x) (= column number) and IP(y) (= row number). Cf. RCLIJ.
STOP	R/S	(0) Stops program execution. May be inserted in programs to wait for input, for example.
STOS	STO f Stack *r*	(0) Stores the entire *stack* in a set of 4 or 8 *registers*, starting at the destination address specified. See RCLS.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
STO+	STO + *r*	(0) Executes the specified operation on r and stores the result (e.g. $r-x$) at the address specified.[24]
STO–	STO – *r*	
STO×	STO × *r*	
STO/	STO / *r*	
sto→kg	g U→ m: f stone → kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
STO↑	STO f Max *r*	(0) {1, 2, 4, 5, 6, 10, 11} Stores the maximum of r and x in the address specified.[24]
	STO ▲ *r*	
STO↓	STO f Min *r*	(0) {1, 2, 4, 5, 6, 10, 11} Stores the minimum of r and x in the address specified.[24]
	STO ▼ *r*	
STRI?	g TEST g STRI?	(0) True if x is an alphanumeric string (like STR? in *HP-42S*).
STRING	f CAT. VARS STRING	*Submenu* of *alpha string* variables defined at execution time. See pp. 113f.
SUM	f STAT f SUM	(–2) { } → {2} Recalls the linear sums ***Σy*** and ***Σx***. Useful in basic 2D vector algebra. Output is labeled in analogy to s.
s_w	f STAT f s_w	(–1) { } → {2} Calculates the *standard deviation* for weighted data (where the weight y of each data point x was entered via Σ+). See pp. 204ff for the formula.

[24] Only legal operations according to the matrices in *Section 2* of the *OM* will work. See also the examples given there.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
s_{xy}	f STAT ▲ s_{xy}	(−1) { } → {2} Calculates the *sample covariance* for the two data sets entered via Σ+, depending on the curve fit model selected. See pp. 204ff for the formula and COV for the *population covariance*.
S.INTS	f CATALOG VARS S.INTS	*Submenu* of *short integer* variables defined at execution time. See pp. 113f.
s.t→kg	g U→ m: ▲ short ton → t	(1) {2, 11}; {1} → {2} Convert masses and times. See pp. 135ff.
s→year	g U→ f s→year	
T_0	g CNST T_0	(−1) { } → {2} Standard temperature (0°C) in *kelvin*.
tan	TRI tan	(1) {2, 3, 8*, 9*, 11, 12}; {1, 4} → {2} Returns the tangent of the angle in **X**. Returns "*Not a Number*" for x = ±90° or equivalents if *flag* **D** is set.
tanh	g EXP g tanh TRI g tanh	(1) {2, 3, 8*, 9*, 11, 12} Returns the hyperbolic tangent of x.
TDISP	g CLK ▲ TDISP *n*	(0) Sets time display format. TDISP **0** and **1** allow for displaying just *hours* and *minutes*, TDISP **2** for *seconds*, too, and $n \geq 3$ also for $n - 2$ digits showing decimal fractions of *seconds*. TDISP **−1** allows for displaying all digits.
TEST	g TEST	*Menu*. See p. 123.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
t_e(x) t_p(x) t(x) t^{-1}(p)	g PROB t: t_e(x) etc.	(1) {2, 11} *Student's t distribution*. The degrees of freedom are stored in **J**. t_e(x) equals Q(t) on *HP-21S*. See *Section 2* of the *OM* for an application example and pp. 204ff for more mathematical details.
TICKS	g P.FN TICKS	(−1) { } → {1} Returns the number of ticks from the real-time clock at execution time. 1 *tick* = 0.1 *s*. Counting starts when the calculator is turned on.
TIME	g CLK TIME	(−1) { } → {5} Recalls the time from the real-time clock at execution (see *Sect. 2* of the *OM* for the output format).
TIMER	f TIMER	Starts the timer application based on the real-time clock and following the timer of *HP-55*. See *Sect. 5* of the *OM* for a detailed description.
TIMES	f CATALOG VARS f TIMES	*Submenu* of *time* variables defined at execution time. See pp. 113f.
T_n	g X.FN Orthog T_n	(2) {2, 11}; {1} → {2} *Chebyshev polynomials of first kind*. See pp. 219f for details.
TONE	g I/O f TONE ***n***	(0) Sounds a tone according to ***n*** (= 1 … 9) .
ton→kg	g U→ m: ▲ ton→kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
TOP?	g TEST g TOP?	(0) Returns ... • false if called with the program pointer being in a subroutine; • true if called in the top routine (i.e. if the program-running *flag* is set and the *SRS* pointer is clear).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
tor→Pa	g U→ F&p: f torr → Pa	(1) {2, 11}; {1} → {2} Converts pressures. See pp. 135ff.
T_p	g CNST T_p	(−1) { } → {2} *Planck* temperature in *Kelvin*; *Planck* time in *seconds*.
t_{PL}	g CNST t_{PL}	
TRANS	f CATALOG FCNS TRANS	Works like $[M]^T$ on p. 100. Maintained for backward compatibility only.
TRI	TRI	*Menu*. See p. 123.
trz→kg	g U→ m: f tr.oz → kg	(1) {2, 11}; {1} → {2} Converts masses. See pp. 135ff.
TVM	g FIN TVM	Application. See *Section 5* of the *OM*.
t:	g PROB t:	*Submenu*. See p. 122.
t⇆	g STK t⇆ *r*	Swaps t and r, in analogy to x⇆
ULP?	g INFO f ULP?	(1) {1, 2, 11} Returns 1 times the smallest power of ten which can be added to x or subtracted from x to actually change the value of x in your *WP 43S* in the mode set. Thus 1 is returned for integers. Indicated in STATUS.
U_n	g X.FN Orthog U_n	(2) {2, 11}; {1} → {2} *Chebyshev polynomials of second kind*. See pp. 219f for details.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
UNITV	f MATX f UNITV	(1) {8, 9} Returns the unit vector for the matrix x (like UVEC in *HP-42S*). Each element of the matrix is adjusted so its overall Euclidean norm becomes 1 (see ENORM); for a vector, its magnitude will become 1.
	g CPX UNITV	(1) {3, 12} Returns a complex number with magnitude $\|r\|$ = 1 in direction of x.
UNSIGN	f BITS ▼ UNSIGN	(0) Sets unsigned mode for mode for operations on *short integers*. Indicated in the status bar. Cf. UNSGN on *HP-16C*. See *Section 2* of the *OM*.
	g INTS ▲ …	
U→	g U→	*Menu*. See p. 123.
VARMNU	g P.FN P.FN2 f VARMNU ***labl***	Creates a variable *menu* using the MVAR instructions following the global label specified. Cf. the *HP-42S Owner's Manual.*
VARS	f CAT. VARS	*Submenu* of variables defined at execution time. See pp. 113f.
VERS?	g INFO g VERS?	(0) Shows your firmware version and build number (see *Section 2* of the *OM*).
VIEW	f VIEW ***r***	(0) Shows ***r*** until the next key is pressed. **Example:** If ***r*** is e.g. a variable called **Test12** containing **−123.45**, VIEW α **Test12** ENTER↑ will display Test12 = -123.45
V_m	g CNST V_m	(−1) { } → {2} Molar volume of an ideal gas at standard conditions in *cubic meter per mol*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
V:	g U→ f V:	Submenu. See p. 135.
V∡	g ∡	(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	g CLK WDAY	(1) {2, 6, 11} → {1} Assumes x containing a *date* in the format selected (or a real number in corresponding format) and returns the name of the respective day and a corresponding integer (Monday = 1).[25]
Weibl Weibl$_e$ Weibl$_p$ Weibl^{-1}	g PROB f Weibl: Weibl etc.	(1) {2, 11} *Weibull distribution* with its shape parameter ***b*** in **I** and its characteristic lifetime ***T*** in **J**. See pp. 204ff 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:	g PROB f Weibl:	Submenu. See p. 122.
WHO?	g INFO g WHO?	(0) Displays credits to the brave men who made this project work.
Wh→J	g U→ E: Wh→J	(1) {2, 11}; {1} → {2} Converts energies. See pp. 135ff.
W$_m$ W$_p$ W^{-1}	g X.FN ▲ W$_m$ etc.	(1) {2, 3, 11, 12}; {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. 234ff for more.

[25] 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)
WSIZE	f BITS ▼ WSIZE *n*	(0) Works almost like on *HP-16C*, but with the parameter $1 \leq n \leq 64$ trailing the command instead of taken from **X**. Reducing *word* size truncates the values in the *stack* as allocated and in **L**. All other memory content stays as is (see *App. B* on pp. 155ff). Increasing the *word* size will add empty bits to each *stack register*. WSIZE **0** sets the *word* size to maximum, i.e. 64 *bits*. WSIZE is indicated in the status bar.
	g INTS ▲ ...	
WSIZE?	g INFO g WSIZE?	(−1) { } → {1} Recalls the *word* size set.
W→hp$_E$	g U→ P: W→hp$_E$ etc.	(1) {2, 11}; {1} → {2} Convert powers. See pp. 135ff.
W→hp$_M$		
W→hp$_{UK}$		
$\bar{x}$	f 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.
$\hat{x}$	f STAT ▲ $\hat{x}$	(1) {2, 11}; {1} → {2} Returns a forecast x for a given y (in **X**) according to the curve fit model chosen. See L.R. for more.
x^2	g EXP x^2	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} Return the square of x.
	f STAT x^2	
	f STAT ▲ x^2	
x^3	g EXP x^3	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} Return the cube of x.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
XEQ	XEQ ***labl***	(0) Executes the function or routine with the label specified. – In *PEM*, inserts a call to the subroutine with the label specified.
$\bar{x}_G$	f STAT g $\bar{x}_G$	(−2) { } → {2} Calculates the *geometric means* of the y- and x-data accumulated and and pushes them on the *stack*. See pp. 213ff for the formula. Output format will be similar to the one of x̄. See also ε , ε_m , and ε_P .
$\bar{x}_H$	f STAT ▲ f $\bar{x}_H$	(−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. SSIZE8). xIN is the recommended way to start an *XROM routine*. Thereafter, SSIZE4 is legal. **Note xIN cannot nest** and *XROM routines* using xIN cannot call user code.
XNOR	f BITS f XNOR	(2) Work in analogy to AND. See p. 18.
XOR	f BITS XOR	
xOUT	XEQ *way*	Cleans and reverts the settings of xIN, taking care of a proper return including the correct setting of ***I*** and the *stack*. Typically, ***way*** = `xOUT_NORMAL` . Generally, xOUT shall be the last command of an *XROM routine*.
$\bar{x}_{RMS}$	f STAT ▲ g $\bar{x}_{RMS}$	(−2) { } → {2} Calculates the *quadratic means* of the y- and x-data accumulated and pushes them on the *stack*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\bar{x}_w$	f STAT f $\bar{x}_w$	(−1) { } → {2} Returns the *arithmetic mean* for weighted data (where the weight y of each data point x was entered via Σ+). See pp. 213ff for the formula. See also s_w and s_{mw}.
$\sqrt[x]{y}$	g EXP $\sqrt[x]{y}$	(2) Returns the x^{th} root of y. Roots of negative integers or reals may return complex numbers if CPXRES is set.
X.FN	g X.FN	*Menu*. See p. 125.
x!	f 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 the biggest factorial allowed for *long integers*.
		(1) {2, 3, 11, 12} Returns $\Gamma(x+1)$. *204.3796629328708* is the max. x for *SP* reals. *2123.5499566246323631* is the max. x for *DP* reals.
x:	g U→ x:	*Submenu*. See p. 135.
x→DATE	g 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	g α.FN x→α	(1) {1, 2, 10, 11} → {7} Interprets x as a character code and converts the integer part of x to the respective character x, similar to XTOA in the *HP-42S*.
x⇆	g STK x⇆ *r*	Swaps x and r, analogous to $x⇆y$. Will be listed like x⇆ J, x⇆ .12, x⇆ →12, etc.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x⪤y	x⪤y	Swaps the *stack* contents ***x*** and ***y***.
x= ?	g TEST x= ? ***r***	(0) Compare ***x*** with ***r***. See x< ? for more.
x≠ ?	g TEST x≠ ? ***r***	
x=+0?	g TEST ▲ x=+0? etc.	(0) {1, 2, 3, 10, 11, 12} These tests are for comparing *short integers* in modes 1COMPL and SIGNMT, and for *long integers*, real or complex numbers if *flag* **D** is set. Then e.g. 0 / (−7) will display −0.
x=-0?		
x≈ ?	g TEST ▲ x≈ ? ***r***	(0) {2, 3, 4, 5, 8*, 9*, 11, 12} Will be true if the rounded values of ***x*** and ***r*** are equal (see ROUND). See x< ? for more.
x< ?	g TEST x< ? ***r*** etc.	(0) {1, 2, 4, 5, 6, 10, 11} Compare ***x*** with ***r***. **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 *Sect. 1* of the *OM* for more.
x≤ ?		
x≥ ?		
x> ?		
ŷ	f STAT ▲ ŷ	(1) {2, 11}; {1} → {2} Returns a forecast ***y*** (in **X**) for a given ***x*** according to the curve fit model chosen. See L.R. for more.
yd.→m	g U→ x: g yd.→m	(1) {2, 11}; {1} → {2} Converts distances. See pp. 135ff.
YEAR	g CLK f YEAR	(1) {2, 6, 11} → {1} Assumes ***x*** containing a *date* in the format selected (or a real number in corresponding format) and extracts the year.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
year→s	g U→ f year→s	(1) {2, 11}; {1} → {2} Converts times. See pp. 135ff.
y^x	y^x	(2) {1, 2, 3, 10, 11, 12} Returns the x^{th} power of y. It allows for raising any positive real number to an arbitrary real power, as well as any negative real number to an arbitrary integer power, all returning real results. Exceeding these boundaries may produce complex results (or errors if CPXRES is not set).
Y.MD	g CLK ▲ Y.MD	(0) Sets the format yyyy-mm-dd for *dates*.
y⇆	g STK y⇆ *r*	Swaps y and r, in analogy to x⇆.
Z_0	g CNST Z_0	(−1) { } → {2} Characteristic impedance of vacuum in *ohm*.
z⇆	g STK z⇆ *r*	Swaps z and r, in analogy to x⇆.
α	g CNST α	(−1) { } → {2} Fine-structure constant.
αINTL	f CAT. CHARS αINTL	*Submenu*. See pp. 113ff.
	g +	*Menu* in *AIM* (see p. 125).
αLENG?	g INFO g αLENG? *r* g α.FN f αLENG? *r*	(−1) { } → {1} Returns the number of characters found in r, similar to ALENG in *HP-42S*.[26]
αMATH	f CAT. CHARS αMATH	*Submenu*. See pp. 113ff.
	g −	*Menu* in *AIM*. See p. 126.

[26] This command will throw an error if there is no string in r at execution time.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
αOFF αON	g P.FN f αOFF etc.	(0) Turn *AIM* off and on, like AOFF and AON in *HP-42S*. Indicated in the status bar.
αPOS?	g INFO g αPOS? ***r*** g α.FN αPOS? ***r***	(−1) { } → {1} Looks in ***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.[26] The ***target*** may be an individual character code or an *alpha string*. αPOS saves a copy of the ***target*** in **L**. It works similar to POSA in *HP-42S*.
αRL	g α.FN αRL ***r***	(0) Rotates ***r*** by ***x*** characters like AROT in *HP-42S*, but with $x \geq 0$. αRL **0** executes as NOP, but loads **L**.[26]
αRR	g α.FN αRR ***r***	(0) Works like αRL but rotates to the right.
αSL	g α.FN αSL ***r***	(0) Shifts the ***x*** leftmost characters out of ***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**.[26]
αSR	g 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.[26]
α.FN	g α.FN	*Menu*. See p. 126.
A…Ω	f CAT. CHARS A…Ω	*Submenu* of Greek letters, see pp. 113ff.
α•	f CAT. CHARS α•	*Submenu*. See pp. 113ff.
	g .	*Menu* in *AIM*. See p. 126.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
α→x	g α.FN α→x *r*	(−1) { } → {1} Pushes the character code of the leftmost character in ***r*** on the *stack* and removes this character from the string, similar to ATOX in *HP-42S*.[26]
β(x,y)	g X.FN ▲ β(x,y)	(2) {2, 3, 11, 12}; {1} → {2} Returns *Euler's Beta* $\mathrm{B}(x,y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}$ with $Re(x) > 0$ and $Re(y) > 0$. Called β here to avoid ambiguity. See $\Gamma(x)$ below.
γ	g CNST γ	(−1) { } → {2} *Newtonian* constant of gravitation (also called G by other authors) in $m^3/_{kg\ s^2}$;
γ_{EM}	g CNST γ_{EM}	*Euler-Mascheroni* constant (for mathematics);
γ_p	g CNST γ_p	proton gyromagnetic ratio (see p. 133).
Γ_{xy}	g X.FN ▲ Γ_{xy}	(2) {2, 11}; {1} → {2} Returns the *lower incomplete Gamma function*. See pp. 234ff for more.
γ_{xy}	g X.FN ▲ f γ_{xy}	(2) {2, 11}; {1} → {2} Returns the *upper incomplete Gamma function*. See pp. 234ff for more.
Γ(x)	g PROB ▲ Γ(x)	(1) {2, 3, 11, 12}; {1} → {2} Returns $\Gamma(x)$. Note x! calls $\Gamma(x+1)$. See also LNΓ.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
δx	f CAT. PROGS δx	Predefined global label for f'(x) and f"(x) – see *Section 4* of the *OM*.
Δ%	f Δ%	(1) {2, 11}; {1} → {2} Returns $100\,\frac{x-y}{y}$ leaving y unchanged, like %CH in *HP-42S*. Use it also for calculating markups or margins as explained in the *OM*, *Sect. 2*.
ε	f STAT g ε	(−2) { } → {2} Calculates the *scattering factors* ε_y and ε_x for *log-normally* distributed sample data and pushes them on the *stack*. This ε_x works for the *geometric mean* $\bar{x}_g$ in analogy to the *standard deviation* s for the *arithmetic mean* $\bar{x}$ but <u>multiplicative</u> instead of additive. See pp. 204ff for more information.
ε_0	g CNST ε_0	(−1) { } → {2} Electric constant or vacuum permittivity in *ampere-second per volt-meter*.
ε_m	f STAT g 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	f STAT g ε_p	(−2) { } → {2} Works like ε but returns the *scattering factors* of the two populations.
ζ(x)	g X.FN ▲ f ζ(x)	(1) {2, 3} Returns *Riemann's Zeta*. See p. 237 for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
λ_C λ_{Cn} λ_{Cp}	g CNST λ_C etc.	(−1) { } → {2} *Compton* wavelength of the electron, neutron, and proton in *meter*.
μ_0 μ_B μ_e μ_e/μ_B μ_n μ_p μ_u μ_μ	g CNST μ_0 etc.	(−1) { } → {2} Magnetic constant or vacuum permeability in *volt-second per ampere-meter*; Bohr magneton in *joule per tesla*; electron magnetic moment in *joule per tesla*; ratio of electron magnetic moment to *Bohr* magneton; neutron and proton magnetic moments, nuclear magneton, and Muon magnetic moment in *joule per tesla*.
π	g π	(−1) { } → {2} Recalls π.
Π_n	f ADV Π_n ***labl***	Computes a product using the routine specified. See *Section 4* of the *OM* for more. **ATTENTION:** Π_n fills all *stack registers* with x before calling the routine specified.
σ	f STAT σ	(−2) { } → {2} Works like s but returns the *standard deviations* of the two *populations* instead. See pp. 204ff.
σ_B	g CNST σ_B	(−1) { } → {2} *Stefan-Boltzmann* constant (see p. 134).

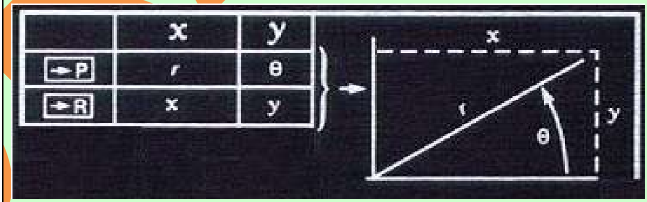
Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\Sigma\,^1/x$	g SUMS ▲ $\Sigma\,^1/x$ etc.	(−1) { } → {2} Recall the corresponding statistical sums, necessary for means and regressions beyond the linear model. Calling these sums by name significantly improves program readability. Note they are stored in dedicated *registers* of your *WP 43S* (see *App. B* on pp. 155ff.). **ATTENTION:** Depending on input data, some of the logarithmic sums may become non-numeric or some of the inverted may become infinite. If this happens no error will be thrown, however, regardless of the status of *flag* **D**. For space reasons, two sums are abbreviated: **Σlnxy** denotes $\sum ln(x)ln(y)$. **Σlny/x** denotes $\sum \frac{ln(y)}{x}$.
$\Sigma\,^1/x^2$		
$\Sigma\,^1/y$		
$\Sigma\,^1/y^2$		
Σln^2x	g SUMS f Σln^2x etc.	
Σln^2y		
Σlnx		
Σlnxy		
Σlny		
Σlny/x	g SUMS g Σlny/x	
Σ_n	f 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	f STAT f σ_w	(−1) { } → {2} Works like s_w but returns the *standard deviation* of the *population* instead. See pp. 204ff.
Σx	g SUMS Σx etc.	(−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see Σln^2x above for more).
Σx^2		
Σx^2y	g SUMS g Σx^2y	
$\Sigma x^2/y$	g SUMS ▲ $\Sigma x^2/y$	
Σx^3	g SUMS ▲ f Σx^3 etc.	
Σx^4		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Σxlny	g SUMS g Σxlny	(−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see $\Sigma\ln^2 x$ above for more).
Σxy	g SUMS Σxy	
Σx/y	g SUMS ▲ Σx/y	
Σy	g SUMS Σy etc.	
Σy^2		
Σylnx	g SUMS g Σylnx	
Σ+ [27]	f 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 and the matrix will be in **L**. {1, 2, 11} Adds one 2D data point to the statistical sums.
Σ− [27]	f STAT f Σ-	{1, 2, 11} Subtracts one 2D data point from the statistical sums.
Φ	g CNST Φ	(−1) { } → {2} Golden ratio and magnetic flux quantum, the latter in *volt-second*.
Φ_0	g CNST Φ_0	
$\chi^2_e(x)$	g PROB χ^2: $\chi^2(x)$ etc.	(1) {2, 11} *Chi-square distribution* (with its degrees of freedom given in **I**). χ^2_e(x) equals Q(χ^2) on *HP-21S*. See *Section 2* of the *OM* for an application example and pp. 204ff for more.
$\chi^2_p(x)$		
$\chi^2(x)$		
$(\chi^2)^{-1}$		

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
χ^2:	g PROB χ^2:	*Submenu*. See p. 122.
ω	g CNST ω	(−1) { } → {2} Angular velocity of the Earth in *radian per second* according to *WGS84* (see footnote 23 on p. 71).
$(-1)^x$	g X.FN ▲ f $(-1)^x$	(1) {1, 2, 3, 8*, 9*, 10, 11, 12} If x is non-integer, returns $cos(\pi\ x)$.
$[M]^T$	f MATX $[M]^T$	(1) {8, 9} Returns the transpose of the matrix x (like TRANS in *HP-42S*). The transpose is another matrix with rows changed by columns. If $\boldsymbol{A}$ is an $\boldsymbol{n}\times\boldsymbol{m}$ matrix and $\boldsymbol{a_{ij}}$ is an element of it then $\boldsymbol{A}^T$ will be an $\boldsymbol{m}\times\boldsymbol{n}$ matrix $\boldsymbol{B}$ with $\boldsymbol{b_{ij}} = \boldsymbol{a_{ji}}$. The transpose is done in-situ and does not require any additional memory.
$[M]^{-1}$	f MATX $[M]^{-1}$	(0) {8, 9} Takes the square matrix in **X** and inverts it in-situ (like INVRT on *HP-42S*).
+	+	(2) Returns $y + x$ for compatible objects. See the tables in *Section 2* of the *OM* for details.
+/−	+/− (for closed input)	(1) 'Unary minus', returns $x \times (-1)$.
±∞?	g INFO g ±∞?	(0) {2, 11} → {1} Tests x for infinity. Returns true 1 for $x = +\infty$, true -1 for $x = -\infty$, and false 0 else.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
–	–	(2) Returns $y - x$ for compatible numeric objects. See the tables in the *OM, Sect. 2,* for details.
-∞	g CNST -∞	(−1) { } → {2} Minus infinity. See p. 134.
×	×	(2) Like – , but returns $y \times x$.
×MOD	g INTS f ×MOD	(3) {1, 2, 10, 11} Returns $(z \times y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. See MOD.
/	/	(2) Like – , but returns $y \times x^{-1}$. Returns $y \operatorname{div} x$ if both y and x are of *data type* 1 or 10; cf. IDIV.
^MOD	g INTS g ^MOD	(3) {1, 2, 10, 11} Returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. See MOD.
→		Reserved symbol for indirect addressing.
→DATE	g 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	f L→ →DEG	(1) {1, 2, 4, 11} → {4} Converts angles as described on pp. 142f.
→DP	g →DP g X.FN ▲ f →DP	(1) {1, 2, 11} → {11}; {3, 12} → {12} Converts x into a *DP* number. Numbers shown as fractions will be displayed as decimal numbers (cf. IMPFRC and PROFRC). Compare →SP.
→D.MS	f L→ →D.MS	(1) {1, 2, 4, 11} → {4} Convert angles as described on pp. 142f.
→GRAD	f L→ →GRAD	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→HR	f CATALOG FCNS →HR	(1) {5} → {2} Operates on *times* like →REAL below. Maintained for backward compatibility only.
→H.MS	f h.ms (for closed input)	(1) {1, 2, 5, 11} → {5} Converts x to a sexagesimal *time* – cf. p. 109.
→INT	f # ***base*** (for closed input)	(1) {1, 2, 10, 11} → {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. Cf. p. 109.
→MULπ	f L→ →MULπ	(1) {1, 2, 4, 11} → {4} Converts angles as described on pp. 142f.
→POL	g →P	{2, 11}; {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***). See the picture and cf. →REC.  For switching the display format of complex numbers, choose POLAR.
→RAD	f L→ →RAD	(1) {1, 2, 4, 11} → {4} Converts angles as described on pp. 142f

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→REAL	f .d (for closed input)	(1) {1, 2, 4, 5, 6, 10, 11} → {2} Converts *x* to an *SP* real number. Any object (e.g. a *time*) tagged sexagesimal will be converted in a decimal number. For {6}, the date format chosen is taken into account. Numbers shown as fractions will be displayed as decimal numbers (cf. IMPFRC and PROFRC). For returning the real part of a complex number, choose RE. For cutting a complex number into its parts, use CC.
→REC	f R←	{2, 11}; {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***). See the picture and cf. →POL on previous page. For switching the display format of complex numbers, choose RECT.
→SP	f SP← g X.FN ▲ f →SP	(1) {1, 2, 11} → {2}; {3, 12} → {3} Converts *x* to an *SP* real number. Numbers shown as fractions will be displayed as decimal numbers (cf. IMPFRC and PROFRC). Cf. →DP.
≷	g STK ≷_ _ _ _	Shuffles the contents of the *stack registers* **X**, **Y**, **Z**, and **T** at execution time. **Examples:** ≷xxyz works like ENTER↑ (but does <u>not</u> disable *automatic stack lift!*), ≷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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		ATTENTION: This is a very powerful command although it does not look it. Note it will affect the <u>bottom four *stack registers* only</u>; there is no connection to **A** … **D**, <u>regardless of *stack* size</u>. Playing with ⁑, you may lose some *stack* contents and make a mess of the *stack* easily.
%	g FIN %	(1) {2, 11}; {1} → {2} Returns $\frac{xy}{100}$, leaving y unchanged.
%MRR	g FIN %MRR	(3) {2, 11}; {1} → {2} Returns the mean rate of return in percent 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	g FIN %T	(1) {2, 11}; {1} → {2} Returns $\frac{100\,x}{y}$, interpreted as % of <u>t</u>otal. Leaves y unchanged.
%Σ	g FIN %Σ	(1) {2, 11}; {1} → {2} Returns $\frac{100\,x}{\sum x}$.
%+MG	g FIN %+MG	(2) {2, 11}; {1} → {2} Calculates a sales price by adding a margin of x % to the cost y, as %MU-Price in *HP-17B*. Formula: $p_{sale} = \frac{y}{1 - \frac{x}{100}}$ You may use %+MG for calculating net amounts as well; just enter a negative percentage in x.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
√x	√x	(1) {1, 2, 3, 8*, 9*, 10, 11, 12}; ({1} → {2}, {2} → {3}) Returns the square root of x. Square roots of non-square *long integers* will return reals. Roots of negative *long integers* or reals will return complex numbers if CPXRES is set.
∞	g CNST ∞	(−1) { } → {2} Infinity. See p. 134.
∫	f 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**. **ATTENTION:** ∫ fills all *stack registers* with x before calling the routine specified in PGMINT. *See Section 4 of the OM for more.*
	g EQN ∫f ∫	Integrates the current equation.
∫f	g EQN ∫f	*Submenus*. See pp. 119f.
∫fdx	f ADV ∫fdx	
|M|	f MATX |M|	(1) {8} → {2}; {9} → {3} Requires a square matrix in **X** and returns its determinant. The original matrix is stored in **L**.
|x|	f |x| or f PARTS f |x|	(1) {1, 2, 4, 10, 11} Returns the absolute (unsigned) value of x. (1) {8*} Returns a real matrix with the absolute values of all input matrix elements. Cf. ENORM.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
	f |x| or g CPX f |x|	(1) {3} → {2}; {12} → {11} Returns the magnitude $\sqrt{\mathrm{Re}(x)^2+\mathrm{Im}(x)^2}$ in **X**. (1) {9*} → {8} Returns a real matrix with the magnitudes of all input matrix elements. Cf. ENORM.
| |	g X.FN ▲ f | |	(2) {2, 3, 11, 12}; {1} → {2} Returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{-1}$, being useful in electrical engineering especially. Returns 0. for x or y being zero.
∡	g ∡ or g CPX f ∡ or f PARTS f ∡	(1) {3} → {2}; {12} → {11} 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|.
∡→	f L→	*Menu* of angular conversions. See p. 127.
🖶ADV	g I/O ▲ 🖶ADV	(0) Prints the current contents of the print buffer and a linefeed. **ATTENTION:** The printer will actually print only when a linefeed is sent to it.
🖶CHAR	g I/O ▲ 🖶CHAR *n*	(0) Sends a single character (with the code specified) to the printer. Character codes n > 127 can only be specified indirectly. 🖶MODE setting will be honored. See 🖶ADV.
🖶DLAY	g I/O ▲ 🖶DLAY *n*	(0) Sets a delay of n ticks (see TICKS) to be used with each linefeed on the printer.
🖶LCD	g I/O ▲ 🖶LCD	(0) Sends the contents of the entire *LCD* to the printer.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
🖶MODE	g I/O ▲ 🖶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 channel. Works for plain ASCII only – no characters will be translated. Line setup is the same as for serial communication: 9600 baud, 8 bits, no parity.
🖶PROG	g I/O ▲ 🖶PROG	(0) Prints the listing of the *current program* (see *Sect. 3 of the OM*), 1 row per step. See 🖶ADV.
🖶REGS	g I/O ▲ f 🖶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 a label. See also 🖶ADV. **ATTENTION for *nn* = 0 :** • For ***sss*** $\in [0; 99]$, printing will stop at **R99**. • For ***sss*** $\in [100; 111]$, printing will stop at **K**. • For ***ss*** ≥ 112, printing will stop at the highest allocated local *register*.
🖶r	g I/O ▲ f 🖶r *r* f 🖶x	(0) Prints the *register* specified, right adjusted, without labeling the output. Note 🖶r **X** is on the keyboard. If you want a heading label, compose the string in **X** first or use 🖶REGS. See 🖶ADV.
🖶STK	g I/O ▲ f 🖶STK	(0) Prints the *stack* contents. Each *register* prints in a separate row starting with a label indicating said *register*. See 🖶ADV.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
🖶TAB	g I/O ▲ f 🖶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	g I/O ▲ f 🖶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	g I/O ▲ f 🖶WIDTH	(−1) Returns the number of print columns that ***x*** would take in the print mode set. See 🖶ADV and 🖶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.
🖶Σ	g I/O ▲ g 🖶Σ	(0) Prints the summation *registers*. Each *register* prints in one row starting with its label. See 🖶ADV.
🖶#	g I/O ▲ g 🖶# *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.
#	f CATALOG FCNS # *n*	For inserting an integer $0 \le n \le 255$ in a single program step. Maintained for backward compatibility to *WP 34S* only.
#B	f BITS g #B	(1) {10} Counts the bits set in ***x*** (like on *HP-16C*).

Predefined Variables Provided

There is a *name* overlap between some constants and predefined variables. Thus, the latter set is kept separate from the other *items*. As required for the *items* above, *names* of variables must be unique.

Variable *name*	Keystrokes	Remarks (see pp. 12ff for general information)
A		Reserved variable for *register* **A**
ACC	f ADV ∫fdx ACC	Reserved real variable for the accuracy of integration (see *Sect. 4* of the *OM*).
B		Reserved variables for *registers* **B**, **C**, and **D**.
C		
D		
FV	g FIN TVM FV	Reserved variable for the future value of your investment or loan in *TVM*.[28]
I		Reserved variable for *register* **I**.
i%/a	g FIN TVM i%/a	Reserved variable for the annual interest rate of your investment or loan in *TVM*.[28]
J		Reserved variables for *registers* **J**, **K**, and **L**.
K		
L		
Mat_A	f MATX SIM EQ Mat A	Reserved variables for solving systems of linear equations (*SLE*, see *Section 2* of the *OM*).
Mat_B	f MATX SIM EQ Mat B	
Mat_X	f MATX SIM EQ Mat X	

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

Variable *name*	Keystrokes	Remarks (see pp. 12ff for general information)
NPER	g FIN TVM n_{PER}	Reserved variable for the <u>total</u> number of • payment periods for your loan or • compounding periods for your investment.
PER/a	g FIN TVM f per/a	Reserved variable for the <u>annual</u> number of • payments for your loan or • compounding periods of your investment.
PMT	g FIN TVM PMT	Reserved variable for the payment per period for your investment or loan in *TVM*.[28]
PV	g FIN TVM PV	Reserved variable for the present value of your investment or loan in *TVM*.[28]
REGS		Reserved variable for the 100×1 matrix of registers – if required.
ST.A		Reserved variables for *stack registers* **A** … **Z**
ST.B		
ST.C		
ST.D		
ST.T		
ST.X		
ST.Y		
ST.Z		
↑Lim	f ADV ∫fdx ↑Lim etc.	Reserved real variables for the upper and lower limit of integration (see the *OM, Sect. 4*).
↓Lim		

Nonprogrammable Commands and Keys

The commands marked violet 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.

Furthermore, all *catalog* and *menu* calls themselves as well as the operations called by EXIT, P/R, USER, α, ↰, ≡▲, ▲, ≡▼, and ▼ are neither programmable nor will they show any input echo in the top numeric row as the other commands do (cf. the *OM*, *Sect. 2*). See also *Section 2: Menus and Catalogs* (on pp. 113ff) for more about this topic.

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 – see the *OM, Section 5*).

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. 109ff for input in **X** instead). Note that characters include digits and punctuation marks as well. The table lists the respective keys beginning top left on the keyboard:

Keystrokes	Situation	Remarks
A … Z	addressing	*Register* or *flag* input. See the *virtual keyboard* in *Section 1* of the *OM* for the letters applicable.
A … Z	entering a label or a variable *name*	Appends the corresponding Latin or Greek letter or digit to the label or variable *name* pending. Use ▼ and ▲ to switch cases for letters. See the virtual keyboard on p. 108 (and cf. *Section 2* of the *OM*).
g A … g O		
f 0 … f 9		

Keystrokes	Situation	Remarks
ENTER↑	arbitrary parameter input pending	If there is no input yet, assumes the default, if applicable. Closes pending input, interprets it as a *register* or *flag* address or a variable *name* or a label or alike, and executes the command. Cf. *Section 1* of the *OM*.
←	arbitrary parameter input pending	Deletes the rightmost character keyed in. If there is nothing left, cancels the pending command, returning to the status of your *WP 43S* as it was before that input was started.
0 … 9	addressing	Numeric parameter input. See *Section 1* of the *OM* for the valid number ranges.
.	addressing	Header for *local registers* or *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 that command was called.

Virtual keyboard in *alpha input mode* (*AIM*). *AIM* is also active when a catalog is entered, so you can use all accessible characters for alphabetic searching (see pp. 117f).

Alphanumeric Input in X and Closing It

The following table shows what will happen when particular keys are pressed with alphanumeric or numeric input in **X** being open still (turn to pp. 106f for command parameter input instead). The table lists the respective keystrokes top left to bottom right on the keyboard:

Keystrokes	in mode(s)	Remarks
f # ***base***	¬ (A, α)	Closes input of a *short integer*
f d.ms		sexagesimal angle
f .d		*date*
f h.ms		sexagesimal *time* in **X**.[29]
A … Z	A, α	Appends the corresponding Latin or Greek letter to the *alpha string* x. Use ▼ and ▲ to switch cases. See the picture on previous page and cf. *Section 2* of the *OM*.
g A … g O		
f #	A, α	Appends # to the *alpha string* x.
CC	¬ (A, α)	Closes input of the first part of a complex number in **X** and waits for input of its second part (see the *Key Response Table* and *Section 2* of the *OM*).
f R↓ (⇩)	A, α	Prefix for the next character becoming a subscript, if applicable.

[29] See *Section 2* of the *OM*. At closure, input will be checked – illegal digits (e.g. 8 in octal input or C in decimal), bases, numbers (e.g. 72 *minutes* in a *time*), 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)	Remarks
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 an *alpha 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*; ○ 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 29 and interprets it. Finally, unless an error had to be thrown, copies x into **Y**.
f x≷	A, α	Appends ≷ to the *alpha string* x.
+/_	¬ (A, α)	Changes the sign of the mantissa or exponent in numeric input as explained in *Section 1* of the *OM*.
f +/_	A, α	Appends ± to the *alpha string* x.
E	¬ (A, α)	Closes input of the mantissa and waits for input of the exponent (see *Section 1* of the *OM*).

Keystrokes	in mode(s)	Remarks
f E (⇧)	A, α	Prefix for the next character becoming a superscript, if applicable.
←	arbitrary input pending	Deletes the last (rightmost) character keyed in. If there is nothing left, cancels the pending input, returning to the status of your *WP 43S* as it was before that input was started.
/	A, α	Appends / to the *alpha string* x.
f ×	A, α	Appends × to the *alpha string* x.
0 … 9	¬ (A, α)	Standard numeric input, appending the corresponding digit to x. Note you can enter … • up to 16 digits plus a sign in the mantissa and up to three 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.
f 0 … f 9	A, α	Appends the respective digit to the *alpha string* x.
▲	A, α	Turns to upper case for the following letter(s).
▼	A, α	Turns to lower case for the following letter(s).
f −	A, α	Appends - to the *alpha string* x.
g INTS A … F	¬ (A, α)	Numeric input for bases >10, appending the corresponding digit to x. See *Section 2* of the *OM* for more. Digits will be checked when input is closed (see the description of ENTER↑ above).
f +	A, α	Appends + to the *alpha string* x.

Keystrokes	in mode(s)	Remarks	
?	A, α	Appends ? to the *alpha string* ***x***.	
.	¬ (A, α)	Inserts a radix mark as selected.	
		Separates *degrees* from *minutes*, *seconds*, and *hundredths of seconds* in angular input, so input format is ddddd.mmsshh d.ms for sexagesimal angles (cf. p. 109 and *Section 2* of the *OM*).	
		Separates *hours* from *minutes*, *seconds*, and fractions of *seconds* in *time* input, so input format is hhhhh.mmssffff h.ms for sexagesimal *times* (cf. p. 109 and *Section 2* of the *OM*).	
Second .	¬ (A, α)	A second . in input indicates a fraction. See *Section 2* of the *OM* for examples. The second . just separates the nominator and the denominator in input. Note you cannot enter E after you entered . twice – but you may delete the second dot while editing the input row.	
,	A, α	Appends , to the *alpha string* ***x***.	
f .	A, α	Appends . to the *alpha string* ***x***.	
R/S	program waiting for arbitrary input	Closes input and starts its checks and interpretation like ENTER↑ above.	
	A, α	Appends a blank space to the *alpha string* ***x***.	
f R/S	A, α	Appends 🖶 to the *alpha string* ***x***.	
EXIT	arbitrary input pending	If there is an open *menu*, closes it. Else closes pending numeric or alphanumeric input and releases it for interpretation.	

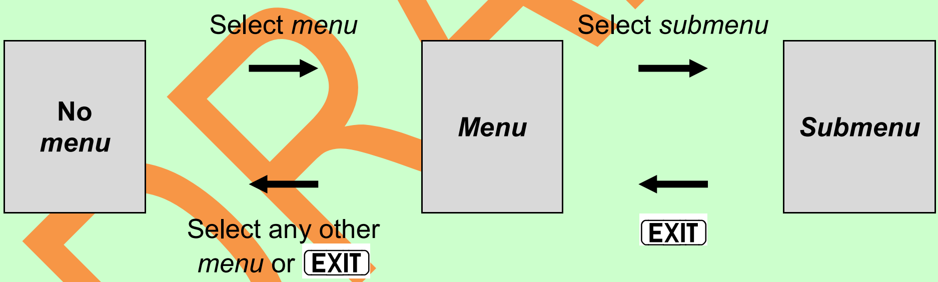
There are many more characters you can enter via the three alpha *menus* or CATALOG CHARS. See pp. 114 and 125ff for these menus and FBR for browsing the entire character sets provided.

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, program labels).

Catalogs are a special kind of *menus* with their contents sorted alphabetically. Your *WP 43S* provides two *catalogs*, CATALOG and CNST. Also some *submenus* of CATALOG are treated as *catalogs* (i.e. A...Z, DIGITS, FCNS, MENUS, αINTL, and the *submenus* of VARS, see next chapter). Within *catalogs*, some special operations ease your path accessing the *items* stored there (as shown on pp. 117f).

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



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. 117f.

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

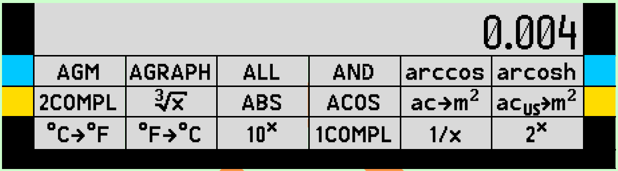
	^	^	^	^	^	^	Remarks
CATALOG:	FCNS	DIGITS	CHARS	PROGS	VARS	MENUS	top branches
FCNS:	°C→°F	°F→°C	10^x	1COMPL	$^1/x$	2^x	contains 678 functions provided
	2COMPL	$\sqrt[3]{x}$	ABS	ACOS	ac→m^2	ac$_{US}$→m^2	
	AGM	AGRAPH	ALL	AND	arccos	arcosh	
	…						
	…	#B					
DIGITS:	0	1	2	3	4	5	digits defined
	6	7	8	9	A	B	
	C	D	E	F	i		
CHARS:	A…Z	A…Ω	αINTL	αMATH	Myα	α•	character branches
A…Z:	A	B	C	D	E	…	plain Latin letters
αINTL:	Ā	Á	Ă	À	…		additional (international) Latin letters, see p. 125
αMATH:	<	≤	=	…			mathematical operators and symbols, see p. 126
A…Ω:	Α	Β	Γ	Δ	…		Greek letters, see p. 126
α•:	!	:	;	…			punctuation marks, see p. 127
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:	...						all variables currently defined, placed following their *data types* – all these *submenus* are empty at startup but will be filled and grow with your creations
CPXS:	...						
DATES:	...						
L.INTS:	...						
MATRS:	...						
REALS:	...						
STRING:	...						
S.INTS	...						
TIMES:	...						
MENUS:	ANGLES	A...Z	A:	BITS	Binom:	Cauch:	*menus* and *submenus* currently defined (shown here at startup, but also this list will grow with your creations) – see above and below for fix menu contents
	CHARS	CLK	CLR	CNST	CPX	CPXS	
	DATES	DIGITS	DISP	EQN	EXP	Expon:	
	...						
	...	↯→					
A:	...						*(sub-) menus* provided unless mentioned above already, see pp. 119ff 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, Sect. 6*); the other are fixed size (**FCNS**, **DIGITS**, and **CHARS**) since all functions, digits, and characters are predefined.

Calling CATALOG will display its top level branches (one row of labels, being pointers to the *submenus* containing all the functions, digits, characters, programs, variables, and *menus* defined at execution time):

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



Within CATALOG branches, browsing by ▲ will advance by only six *items* per keystroke (and ▼ will go back by six).[30] Select an *item* by pressing the corresponding top row key (headed by a *prefix* if applicable); e.g. call a function by pressing the corresponding *softkey*. EXIT will just leave CATALOG without doing anything.

All the functions available on your *WP 43S* are stored in CATALOG'FCNS. Most of them are also found (and easier to access) in other predefined *menus*. Each and every command featured on your *WP 43S*, the keystrokes calling it, and the necessary particular explanations are also listed for your reference in the *IOI* on pp. 12ff.

Remember all labels printed underlined on the keyboard point to *menus*. All *menus* available are found in CATALOG'MENUS; predefined *menu names* are listed in the *IOI* as well; their particular contents are printed in next chapter. Individual *items* may appear in more than one *menu* and also on the keyboard.

See *Section 6* of the *OM* to learn how to customize your *WP 43S* by creating and filling your own *menus* (assignable to your favorite keyboard locations) and accessing the functions you stored therein. You may as well assign your favorite individual functions to almost any location on the keyboard. Actually, you can design your very own *WP 43S* user interface.

---

[30] Navigating in 'CATALOG, *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.

Accessing Cataloged Items Rapidly

You can browse a *catalog* like any other *menu* just using ▲ and ▼ as explained in previous chapter. In CNST and major parts of CATALOG (FCNS, MENUS, PROGS RAM or FLASH, CHARS αINTL, and the *submenus* of VARS), however, you may reach your target significantly faster taking advantage of the alphabetic access method demonstrated here.

Assume we are looking for the function FS?S, for **example**:

1	User input	CATALOG FCNS
	Echo	**Your *WP 43S* displays the first *view* in this *catalog*[31]**

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

2	User input	First character of the *item* desired (e.g. F)
	Echo	**Your *WP 43S* displays a *view* starting with the first *item* starting with this character[32]**

[31] ... 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.

[32] 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 also enter Greek letters in such a search using prefix g, e.g. g + A for α – though watch the sorting order printed at the beginning of the *IOI*. Also some 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 positions in the *menu section* deviating from the ones you see in simple browsing using just ▼ or ▲.

You may put in more than one character (see overleaf) – 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 sequence 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.

	e.g.					
	FLASH?	FLOOR	FP	FP?	$F_p(x)$	FS?
	FC?S	$F_e(x)$	FF	FIB	FILL	FIX
	FAST	FB	FBR	FC?	FC?C	FC?F
3 User input	Second character of the *item* desired (e.g. S)					
Echo	**Your *WP 43S* displays a *view* starting with the first *item* starting with this sequence** e.g.					
	fz_{US}→m^3	f'(x)	f"(x)	GAP	GaussF	GCD
	fm.→m	fr→dB	ft.→m	ft_{US}→m	ft.→m	fz_{UK}→m^3
	FS?	FS?C	FS?F	FS?S	F(x)	$F^{-1}(p)$
4 User input	Press the corresponding *softkey* e.g. for FS?S					
Echo	**Your *WP 43S* executes or inserts the command, recalls the constant, or inserts the letter selected.** **Result** (in this example after specifying the *flag* number): true 1					

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 750);
- any constant provided can be recalled by g CNST + 3 keystrokes maximum if you know its first character;
- any letter provided can be inserted by f CATALOG CHARS αINTL (or in *AIM* by g +) + 3 keystrokes maximum.

Further Menus and Their Contents

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

Different *views* within one *menu* are separated by a dashed line, *submenus* by a double line.

Menu	^	^	^	^	^	^	Remarks
ADV	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx	advanced operations, see *Sect. 4 of the OM*
			f"(x)				
∫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	
	SL	RL	RLC	RRC	RR	SR	
	LJ					RJ	
	1COMPL	2COMPL	UNSIGN	SIGNMT	LZON	WSIZE	
					LZOFF		
CLK	DATE	→DATE	DATE→	WDAY	TIME	x→DATE	date and time functions (first view) and settings (2nd view)
	J→D	D→J		DAY	MONTH	YEAR	
	SETTIM	TDISP	SETDAT	D.MY	Y.MD	M.DY	
	CLK12	CLK24				J/G	
CLR	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX	almost as in *HP-42S*
	CLREGS	CLPall	CLFall		CLLCD	CLSTK	
	CLall					RESET	
CNST							*catalog* of constants, see pp. 127ff

Menu	⌃	⌃	⌃	⌃	⌃	⌃	Remarks
CPX	dot	cross	UNITV	Re	conj	Re⇄Im	special complex functions and settings (third row)
	CX→RE	RE→CX	sign	Im	|x|	∡	
	CPXi	CPXj			RECT	POLAR	
DISP	FIX	SCI	ENG	ALL	ROUNDI	ROUND	display rounding and shifts, formats and settings, mostly for reals
	SDL	SDR			RDP	RSD	
	SCIOVR	ENGOVR	MULT×	MULT·	RDX.	RDX,	
	GAP					DSTACK	
	CHINA	EUROPE	INDIA	JAPAN	UK	USA	
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	x^2	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	
FLAGS	SF	FS?	FF	STATUS	FC?	CF	
	FS?S	FS?C	FS?F	FC?F	FC?S	FC?C	
						CLFall	
INFO	SSIZE?	MEM?	RM?	SMODE?	WSIZE?	KTYP?	system information
	LocR?	FLASH?	ULP?	NEIGHB	SDIGS?	BATT?	
	WHO?	VERS?	DIM?	±∞?	αPOS?	αLENG?	

Menu	^	^	^	^	^	^	Remarks
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	LZON	WSIZE	
					LZOFF		
I/O	BEEP	LOAD	LOADP	LOADR	LOADSS	LOADΣ	data exchange incl. the PRINT commands of *HP-42S*
	QUIET	TONE			RECV	SEND	
	⎙ADV	⎙CHAR	⎙DLAY	⎙LCD	⎙MODE	⎙PROG	
	⎙r	⎙REGS	⎙STK	⎙TAB	⎙USER	⎙WIDTH	
	⎙Σ	⎙#					
LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG	
	DEC					INC	
MATX	NEW	[M]⁻¹	|M|	[M]ᵀ	SIM EQ	EDIT	matrix operations (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	DEG	RAD	GRAD	MULπ	RECT	POLAR	mode settings (top six almost as in *HP-42S*)
	FAST	SLOW	RM	QUIET	REALRE	CPXRES	
	DENMAX	DENANY	DENFAC	DENFIX	SSIZE4	SSIZE8	
	SETSIG						
MyMenu							will show up out of *AIM* [33]

Menu							Remarks
Myα							will show up in *AIM*[33]
PARTS	IP	FP	MANT	EXPT	sign	DECOMP	some overlaps with *HP-42S* CONVERT
	ROUNDI	ROUND			|x|	∡	
					Re	Im	
PROB	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:	combinations, permutations, random number generator and 15 probability distributions. Selecting one (e.g. Norml) opens a submenu featuring entries for its *PDF* (or *PMF*), *CDF*, error probability & quantile function
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:	
		NBin:	Geom:	Hyper:	Binom:	Poiss:	
	RAN#	SEED				Γ(x)	
Binom:	Binom_p	Binom			Binom_e	Binom^{-1}	
Cauch:	Cauch_p	Cauch			Cauch_e	Cauch^{-1}	
Expon:	Expon_p	Expon			Expon_e	Expon^{-1}	
F:	$F_p(x)$	$F(x)$			$F_e(x)$	$F^{-1}(p)$	
Geom:	Geom_p	Geom			Geom_e	Geom^{-1}	
Hyper:	Hyper_p	Hyper			Hyper_e	Hyper^{-1}	
LgNrm:	LgNrm_p	LgNrm			LgNrm_e	LgNrm^{-1}	
Logis:	Logis_p	Logis			Logis_e	Logis^{-1}	
NBin:	NBin_p	NBin			NBin_e	NBin^{-1}	
Norml:	Norml_p	Norml			Norml_e	Norml^{-1}	
Poiss:	Poiss_p	Poiss			Poiss_e	Poiss^{-1}	
t:	$t_p(x)$	$t(x)$			$t_e(x)$	$t^{-1}(p)$	
Weibl:	Weibl_p	Weibl			Weibl_e	Weibl^{-1}	
χ^2:	$\chi^2_p(x)$	$\chi^2(x)$			$\chi^2_e(x)$	$(\chi^2)^{-1}$	

[33] ... 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 *multi-view* here).
	PSTO	PRCL	αOFF	αON	CONST	PUTK	
	R-CLR	R-COPY	R-SORT	R-SWAP	LocR	PopLR	
P.FN2	MENU	KEYG	KEYX	CLMENU	EXITall	RTN+1	
	SDL	SDR	MSG	NOP	VARMNU	MVAR	
	BACK	CASE	SKIP	AGRAPH	PIXEL	POINT	
STAT	Σ+	$\bar{x}$	s	σ	s_m	x^2	for sample statistics.
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	SUM	
	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	PLOT	
	L.R.	r	s_{xy}	$\hat{x}$	$\hat{y}$	x^2	for curve fitting and 2d sample statistics.
		$\bar{x}_H$	cov				
		$\bar{x}_{RMS}$					
	LinF	ExpF	LogF	PowerF		BestF	for choosing the fit model(s)
	OrthoF	GaussF	CauchF	ParabF	HypF	RootF	
STK	x⇄	y⇄	z⇄	t⇄	⇄	DROPy	*stack* related operations.
	SSIZE4					SSIZE8	
SUMS	n	Σx	Σx^2	Σxy	Σy^2	Σy	all the sums necessary for the statistics in STAT.
		$\Sigma \ln x$	$\Sigma \ln^2 x$	$\Sigma \ln xy$	$\Sigma \ln^2 y$	$\Sigma \ln y$	
	$\Sigma x^2 y$	$\Sigma x \ln y$		$\Sigma \ln y/x$		$\Sigma y \ln x$	
	$\Sigma x^2/y$	$\Sigma 1/x$	$\Sigma 1/x^2$	$\Sigma x/y$	$\Sigma 1/y^2$	$\Sigma 1/y$	
	Σx^3	Σx^4					
TEST	x< ?	x≤ ?	x= ?	x≠ ?	x≥ ?	x> ?	binary test commands.
	INT?	EVEN?	ODD?	PRIME?	LEAP?	FP?	
	ENTRY?	KEY?	LBL?	STRI?	CONVG?	TOP?	
	x=+0?	x=-0?	x≈ ?	MATR?	CPX?	REAL?	
	SPEC?	NaN?		M.SQR?		DBL?	

Menu	^	^	^	^	^	^	Remarks
TRI	sin	arcsin	cos	arccos	tan	arctan	trigonometric & hyperbolic functions (cf. EXP).
	sinh	arsinh	cosh	arcosh	tanh	artanh	
U→	E:	P:	year→s	F&p:	m:	x:	unit conversions (see pp. 136ff).
	°C→°F	°F→°C	s→year		V:	A:	
	power ratio → dB	dB → power ratio			field ratio → dB	dB → field ratio	
A:	acre → m^2	m^2 → acre			acre$_{US}$ → m^2	m^2 → acre$_{US}$	units of area
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	
m:	lb.→kg	kg→lb.	cwt→kg	kg→cwt	oz→kg	kg→oz	units of mass
	stone → kg	kg → stone	short cwt→kg	kg → sh.cwt	tr.oz → kg	kg → tr.oz	
	ton→kg	kg→ton	short ton → kg	kg → short ton	carat → kg	kg → carat	
P:	hp$_E$→W	W→hp$_E$	hp$_{UK}$→W	W→hp$_{UK}$	hp$_M$→W	W→hp$_M$	units of power
V:	gl$_{UK}$→m^3	m^3→gl$_{UK}$	qt.→m^3	m^3→qt.	gl$_{US}$→m^3	m^3→gl$_{US}$	units of volume
	floz$_{UK}$ → m^3	m^3 → floz$_{UK}$	barrel → m^3	m^3 → barrel	floz$_{US}$ → m^3	m^3 → floz$_{US}$	
x:	au→m	m→au	ly→m	m→ly	pc→m	m→pc	units of length
	mi.→m	m→mi.	nmi.→m	m→nmi.	ft.→m	m→ft.	
	in.→m	m→in.			yd.→m	m→yd.	
	fathom → m	m → fathom	point → m	m → 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	
	sinc	W_m	W_p	W^{-1}	β(x,y)	γ_{xy}	
	Γ_{xy}	ζ(x)	$(-1)^x$	→DP	→SP	||	
Orthog	H_n	L_m	$L_{m\alpha}$	P_n	T_n	U_n	orthogonal polynomials
	H_{np}						
αINTL	Ā ā	Á á	Ă ă	À à	Ä ä	Ã ã	[α] *catalog* of international letters.[34] All letters except 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. You will reach each and every letter by 3 keystrokes maximum since letters can be accessed alphabetically (cf. pp. 117f).
	Â â	Å å	Æ æ	Ą ą	Ć ć	Č č	
	Ç ç	Đ đ	Ď ď	Ē ē	É é	Ě ě	
	È è	Ë ë	Ê ê	Ė ė	Ĕ ĕ	Ę ę	
	Ğ ğ	Ī ī	Í í	Ĭ ĭ	Ì ì	Ï ï	
	Î î	Į į	İ i	I ı	Ł ł	Ĺ ĺ	
	Ľ ľ	Ń ń	Ň ň	Ñ ñ	Ō ō	Ó ó	
	Ŏ ŏ	Ò ò	Ö ö	Õ õ	Ô ô	Ø ø	
	Œ œ	Ř ř	Ŕ ŕ	ß	Ś ś	Š š	
	Ş ş	Ť ť	Ţ ţ	Ū ū	Ú ú	Ŭ ŭ	
	Ù ù	Ü ü	Ű ű	Û û	Ů ů	Ų ų	
	Ŵ ŵ	Ŷ ŷ	Ý ý	Ÿ ÿ	Ź ź	Ž ž	
	Ż ż						

[34] See https://de.wikipedia.org/wiki/Liste_lateinischer_Alphabete#Erweiterungen .

Menu	^	^	^	^	^	^	Remarks
αMATH	<	≤	=	≈	≥	>	[α] for comparison symbols, parentheses & brackets, as well as more mathematical and related symbols. You can reach every character by 3 keystrokes maximum.
	{	[	(	)	]	}	
	× / · [35]	÷	∫	$_\infty$	$^\infty$	∞	
	¬	∧	∨	≠	|	&	
	∡	∟	⊥	∛	√	$\sqrt[x]{}$	
	$\bar{x}$	$\bar{y}$	$\hat{x}$	$\hat{y}$	$\bar{x}$	$\bar{y}$	
	:=	≙	≘	E	ℂ	ℝ	
	⊙	⊙	⊕				
	±	^	T	−1	ℏ		
α.FN	x→α	αRL	αRR	αSL	αPOS?	α→x	dedicated functions for *alpha strings*, plus font browser
					αLENG?		
	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. [36] 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	

---

[35] With startup default settings, the multiplication dot is found here and the multiplication cross is called via f × in *AIM*. If MULT· is set, however, this dot is called via f × in *AIM* and the multiplication cross via αMATH.

[36] The Greek alphabet (sic!) goes alpha, beta, gamma, delta, e-psilon, zeta, eta, theta, iota, kappa, lambda, my, ny, xi, o-micron, pi, rho, sigma, tau, y-psilon, phi, chi, psi, o-mega. Note ancient Greek H, Θ, and Y are pronounced like Finnish ÄÄ, T, and Y – modern Greek Θ like English Th, H and Y both like Finnish I. Finnish Y is pronounced like German Ü or French U. Think of Nils Holgersson's goose Yksi (coming first before Kaksi, Kolme, Neljä, Viisi, and Kuusi for obvious reasons – there is no goose named Seitsemän in that novel).

Menu	^	^	^	^	^	^	Remarks
α•	!	:	;	'	"	@	[α] for punctuation marks, currency symbols, arrows, and further special characters.
	¡	¿	§	_	~	\	
	$	€	%	&	£	¥	
	←	↑	↕	↓	→	⤒	
	«	»	ø	🕓	•	*	
	⌛	❗	U	🔋		*	

Menu	^	^	^	^	^	^	Remarks
∡→	→DEG	→RAD	→GRAD		→D.MS	→MULπ	angular conversions, cf. pp. 142f.
	DEG→	RAD→	GRAD→		D.MS→	MULπ→	
	D→R	R→D		D→D.MS	D.MS→D		

Constants

Your *WP 43S* contains a *catalog* of 80 physical, astronomical, and mathematical constants sorted alphabetically:

G	G_0	G_C	g_e	$GM_\oplus$	$g_\oplus$
c_1	c_2	e	e_E	F_α	F_δ
1/2	a	a_0	a_M	$a_\oplus$	c

Names of astronomical and mathematical constants are printed on colored background in the table starting overleaf. Values of physical constants (including their relative standard deviations in red print below) are printed on light background if they are exactly defined or almost exactly known – the darker the background, the less precisely the particular value is known.[37]

[37] For most of the physical constants, their precise numeric values (incl. their units) and their relative standard deviations (*SD*) are from *CODATA 2018*, copied in May 2019. These are the best values known in the scientific community, agreed on by the national standards institutes worldwide (e.g. by NIST and PTB). Note that all of them feature less than 16 significant digits.

Here are the contents of CNST (printing commas as radix marks for better visibility and multiplication dots for space reasons). Formulas are printed if applicable.

Name	Numeric value and rel. *SD*	Remarks
1/2 (0)[38]	0,5	Trivial but helpful constant for some iterations.
a	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*.

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

By the way, the terms *resolution*, *precision*, and *accuracy* are confused frequently in measuring. In a nutshell, *resolution* is the least significant digit a measuring instrument indicates. Using this instrument for measuring the same object under identical conditions multiple times, you get an idea about its *repeatability* (or *precision*); this can be no better than its *resolution* but may be significantly worse – a factor of ten or more may be observed easily in real life. *Accuracy* of a measuring instrument, however, can never be better than its *repeatability*.

Since you cannot know anything about a real-life object or process any better than you can measure it, these considerations are of fundamental importance. We recommend watching them – in your very own interest.

[38] The counting numbers in parentheses are to support determination of parameters for CONST – see the *IOI*.

Name	Numeric value and rel. *SD*	Remarks
c (5)	$2{,}997\ 924\ 58 \cdot 10^{8}\ {}^{\mathrm{m}}/_{\mathrm{s}}$ *(exact)*	**Speed of light in vacuum** $\approx 300\ 000 \frac{km}{s} = 300 \frac{km}{ms} = 300 \frac{m}{\mu s} = 30 \frac{cm}{ns} = 0{,}3 \frac{mm}{ps}$ etc.
c_1	$3{,}741\ 771\ 852\ldots \cdot 10^{-16}\ \mathrm{W\,m^2}$ *(exact)*	First radiation constant $c_1 = 2\pi\, h\, c^2$
c_2	$0{,}014\ 387\ 768\ 77\ldots\ \mathrm{m \cdot K}$ *(exact)*	Second radiation constant $c_2 = {}^{hc}/_{k}$
e	$1{,}602\ 176\ 634 \cdot 10^{-19}\ \mathrm{A\,s}$ *(exact)*	**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	$96\ 485{,}332\ 12\ldots\ {}^{\mathrm{A\,s}}/_{\mathrm{mol}}$ *(exact)*	*Faraday* constant $F = e\, N_A$
F_α	2,502 907 875 095 892 8…	*Feigenbaum's* α and δ
F_δ	4,669 201 609 102 990 6…	
G	$6{,}674\ 30 \cdot 10^{-11}\ {}^{\mathrm{m^3}}/_{\mathrm{kg\,s^2}}$ *$(2{,}2 \cdot 10^{-5})$*	*Newtonian* constant of gravitation; also known as γ from other authors. See $GM_\oplus$ below for a more precise value.
G_0 (15)	$7{,}748\ 091\ 729\ldots \cdot 10^{-5}\ /\Omega$ *(exact)*	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	−2,002 319 304 362 56 *$(1{,}7 \cdot 10^{-13})$*	*Landé's* electron g-factor

Name	Numeric value and rel. *SD*	Remarks
$GM_{\oplus}$	$3{,}986\,004\,418 \cdot 10^{14}\ \mathrm{m^3/s^2}$ *(2,0·10⁻⁹)*	*Newtonian* constant of gravitation times the Earth's mass with its atmosphere included (according to *WGS84* [39])
$g_{\oplus}$	$9{,}806\,65\ m/s^2$ *(per def.)*	Standard earth acceleration
h (20)	$6{,}626\,070\,15 \cdot 10^{-34}\ \mathrm{J\,s}$ *(exact)*	***Planck constant***
ħ	$1{,}054\,571\,817\ldots \cdot 10^{-34}\ \mathrm{J\,s}$ *(exact)*	Reduced *Planck* constant $\hbar = h/2\pi$
k	$1{,}380\,649 \cdot 10^{-23}\ \mathrm{J/K}$ *(exact)*	***Boltzmann constant*** $k = R/N_A$
K_J	$4{,}835\,978\,484\ldots \cdot 10^{14}\ \mathrm{Hz/V}$ *(exact)*	*Josephson* constant $K_j = 2e/h$
l_{PL}	$1{,}616\,255 \cdot 10^{-35}\ \mathrm{m}$ *($1{,}1 \cdot 10^{-5}$)*	*Planck* length $l_{PL} = t_{PL} c$
m_e (25)	$9{,}109\,383\,701\,5 \cdot 10^{-31}\ \mathrm{kg}$ *($3{,}0 \cdot 10^{-10}$)*	Electron mass $\triangleq 511{,}00\ \mathrm{keV}$
M_{Moon}	$7{,}349 \cdot 10^{22}\ \mathrm{kg}$ *($5 \cdot 10^{-4}$)*	Mass of the Moon
m_n	$1{,}674\,927\,498\,04 \cdot 10^{-27}\ \mathrm{kg}$ *($5{,}7 \cdot 10^{-10}$)*	Neutron mass $\triangleq 939{,}57\ \mathrm{MeV}$
m_p	$1{,}672\,621\,923\,69 \cdot 10^{-27}\ \mathrm{kg}$ *($3{,}1 \cdot 10^{-10}$)*	Proton mass $\triangleq 938{,}27\ \mathrm{MeV}$
m_{PL} (30)	$2{,}176\,435 \cdot 10^{-8}\ \mathrm{kg}$ *($1{,}1 \cdot 10^{-5}$)*	*Planck* mass $m_{PL} = \sqrt{\hbar c / G} \approx 22\ \mu\mathrm{g}$

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

Name	Numeric value and rel. *SD*	Remarks
m_p/m_e	1 836,152 673 43 *(6,0·10^{-11})*	Proton to electron mass ratio
m_u	1,660 539 066 60·10^{-27} kg *(3,0·10^{-10})*	Atomic mass constant $= 10^{-3}\ ^{\mathrm{kg}}/_{N_A}$
$m_u c^2$	1,492 418 085 60·10^{-10} J *(3,0·10^{-10})*	Energy equivalent of the atomic mass constant $\approx 931{,}49\ \mathrm{MeV}$
m_μ	1,883 531 627·10^{-28} kg *(2,2·10^{-8})*	Muon mass $\triangleq 105{,}66\ \mathrm{MeV}$
$M_\odot$ (35)	1,989 1·10^{30} kg *(5·10^{-5})*	Mass of the Sun
$M_\oplus$	5,973 6·10^{24} kg *(5·10^{-5})*	Mass of the Earth. See **GM$_\oplus$** above for a more precise value.
N_A	6,022 140 76·10^{23} / mol *(exact)*	***Avogadro's* number**
NaN	Not a number	See the corresponding entry in *Section 5 of the OM.*
p_0	101 325 Pa *(per def.)*	Standard atmospheric pressure
R (40)	8,314 462 618… $^{\mathrm{J}}/_{\mathrm{mol\ K}}$ *(exact)*	Molar gas constant
r_e	2,817 940 326 2·10^{-15} m *(4,5·10^{-10})*	Classical electron radius $r_e = \alpha^2\, a_0$
R_K	25 812,807 45… Ω *(exact)*	*Von Klitzing* constant $R_K = {}^{h}/_{e^2}$
R_{Moon}	1,737 530·10^{6} m *(5·10^{-7})*	Mean radius of the Moon

Name	Numeric value and rel. *SD*	Remarks
R∞	$10\,973\,731{,}568\,160\ /\mathrm{m}$ *($1{,}9 \cdot 10^{-12}$)*	*Rydberg* constant $R_\infty = \frac{\alpha^2 m_e c}{2h}$
R⊙ (45)	$6{,}96 \cdot 10^{8}\ \mathrm{m}$ *($5 \cdot 10^{-3}$)*	Mean radius of the sun
R⊕	$6{,}371\,010 \cdot 10^{6}\ \mathrm{m}$ *($5 \cdot 10^{-7}$)*	Mean radius of the Earth
Sa	$6{,}378\,137\,0 \cdot 10^{6}\ \mathrm{m}$ *(per definition)*	Semi-major axis … according to *WGS84* (see footnote 39 on p. 130)
Sb	$6{,}356\,752\,314\,2 \cdot 10^{6}\ \mathrm{m}$ *($1{,}6 \cdot 10^{-11}$)*	Semi-minor axis
Se²	$6{,}694\,379\,990\,14 \cdot 10^{-3}$ *($1{,}5 \cdot 10^{-12}$)*	First eccentricity squared
Se'² (50)	$6{,}739\,496\,742\,28 \cdot 10^{-3}$ *($1{,}5 \cdot 10^{-12}$)*	Second eccentricity squared
Sf⁻¹	298,257 223 563 *(per def.)*	Flattening parameter
T₀	$273{,}15\ \mathrm{K}$ *(per definition)*	= 0°C, standard temperature
Tp	$1{,}416\,785 \cdot 10^{32}\ \mathrm{K}$ *($1{,}1 \cdot 10^{-5}$)*	*Planck* temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_P c^2}{k} = \frac{E_P}{k}$
tPL	$5{,}391\,245 \cdot 10^{-44}\ \mathrm{s}$ *($1{,}1 \cdot 10^{-5}$)*	*Planck* time $t_{PL} = {}^{l_{PL}}/_{c}$
Vm (55)	$0{,}022\,413\,969\,54\ldots\ {}^{\mathrm{m}^3}/_{\mathrm{mol}}$ *(exact)*	Molar volume of an ideal gas at standard conditions $V_m = \frac{RT_0}{p_0} \approx 22{,}4\ {}^{l}/_{\mathrm{mol}}$

Name	Numeric value and rel. *SD*	Remarks
α	7,297 352 569 3·10^{-3} *($1{,}5 \cdot 10^{-10}$)*	Fine-structure constant $\alpha = \frac{e^2}{2\varepsilon_0\, h\, c} \approx \frac{1}{137}$
γ	6,674 30·10^{-11} $\mathrm{m^3/_{kg\,s^2}}$ *($2{,}2 \cdot 10^{-5}$)*	*Newtonian* constant of gravitation; also known as G from other authors. See $GM_\oplus$ below for a more precise value.
γ_{EM}	0,577 215 664 901 532 9…	*Euler-Mascheroni* constant
γ_p (60)	2,675 221 874 4·10^{8} $\mathrm{Hz/_T}$ *($4{,}2 \cdot 10^{-10}$)*	Proton gyromagnetic ratio $\gamma_p = 4\pi\, {}^{\mu_p}/_h$
ε_0	8,854 187 812 8·10^{-12} $\frac{\mathrm{A\,s}}{\mathrm{V\,m}}$ *($1{,}5 \cdot 10^{-10}$)*	Vacuum electric permittivity $\varepsilon_0 = {}^1/_{\mu_0 c^2}$ (note the so-called *Coulomb's* constant is just ${}^1/_{4\pi\varepsilon_0} = {}^{\mu_0 c^2}/_{4\pi} = c^2\, 10^{-7}\, \frac{\mathrm{V\,s}}{\mathrm{A\,m}} \approx 8{,}987\,55 \cdot 10^9\, {}^{\mathrm{V\,m}}/_{\mathrm{A\,s}}$)
λ_C	2,426 310 238 67·10^{-12} m *($3{,}0 \cdot 10^{-10}$)*	*Compton* wavelengths of the electron $\lambda_C = {}^h/_{m_e c}$, neutron $\lambda_{Cn} = {}^h/_{m_n c}$, and proton $\lambda_{Cp} = {}^h/_{m_p c}$, respectively
λ_{Cn}	1,319 590 905 81·10^{-15} m *($5{,}7 \cdot 10^{-10}$)*	
λ_{Cp}	1,321 409 855 39·10^{-15} m *($3{,}1 \cdot 10^{-10}$)*	
μ_0 (65)	1,256 637 062 12·10^{-6} $\frac{\mathrm{V\,s}}{\mathrm{A\,m}}$ *($1{,}5 \cdot 10^{-10}$)*	Vacuum magnetic permeability
μ_B	9,274 010 078 3·10^{-24} $\mathrm{J/_T}$ *($3{,}0 \cdot 10^{-10}$)*	*Bohr* magneton $\mu_B = {}^{e\hbar}/_{2m_e}$

Name	Numeric value and rel. *SD*	Remarks
μe	$-9{,}284\,764\,704\,3\cdot10^{-24}$ J/T *(3,0·10⁻¹⁰)*	Electron magnetic moment
μe/μB	$-1{,}001\,159\,652\,181\,28$ *($1{,}7\cdot10^{-13}$)*	Ratio of electron magnetic moment to *Bohr's* magneton
μn	$-9{,}662\,365\,1\cdot10^{-27}$ J/T *($2{,}4\cdot10^{-7}$)*	Neutron magnetic moment
μp (70)	$1{,}410\,606\,797\,36\cdot10^{-26}$ J/T *($4{,}2\cdot10^{-10}$)*	Proton magnetic moment
μu	$5{,}050\,783\,746\,1\cdot10^{-27}$ J/T *($3{,}1\cdot10^{-10}$)*	Nuclear magneton $\mu_u = {e\hbar}/{2m_p}$
μμ	$-4{,}490\,448\,30\cdot10^{-26}$ J/T *($2{,}2\cdot10^{-8}$)*	Muon magnetic moment
σB	$5{,}670\,374\,419\ldots\cdot10^{-8}\ \frac{\mathrm{W}}{\mathrm{m^2K^4}}$ *(exact)*	*Stefan-Boltzmann* constant $\sigma_B = \frac{2\pi^5 k^4}{15h^3c^2}$
Φ	$1{,}618\,033\,988\,749\,894\,8\ldots$	Golden ratio $\Phi = \frac{1}{2}(1+\sqrt{5})$
Φ0 (75)	$2{,}067\,833\,848\ldots\cdot10^{-15}$ V s *(exact)*	Magnetic flux quantum $\Phi_0 = \frac{h}{2e}$
ω	$7{,}292\,115\cdot10^{-5}$ rad/s *($2\cdot10^{-8}$)*	Angular velocity of the Earth according to *WGS84* (see footnote 39 on p. 130)
-∞	$-\infty$	Note both these 'constants' are counted as numeric values in your *WP 43S*.
∞	∞	
xxx	XXX	Auxiliary numeric constants (cf. *WP 34S*)

Some orders of magnitude found in nature are not stored in CNST. Although they are known with one or two digits precision only they may be helpful nevertheless:

Radius of an atomic nucleus	$\sim 10^{-15}\ \mathrm{m}$
Radius of an atom [40]	$\sim 10^{-10}\ \mathrm{m}$
Radius of the observable universe	$\approx 45 \times 10^{9}\ l.y. \approx 4.3 \times 10^{26}\ \mathrm{m}$
Amount of stars in observable universe	$\sim 10^{23}$
Amount of atoms in the Sun	$\sim 10^{57}$
Amount of atoms in observable universe	$\sim 10^{80}$
Baryonic mass in observable universe	$\sim 10^{53}\ \mathrm{kg}$

Note these quantities are all well within the range of *SP* numbers on your *WP 43S* (see *App. B*). Precision of physical constants is seldom better known than twelve digits. Please take these facts into account when assessing small differences as well as talking about very small or very large numbers.

[40] 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. Think about it!

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

Unit Conversions

Your *WP 43S* features 14 angular conversions provided in ∡→ (see p. 127) and 88 unit conversions in U→. The structure of U→ follows various branches as explained in the *OM*, *Section 5*. Its top *view* looks like this:

with E: standing for the *submenu* of energy unit conversions, P: for power, F&p: for force and pressure, m: for mass, x: for length, A: for area, and V: for volume. See pp. 124f for further details of the structure.

Conversions contained in U→ either begin or end in basic *SI* units. Beyond them and products or powers of these, knowledge of the following *SI derived units* carrying special names may be helpful in your further calculations and communication:

Quantity	**Unit**	**Symbol and formula**
Temperature	*degree Celsius*	$\vartheta[°\mathrm{C}] = \mathrm{T}[\mathrm{K}] - 273.15$
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*	$1\ \mathrm{W} = 1\ \mathrm{V\ A\ s} = 1\ ^{\mathrm{J}}/_{\mathrm{s}}$
Electric potential	*volt*	$1\ \mathrm{V} = 1\ ^{\mathrm{W}}/_{\mathrm{A}}$
Charge	*coulomb*	$1\ \mathrm{C} = 1\ \mathrm{A\ s}$

Quantity	Unit	Symbol and formula
Capacitance	*farad*	$1\ \mathrm{F} = 1\ {}^{\mathrm{C}}/_{\mathrm{V}} = 1\ {}^{\mathrm{A\,s}}/_{\mathrm{V}}$
Conductance	*siemens*	$1\ \mathrm{S} = 1\ {}^{\mathrm{A}}/_{\mathrm{V}}$
Resistance	*ohm*	$1\ \Omega = 1\ {}^{\mathrm{V}}/_{\mathrm{A}}$
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*	$1\ \mathrm{Gy} = 1\ {}^{\mathrm{J}}/_{\mathrm{kg}}$

For talking about inputs and results, knowing the symbols and names of the *SI* prefixes as listed here is beneficial:

Prefix symbol	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 symbol	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 conversions featured in U→ and ∡→ are explained in alphabetical order in the following two tables. Therein, numbers are either exact (printed on white background) or generally rounded to six significant digits; they are for your orientation only (your *WP 43S* uses more precise values). And commas are used as radix marks for better visibility.

Softkey	**Calculation**	**Remarks**	**Branch**
°C→°F	× 1,8 + 32		U→
°F→°C	– 32) / 1,8		
acre → m²	× 4 046,86	These *acres* are based on the '*international feet*', see below	U→ f A:
acre_{US} → m²	× 4 046,87	These *acres* are based on the *U.S. survey feet*, see below	
atm→Pa	× 1,013 25×10⁵ → $\times 1{,}013\,25\times10^{5}$	*Atmospheres*	U→ F&p:
au→m	$\times 1{,}495\,98\times10^{11}$	*Astronomic units*	U→ x:
barrel → m³	× 0,158 987	*Barrel (U.S.)* of oil, abbr. *bbl*	U→ f V:
bar→Pa	$\times 10^{5}$	1 mbar = 1 hPa	U→ F&p:
Btu→J	× 1 055,06	*British thermal units*	U→ E:
cal→J	× 4,186 8	*Calories*	
carat → kg	× 0,000 2		U→ m:
cwt→kg	× 50,802 4	(*Long*) *hundredweight* = 112 lbs	
dB → field ratio	$10^{R_{dB}/20}$	*Decibels*	U→ ▼
dB → power ratio	$10^{R_{dB}/10}$		
fathom → m	× 1,828 8	1 *fathom* := 6 *feet*	U→ x:
ft.→m	× 0,304 8	The '*international feet*' of 1959 – 1 *foot* := $^1/_3$ *yard*	
field ratio→dB	$20\,lg\left({}^{a_1}/_{a_2}\right)$	Also known as amplitude ratio	U→ ▼

Softkey	Calculation	Remarks	Branch
floz$_{UK}$ → m^3	× 2,841 31×10^{-5}	*Fluid ounces*	U→ f V:
floz$_{US}$ → m^3	× 2,957 35×10^{-5}		
gl$_{UK}$→m^3	× 4,546 09×10^{-3}	*Gallons*	
gl$_{US}$→m^3	× 3,785 42×10^{-3}		
hp$_E$→W	× 746	*Electric horsepower*	U→ P:
hp$_M$→W	× 735,498 8	*'Metric' horsepower* (equivalent to PS in German)	
hp$_{UK}$→W	× 745,699 9	*British Imperial horsepower*	
in.→m	× 0,025 4	1 *inch* := $^1/_{12}$ *foot* and 1 *inch* := 1 000 *mil*	U→ x:
in.Hg → Pa	× 3 386,39	*Inches of mercury*	U→ F&p:
J→Btu	/ 1 055,06	*Joule*	U→ E:
J→cal	/ 4,186 8		
J→Wh	/ 3 600		
kg → carat	/ 0,000 2	1 t [*(metric) ton*] = 1 000 kg	U→ m:
kg→cwt	/ 50,802 4		
kg→oz	/ 0,028 349 5		
kg→lb.	/ 0,453 592		
kg → sh.cwt	/ 45,359 2		
kg → short ton	/ 907,185		
kg → stone	/ 6,350 29		
kg→ton	/ 1 016,05		
kg → tr.oz	/ 0,031 103 5		
lbf→N	× 4,448 22	*Pounds force*	U→ F&p:
lb.→kg	× 0,453 592	*Pounds;* 1 lb := 16 *ounces*	U→ m:

Softkey	Calculation	Remarks	Branch
ly→m	× 9,460 73×10^15^	*Light years*	U→ x:
m² → acre	/ 4 046,86	*Square meter*; 1 a [*are*] = 100 m², 1 ha [*hectare*] = 10 000 m², 1 km² = 100 ha = 1 000 000 m²	U→ f A:
m² → acre$_{US}$	/ 4 046,87		
m³ → barrel	/ 0,158 987	*Cubic meter*; 1 l [*liter*] = 10⁻³ m³, 1 ml [*milliliter*] = 1 cm³	U→ f V:
m³ → floz$_{UK}$	/ 2,841 31×10⁻⁵		
m³ → floz$_{US}$	/ 2,957 35×10⁻⁵		
m³→gl$_{UK}$	/ 4,546 09×10⁻³		
m³→gl$_{US}$	/ 3,785 42×10⁻³		
m³→qt.	/ 1,1365×10⁻³		
mi.→m	× 1 609,344	1 *mile* = 1 760 *yards*	U→ x:
m→au	/ 1,495 98×10^11^	*Meter*	
m → fathom	/ 1,828 8		
m→ft.	/ 0,304 8		
m→in.	/ 0,025 4		
m→ly	/ 9,460 73×10^15^		
m→mi.	/ 1 609,344		
m→nmi.	/ 1 852		
m→pc	/ 3,085 68×10^16^		
m → point	/ 352,778×10⁻⁶		
m → survey foot$_{US}$	/ 0,304 801		
m→yd.	/ 0,914 4		
nmi.→m	× 1 852	*Nautical miles*	

Softkey	Calculation	Remarks	Branch
N→lbf	/ 4,448 22	*Newton*	U→ F&p:
oz→kg	× 0,028 349 5	1 *ounce* := $^{1}/_{16}$ *pound*	U→ m:
Pa→atm	/ 1,013 25×10^{5}	*Pascal*; 1 mbar = 1 hPa	U→ F&p:
Pa→bar	/ 10^{5}		
Pa → in.Hg	/ 3 386,39		
Pa→psi	/ 6 894,76		
Pa → torr	/ 133,322		
pc→m	× 3,085 68×10^{16}	*Parsec*	U→ x:
point → m	× 352,778×10^{-6}	*(Typographical) point*	
power ratio →dB	$10\ lg(^{p_1}/_{p_2})$		U→ ▼
psi→Pa	× 6 894,76	*Pounds per square inch*	U→ F&p:
qt.→m^{3}	× 1,1365×10^{-3}	*(Imperial) quart*	U→ f V:
short cwt → kg	× 45,359 2	1 *short hundredweight* = 100 lbs	U→ m:
short ton → kg	× 907,185	1 *short ton* = 2 000 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→
ton→kg	× 1 016,05	1 *Imperial ton* = 200 *(long) cwt* = 2 240 lbs	U→ m:
torr → Pa	× 133.322	1 *torr* = 1 mm Hg	U→ F&p:
tr.oz → kg	× 0,031 103 5	*Troy ounces*	U→ m:
Wh→J	× 3 600	*Watt-hours*	U→ E:

Softkey	**Calculation**	**Remarks**	**Branch**
W→hp$_E$	/ 746	*Watt*	U→ P:
W→hp$_M$	/ 735,498 8		
W→hp$_{UK}$	/ 745,699 9		
yd.→m	× 0,914 4	1 *yard* := 3 *feet* and 2 *yards* := 1 *fathom*	U→ x:
year→s	× 31 556 952	= 365,242 5 × 24 × 60^2	U→

L→ …	**Remarks**
DEG→	Takes an integer or real [41] x as an angular input in *decimal* or *sexagesimal degrees*, respectively, and converts it to the current *ADM*.
D.MS→	
D.MS→D	Takes an integer or real [41] x as an angular input in *sexagesimal degrees* (formatted `ddddd.mmsshh`) and converts it to an *angle* in *decimal degrees* (corresponding to the old command H.MS→H).
D→R	Takes an integer or real [41] x as an angular input in *decimal degrees* and converts it to … … *radians*.[42]
D→D.MS	Takes an integer or real [41] x as an angular input in *decimal degrees* and converts it to … … *sexagesimal degrees* (corresponding to the old command H→H.MS).

[41] If x is neither integer nor real (i.e. neither *data type* 1, 2, nor 11), error 24 will be thrown. Conversions of integer values to *angles* in *sexagesimal degrees* do not make real sense but are allowed nevertheless.

[42] Note that angles given in *radians* cannot represent full circles (or simple fractions of π like π/2, π/4, π/5, π/8, π/10, etc.) exactly but with an accuracy of 16 digits "only". If you want to avoid rounding errors caused by that, *multiples of* **π** may be a better choice here. This applies especially to large numeric inputs in trigonometric functions since those will be reduced to values between −π and +π before calculating (as mentioned in *Section 2* of the *OM*).

L→ …	Remarks
GRAD→	Takes an integer or real[41] x as an angular input in … *… grad/gon …* … and converts it to the current *ADM*.
MULπ→	Takes an integer or real[41] x as an angular input in … *… multiples of π …* … and converts it to the current *ADM*.
RAD→	Takes an integer or real[41] x as an angular input in … *… radians*[42] … … and converts it to the current *ADM*.
R→D	Takes an integer or real[41] x as an angular input in *radians* and converts it to *decimal degrees* (equaling the old command R→D).[42]
→DEG →D.MS →GRAD →MULπ →RAD	If x is an integer or real (*data type* 1, 2, or 11), takes it as an angular input in the current *ADM* and converts it to *decimal degrees*, *sexagesimal degrees*, *grad/gon*, *multiples of π*, or *radians*, respectively.[42] If x is a tagged real (*data type* 4), on the other hand, this information is used in conversion (e.g. if $x = \mathbf{1.5}\,\boldsymbol{\pi}$ then →GRAD will return $\mathbf{300}^{g}$ regardless of current *ADM*). If x is neither of *data type* 1, 2, 4, nor 11, error 24 will be thrown.

Angular output is tagged always.

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 have already learned how to call *items* appearing on the keyboard and in *menus* (including *catalogs*). In *Section 6* of the *OM*, you have seen that you can store *items* in user *menus* and/or assign them to specific locations on your *WP 43S*. In the following you will learn about one more way you can use for calling and executing operations:

- Using XEQ followed by the *name* of the operation typed in *AIM*.

Furthermore, we will summarize the functions requiring parameters.

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* thereafter; 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. it will be checked for required parameters (cf. overleaf);
      i. if true, you will be prompted for these parameters; then the function will be executed. End.
      ii. else the function will be executed. End.
   b. else error 7 (No such function) will be thrown (see *App. C*). End.

Operations Requiring Trailing Parameters

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

Operations requiring one trailing parameter

Operation	**Numeric param.**	**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	# of decimals	
ASR MASKL MASKR RL RLC RR RRC SL SR WSIZE	Number of bits	
BACK CASE SKIP	# 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	*Flag* number	
CONST	Constant number	
DSTACK	# of *stack registers*	
ERR MSG	Error number	
f'(x) f''(x) GTO LBL LBL? PGMINT PGMSLV XEQ Π_n Σ_n		Program label
GTO.	# of program step	Label

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

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

XYZ → ST.X and it will take its parameter from **X** like a good old *RPN* command instead.

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

Operation requiring four trailing parameters	**First to fourth 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 *data type* (*DT*) of the lowest *stack register(s)* regardless of specific input values, as mentioned at various locations in the *OM*. These operations are collected in the list here:

Input *DT*	**Operation(s)**	**Output *DT***	**Output *registers* involved**
1	1/x $\sqrt[3]{x}$ AGM ALL cos ENG erf erfc e^x FIX f' f" gd gd^{-1} IMPFRC $J_y(x)$ LN LNβ LNΓ LOG_{10} LOG_2 LOG_xy MANT POISS… PROFRC 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 [43]	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

[43] The functions printed on yellow background will return *long integers* (*data type 1*) wherever possible.

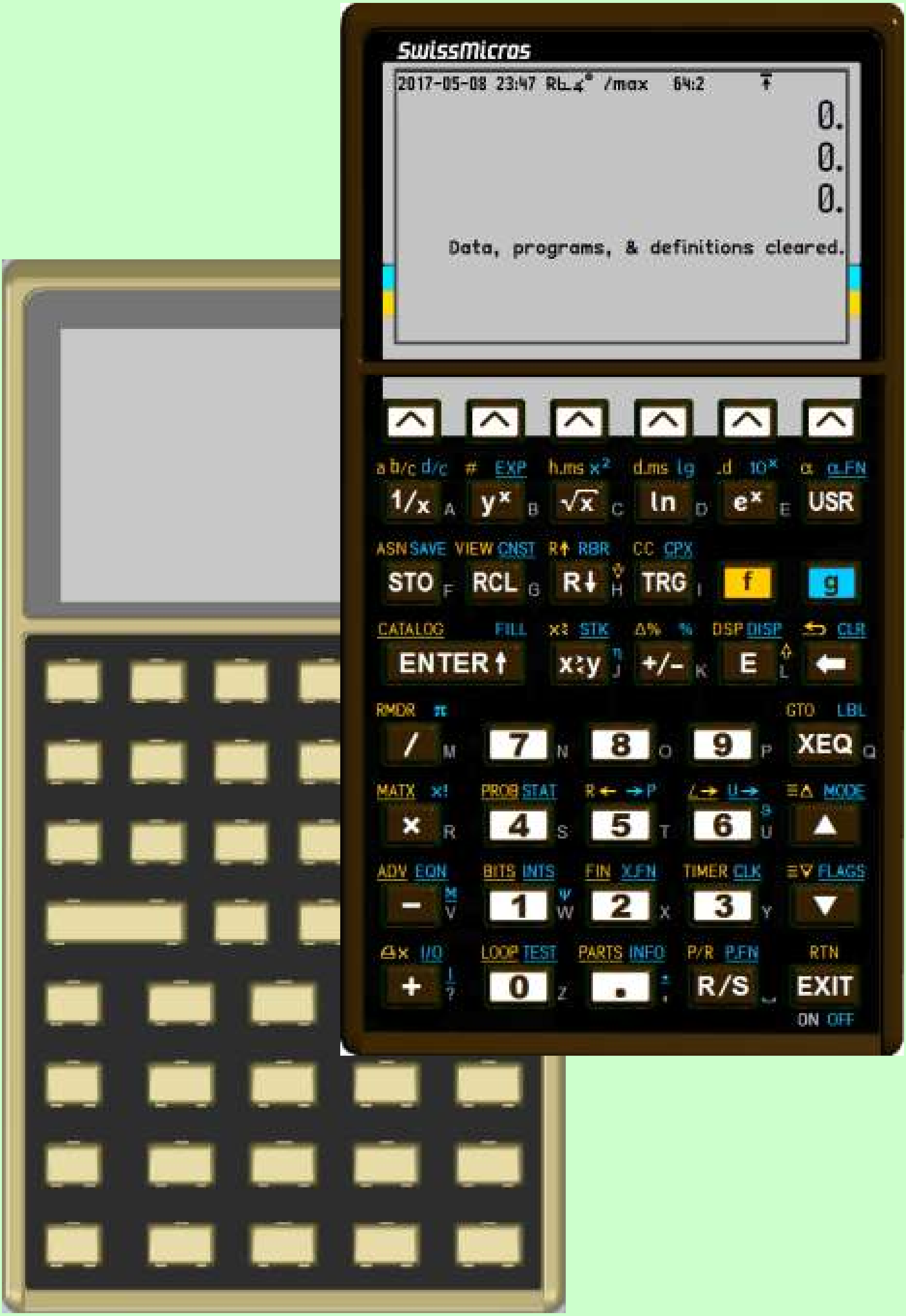
Input *DT*	Operation(s)	Output *DT*	Output *registers* involved
2	AND CEIL DATE→ DAY D→J EXPT FLOOR IDIV IDIVR IP MONTH NAND NEXTP NOR NOT OR ROUNDI SIGN WDAY XNOR XOR YEAR ±∞?	1	**X**
	DECOMP	1	**X, Y**
	SLVQ	2 or 3	**X, Y, Z**
	RE→CX	3	**X**
	arccos arcsin arctan and all angular conversions	4	**X**
	→H.MS	5	**X**
	x→DATE →DATE	6	**X**
	x→α	7	**X**
	M.GET M.NEW	8 or 9	**X**
	→INT	10	**X**
	→DP	11	**X**
3	ABS CROSS IM RE |x| ∡	2	**X**
	CX→RE	2	**X, Y**
	SLVQ	2 or 3	**X, Y, Z**
	→DP	12	**X**
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**

Input *DT*	Operation(s)	Output *DT*	Output *registers* involved
8	M.DIM?	1	**X, Y**
	DET DOT ENORM |M|	2	**X**
	Σ+	2	**X, Y**, statistics
	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_xy →REAL	2	**X**
	x→α	7	**X**
11	AND CEIL DATE→ DAY D→J EXPT FLOOR IDIV IDIVR MONTH NEXTP NAND NOR NOT OR ROUNDI SIGN WDAY XNOR XOR YEAR ±∞?	1	**X**
	DECOMP	1	**X, Y**
	→SP	2	**X**
	arccos arcsin arctan	4	**X**
	→H.MS	5	**X**
	x→α	7	**X**
	→INT	10	**X**
	RE→CX	12	**X**
12	arccos arcsin arctan →SP	3	**X**
	CROSS |x| ∡	11	**X**

APPENDIX A: HARDWARE

Overall dimensions:	wedge-shaped: 77 mm × 144 mm × 13 mm or 8 mm (see p. 151)
Mass with battery:	~100 g
LCD dimensions:	about 58.8 mm × 35.3 mm visible area, 400 × 240 quadratic pixels monochrome
Processor:	STMicroelectronics STM32L476 incl. RTC (see https://www.st.com/en/evaluation-tools/32l476gdiscovery.html for the development board used by *SwissMicros*) running at 25 MHz on battery power or 80 MHz connected to USB (see below).
Memory:	1 MB *FM*, 128 kB *RAM* (see *App. B* on pp. 155ff), 8 MB additional *FM* on a QSPI chip; user *RAM* is some 75 kB, user *FM* is 6 MB.
Power supply:	3 V by one *CR2032* coin cell; alternative power supply through USB port; typical average currents drawn for power on and busy: 4.2 mA; idle: 0.1 mA; power off: 3 µA.
Buzzer frequency:	≥ 1 Hz up to > 20 kHz in steps of 1 Hz.
I/O:	infrared printer port, standard micro-USB port.
Self-test:	initiated by xxx
Keyboard overlays:	Three short slots on either side of the keyboard are provided in the calculator case for easy fixing overlay sheets with your personal layouts printed on them. See *App. F* for more (pp. 199f).

Seven pictures of the hardware are displayed here and on the three 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→ MODE
× 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 left and below. These operations are at your own risk.

This picture shows the *PCB* of an early *DM42* of spring 2017.

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

APPENDIX B: MEMORY MANAGEMENT

Data Types

There are twelve *data types* 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 ± 2 147 483 648 (i.e. 32 *bits*),
- 39-digit reals for internal calculations not exposed to the user,
- *flag* words for storing 128 (i.e. 112 global plus 16 local) user *flags* and the same amount of system *flags*,[44]
- two *data types* for two kinds of *menus*,
- a *data type* of variable length for storing *configurations* (modes and user assignments, see STOCFG and RCLCFG),
- another one for *expressions* in EQN (see *Section 4* of the *OM*),
- two more for program steps and routines (see *Section 3* of the *OM*).

A 4-*byte* header is specified for each object of each *data type*:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">pointer to the data</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td colspan="5">pointer to the variable name</td><td>0</td><td>0</td><td colspan="5">data information[45]</td><td colspan="4">data type (as specified below) − 1</td></tr>
</table>

[44] Up to 256 flags would be possible.

[45] E.g. base of a *short integer*, angular unit for an *angle*.

	Data type **number and meaning**	**Size** [*bytes*]
1	ℤ *Long integer*[46]	≥ 4 + 2 + 4 = 14…1030
2	ℝ Real number[47] (*real16*)	4 + 8 = 12
3	ℂ Complex number (*complex16*)	4 + 2 × 8 = 20
4	*Angle*[48] (*angle16*)	4 + 8 = 12
5	*Time*[49]	4 + 8 = 12
6	*Date*[50]	4 + 4 = 8
7	*Alpha string* (each character requires 16 *bits*)[51]	4 + 2 + ***n*** × 2
8	Real matrix (featuring ***n*** rows and ***m*** columns)	4 + ***n*** × ***m*** × 8
9	Complex matrix (featuring ***n*** rows and ***m*** columns)	4 + ***n*** × ***m*** × 16
10	*Short integer*[52]	≤ 4 + 8 = 8 or 12
11	Double precision real number (*real34*)	4 + 2 × 8 = 20
12	Double precision complex number (*complex34*)	4 + 2 × 16 = 36
13	Double precision *angle* (*angle34*)	4 + 2 × 8 = 20
14	Double precision mathematical constant[53]	4 + 0 = 4

[46] This *data type* is for number theory kind of problems. 2 *bytes* are for the size (in *bits*) of the integer following. 4 *bytes* allow for signed integers up to $2^{31} \approx 2 \times 10^{9}$, 8 *bytes* for $2^{63} \approx 9 \times 10^{18}$. Size is increased in steps of 4 *bytes* if required (look up the display limits further below). Maximum integer size is 3328 *bits* (= 416 *bytes*) equivalent to 1002 decimal digits.

[47] As in the *WP 34S*, standard *SP* reals feature 64 *bits* and 16 digits precision. See the chapter after next.

[48] A tagged *angle* is stored as a real number, just with a specific header.

[49] A *time* or time interval is stored as a real number of *seconds* internally, just with a specific header. A day corresponds to 86 400 s, a year to 31 556 952 s.

[50] A *date* is stored as its Julian day number internally. Four *bytes* do for $>10^{7}$ years.

[51] 2 *bytes* are for the size (in *bytes*) of the string following, including the trailing zero. The size must be even.

[52] This *data type* is for computer science problems. Most probably, such a storage space will be either 4 + 4 or 4 + 8 *bytes* long.

[53] The other constants are far within *SP*. A pointer is sufficient in any case.

	Data type **number and meaning**	**Size** [*bytes*]
15	Extended precision real (39 digits, exclusively for internal use)	4 + 32 = 36
16	*Label* (each character requires 16 *bits*)	4 + 7 × 2 = 18
17	System integer (for internal use only)	4 + 4 = 8
18	System and user *flags* (128 *flags* each)	4 + 2 = 6
19	User-created *menu* (limited to 1 *view*)	4 + 18 × 7 × 2 = 256
20	Predefined *menu* (featuring ***n*** *views*)	4 + ***n*** × 18 × 14
21	*Configuration* (as stored by STOCFG)[54]	4 + M
22	Program step, may be stored as *alpha string* (7)[55]	4 + M
23	Program, containing ***n*** program steps	4 + M
24	*Expression* (for members of EQN), may be stored as *alpha string* (7)	4 + M
25	*Directory* (proposed by P. 2012-12)	4 + M
26		

Data types 7 - 9 and 20ff are of 'infinite' size limited by available memory (M) only. Individual size of each object is fixed though.

As mentioned above, any object of any *data type* will take one storage space only: one *register* or one variable. In consequence, *register* lengths in your *WP 43S* may vary considerably. You do not have to bother – the operating system of your *WP 43S* will take care of all the necessary administration. Thus, the amount of *RAM* required for data storage is not fixed. Data and programs allocate their memory from the same large pool.

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

[55] The size will vary depending on parameters. Exact limits and methods are not decided yet.

Statistical Summation Registers

Your *WP 43S* features a block of 23 special *registers* for storing statistical sums (like the *WP 34S* and *WP 31S* had 14 before). These statistical *registers* neither overlap nor interfere with any general purpose *registers* unlike they did on *HP's* pocket calculators. The contents of these *registers* can be recalled using their names; please see pp. 59 and 93f.

And like on our 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 the memory is returned to the pool by [CLΣ].

Range of Standard (*SP*) Real Numbers

Your *WP 43S* can calculate with reals of more than 750 orders of magnitude. Floating point numbers within $10^{-383} \leq |x| < 10^{+385}$ may be entered directly easily.

Within this range, your *WP 43S* calculates with 16 digits precision (thus, it can display up to 16 digits in *startup default* display format). Results should be accurate within $\pm 1\times10^{-15}$ (e.g. ***n*** [1/x] ***2n*** [×] returns a plain 2 for all primes < 500 – just [7] [1/x] [1][4] [×] returns $2 + 1\times10^{-15}$; the same results for 79 and 89, and for 97 a $2 - 1\times10^{-15}$ is returned). Results $|x| < 10^{-398}$ are set to zero. For results $|x| \geq 10^{+385}$, error 4 or 5 will appear unless *flag* **D** is set (see *App. C*).

All these effects are caused by the **internal representation of reals**: *Standard floating point numbers* are stored in eight bytes using an internal format as follows:

- Real zero is stored as integer zero, i.e. all bits cleared.
- The mantissa of a real number (also known as *significand* in this context) is encoded in five groups of three digits. Each such group is packed into 10 bits <u>straight forward</u>, meaning e.g. 555_{10} = 10 0010 1100_2 or 999_{10} = 11 1110 0111_2 = $3E7_{16}$. So the 15 rightmost decimal digits of the *significand* take the least

significant 50 bits. Trailing zeroes are omitted, so the *significand* will be right adjusted.

- The most significant (64^{th}) bit takes the sign of the mantissa.
- The remaining 13 bits are used for the exponent and the leftmost digit of the mantissa. Of those 13, the lowest 8 are reserved for the exponent. For the top 5 bits it becomes complicated.[56] If they read
  - `00ttt`, `01ttt`, or `10ttt` then `ttt` takes the leftmost digit of the *significand* ($0 – 7_{10}$), 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 become the leftmost digit of the *significand*. If `uu` reads 00, 01, or 10 then these two will be the most significant bits of the exponent. If `uu` reads 11 instead, there are codes left for encoding special numbers (e.g. infinities).

In total, we get 16 digits for the mantissa and a bit less than 10 bits for the exponent: its maximum is $10\ 1111\ 1111_2$ (i.e. 767_{10}). For reasons becoming obvious below, 398 must be subtracted from the value in this field to get the true exponent of the number represented. The 16 digits of the *significand* allow for a range from 1 to almost 10^{16}.

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:

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 14 bits in binary representation	Stored exponent
1.	`22 38 00 00 00 00 00 01`		`0010 0010 0011 10`	398
−1.	`A2 38 00 00 00 00 00 01`		`1010 0010 0011 10`	398
111.	`22 38 00 00 00 00 00 6F`	`06F`	`0010 0010 0011 10`	398

[56] Don't blame us – this part follows the standard IEEE 754.

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 14 bits in binary re-presentation	Stored expo-nent
111.111	22 2C 00 00 00 01 bC 6F	06F 06F	0010 0010 0010 11	395
−123.000123	A2 20 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0010 00	392
9.99×10^{99}	23 bC 00 00 00 00 03 E7	3E7	0010 0011 1011 11	495
1×10^{-99}	20 AC 00 00 00 00 00 01		0010 0000 1010 11	299
1×10^{-383}	00 3C 00 00 00 00 00 01		0000 0000 0011 11	15
$0.000\,000\,000\,01 \times 10^{-383}$	00 04 00 00 00 00 00 01		0000 0000 0001 00	4

This last number is the smallest that can be entered directly from the keyboard. Dividing it by 10^4 results in 1×10^{-398}, being stored as hexadecimal 1. Divide this by 1.999 999 999 99 and the result will remain 1×10^{-398} 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 high end of our numeric range now:

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 14 bits	Stored exp.
$9.999\,999\,999\,99 \times 10^{384}$	77 FF E7 F9 FE 7F 78 00	9 3E7 3E7 3E7 3dE 000	0111 0111 1111 11	767

This number (featuring 12 times the digit 9) is the maximum which can be keyed in directly. Adding 9.999×10^{372} to it will display 1×10^{385} …

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 14 bits	Stored exp.
1×10^{385}	77 FF E7 F9 FE 7F 9F E7	9 3E7 3E7 3E7 3E7 3E7	0111 0111 1111 11	767

… being stored as $9.999\,999\,999\,999\,999 \times 10^{384}$. This is the greatest number representable in this format. Thus, the greatest *significand* possible is 9 999 999 999 999 999 $= 10^{16} - 1$;

All this follows *decimal64* floating point format, though not exactly. Additionally, your *WP 43S* features three 'special reals':

Floating point 'number'	Hexadecimal value stored	Top byte in binary representation
$+\infty$	78 00 00 00 00 00 00 00	0111 1000
$-\infty$	F8 00 00 00 00 00 00 00	1111 1000
NaN	7C 00 00 00 00 00 00 00	0111 1100

An exponent is not applicable here. **These 'three special reals' may be legal results of your *WP 43S* <u>if flag D is set</u>** – no error will be thrown then. 'Not a number' (*NaN*) covers outputs exceeding real domain or being undefined (like poles) – see the corresponding entry in *Section 5* of the *OM* and examples in next chapter. Note that **+∞ and −∞ may be also legal inputs while *NaN* is not.**

Calculations with Double Precision (*DP*) Real Numbers

Your *WP 43S* uses *single precision* (*data type* 2) in real number calculations per default, wherein 16 digit precision is reached in all calculations. Additionally, you may use *double precision* reals (*data type* 11), allowing for 34 digits instead of 16 (see below).

☞ Matrix commands will not work with *DP* numbers.

☞ *DP* allows for more precise calculations. While some computations will reach high accuracy, **we do not warrant 34 digit precision in all calculations with *DP* reals.** [57]

DP reals are stored coarsely following *decimal128* packed coding, though with some exceptions. The lowest 110 bits take the rightmost 33 digits of the *significand*. Going left, a 12 bit exponent field follows, then

[57] Not all functions are expanded to *DP*, some stay in *single precision* or merely a little bit more.

The *WP 43S* software is based on the *decNumber library* supporting arbitrary precision *BCD* numbers. As mentioned at some places in the *IOI*, internal computations are carried out with 39 digits. Actually this is the minimum; some modulo calculations are performed with a few hundreds of digits to avoid cancellation (e.g. 2π features 451 digits for proper reduction to the standard range for trigonometric functions).

More elaborate algorithms are coded as *DP* keystroke programs to save flash space (for the cost of execution speed and the loss of a few digits of accuracy in *DP* mode). The internal formats used for storing numbers in your *WP 43S* (as shown just above for *single precision* and below for *DP*) need to be converted back and forth from and to the *decNumber* format. This is a lot of overhead and doesn't come for free in terms of execution speed.

There is a quasi standard to find out about processors and test accuracy of calculators to some extent – compute $arcsin\{arccos[arctan(tan\{cos[sin(9°)]\})]\}$. An ideal calculator with an infinite internal precision would return exactly 9 without cheating. Real calculators (all computing with a finite number of digits) deviate for obvious reasons. Your *WP 43s* returns

- 9,000 000 000 029 361 = $9 + 3 \cdot 10^{-11}$ for an *SP* argument and
- 8,999 999 999 999 999 999 999 999 999 937 535 = $9 - 6{,}246\,5 \cdot 10^{-29}$ for a *DP* argument.

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

Another simple test discussed in the internet: Enter 1,000 000 1 and then execute x^2 just 27 times. Your *WP 43s* will return

- 674 530,470 *539 687 4* for an *SP* argument and
- 674 530,470 741 084 559 382 689 1*84 727 772 2* for a *DP* argument.

The latter is the most precise result known of a pocket calculator so far (*WP 34S* and *Free42* concur, computing with 34 digits as well). Nevertheless, only the first 25 digits of this *DP* output are correct! Calculating with unlimited precision instead returns here 674 530,470 741 084 559 382 689 178 029 746 812 844 4 for the first 40 of the 10^9 digits of the complete result.

Please take this information into account when assessing small deviations or many decimals returned by your *WP 34S*.

5 bits used and coded exactly as in *SP*, and finally the sign bit. The maximum absolute value of the stored exponent is $10\ 1111\ 1111\ 1111_2 = 12\ 287_{10}$. For reasons analogous to those explained on pp. 158ff, 6176 must be subtracted from this value to get the true exponent of the floating point number represented. Thus, *data type* 11 could support 34-digit numbers within $10^{-6143} \leq |x| < 10^{+6145}$ (keyboard input is limited to $10^{-999} \leq |x| < 10^{+999}$, cf. p. 135 for reasons). Coding works in full analogy to the way described for *SP* in previous chapter.

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, stored as hexadecimal 1.

Divide this by 1.999 999 999 999 999 999 999 999 999 999 999 and the result will remain 10^{-6176}. Divide it by 2 instead and the result will become zero.

Full 34-digit precision of a *DP* number in **X** may be displayed by SHOW as a *temporary information* in small font (see p. 73). Remember not every such number may be true to 34 digits – cf. p. 162. And errors accumulate as explained in footnote 37 on pp. 127f.

Returning to *SP* (via →SP) with input exceeding the *SP* number range explained on pp. 158ff will cause 0 or ±∞ (or an overflow) being displayed instead.

As mentioned above, some calculations are executed in "*internal high precision*" even for *SP* arguments. "*Internal high precision*" means even more digits than *DP* – it may go up to some hundred digits in special cases.

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

Special Results

Within this chapter, flag **D** is presumed to be set. Thus, infinities and non-numeric results are legal – no error message will be thrown if such results happen to occur. **NaN** covers poles as well as regions where a function result is not defined at all.

The following monadic functions, if called with REARE set (i.e. with ℝ lit and *flag* I cleared), return either ±∞ or NaN under the conditions stated below:

Input x	Operation(s)	Output [58]
0.	1/x	∞
0 or 0.	ln, lg, lb x	−∞
$-\infty \leq x < 0$		NaN
0 or 0.	Γ(x)	NaN
$\lvert \mathrm{Re}(x) \rvert > 1$	arccos, arcsin	NaN
$\mathrm{Re}(x) < 1$	arcosh	NaN
$\mathrm{Re}(x) \geq 1$	artanh	NaN
90° or equivalents in other *ADM*	tan	−∞ (*WP 34S* returns NaN at this pole)
−∞	arctan	-90.° or equivalents
∞		90.° or equivalents
−∞	e^x, 10^x, 2^x	0.
−∞ or ∞	1/x, sinc	0.
−∞ or ∞	x^2	∞
∞	tanh	1.
∞	ln, e^x, $\sqrt{x}$, lg, 10^x, lb x, sinh, cosh	∞
−∞ or ∞	cos, sin, tan, arcosh, arsinh, artanh	NaN

58 In this chapter, results were crosschecked against the *WP 34S* wherever possible. Deviations are highlighted. Red results concur with the *WP 34S* but are considered wrong nevertheless.

And this is the respective table for <u>dyadic</u> functions:

Input y	x	Operation	Output
∞	arbitrary $x \neq -\infty$	+	∞ [59]
−∞	arbitrary $x \neq \infty$		−∞ [59]
−∞	∞	+	NaN [59]
∞	arbitrary $x \neq \infty$	−	∞ [60]
−∞	arbitrary $x \neq -\infty$		−∞ [60]
−∞	−∞	−	NaN
∞	∞		
∞	arbitrary $x > 0$	×	∞ [59]
−∞	arbitrary $x < 0$		
∞	arbitrary $x < 0$	×	−∞ [59]
−∞	arbitrary $x > 0$		
0 or 0.	−∞ or ∞	×	NaN [59]
$0 < y \leq \infty$	0.	/	∞
$-\infty \leq y < 0$			−∞
−∞ or ∞	−∞ or ∞	/	NaN
0 or 0.	0.	/, y^x	NaN
$-\infty < y < 0$	non-integer x	y^x	NaN
−∞ or ∞	0. or 0		
−∞	odd $x > 0$	y^x	−∞
−∞	even $x > 0$		∞
∞	arbitrary $x > 0$	y^x	∞
arbitrary $y \neq 0$	−∞	y^x	0.
	∞		∞
0.	$0 < x < \infty$	$\log_x y$	−∞

[59] Swapping x and y will return the same result here.

[60] Swapping x and y will return the result times -1.

The functions printed on light yellow background in the two tables above will return NaN also with complex results allowed. Others will change their output when ℂ is lit (i.e. *flag* **I** set). Some particular returns of elementary transient functions operating near ±∞ are listed here: [61]

Input Re(x)	Im(x)	r(x)	φ(x)	Op.	Output
−∞	—	—	—		∞∡90° = 0.+i×∞
	0	∞	180°	√x	NaN+i×∞ (*WP 34S* returns 0.+i×∞)
-10^{999}		10^{999}			→ ∞∡90° = 0.+i×∞
0.	10^{999}	10^{999}	90°	x²	→ ∞∡180° = −∞+i×0.
	∞	∞			−∞+i×NaN (see remark 2 below)
−∞	—	—	—		−∞
	0	∞	180°	∛x	∞∡45° = ∞+i×∞ (*34S*: NaN+i×NaN)
-10^{999}		10^{999}			$1.\times10^{333}$∡60° = $5.\times10^{332}$ + i×8.660 254 037 8×10^{332}
		10^{333}	60°	x³	$1.\times10^{999}$∡−180° = $-1.\times10^{999}$ + i×0.
−∞	—	—	—		−∞
	0	∞	180°	x³	NaN+i×NaN (see remark 3 below)
-10^{999}		10^{999}			$-1.\times10^{2997}$+i×0. $\to -\infty + i \times 0$
∞	—	—	—		∞
	0	∞	0°	ln	∞+i×∞ (*WP 34S* returns ∞+i×0.)
10^{999}		10^{999}			2 300.282 507 9+i×0. $\to \infty + i \times 0$
-10^{999}	0	10^{999}	180°	ln	2 300.282+i×3.141 592 $\to \infty + i\pi$
−∞		∞			∞+i×∞ (*WP 34S* $= \infty + i\pi$)
	—	—	—		NaN

[61] Following an article about the *HP-71*, infinities should be treated in polar notation (see *http://hparchive.com/Journals/HPJ-1984-07.pdf*, p. 27, left column for the reasons).

Input Re(x)	Im(x)	r(x)	φ(x)	Op.	Output
∞	∞	∞	45°	ln	∞+i×∞
∞	−∞	∞	−45°	ln	∞−i×∞
−∞	∞	−∞	135°	ln	∞+i×∞
−∞	−∞	−∞	−135°	ln	∞−i×∞
0.	10^{999}	0.	0°	ln	$\rightarrow \infty + i\,{}^{\pi}/_{2}$
	∞				∞+i×∞
0.	−∞				∞−i×∞
0.	0.				NaN+i×NaN
0.	—	—	—		−∞
0.	∞	∞	90°	e^x	NaN+i×NaN (see remark 2 below)
	10^{999}	10^{999}			1.0 ∡ −45° (*WP 34S*: NaN+i×NaN) =0.707 106 781 186… − i×0.707 1… $= \frac{1}{2}(\sqrt{2} - i\sqrt{2})$
0.	−∞	∞	−90°	e^x	NaN+i×NaN (see remark 2 below)
	-10^{999}	10^{999}			1.0 ∡ 45° (*WP 34S*: NaN+i×NaN) =0.707 106 781 186… + i×0.707 1… $= \frac{1}{2}(\sqrt{2} + i\sqrt{2})$
−∞	0	∞	180°	e^x	0.+i×0.
-10^{999}	10^{-999}	10^{999}			0.+i×0. (see remark 2 below)
−∞		∞			NaN+i×NaN (see remark 2 below)
−∞	∞	−∞	135°	e^x	NaN+i×NaN
−∞	−∞	−∞	−135°	e^x	NaN+i×NaN

Computation of lg and lb x is derived from ln. The same applies for e^x, 10^x, and 2^x.

1. Note that $f^{-1}(f(\boldsymbol{x})) = \boldsymbol{x}$ may not hold in such special cases since ±∞ is not a usual number (so inversions may include operations with non-numeric results).
2. $\lim_{x\to\infty} f(x) \approx f(10^{999})$ may deviate significantly from $f(x=\infty)$; the same applies to $\lim_{x\to 0} f(x) \approx f(10^{-999})$ and $f(0)$ as well as to $\lim_{x\to -\infty} f(x) \approx f(-10^{999})$ and $f(x=-\infty)$.
3. And although $x \equiv x + i \times 0$, there may be $f(x) \neq f(x + i \times 0)$ above (this may be a bug though).

At the bottom line, we hope confusion is limited (and I recommend keeping off ±∞ in complex plane).

Program Step Size

Program step size is assumed to be 4 *bytes* typically. But compare *data type* 22 on p. 157.

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, RBR, s, STATUS, 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, the following messages will be displayed. They are listed below in alphabetical order (*EC* means *error code*):

	EC	Explanations, countermeasures and examples
An argument exceeds the function domain	1	{1, 2, 3, 4, 10, 11, 12} 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 or *flag* **I** is set), by 0^0, $x/0$, $0/0$, $\Gamma(0)$, $\tan(\pm 90°)$ and equivalents, by $\operatorname{artanh}(x)$ for $\lvert \operatorname{Re}(x) \rvert \ge 1$, by $\operatorname{arcosh}(x)$ for $\operatorname{Re}(x) < 1$, etc.[62]
Bad time or date input	2	{2, 5, 6, 11} Invalid date format or incorrect *date* or *time* in input, e.g. month > 12, day > 31. Will be thrown as soon as the input is closed.
Cannot delete a predefined item	27	Self-explanatory.
Distribution parameter out of valid range	16	{1, 2, 11} A parameter specified in **I**, **J**, or **K** is out of valid range for the distribution function called (e.g. if LGNRM is called with $j < 0$).
Flash memory is full	23	Delete a program from *FM* to regain space.

[62] Note that e.g. tan(90°) and logs of 0 are legal operations on {1, 2, 3, 11, 12} if *flag* **D** is set. See the end of this appendix.

	EC	Explanations, countermeasures and examples
Flash memory is write protected	19	There was an attempt to edit or delete program steps in *FM*. See PRCL and PSTO to circumvent.
Function to be coded for that data type	30	Functions may not be coded yet during FW development.
Illegal digit in integer input for this base	9	{10} E.g. 2 in binary or 9 in octal input. Will be thrown as soon as the respective base is entered (i.e. as soon as input is closed).
Illegal input data type for this operation	24	… called. Convert what is necessary. Cf. "*operation is undefined in this mode*".
Input is too long	10	Keyboard input is too long for the buffer. (This error is not used currently. Only alpha input is limited presently.)
Invalid or corrupted data	18	Set when there is a checksum error either in *FM* or as part of a serial download. Also set 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 root found	20	{2, 11} The *Solver* did not converge.
No such function	7	Returned when calling a nonexistent function via XEQ α … ENTER↑ (check for typos!) or running a routine containing a nonprogrammable command.

	EC	Explanations, countermeasures and examples
`No such label found`	6	Attempt to address an undefined label.
`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, 11, 12} • A number exceeds the valid range. This can be caused by specifying decimals > 16, *word size* > 64, negative *flag* numbers, short integers $\geq 2^{64}$, *hours* or *degrees* $> 9\,000$, invalid *dates* or *times*, denominators $\geq 9\,999$, 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 (e.g. R-COPY) attempts accessing invalid *register* addresses.
`Overflow at +∞`	4	{1, 2, 3, 8, 9, 11, 12} unless *flag* **D** 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, 11, 12} unless *flag* **D** is set • Division of a number < 0 by 0. • Divergent sum or product or integral. • Negative overflow (see p. 158). • Logarithm of 0 (note a logarithm of -0 returns NaN).
`Please enter a NEW name`	26	Trying to define a new variable or user *menu* with a *name* already in use.

	EC	Explanations, countermeasures and examples
RAM is full	11	May be caused by attempts to write too large routines, allocate too many variables, and the like (see pp. 155ff for the space required by different *data types*). May happen also in program execution due to dynamic allocations (see *Sect. 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 e.g. SSIZE8 and STOS 93 .
Too few data points for this statistic	15	{2} A statistical calculation was attempted with too few data, e.g. *regression* or *standard deviation* for less than 2 points.
Undefined op-code	3	An instruction with an undefined operation code occurred. Should never happen – but who knows?
Word size is too small	14	{10} Input or *register* content is too great to be handled by the *word* size currently set.
	25	Left unused for *WP 34S* compatibility

If *flag* **D** 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 CNST on pp. 127ff and the tables on pp. 161ff). 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
HEWLETT PACKARD
42S
RPN SCIENTIFIC
x: 812.6000
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 in a way. The first table shows the functions of the *HP-42S* and the corresponding ones of your *WP 43S* unless they carry identical names and are either both keyboard accessible or both stored in a *catalog* or *menu*. There is an analog table for *HP-16C* functions starting on p. 180, one for the *HP-21S* on p. 182, and another one for the *WP 34S* on p. 183. Functions newly introduced with *WP* calculators are compiled on pp. 187ff.

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 I/O
AIP	Dispensable	You can merge text and numeric data easily 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 general purpose *registers*.
ALPHA	α	See the description of *AIM* in *Sect. 2* of the *OM*.

HP-42S	*WP 43S*	**Remarks**
AOFF	αOFF	In α.FN
AON	αON	In α.FN
ARCL	Dispensable	Any *register* or variable can take an *alpha 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 an *alpha string*. Simply press STO instead.
ATANH	artanh	In EXP
ATOX	α→x	In α.FN
AVIEW	Dispensable	Any *register* or variable can take an *alpha 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 I/O
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
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	M.OLD	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 I/O

HP-42S	*WP 43S*	**Remarks**
PRGM	P/R	
PRINT	I/O	🖶x is on the keyboard.
PRLCD	🖶LCD	In I/O
PROFF	CF T	
PROMPT	Dispensable	Use VIEW , STOP instead.
PRON	SF T	
PRP	🖶PROG	In I/O
PRSTK	🖶STK	In I/O
PRUSR	🖶USER	In I/O
PRV	🖶r	In I/O
PRX	🖶r X	Press 🖶x.
PRΣ	🖶Σ	In I/O
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 general purpose *registers* always.
SLOPE	L.R.	In STAT
SOLVE	SLV	In ADV
SQRT	√x	
SST	≡▼ (▼)	Shortcut works if no *multi-view menu* is open.
STR?	STRI?	In TEST

HP-42S	*WP 43S*	**Remarks**
TOP.FCN	Dispensable	Obsolete since no top functions are overwritten.
TRACE	SF T	
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 general purpose *registers* always. Statistical registers are separate.
ΣREG?		
→DEC	→INT 10	Press # 1 0 ... 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 their appearance on its keyboard, starting top left. As for the *HP-42S*, only functions carrying different names on both calculators are listed.

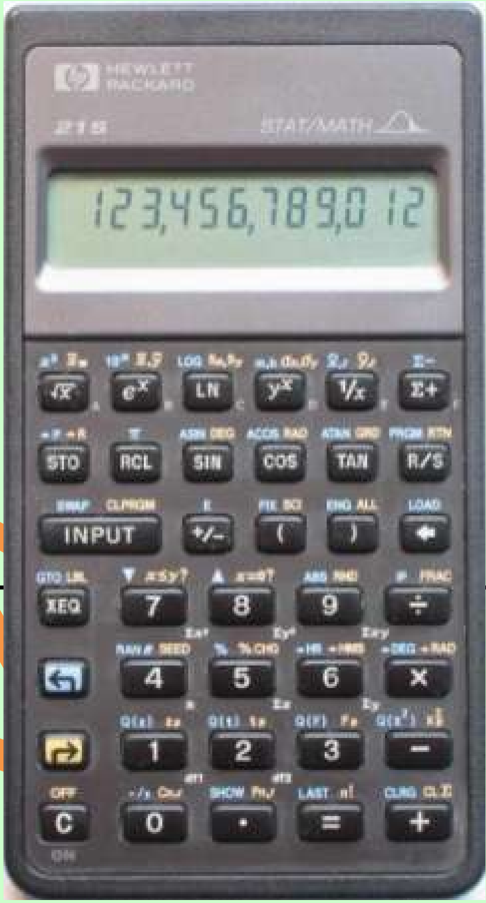
HP-16C	*WP 43S*	Remarks
RL , RLn	RL	In BITS
RR , RRn	RR	In BITS
RLC , RLCn	RLC	In BITS
RRC , RRCn	RRC	In BITS
÷	/	(see also *ISO 80000-2*: "The symbol ÷ should not be used.")
DBL÷	DBL/	In INTS (see also *ISO 80000-2*: "The symbol ÷ should not be used.")
x≷(i)	Superfluous	Any *register* may be used for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

HP-16C	*WP 43S*	**Remarks**
(i)	Dispensable	Any *register* may be used for indirection.
I		
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> ?	
x>0		
FLOAT	FIX	In DISP
MEM	STATUS	In FLAGS
CHS	+/_	
<, >	Dispensable	64 *bits* can be displayed in one row.
LSTX	RCL L	
x≠y	x≠ ?	In TEST. Note far more tests are covered here.
x≠0		
x=y	x= ?	
x=0		

Corresponding Operations on *HP-21S*

The table for the functions of *HP-21S* (starting overleaf) follows the same rules as the one for *HP-16C*. The *HP-21S*, however, is 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
$\bar{x}_w$	$\bar{x}_w$	In STAT
$\bar{x},\bar{y}$	$\bar{x}$	
s_x,s_y	s	
m,b	L.R.	
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
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.
ABS	|x|	In PARTS
RND	ROUND	

HP-21S	*WP 43S*	**Remarks**
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 ($\mu=0$, $\sigma=1$). And the implementation of 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)	$t_e(x)$	
t_p	$t^{-1}(p)$	
Q(F)	$F_e(x)$	
F_p	$F^{-1}(p)$	
$Q(\chi^2)$	$\chi^2{}_e(x)$	
$\chi^2{}_p$	$(\chi^2)^{-1}$	
Cn,r	COMB	In PROB
Pn,r	PERM	
LAST	RCL L	
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. Thus, your *WP 43S* allows for *softkeys* – the *WP 34S* can only carry four *hotkeys* instead. Also dealing with matrices is greatly eased by the large high resolution dot matrix display of your *WP 43S*; thus some elementary matrix commands of the *WP 34S* are not required anymore on your *WP 43S*.

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

WP 34S	***WP 43S***	**Remarks**
ANGLE	∡	
Binom_u	Binom_e	
Cauch_u	Cauch_e	
CLα	**0** STO K	Check the *OM* for the conditions when this register is used.
CONST	CNST	For keyboard space reasons.
DBLOFF DBLON	Dispensable	Your *WP 43S* features *DP data types* – it does neither need nor feature a *DP* mode. Use **→DP** to convert individual data to *DP*; use **→SP** to reconvert *DP* data to *SP*.
Expon_u	Expon_e	
$F_u(x)$	$F_e(x)$	
dRCL	Dispensable	Your *WP 43S* features various *data types*.
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 the *HP-42S* (AGRAPH, CLLCD, PIXEL) plus POINT.
Geom_u	Geom_e	
GTOα	Dispensable	Use GTO with an appropriate parameter instead.
H.MS+, H.MS−	Dispensable	Your *WP 43S* features a dedicated *data type* for *times*, so + and − suffice for adding or subtracting sexagesimal times, respectively.
INTM?	Dispensable	Your *WP 43S* features dedicated *data types* for integers – it does neither need nor feature an integer mode.

WP 34S	*WP 43S*	**Remarks**
iRCL	Dispensable	Your *WP 43S* features various *data types*.
I_x	I_{xyz}	This is a triadic function after all.
$Lgnrm_u$	$LgNrm_e$	
L_n	L_m	Renamed to avoid search conflict with LN.
$L_{n\alpha}$	$L_{m\alpha}$	Renamed in consequence to L_m.
$Logis_u$	$Logis_e$	
MROW+×, MROW×	Dispensable	Obsolete matrix commands.
MROW⇆	M.R⇆R	
M+×	Dispensable	Obsolete matrix command.
M^{-1}	$[M]^{-1}$	
M-ALL, M-COL, M-DIAG, M-ROW	Dispensable	Obsolete matrix commands.
M×	Dispensable	Your *WP 43S* features two dedicated *data types* for matrices. Thus you can simply multiply two matrices using $\boxed{\times}$ 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.
$Norml_u$	$Norml_e$	
$Poiss_u$	$Poiss_e$	
REALM?	Dispensable	Your *WP 43S* features a dedicated *data type* for reals – it does not need a real mode.
REGS, REGS?	Dispensable	The number of global general purpose *registers* is fixed to 100 on your *WP 43S*.

WP 34S	***WP 43S***	**Remarks**
SENDA, SENDP, SENDR, SENDΣ	SEND	SEND combines all those four commands of the *WP 34S*.
SEPOFF, SEPON	GAP	
SHOW	RBR	
sRCL	Dispensable	Your *WP 43S* features various *data types*.
TRANSP	$[M]^T$	
TSOFF	GAP 0	
TSON	GAP 3	
$t_u(x)$	$t_e(x)$	
VIEWα, VWα+	Dispensable	Simply use VIEW instead; *alpha strings* are just another *data type*. You can combine text and numeric data easily using + as shown in *Sect. 2* of the *OM.*
$Weibl_u$	$Weibl_e$	
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.
αDATE, αDAY	Dispensable	You can combine text and numeric data easily using + as shown in *Section 2* of the *OM.*
αGTO	Dispensable	Use GTO with an appropriate parameter instead.
αIP, αMONTH	Dispensable	You can combine text and numeric data easily using + as shown in *Section 2* of the *OM.*
αRCL, αRC#	Dispensable	Your *WP 43S* features various *data types* and 'knows' which type is in the *register* specified. Appending *alpha strings* is done by +.
αSTO	Dispensable	Simply press STO instead (any *register* can take an *alpha string*).

WP 34S	*WP 43S*	**Remarks**
αTIME	Dispensable	See 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.
$\chi^2{}_u(x)$	$\chi^2{}_e(x)$	
→H	→HR	
🖶PLOT	n/a	See gCLR.
🖶$^C r_{XY}$	Dispensable	Use 🖶r instead. [🖶x] is on the keyboard.
🖶α, 🖶α+, 🖶+α	Dispensable	You can combine text and numeric data easily using [+] as shown in *Section 2* of the *OM*. Then use 🖶r. [🖶x] is on the keyboard.
🖶?	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 340 operations, not counting the unit conversions and constants provided; 55 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**	●	●
Binom Binom_p (of *Binomial distribution*)	●	●	new
B_n B_n^* CEIL FLOOR	●	—	new
Cauch Cauch_p Cauch_e Cauch^{-1}	●	●	new
CauchF GaussF HypF ParabF RootF	**new**	—	—
CLCVAR	**new**	—	—
CLFall CLK12 CLK24 CLPall CONJ CONVG? COV	●	—	new
CPXi CPXj CX→RE RE→CX	**new**	—	—
DATE TIME	●	—	(●)
DATE→ DAY MONTH YEAR →DATE	●	—	new
DBL?	**modified**	—	new
DEC DSL INC ISE	●	—	new
DECOMP	●	●	new
DEG→ D.MS→ GRAD→ RAD→	●	—	new
DROP	●	—	new
DROPy DSTACK	**new**	—	—
D→J J→D	●	—	new
EIGVAL EIGVEC	**new**	—	—
ENGOVR SCIOVR	●	—	new
ENTRY?	●	—	new
EQ.DEL EQ.EDI EQ.NEW	**new**	—	—

Command	*WP 43S*	*WP 31S*	*WP 34S*
erf erfc ERR MSG	●	—	new
EVEN? ODD?	●	—	new
Expon Expon_p Expon_e Expon^{-1}	●	●	new
EXPT MANT	●	—	new
FAST SLOW	●	—	new
FBR	**new**	—	—
FC?F FC?S FF FS?F FS?S	●	—	new
FIB	●	—	new
FILL	●	●	new
FLASH? FP?	●	—	new
$F_p(x)$ F(x) (of *F distribution*)	●	●	new
f' f''	**new**	—	—
f'(x) f''(x)	**extended**	—	new
GAP	**extended**	●	new
GCD LCM	●	●	new
g_d g_d^{-1}	●	—	new
Geom Geom_p Geom_e Geom^{-1}	●	—	new
H_n H_{nP} L_m $L_{m\alpha}$ P_n T_n U_n	●	—	new
Hyper Hyper_p Hyper_e Hyper^{-1}	**new**	—	—
IDIV	●	—	new
IDIVR IM RE	**new**	—	—
IMPFRC PROFRC	●	●	new
INT? I_{xyz} $I\Gamma_p$ $I\Gamma_q$	●	—	new
$J_y(x)$	**new**	—	—
J/G	**extended**	●	new
KEY? KTYP? LBL? LEAP?	●	—	new
LgNrm LgNrm_p LgNrm_e LgNrm^{-1}	●	—	new

Command	***WP 43S***	*WP 31S*	*WP 34S*
LNβ LNΓ LOADP LOADR LOADSS LOADΣ LocR LocR? LOG$_2$ LOG$_x$y	●	—	new
LOAD SAVE	●	●	new
Logis Logis$_p$ Logis$_e$ Logis^{-1}	●	●	new
max min MIRROR	●	—	new
MOD	●	●	new
MULT× MULT· MULπ MULπ→	**new**	—	—
M.LU M.SQR? NAND NaN? NEIGHB NOR	●	—	new
NBin NBin$_p$ NBin$_e$ NBin^{-1}	**new**	—	—
NEXTP PRIME?	●	●	new
Norml Norml$_p$ Norml$_e$ Norml^{-1}	●	●	new
nΣ (callable by name)	●	●	new
OrthoF PLOT POINT	**new**	—	—
PAUSE	●	—	extended
Poiss Poiss$_p$ Poiss$_e$ Poiss^{-1}	●	●	new
PopLR PRCL PSTO PUTK	●	—	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**	—	—

Command	***WP 43S***	*WP 31S*	*WP 34S*
SDL SDR SETCHN SETEUR SETIND SETJPN SETUK SETUSA	●	—	new
SETDAT SETTIM	●	—	(●)
SIGNMT sinc	●	—	new
SL SR	●	—	extended
SLVQ SMODE? SPEC?	●	—	new
s_m s_{mw} s_w	●	●	new
SSIZE4 SSIZE8 SSIZE?	●	●	new
STATUS	**extended**	—	extended
s_{xy}	●	—	new
TDISP	**new**	—	—
TICKS	●	—	new
TIMER	●	—	(●)
TOP? ULP?	●	—	new
$t_p(x)$ $t(x)$ (of *t distribution*)	●	●	new
t⇆ y⇆ z⇆ ⇆	●	—	new
(UNDO)	●	new	—
V∡	**new**	—	—
VERS? WDAY WHO?	●	●	new
Weibl Weibl$_p$ Weibl$_e$ Weibl^{-1}	●	●	new
W_m W_p W^{-1} WSIZE? $\bar{x}_G$ XNOR	●	—	new
x→DATE	**new**	—	—
x< ? x≤ ? x= ? x≠ ? x≥ ? x> ?	●	—	extended
x= +0? x= -0? x≈ ?	●	—	new
Y.MD	●	●	new
αLENG?	**extended**	—	●
αPOS?	**extended**	—	—
αRL αRR αSL αSR	●	—	extended
$\beta(x,y)$ Γ_{xy} γ_{xy} ε ε_m ε_p $\zeta(x)$ Π Σ σ_w	●	—	new

Command	*WP 43S*	*WP 31S*	*WP 34S*
Σ^1/x Σ^1/x^2 Σ^1/y Σ^1/y^2 $\Sigma\ln y/x$ $\Sigma x^2/y$ Σx^3 Σx^4 $\Sigma x/y$	**new**	—	—
$\Sigma\ln^2x$ $\Sigma\ln^2y$ $\Sigma\ln x$ $\Sigma\ln xy$ $\Sigma\ln y$ Σx Σx^2 Σx^2y $\Sigma x\ln y$ Σxy Σy $\Sigma y\ln x$ Σy^2 (callable by names)	●	●	new
$\chi^2{}_p(x)$ $\chi^2(x)$ (of *chi-square distribution*)	●	●	new
$(-1)^x$ ×MOD ^MOD	●	—	new
±∞?	**new**	—	—
→DEG →RAD	●	●	new
→DP →SP	**new**	—	—
→D.MS →MULπ	**new**	—	—
→GRAD	●	—	new
→INT →REAL	**new**	—	—
🖶ADV 🖶CHAR 🖶r 🖶REGS 🖶TAB 🖶# 🖶MODE	●	—	(new)
🖶WIDTH	**extended**	—	(new)
			●	●	new

The statements in parentheses in the rightmost column refer to the *WP 34S* with optional quartz and capacitors installed.

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. The manuals listed below are entirely contained in a document set distributed by the *Museum of HP Calculators* (see http://www.hpmuseum.org/cd/cddesc.htm).

Topic	Recommended literature
General calculation examples and applications	All vintage *HP* calculator manuals can be recommended
Root finding and numeric integration	*HP-34C OH and Programming Guide* *HP-15C Owner's Handbook* *HP-15C Advanced Functions Handbook* *HP-42S Programming Examples and Techniques*
Statistical distributions and their application	*HP-21S Owner's Manual*
Financial calculations	*HP-17BII+ User's Guide*
Manipulating *short integers*	*HP-16C Owner's Handbook*
Programming	*HP-42S Owner's Manual* [63] *HP-42S Programming Examples and Techniques* [64]

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 70 on p. 213 below as well as the last paragraph on p. 16 of the *OM*).

[63] If you want this manual only, you can download it for free at http://www.hp41.net/forum/fileshp41net/manuel-hp42s-us.pdf .

[64] If you just want this manual, you can download it for free at http://www.hp41.net/forum/fileshp41net/hp42s-programming-examples.pdf .

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

Under Windows, you can …

a) use **MSYS2 MinGW 64-bit**, a runtime environment for gcc. You get it here (download the latest version): https://sourceforge.net/projects/mingw-w64/files/External%20binary%20packages%20%28Win64%20hosted%29/MSYS%20%2832-bit%29/ . Install. Then start it. It opens a DOS window. Enter therein:

cd wp43s	for changing to the proper directory.
git pull	for pulling the changed files from gitlab repository.[65]
make	for building a new `wp43s.exe`.[66]
rm backup.bin	for starting with the simulator reset to default.
./wp43s	for starting the simulator.

The simulator window will open (looking like one of the pictures overleaf though larger).

b) alternatively do the following:

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

Open `README.md` and proceed as described therein.

Eventually run `wp43s.exe`.

The simulator window will open (looking like one of the pictures overleaf though larger).

---

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

[66] There may be files updated by **git pull** but no new build possible sometimes. Then **make** will throw a corresponding message. – There may be also other obstacles; then **make mrproper** will clean the field for a subsequent **make**.

(The pictures printed here show an earlier keyboard layout. The landscape window will open if screen resolution does not suffice for portrait display.)

Operate the simulator with the mouse. Digits, [.], [ENTER↑], [+], [−], [×], and [/] may also be entered via the numeric keypad directly, [←] via [←Backspace], [▲] and [▼] via the cursor keys. Further computer keyboard shortcuts to simulator keys are listed in the table printed overleaf.

^	^	^	^	^	^
F1	F2	F3	F4	F5	F6
1/x	y^x	TRI	ln	e^x	$\sqrt{x}$
i	**y**	**t**	**l**	**e**	**q**
STO	RCL	R↓	CC	f	g
s	**r**	Page↓	**C**	**f**	**g**
ENTER↑		x≷y	+/-	E	←
		tab	**c**	**E**	

/	7	8	9	XEQ
				X
×	4	5	6	▲
−	1	2	3	▼
+	0	.	R/S	EXIT
		. or ,	ctrl	esc

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

… **x** copies the full content of **X** to the clipboard.

… **z** copies the full contents of all 12 lettered registers to the clipboard.

… **Z** copies the full contents of all 112 registers to the clipboard.

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

APPENDIX F: FLASHING AND UPDATING YOUR *WP 43S*

There are two ways to get your *WP 43S*, in principle:

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

Way 1 allows you to repurpose a *DM42* you own already, so you may save costs – on the other hand, you will have to live with stickers on the keys then. This way is explained in next chapter. The chapter thereafter describes the way to update your *WP 43S* when a new firmware becomes available.

How to Flash Your *WP 43S*

Start your computer. Take your *DM42* and turn it on. Then press

SETUP

5 System

2 Enter system menu

4 Reset to *DMCP* menu

6 Activate *USB* Disk

Now connect to *USB* using a proper data transfer cable. The flash disk of your *DM42* should show up as an external mass storage volume.

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

Copy `keymap.bin` from there to the *DM42* flash disk. This will relocate keys to match the *WP 43S* layout.

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

Flash `WP43S.pgm` just like the demonstration program `SDKdemo.pgm` (https://github.com/swissmicros/SDKdemo).

Then press from *Free42*: [ ] SETUP 5 2 4 3 `WP43S.pgm` ENTER ENTER . Wait for xxx. Then press EXIT EXIT.

The keyboard layout is also that of *WP 43S* when navigating in the system part of the *DM42* (thanks to `keymap.bin`). The two files `Key_stickers.xcf` and `WP43S_overlay.xcf` are *GIMP* images to make your life easier. Print them, cut, and apply (see the picture) as long as you converted a *DM42 to get your WP 43S.*

To leave the *WP 43S* program:

Enter g MODE ▲ and select g SYSTEM to return to the *DMCP* system.

To retrieve the original *DM42* keyboard layout:

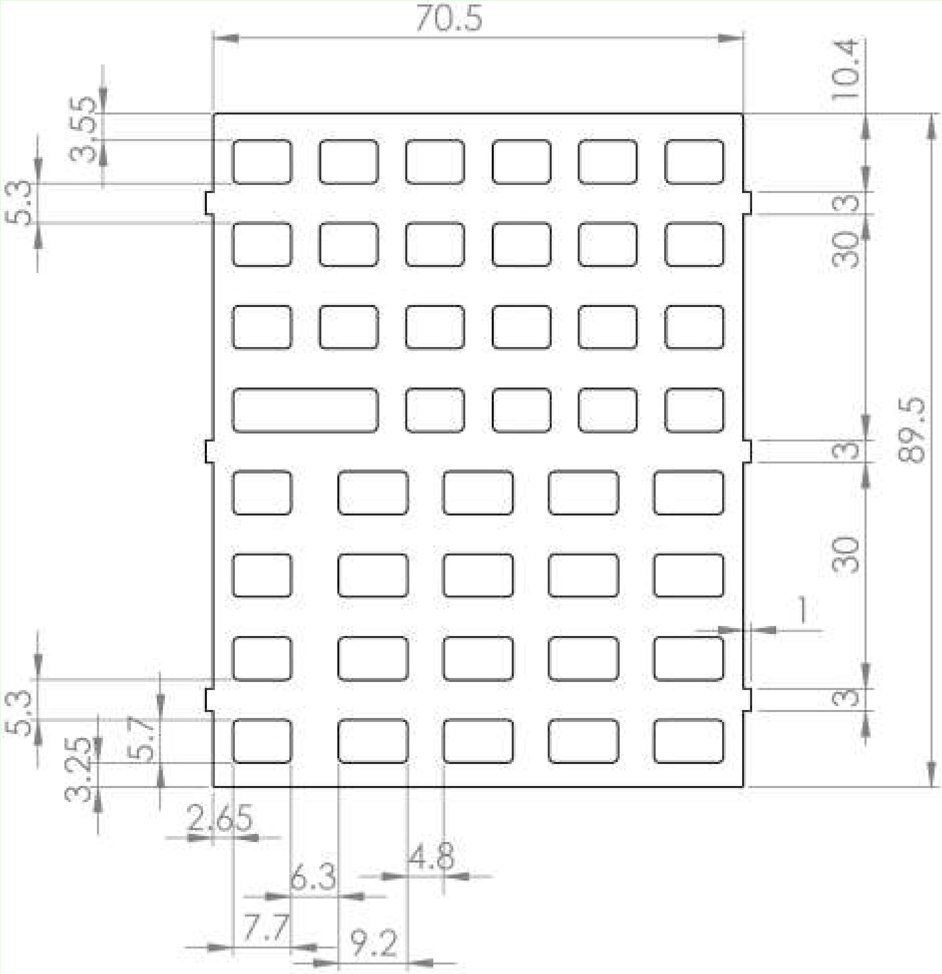
Copy the file `original_DM42_keymap.bin` to the *DM42* flash disk, rename it `keymap.bin` and RESET the *DM42*. Look here for more information: http://www.swissmicros.com/dm42/devel/dmcp_devel_manual/

How to Update Your *WP 43S*

xxx

Overlays

See below the drawing for an overlay – all dimensions are given in mm. Note the overall width is 72.5 mm.


70.5
10.4
3.55
5.3
3
30
3
89.5
3
30
1
3
5.3
5.7
3.25
2.65
4.8
6.3
7.7
9.2

APPENDIX G: TROUBLESHOOTING GUIDE

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

1. Open your calculator by unfastening the two bolts at the top of its backside. You will probably find a printed circuit board (*PCB*) whose top looks like this ➔ (cf. p. 154 for a *PCB* of an early *DM42*)

   In any case, you will see two small buttons, one labeled RESET.

   The other one is called PGM.

2. Now, do the following:

   a. Press and hold the PGM button.

   b. Press and release the RESET button.

   c. Release the PGM button.

   This sequence shall reset your *DM42* and put it in bootloader mode.[67]

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

---

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

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 domain functions.

Wherever complex numbers may be valid input or output, the command *name* is printed on light yellow background. 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 – otherwise 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. 237 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. 26 and 60, 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. 204): In a *Galton box*[68] (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). $P_{y,x} = \frac{\Gamma(y+1)}{\Gamma(y-x+1)}$ 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, too.
FIB	For integers, FIB returns the Fibonacci number f_n with $n = x$. These 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. Else 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.

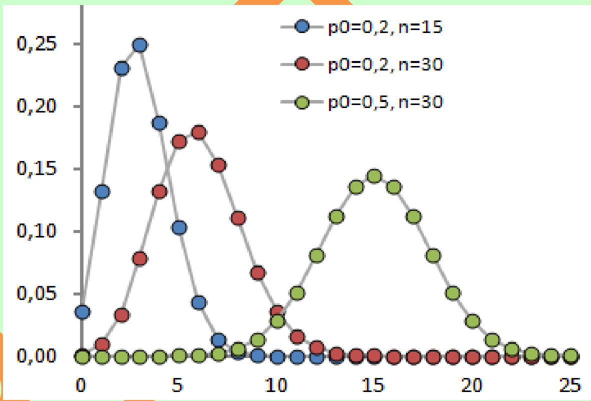
[68] Translator's note: This is called «Planche de Galton» in French, "Galtonbrett" in German, and "macchina di Galton" in Italian. Note the subtle differences in naming. Galton invented his box in 1889.

Statistical Distributions

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 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* p_0 in **I** and the *sample size* n in **J**.



BINOM$_P$ returns

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

(see COMB on p. 203 for the explanation of the notation).

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

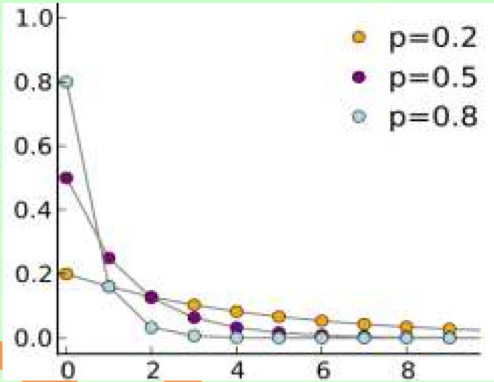
The *binomial distribution* is fundamental for error statistics in industrial sampling, e.g. for designing test plans.

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

.03 STO J **15** STO K **0** PROB g Binom: Binom returning 0.633 – so the odds are almost two out of three that you will not detect any defect in your sample! [69]

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

Geom: *Geometric distribution*:



GEOM$_P$ returns

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

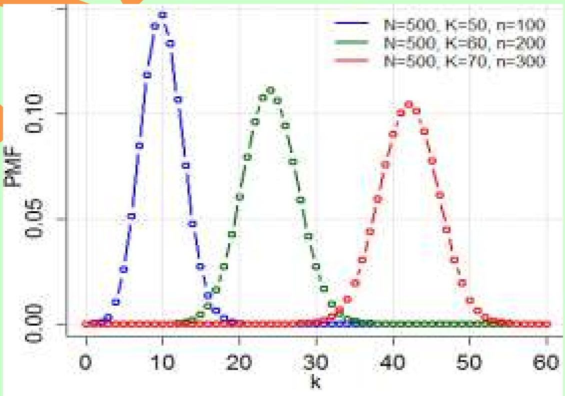
GEOM returns

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

being the probability for a first success after $\boldsymbol{m = x}$ *Bernoulli* experiments. The probability $\boldsymbol{p_0}$ for a success in each such experiment must be specified in $\mathbf{I}$.

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

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



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

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. 203 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 'large' batches (***n₀*** > 10) and 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 $\mathbf{X}$, the *gross probability of a success in a single draw* ***p₀*** in $\mathbf{I}$, and ***n*** in $\mathbf{J}$.

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

$= C_{n-1,f-1} \cdot p_0^f \cdot (1-p_0)^{n-f}$ (see COMB on p. 203 for the explanation of the notation and cf. BINOM).

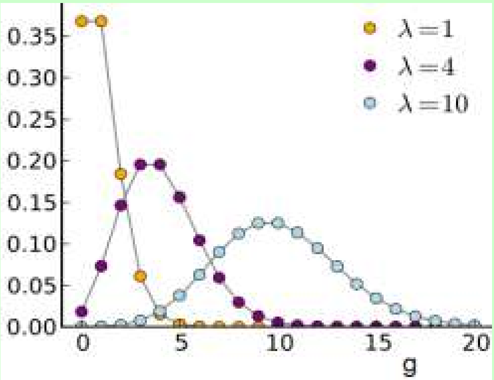
Start reading here for more: http://en.wikipedia.org/wiki/Negative_binomial_distribution .

Poiss: *Poisson distribution* with the *number of successes* ***g*** in $\mathbf{X}$ and the *Poisson parameter* ***λ*** in $\mathbf{J}$.

POISS$_P$ computes

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

and POISS returns the corresponding *CDF* for the



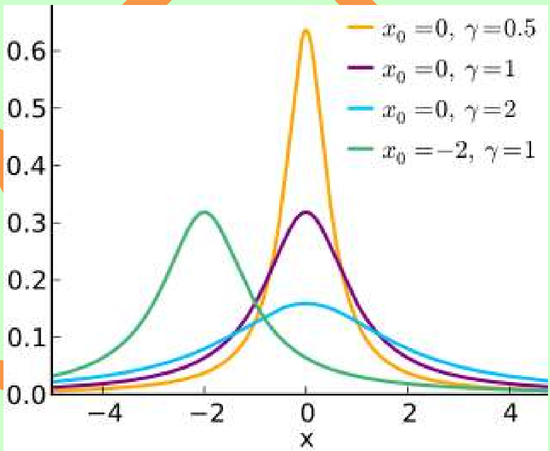
maximum number of successes ***m*** in $\mathbf{X}$.

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 ***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 $\mathbf{I}$ and the *shape* γ in $\mathbf{J}$.



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

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

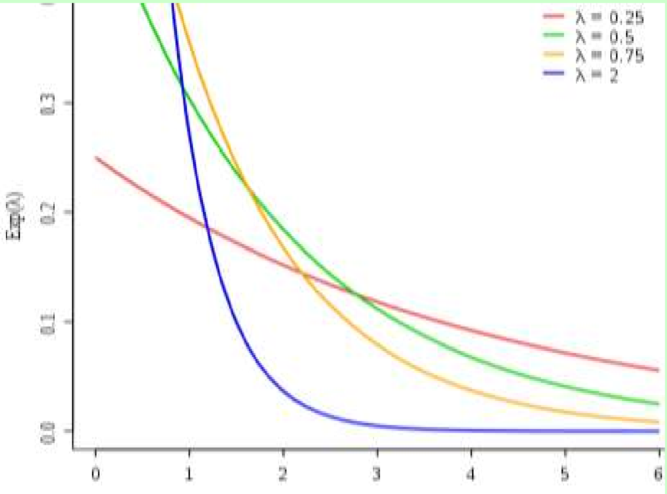
CAUCH^{-1} returns $F_{Ca}^{-1}(p) = x_0 + \gamma \tan\left[\pi \cdot \left(p - \frac{1}{2} \right) \right]$.

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

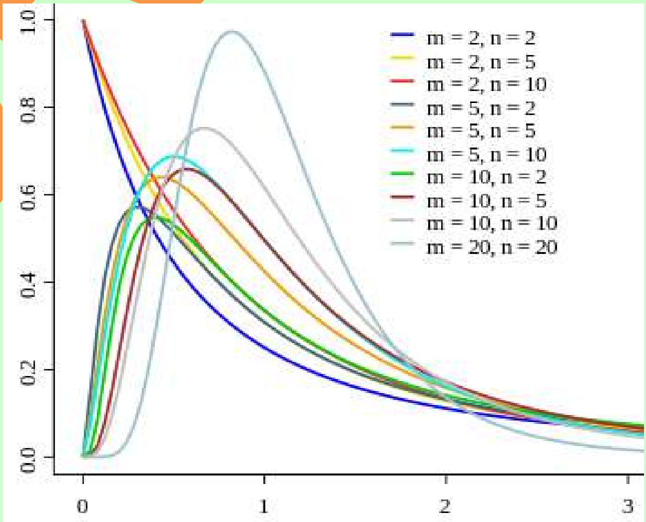
EXPON returns $F_{Ex}(x) = 1 - e^{-\lambda x}$.



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 diagram shows the *PDF* plotted for different *degrees of freedom* ***m*** *and* ***n*** corresponding to ***i*** and ***j***.



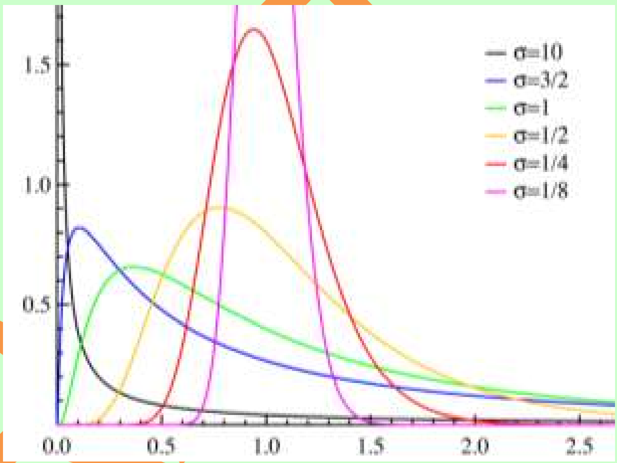
Read here for more information:

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

LgNrm: *Log-normal distribution* with the parameters $\mu = \ln \bar{x}_g$ in $\mathbf{I}$ and $\sigma = \ln \varepsilon$ in $\mathbf{J}$ (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 returns $F_{Ln}(x) = \Phi\left(\frac{ln(x)-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized normal CDF* as presented on p. 212.



Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3669.htm

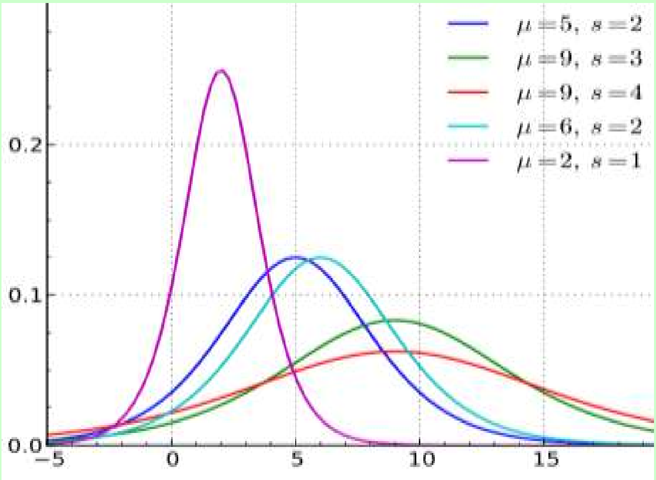
Logis: *Logistic distribution* with an arbitrary *mean* μ given in $\mathbf{I}$ and a *scale parameter* **s** in $\mathbf{J}$.

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

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

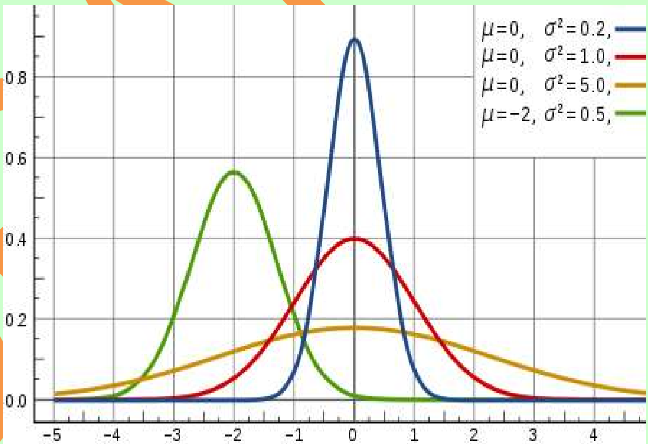
LOGIS returns $F_{Lg}(x) = \frac{1}{1+e^{-\xi}}$.

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 an arbitrary *mean* $\boldsymbol{\mu}$ given in $\mathbf{I}$ and an arbitrary *standard deviation* $\boldsymbol{\sigma}$ in $\mathbf{J}$. The red curve (for $\boldsymbol{\mu} = 0$ and $\boldsymbol{\sigma} = 1$) is the *standardized normal* (a.k.a. *Gaussian*) *distribution* φ.

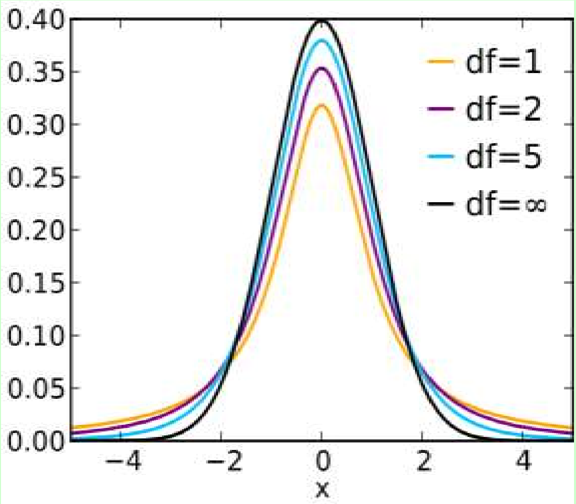


NORML$_P$ returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} = \varphi\left(\frac{x-\mu}{\sigma}\right)$ and

NORML returns $F_N(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standard normal* (or *Gaussian*) *CDF* as presented on p. 212.

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 picture shows its *PDF* plotted for different *degrees of freedom*. For ***df*** $\to \infty$, the shoulders of $t(x)$ shrink and it approaches the *PDF* of the *standard normal distribution* (compare the red *Gaussian* curve at NORML on p. 210).



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**.

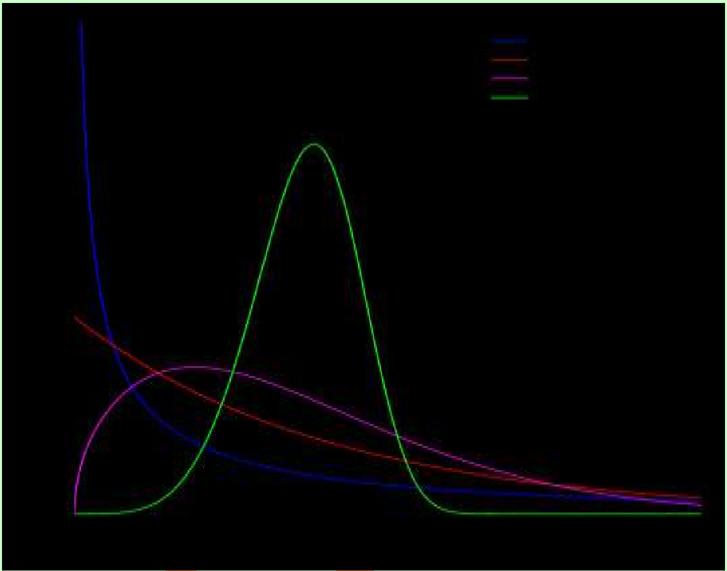
WEIBL$_P$ returns $f_W(t) = \frac{b}{T} \cdot \left(\frac{t}{T}\right)^{b-1} e^{-\left(t/T\right)^b}$ for $t \geq 0$, else 0. This is a very flexible function – see the curves plotted overleaf.

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

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

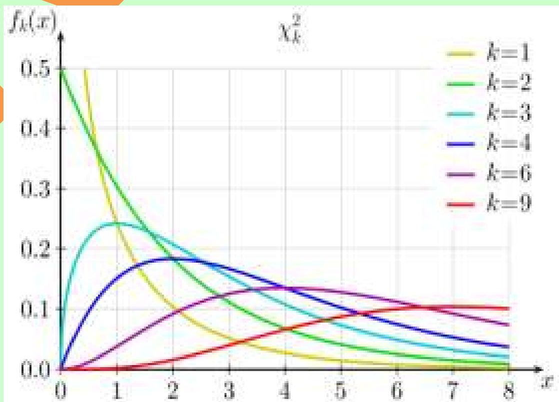
Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm .

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



φ(x) and **Φ(x):** $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is the *standardized normal PDF* (i.e. the famous *Gaussian* bell curve as drawn in red under NORML on p. 210), while $\Phi(x) = \int_{-\infty}^{x} \varphi(\tau) d\tau$ is the corresponding *CDF* (cf. the *error function* on p. 234). Take NORML instead.

χ^2(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 diagram 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 Fitting

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

- For samples drawn out of a *Gaussian* (additive) process, the expected value is the *arithmetic mean* (or *average*) and its uncertainty is given by its *standard error* (see $\overline{x}$ and s_m).
- For samples drawn out of a *log-normal* (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.

Be assured not everything is *Gaussian* in real world! [70] Process features can be detected (and should be checked well in advance of calculating e.g. means) using suitable tests – turn to applicable statistical reference literature.

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

---

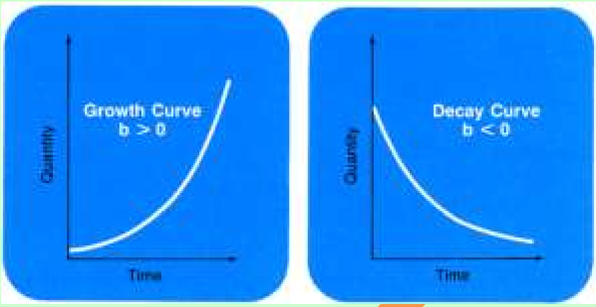
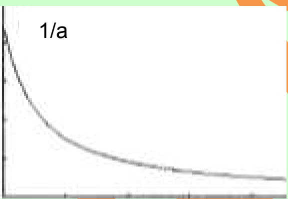
[70] Generally, the statistical model shall be chosen that matches observations best – within their statistical errors. In real life cases, however, dramatic deviations from the model distribution are frequently found – then you cannot expect the calculated consequences matching reality any better.

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 (cf. also D.T. recently).

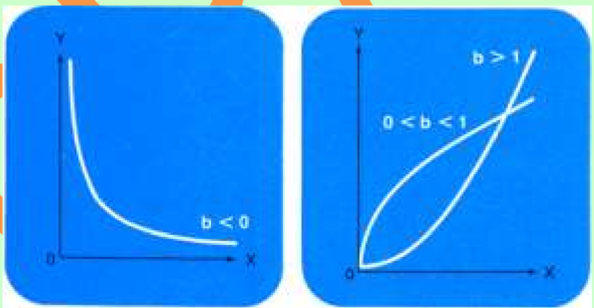
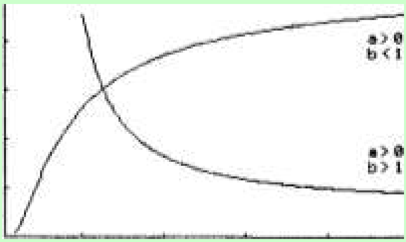
By the way: Since the *PDF* of the *Gaussian* distribution will never reach zero, this statistical model tells you to expect individual items far, far away from the mean value when your sample becomes large enough. This, however, does not match reality. So we must conclude nothing at all is really *Gaussian* 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.

Name	Remarks (see pp. 12ff for general information)
CauchF	Selects a Cauchy (a.k.a. Lorentz, Breit-Wigner) peak fit model $R(x) = {}^{1}/_{[a_0\,(x+a_1)^2+a_2]}$ for least squares regression.[71] See p. 207 for the shapes of such peaks.
CORR	For any set of data points (x_i, y_i), the *coefficient of correlation* is $r = \frac{s_{xy}}{s_x \cdot s_y}$. See s_{XY} and s below. For an arbitrary fit model $R(\boldsymbol{x})$, $r^2 = 1 - \frac{\sum [R(x_i) - y_i]^2}{\sum (\bar{y} - y_i)^2}$ 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* 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* $n-2$ and *confidence level* 99% (see p. 211).
COV	For any set of data points (x_i, y_i), the *population covariance* is $$COV_{xy} = \frac{1}{n^2}\left(n \sum x_i y_i - \sum x_i \sum y_i\right)$$ Compare s_{XY} below.

[71] Note that *least squares regression* is best for data point errors in direction ***y*** being significantly greater than errors in direction ***x***. See pp. 219ff for the formulas and more.

Name	Remarks (see pp. 12ff for general information)
ExpF	Selects the exponential curve fit model $R(x) = a_0 e^{a_1 x}$ for least squares regression.[71] Generally, this will be a good choice if the  measured data follow the shape of one of the two curves pictured here (think of human population growth or nuclear decay, for instance).[72]
GaussF	Selects a Gauss peak fit model $R(x) = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$ for least squares regression.[71] See p. 210 for the shapes of such peaks.
HypF	 Selects the hyperbolic fit model $R(x) = {}^{1}/_{(a_0 + a_1 x)}$ for least squares regression.[71]
LinF	Selects the linear fit model $R(x) = a_0 + a_1 x$ for least squares regression.[71] Generally, this will be a good choice if the measured data follow a straight line, raising or falling (but compare ORTHOF).
LogF	 Selects the logarithmic curve fit model $R(x) = a_0 + a_1 ln(x)$ for least squares regression.[71] Generally, this will be a good choice if the measured data follow a curve looking like drawn at left.

[72] Color plots on this page and the next are taken from the *HP-27* manual; $b = a_1$ on your *WP 43S*.

Name	Remarks (see pp. 12ff for general information)
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* if $$\left\|a_i\right\| / s(a_i) > t_{n-2}^{-1}(0.995)\ ,$$ with the right side being the inverse of the *t distribution* for the *degrees of freedom* $n-2$ and 99% *confidence* (cf. p. 211).
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*** (precisely: their variances are equal). The sum of squared distances of the data points to the fit line will be minimized. This model is called *orthogonal regression*. See pp. 219ff and the *OM* for more.
ParabF	Selects a parabolic fit model $R(x) = a_0 + a_1 x + a_2 x^2$ for least squares regression.[71]
PowerF	Selects the power curve fit model $R(x) = a_0\, x^{a_1}$ for least squares regression.[71]  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*).[72]
RootF	 Selects the root curve fit model $R(x) = a\, b^{1/x} = a_0 a_1^{1/x}$ for a least squares regression.[71]

Name	Remarks (see pp. 12ff for general information)
s_{XY}	For any set of data points (x_i, y_i), the *sample covariance* is $$s_{xy} = \frac{1}{n(n-1)}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$$ Compare COV above.
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* $\bar{x}$) is $s_{mx} = {}^{s_x}/_{\sqrt{n}}$
s_W , s_{mW}	The *sample SD* for weighted data (where the weight $\boldsymbol{y_i}$ of each data point $\boldsymbol{x_i}$ was entered via $\boxed{\Sigma+}$) is $$s_w = \sqrt{\frac{\sum y_i \sum y_i x_i^2 - (\sum y_i x_i)^2}{\sum y_i\,(\sum y_i - 1)}}$$ And the corresponding *standard error* (the *SD* of the *mean* $\bar{x}_w$) is $$s_{mw} = \frac{1}{\sum y_i}\sqrt{\frac{\sum y_i \sum y_i x_i^2 - (\sum y_i x_i)^2}{\sum y_i - 1}}$$
$\bar{x}$	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 $\bar{x}_H = {}^{n}/_{\sum \frac{1}{x_i}}$.
$\bar{x}_{RMS}$	The *quadratic mean* is calculated as $\bar{x}_{RMS} = \sqrt{\frac{1}{n}\sum x_i^2}$.

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

About the Curve Fitting Models Provided

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

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

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

1. It accumulates the 22 sums listed on pp. 93f and increments ***n***.
2. The evaluation depends on the fit model you select (cf. pp. 215ff):
   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

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

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

   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. 219 with the transformed results.

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

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

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

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

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

The standard errors of $a_{0,tPOW}$ and $a_{1,tPOW}$ can be calculated using the formulas for LINF on p. 219 with the transformed results.

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

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

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

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

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

The standard errors of $a_{0,tLOG}$ and $a_{1,tLOG}$ can be calculated using the formulas for LINF on p. 219 with the transformed results.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

About Error Propagation

Experimental data are always attended with errors (cf. footnote 37), caused by e.g. the uncertainty of the measuring method, the instrument used, and environmental variations. Even under controlled environmental and measuring conditions, random errors remain. These errors must be taken into account for a proper estimation of the uncertainty of your results computed using those experimental data. For about 200 years, *Gauß' least squares method* can be employed for this task.

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

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

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

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

And with your *WP 43S*, the following algorithm will do for computing ***ΔR***, even if f is '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 with the help of your *WP 43S*.

For 'small' errors or less curvature, the following simpler algorithm will do, requiring down to half as many steps only:

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

Being through with all n parameters, you will end with

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

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

Solving Differential Equations

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

First, we solve one-dimensional problems of the kind

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

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

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

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

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

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

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

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

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

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

Let us drop the restriction for b now. Replacing *a* in the previous set of equations by the right side of the differential equation on p. 226, 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

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

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

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

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

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

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

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

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

$$f_{i+1} = f_i + \left[\frac{df}{dt}\right]_{i-1/2}\Delta t$$

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

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

And we know $F = {}^{G\,m\,M}\!/_{(x^2+y^2)}$, 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:

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

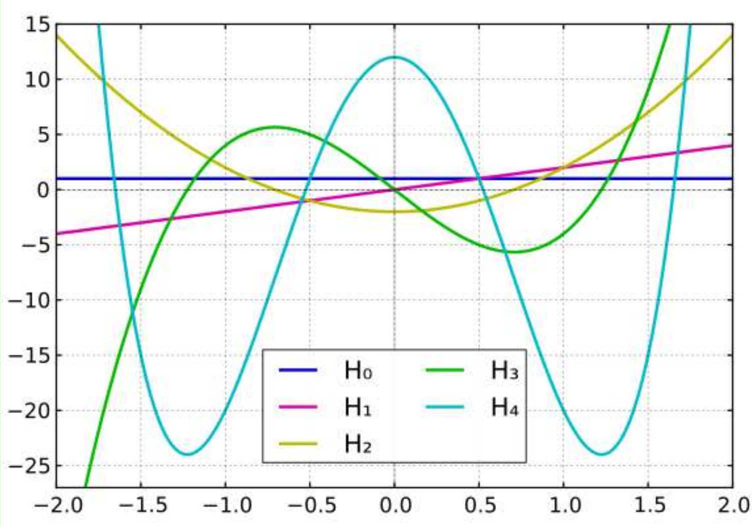
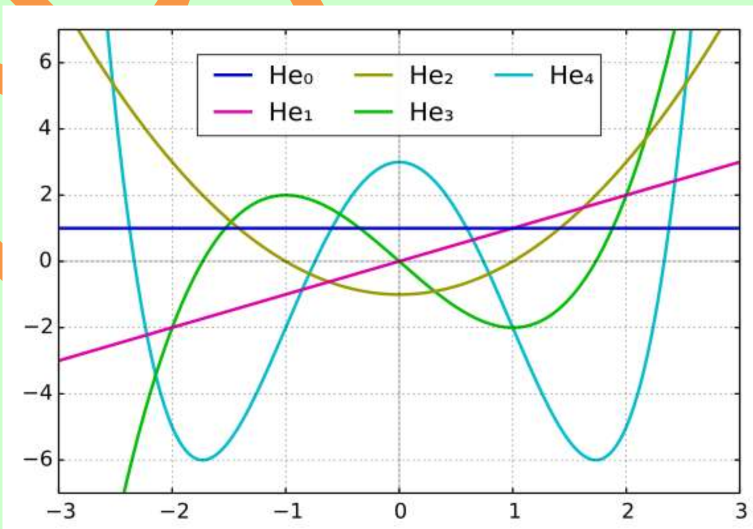
$$\left(\frac{dx}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dx}{dt}\right)_{i-\frac{1}{2}} + K_x \Delta t \qquad \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dy}{dt}\right)_{i-\frac{1}{2}} + K_y \Delta t$$

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

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.

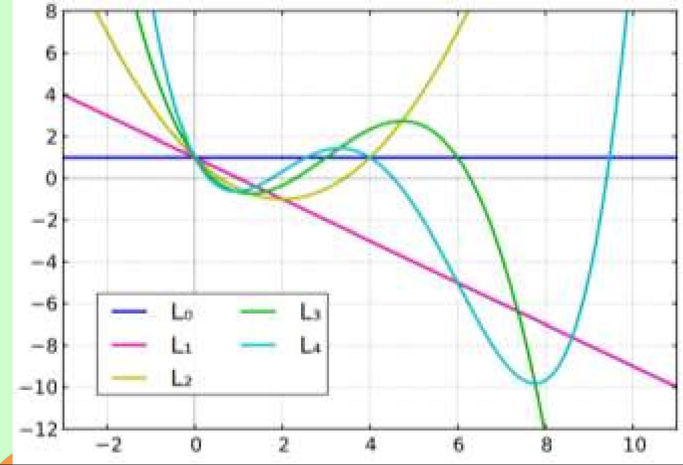
Name	Remarks (see pp. 12ff for general information)
	
H_{np}	***Hermite*** *polynomials* for physics: $H_{np}(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in **Y**, solving the same differential equation. See the first five polynomials plotted below. 

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

***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 $x\cdot f''(x)+(1-x)\cdot f'(x)+n\cdot f(x)=0$.

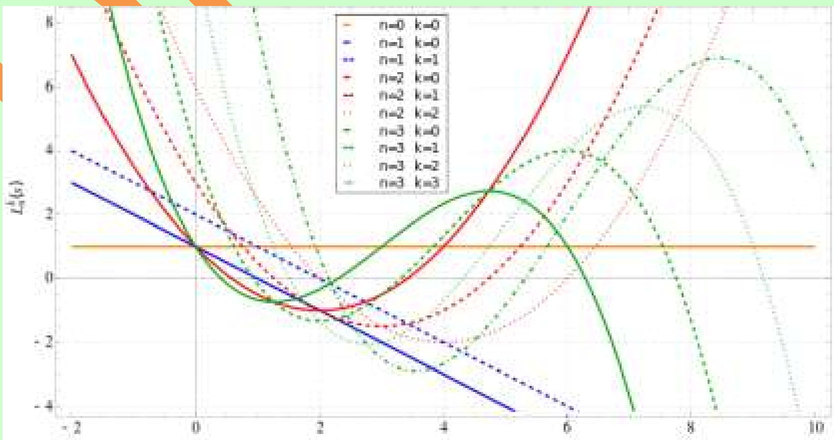
See the first five *Laguerre polynomials* plotted here.



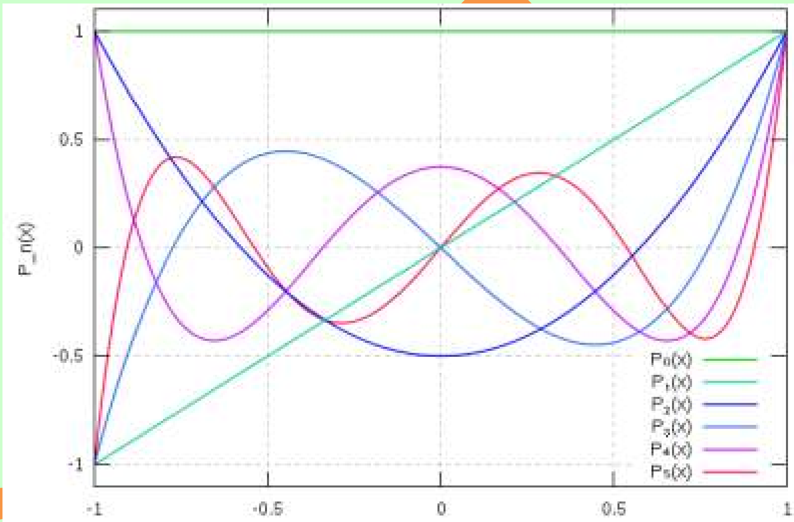
</td></tr>
<tr><td>$L_{m\alpha}$</td><td>

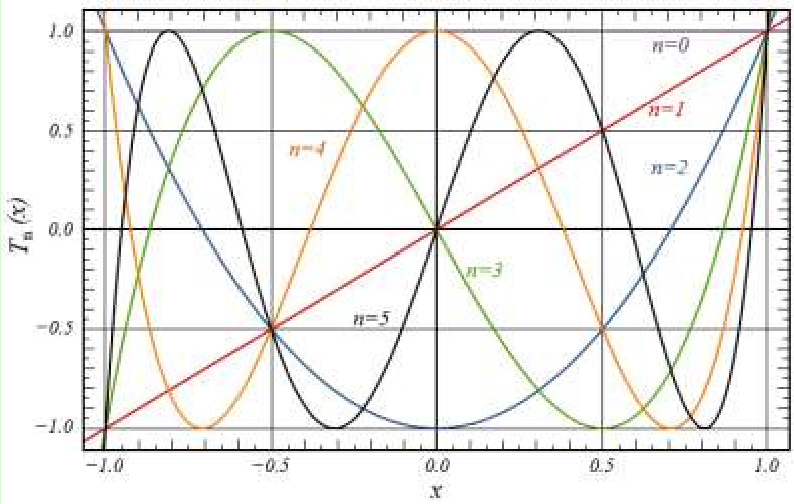
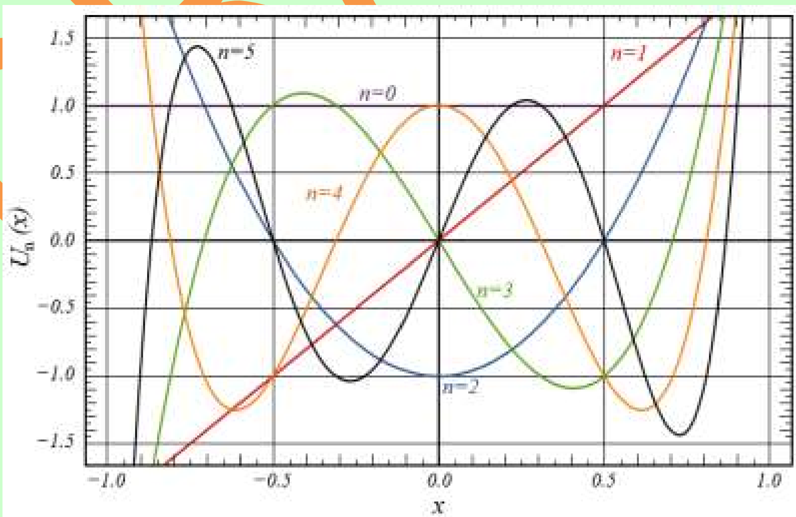
***Laguerre's generalized** polynomials* (compare L_n above):

$L_n^{(\alpha)}(x)=\frac{x^{-\alpha}e^x}{n!}\cdot\frac{d^n}{dx^n}\left(x^{n+\alpha}e^{-x}\right)$ with ***n*** in $\mathbf{Y}$ and α in $\mathbf{Z}$. Some of them are plotted below (k = α).



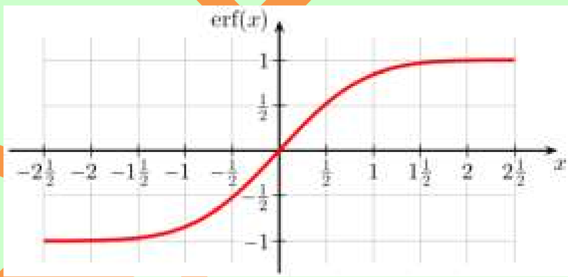
</td></tr>
</table>

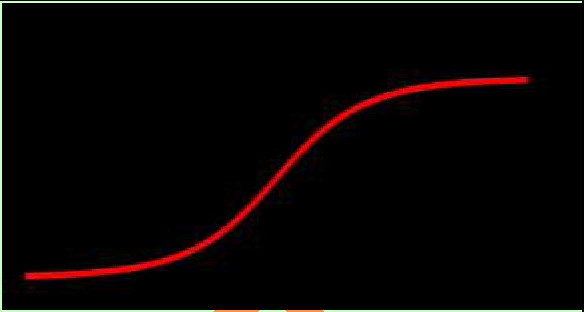
Name	Remarks (see pp. 12ff for general information)
P_n	***Legendre*** *polynomials*: $P_n(x) = \frac{1}{2^n n!} \cdot \frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right]$ with ***n*** in $\mathbf{Y}$, solving the differential equation $$\frac{d}{dx}\left[\left(1-x^2\right)\cdot\frac{d}{dx}f(x)\right]+n(n+1)f(x)=0 \cdot$$ See the first six polynomials plotted here. 
T_n	***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 $T_0(x)$ ... $T_5(x)$.

Name	Remarks (see pp. 12ff for general information)
	
U_n	***Chebyshev*** *polynomials of* ***second*** *kind* $U_n(x)$ 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 $U_0(x)$ ... $U_5(x)$: 

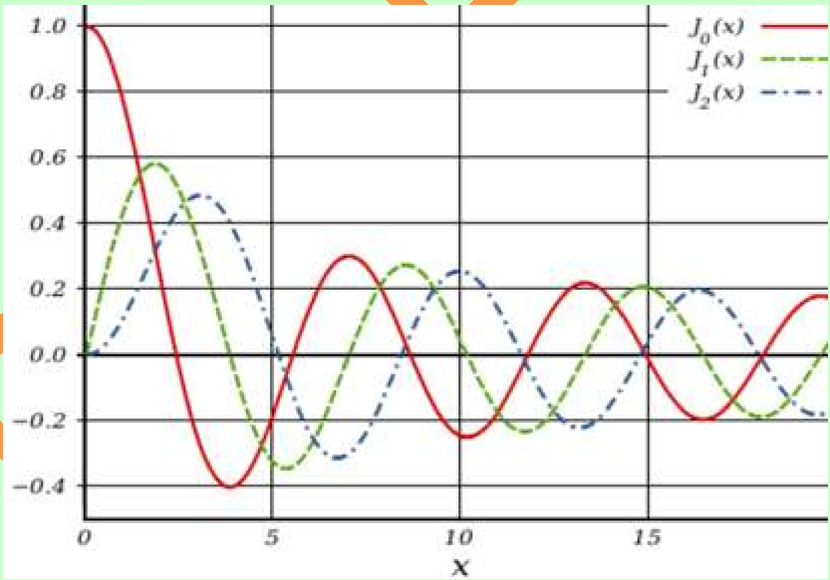
Even More Mathematical Functions

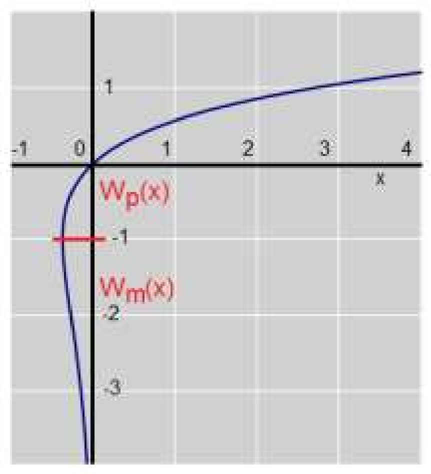
All the following functions are found in X.FN. Some of them are for pure mathematics only but were useful at some stages of the *WP 34S* or *WP 43S* projects, so we made them accessible for the public.

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 $erf\left(\frac{x}{\sqrt{2}}\right) = 2 \cdot \Phi(x) - 1$ with $\Phi(x)$ representing the *standard normal CDF* as described on p. 212. 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 *standard normal distribution*.

Name	Remarks (see pp. 12ff for general information)
g_d	 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 Gudermannian 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* $\frac{\beta_x(x,y,z)}{B(y,z)}$ with $\beta_x(x,y,z) = \int_0^x t^{y-1}(1-t)^{z-1}\,dt$ being the *incomplete Beta function* and $B(y,z)$ being *Euler's Beta function* (see p. 90 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. 90 for $\Gamma(x)$.
$I\Gamma_q$	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. 90 for $\Gamma(x)$.

See here for more: https://en.wikipedia.org/wiki/Incomplete_gamma_function

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)
W_p , W_m	Returns *Lambert's W* with its principal branch (called W_p here) and its negative branch (called W_m for <u>m</u>inus). The connecting point is (−1/e, −1). The diagram 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 .
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 : $\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{\pi}{2}x\right)\cdot\Gamma(1-x)\cdot\zeta(1-x)$. Note the different vertical scales for negative and positive $\boldsymbol{x}$ in the plot overleaf.

Name	Remarks (see pp. 12ff for general information)
	 Look here for more: http://mathworld.wolfram.com/RiemannZetaFunction.html .

Note the *error function* as well as *Laguerre*, *Legendre*, and *Bessel functions* were already provided on the *Commodore M55* pocket calculator of 1976/77 (featuring 55 keys).

Beyond what is printed in this appendix, you will find lots of information about the special functions implemented in your *WP 43S* in the internet in addition. Generally speaking, *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. 204ff) is a competent source. And *Mathworld* (quoted on pp. 234ff) may contain more details than you ever want to know. Further references are found at these sites.

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 $r.00_3$ there, returning to FACT$_2$ with $x = 2$. This x will be multiplied by $r.00_2$ there, returning to FACT with $x = 6$, where it will be multiplied by $r.00$ and will finally become 24.

Xxx

APPENDIX J: RELEASE NOTES

	Date	Release notes
0	29.11.12	**Official project start with the first publication of a layout on the forum of the *MoHPC*** ([*https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685*](https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685))**.** There are found, however, 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 Steinmann* for information.
0.4	28.3.16	Renamed LOGS to EXP and EEX to E. Added hardware information from 2^{nd} manufacturer.
0.5	29.10.16	Returned EEX. Changed the 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 the 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*.

	Date	Release notes
0.7	2.4.18	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, HYP, 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 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 20.9.18	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. Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9	3.1.19	Removed the *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 functionality of EXIT and 1/x to match *HP-42S*. Added a chapter about curve fitting. Modified functionalities of ENTER↑ and ↶. Moved the printer character to f R/S. 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. 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.

	Date	Release notes
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. Corrected.
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*. Corrected.
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*. Corrected.

	Date	Release notes
0.13	10.11.19	Expanded the alpha keyboard. Modified CPX, INTS, MODE, α●, SHOW, and STATUS. Refined sorting of *items*, ALL, CX→RE, RE→CX, RBR, and RM. Started filling *App. F* and *G*. Refined *App. 2*. 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* has more than three rows, its current view is limited by a dashed line indicating that it is a *multi-view menu* and [▲] or [▼] can be used to display the additional *views* of this *menu*.

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 general purpose *registers* (00 ... 99). And there are **I**, **J**, and **K** serving special purposes in matrix handling (see p. vii), probability distributions (see p. viii), 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*. Of these, the following have a special meaning if set:

103	**T**	sets print mode to Tracing
105	**B**	Overflow
106	**C**	Carry
107	**D**	allows for infinite and non-numeric results (“Danger”)
109	**I**	allows for complex results

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. v for more.

Available memory: [INFO] MEM? (or [FLAGS] STATUS) displays the amount of free memory. Use CLP for clearing programs or clear variables to free memory that is no longer needed.

DATA TYPES

Long integers are the simplest type of data. Any number you enter without using [.], [E], or [CC] 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 (set RECT and press **1.23** [CC] **4.56** [+/-][ENTER↑]) or its magnitude and phase like -7.89 ∡ 120° in polar mode (set POLAR and press **7.89** [+/-][CC] **120** [ENTER↑]).

Angles: Any real number input trailed by [d.ms] is interpreted as an *angle* in *sexagesimal degrees*. *Angles* may be entered as well in *decimal degrees*, *radians*, *multiples of π*, or *gradians*. Choose the appropriate angular display mode via [MODE] (see overleaf).

Times: Any real number input trailed by [h.ms] is interpreted as a sexagesimal *time*. It will be displayed like 23:45:43.210 9 with as many decimal *seconds* as needed.

Dates: Any real number input trailed by [.d] is interpreted as a *date* in the format selected (yyyy.mmdd for Y.MD or dd.mmyyyy for D.MY or mm.ddyyyy for M.DY).

Matrices: see p. vii.

Short integers: Any numeric input trailed by [#] and a legal base 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 strings: Enter *alpha input mode* (*AIM*) by pressing [α]. Data entered in *AIM* become an alphanumeric string when closed (unless they are function parameters). International (e.g. accented) letters are found in [g][+]. Greek letters are accessed via [g] plus the corresponding Latin letter. 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

SETSIG					
DENMAX	DENANY	DENFAC	DENFIX	SSIZE4	SSIZE8
FAST	SLOW	RM	QUIET	REALRE	CPXRES
DEG	RAD	GRAD	MULπ	RECT	POLAR

DEG Selects *degrees* as angular display mode (*ADM*).
RAD Selects *radians* as *ADM*.
GRAD Selects *gradians*, a.k.a. *gon*, as *ADM*.
MULπ Selects *multiples of* π as *ADM*.

RECT Sets rectangular format for displaying complex numbers.
POLAR Sets polar format for displaying complex numbers.

FAST Sets processor to normal speed.
SLOW Reduces processor speed (approx. two times slower than fast).

RM Sets rounding mode.

QUIET Disables or enables (toggles) the beeper.

REALRE Allows only real results.
CPXRES Allows also complex results of real calculations (e.g. $\sqrt{-1}$).

DENMAX Sets the maximum denominator for calculating with fractions.
DENANY Any denominator up to DENMAX is legal.
DENFAC Any integer factor of the maximum denominator is legal
DENFIX Sets DENMAX as fixed denominator.

SSIZE4, SSIZE8 Sets *stack* size to 4 (or 8) *registers*.

SETSIG Sets calculator precision (1 ... 34 significant digits).

DISPLAY FORMATS

DISP

CHINA	EUROPE	INDIA	JAPAN	UK	USA
GAP					DSTACK
SCIOVR	ENGOVR	MULT×	MULT·	RDX.	RDX,
SDL	SDR			RDP	RSD
FIX	SCI	ENG	ALL	ROUNDI	ROUND

FIX Fixed number of decimals.
SCI, ENG Scientific (or engineering) notation.

ALL Displays all digits required as far as possible.

ROUND Rounds a *time*, real, or complex x to current display format.
ROUNDI Rounds to next integer.
RDP Rounds x to n decimal places (1 ... 99, think of FIX)
RSD Rounds x to n significant digits (1 ... 34, think of SCI).

SDL, SDR Shifts digits left (right) by n decimal positions.

SCIOVR, ENGOVR Specify the display format if ALL is not viable anymore.

MULT×, MULT· Select the multiplication symbol.

RDX., RDX, Select the radix mark.

GAP Selects a digit group gap inserted after every n digits.

DSTACK Sets the number of *stack registers* displayed (1 ... 4).

CHINA, EUROPE, INDIA, JAPAN, UK, USA Set local display preferences.

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 the current program only.

Smart program menu: XEQ PROGS displays all programs (actually: 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).

The Run/Stop key: Pressing R/S runs the current program beginning with the current step or stops a 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 be searched alphabetically.

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 complete 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 providing 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					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 *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 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 everything but the modes set.
RESET	Resets the *WP 43S* to *startup default*.

CATALOG VARS ... ⌃ ⬅ allows for deleting the user variable selected.

CATALOG MENUS ... ⌃ ⬅ allows for deleting the user *menu* selected.

CATALOG PROGS ... ⌃ ⬅ allows for deleting the user program selected.

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***.
≡△	moves it to previous step (use ▲ or ▼ unless a
≡▽	moves it to next step *multi-view menu* is displayed).
⬅	deletes the *current program step* entirely.

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. iv).

Global labels can be accessed from anywhere in memory (thus, they should be unique). Global labels are alphanumeric and up to 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 calling routine only.

Tests (Do if True, Skip if False)

When a binary test step is executed, the program step immediately following said step is executed if the test result is “true”; if the result is “false”, the step following the test step is skipped.

Looping

ISE, ISG, ISZ, DSE, DSL, and DSZ (found in LOOP) control looping. Each accesses a variable or *register* containing a loop control number in the form cccccc.fffii with cccccc being the current counter value, fff the final counter value, and ii the increment (or decrement) size (default is 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). E.g. the program segment pictured here counts from 1 to 52 by threes (executing the loop 18 times) and then beeps.

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

Using a Variable Menu

A *variable menu* may be displayed by the *Solver* or the numeric integrator (see p. xif) or by VARMNU within a program. Each label in the *menu* represents a variable. While the *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*.

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 variable. If the variable already exists, DIM re-dimensions it.

For editing the matrix in X, use MATX EDIT .

For editing 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 new row (**Grow mode**).

WRAP and GROW are in the f-shifted row of the *Matrix Editor menu*.

Matrix arithmetic: +, −, ×, and / work for matrices just as for individual numbers. Advanced functions often operate on the individual matrix elements. Any time a matrix is used in a mathematical operation with a complex object, the result will be a complex matrix.

To solve a system of simultaneous linear equations represented by the matrix equation $(A)\vec{X} = \vec{B}$:

1. Key in 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**, **Mat_B**, and **Mat_X**.
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				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			$Binom_e$	$Binom^{-1}$

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#.

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

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

Geom: d *Geometric distribution* ($i = p_0$ = gross probability of a success)

Binom: d *Binomial d.* ($i = p_0$, $j = n$ = sample size)

Cauch: c *Cauchy-Lorentz* (a.k.a. *Breit-Wigner*) *d.* (i = location, j = shape)

Expon: c *Exponential d.* (i = rate)

F: c *Fisher's F d.* (i = degrees of freedom $(dof)_1$, $j = dof_2$)

Hyper: d *Hyperbolic d.* ($i = p_0$, $j = n$, k = batch size)

LgNrm: c *Log-normal d.* ($i = \mu$, $j = \sigma$)

Logis: c *Logistic d.* ($i = \mu$, j = scale parameter)

NBin: d *Negative Binomial d.* ($i = p_0$, $j = n$)

Norml: c (General) *normal d.* ($i = \mu$, $j = \sigma$)

Poiss:	d	*Poisson d.*	($\boldsymbol{i}$ = $\boldsymbol{n}$ $\boldsymbol{p_0}$ = *Poisson* parameter)
t:	c	*Student's t d.*	($\boldsymbol{i}$ = *dof*)
Weibl:	c	*Weibull d.*	($\boldsymbol{i}$ = shape, $\boldsymbol{j}$ = characteristic lifetime)
χ^2:	c	*Chi-square d.*	($\boldsymbol{i}$ = *dof*)

Following naming convention holds for most distributions, e.g. for the *normal d.*: **Norml$_p$** denotes the *probability density function*, **Norml** the *cumulated d. function*, **Norml$_e$** 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 given in **X** ($0 \le x \le 1$).

STATISTICS

Statistical data are accumulated in 23 dedicated summation *registers*, 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 mistakes or remove erroneous data,

- either press ↶ (for the very last data point)
- or recall the (earlier) incorrect y and x data in the *stack* and press Σ- .

Data Evaluation and Analysis

	OrthoF	GaussF	CauchF	ParabF	HypF	RootF
	LinF	ExpF	LogF	PowerF		BestF
		$\bar{x}_{RMS}$				
		$\bar{x}_H$	cov			
	L.R.	r	s_{xy}	$\hat{x}$	$\hat{y}$	x^2
STAT	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	PLOT
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	SUM
	Σ+	$\bar{x}$	s	σ	s_m	x^2

$\bar{x}$, 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.
$\bar{x}_H$, $\bar{x}_{RMS}$	Harmonic and quadratic mean values.
SUM	Returns Σx and Σy.
PLOT	See the *ReM*, *App. K*.

L.R.	Computes the parameters a_0 and a_1 (and a_2 , if applicable) of the fit model(s) 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 following the fit model 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 $y = a_0\, x^{a_1}$.
RootF	Root fit model:	$y = a_0 a_1^{1/x}$.
HypF	Hyperbolic fit model:	$y = 1/(a_0 + a_1 x)$.
ParabF	Parabolic fit model:	$y = a_0 + a_1 x + a_2 x^2$.
CauchF	Cauchy peak fit model:	$y = 1/[a_0\,(x + a_1)^2 + a_2]$.
GaussF	Gauss peak fit model:	$y = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$.

BestF	Blindly selects the model returning the best correlation coefficient.
OrthoF	Works like LINF but assumes equal errors in x and y. ORTHOF is not part of the pool BESTF investigates.

ADVANCED OPERATIONS

EQN is for interactive editing, storing, recalling, solving, integrating, and deriving equations.

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

Interactive Operations on Equations

EQN

…					
DELETE					
NEW	EDIT	f''	f'	∫f	Solver

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 $\mathbf{X}$ (cf. p. vi).

Σ_n *label* calculates the sum of the terms given by the routine specified, using the loop control number given in $\mathbf{X}$ (cf. p. vi).

SOLVE *var* solves for an unknown variable in an expression, given values for all the other variables. The expression *$f(x_1, x_2, \ldots)$* shall be written as a program (let's call it **AB**, for example):

- **AB** must <u>begin with a global label.</u>

- 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

				LZOFF	
1COMPL	2COMPL	UNSIGN	SIGNMT	LZON	WSIZE
LJ					RJ
SL	RL	RLC	RRC	RR	SR
SB	BS?	#B	FB	BC?	CB
NAND	NOR	XNOR		MIRROR	ASR
AND	OR	XOR	NOT	MASKL	MASKR

BITS

AND, OR, XOR, NAND, NOR, XNOR   Boole's binary operators.

NOT   Inverts every bit in **X**.

MASKL, MASKR   Create masks of x bits on the left (or right) side.

MIRROR   Reflects all bits.

ASR ***n***   Arithmetic shift x right by ***n*** places.

SB, FB, or CB ***n***   Sets, flips, or clears bit #***n*** in x.

BS?, BC?   Checks if bit #***n*** in x is set (or clear).

#B   Returns the number of bits set in x.

SL, SR ***n***   Shift x left (or right) by ***n*** places.

RL, RR ***n***   Rotate x left (or right) by ***n*** places.

RLC, RRC ***n***   Rotate x left (or right) by ***n*** places through Carry.

LJ, RJ   Adjust the bits set in x to the left (or right).

1COMPL, 2COMPL   1's (2's) complement mode.

UNSIGN   Unsigned mode.

SIGNMT   Sign-and-mantissa mode.

WSIZE   Sets the word size (1 ... 64 bits).

				LZOFF	
1COMPL	2COMPL	UNSIGN	SIGNMT	LZON	WSIZE
DBL /	DBLR	DBL ×	^MOD	CEIL	GCD
IDIV	RMD	MOD	×MOD	FLOOR	LCM
A	B	C	D	E	F

INTS

A ... F   Digits for bases >10.

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

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.

OPERATIONS ON ALPHANUMERIC STRINGS

Connect strings by pressing $\boxed{+}$. Then x will be appended to the string y. With numeric data in $\mathbf{X}$, 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 the substring x begins in the string in ***s***.
FBR	Displays all characters defined in both fonts.

BACKGROUND CONSIDERATIONS AND FACTS

This is for recording and explaining some of the boundary conditions considered and the settings chosen for the *WP 43S* in the course of this project.

Alpha Register

For long I thought we could do without a dedicated *alpha register*. 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 this task.

Angles

Originally, a separate *data type* 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 *data type* 4 returned with v0.10 for sake of keeping algebraic operations simple and avoiding special purpose 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 *grads*) but 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 in various *integer sign modes* and bases. *Radians*, however, did not fit into this concept due to the necessity for high precision storage of $\boldsymbol{\pi}$ for modulo calculations and rounding errors.

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 2π suffice to warrant 16 digits accuracy of respective function results for the number range of *SP* reals.

Character Sets

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

The so-called **'numeric' font** uses a matrix of up to 16 × 32 px (variable width, fixed height). Therein, the punctuation space (2008_{16}, 8 px wide) is employed for separating groups of digits in longer numbers – following *ISO 80000-1* for an unambiguous numeric display. This font is generally used for numeric output of the *WP 43S*. It is also employed for echoing numeric input unless too long. It can be used for echoing command input as well – screen space suffices.

In total, six blank characters are provided 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 CNST.

Optionally, narrow digits can be used in complex numbers or in matrices or in *short integers* of small base where space may be scarce (see pp. xviii ff).

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 exceptions, however: see e.g. the multiplication dot at $00B7_{16}$ and the root symbols in row 2210_{16}.

Characters with codes < 0020_{16} are for control purposes; some of them (4, 10_{10}, 27_{10}) may be useful for printer control (e.g. of an *HP 82240 A/B*).

Many characters are 8 px wide as digits – they will help where a constant character spacing is wanted.

There is a number of super- and subscripts provided. They allow for displaying all the *items* featured on the *WP 43S*. Arbitrary numeric indices or exponents are possible as well.

Eight blank characters are provided (listed at right with their hexadecimal addresses and their widths). Using them, any spacing is feasible.

	Character code	px
Standard space	20_{16}	10
m space	2003_{16}	12
m/3 space	2004_{16}	4
m/4 space	2005_{16}	3
m/6 space	2006_{16}	2
Figure space	2007_{16}	8
Punctuation sp.	2008_{16}	4
Hair space	$200A_{16}$	1

This small character set allows for correctly spelling the languages of more than 3.5×10^9 people using either Greek or Latin alphabets:

> Afrikaans, Aymara, Bahasa Indonesia, Bahasa Melayu, Basa Jawa, Basa Sunda, Bosanski, Català, Cebuano, Česky, Cymraeg, Dansk, Deutsch, Eesti, Ελληνικά, English, Español, Euskara, Français, Gaeilge, Galego, Hrvatski, Italiano, Kiswahili, Kreyòl, Kurdî, Lietuvių, Magyar, Malagasy, Nāhuatl, Nederlands, Nihongo (Rōmanji), Norsk, Ŏzbek tili, Polski, Português, Quechua, Română, Shqip, Slovenščina, Slovensky, Srpski, Suomi, Svenska, Tagalog, Tatarça, Türkçe, Türkmençe, Vlaams, Wallon, and Zhōngwén (hànyǔ pīnyīn).

This makes the WP 43S the most versatile calculator available worldwide. If you know of further living languages covered (with one million speakers or more) beyond the ones listed, please tell us.

Turn to the *OM* for examples where these characters are used.[73] See here some sample strings in either font, 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

Complex Infinities

Since infinities are counted as numeric data, also complex infinities are part of the complex number plane the WP43S operates on. There are eight 'complex infinities' possible only:

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 at the positive real axis.

[73] 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.

Display Limits

Due to the character sizes and their design (cf. pp. xvi f), 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.[74]

For double precision numbers, a 33-digit mantissa plus exponent can be shown using the small font:

$-8.123\ 456\ 789\ 012\ 345\ 678\ 901\ 234\ 567\ 890\ 12\times10^{-925}$,

taking 399 px (= 34 × 8 + 10 × 4 + 9 + 13 + 4 × 8 + 1) for displaying.

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: $-1.234\ 567\ 890\ 12$.

[74] One 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.

Complex numbers in Cartesian notation require 1 + 15 + 12 + 15 + 1 = 44 px for 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):

-4.234×10^{-925}+i×4.234×10^{-925} .

Using 8 px wide multiplication dots instead, only 1 + 15 + 12 + 8 + 1 = 37 px are necessary for +i·. Thus, we can show one decimal more since 15 + 2 × (5 × 16 + 3 × 8 + 16 + 4 × 13) + 37 + 1 = 397 px:

-4,234 5·10^{-925}+i·4,234 5·10^{-925} .

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

-6.234 5×10^{-925}+i×6.234 5×10^{-925} ,

-6.234 56·10^{-925}+i·6.234 56·10^{-925} .

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 ∡-0.234 567 8π .

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 *Sect. 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*:

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

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

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

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 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 294 digits. *Long integers* ≥ 10^{294} shall be trailed by an exponent again taking 9 + 13 + 3 × 8 = 46 px in worst case; thus, display precision may be reduced to 288 digits minimum then.

For unsigned ***short integers***, up to 21 *bits* may be shown in 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 narrow 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*:

Base ▼	Allowable size of digits for display			
	large	narrow	small	special
2	21 *bits*	25 *bits*	44 *bits*	64 *bits*
3	31 *bits*	38 *bits*		
4	38 *bits*	44 *bits*	64 *bits*	
5	46 *bits*	55 *bits*		
6	51 *bits*	62 *bits*		
7	56 *bits*		64 *bits*	
8	57 *bits*	64 *bits*		
> 8	64 *bits*			

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

$[-6{,}609{\cdot}10^{-19}\ \ -6{,}609{\cdot}10^{-19}\ \ -1{,}609{\cdot}10^{-19}\ \ -1{,}609{\cdot}10^{-19}]$

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

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

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

$[\ldots\ -6{,}609\,2{\cdot}10^{-19}\ \ -6{,}609\,2{\cdot}10^{-19}\ \ -1{,}609\,2{\cdot}10^{-19}\ \ldots]$.

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. Within the space of 3 standard numeric output rows, 3 × 32 + 2 × 5 = 106 px are available, allowing for 5 matrix rows (5 × 20 + 4). Thus, 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

[... -6.609 226+i·6.609 226   -1.609 226+i·1.609 226 ...]

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

[... -6.60E^{-199}+i·6.60E^{-199} -1.60E^{-199}+i·1.60E^{-199} ...] .

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:

[... -6,609+i·6,609   -6,609+i·1,609   -1,609+i·1,609 ...] .

One row of **alphanumeric text** will typically take some 40 characters. The actual number will vary depending on their individual widths as mentioned above.

The *status bar* is a good example for such an alphanumeric row: Loaded to maximum, it might look like

2017-05-08 23:49 RL∡π /3 546f 64:u$_c^o$ A

containing 45 characters.

Putting the alphabet in a row allows for

abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMN

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 your *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 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 as follows:

- Each regular alphanumeric row (e.g. labels or status or program steps) requires 20 px vertically (cf. pp. xvif). There shall be at least one pixel separating it from the next row.
- Hence, each *menu* row takes (counting bottom-up) 1 + 20 + 1 + 1 = 23 px (the first pixel is for the distance to the black frame, the last for its upper frame line). Thus, three *menu* rows require 69 px. In <u>U→</u>, extra-large labels may appear (cf. p. 127): 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 minimum remain.
- Each regular numeric output requires 32 px vertically (cf. pp. xvif) plus 5 px separating it from the next such row. Thus, for 4 rows we need 4 × 32 + 3 × 5 = 143 px. We put the remaining 4 px below this output block.

- If there is a short alpha string in any stack *register*, its ground line is positioned where the ground line of the respective numeric output would have been.
- With an alpha string needing two rows, 20 + 1 + 20 + 5 = 46 px are required vertically where 37 px are available for one numeric row. So a second numeric row has to go here – up to three stack *registers* can be displayed only now.

  Two subsequent alpha strings of this kind, e.g. after **ENTER↑**, require 92 px. Three regular numeric rows cover 111 px, so here is no further loss. The ground line of the lowest alpha string is positioned where the ground line of the respective numeric output would have been. The other string is positioned in the center of the free space remaining.
- An alpha string needing three rows requires 67 px which are still covered by two regular numeric rows. Two such strings, however, push any other stack register out of the screen. Then string y shall have its ground line where the second numeric row would have it.
- In *PEM*, on the other hand, 147 = 7 × 21 corresponds to a block of 7 alphanumeric program rows.
- If there will be more space left, we will put it below the numeric block. This is for clear separation from the *menu section* and avoiding vertical output jitter. (gibt's diesen Fall überhaupt noch?)
- With SHOW, also pure numeric output may require more than one row – the small font will be used there as well. Cf. pp. 73 and xxi.

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.

Equations

Equations are entered in <u>EQN</u> 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 each operator like +, -, ·, ×, /, !, 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.

Closing 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.

Menus

Menu size corresponds to keystroke efficiency; optimum 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 functions. Besides function visibility, an operation presented in the unshifted row of top *menu view* is more efficient than a shifted function presented on the keyboard – if used more than just once.

Generally, I separated status setting from "acting" operations in different *views* or rows at least.

Number Range

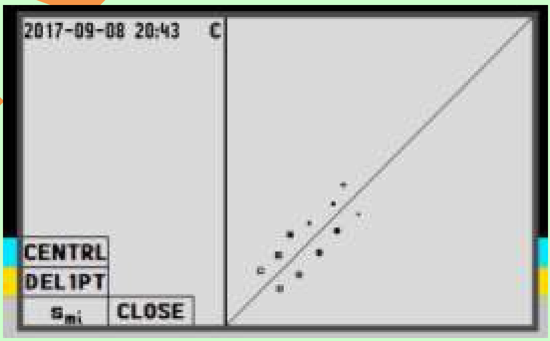
A number range up to 10^{99} is sufficient for almost all real-world problems – else common scientific calculators would feature a larger numeric range generally. So we can conclude that the *SP* number range (see p. 158) is sufficient by far for solving what has to be solved.

For sake of consistency, maximum numbers of different *data types* should match. I.e. the maximum absolute value allowed for a long integer should be approximately equal to the respective values for a real and a complex number. *DP* numbers are implemented for extended precision, not for extended range.

Plotting?

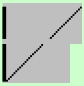
It is mentioned elsewhere we are not out for creating a graphing calculator. There is, however, a very useful application where a basic scatter plot of measured data points would support decision making significantly (see the third industrial statistics application example in *Section 2* of the *OM*). This would require the statistical data (e.g. max. 100 data points, i.e. 100 pairs of x and y values) to be stored point by point in a matrix, not just summed up as in the *WP 34S* and earlier *RPN* calculators.

Plotting could be called by a command PLOT stored in STAT displaying the data points collected in a quadratic diagram. Both axes shall reach from minimum value measured to maximum value measured (plus a little extension which can be calculated based on the data points). Axis scales are not required for analysis so I omitted them. Drawing area has to be quadratic (240 × 240 px for data, 242 × 240 px incl. the vertical axis). Hence *softkeys* can be positioned on one side of the diagram (max. 3 × 2 labels) still. The *status bar* would be partially overwritten by the diagram. See the sketch (to scale) for general screen layout and various data point symbols for checking visibility.



CLLCD is modified for clearing just the screen section required for the diagram.

Four characters are provided in small font for ‘drawing’ the vertical axis and the 45° line:



But both the axis and the line can be created using AGRAPH and PIXEL as well.

Data points can then be plotted using POINT (containing 3×3 px) – positioning them properly in the diagram, however, will require some background calculations best performed by a program. For POINT, also some of the control modes of *HP-42S* shall be implemented in analogy (the picture below is copied from the *HP-42S OM*, p. 137; settings 1 and 3 should do for our plotting):

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.

Softkey functions could be ...

- CENTRL for fitting the center line to the data points and plotting it for checking deviations from the 45° line (some background calculations in L.R. using ORTHOF for <u>orthogonal</u> regression are required for the plot, setting 1 in the picture above will do while plotting);
- DEL1PT turned obsolete;
- s_{mi} for calculating the minimum experimental standard deviation of the measuring instrument (some background calculations required again); and
- CLOSE for closing the plot screen, returning to normal display.

It may be beneficial to define a general origin for graphics at a location deviating from 0, 0 (i.e. the bottom left corner of the *LCD*) – the point 158, 0 may be a useful origin. This would allow for creating also other graphics than just the correlation diagrams mentioned above, while reserving a 'protected screen space' for up to six *softkeys*. Any user may do his own in this almost quadratic drawing area then, using the commands AGRAPH, CENTRL, CLLCD, CLOSE, PIXEL, PLOT, POINT, and s_{mi}.

Precision and Accuracy

As mentioned above more than once, there are inevitable errors in each calculation step, frequently caused by rounding to the internal finite precision the calculator features. Already a simple fraction like $^1/_3$ stored as an *SP* real number deviates from the truth by more than 3×10^{-17}. During calculations, such errors accumulate as elaborated e.g. in footnote 57.

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° *SP*, you can nevertheless get 10 digits precision in the result since the input of 9° is exact (but note you lose one digit precision 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 (seems *Free42* works alike since the standard test results match). Following these, this is implemented for the *WP 43S* as well.

Luckily, real-world problems are usually far less precisely defined than the internal precision of the *WP 43S*. Compare the set of physical and astronomical constants provided.

Prefixes

Prefixes [f] and [g] passed without any discussion for six years until 2019-06. Alternatively, prefixes [↰] and [↱] could have been chosen but their typography leaves less freedom for label placing.

Sorting in Detail

There is no international standard for sorting characters; we had to invent our own order. Sorting of *items*, variable *names*, alphanumeric strings, etc. on the *WP 43S* works as listed below, top down and left to right: [75]

␣ 0020	2003	2004	…	2007	2008	200a
0 0030	₀ 220e	° 00b0	⁰ 2070	₀ 2080		
1 0031	1 2027	½ 00bc	¼ 00bd	¹ 00b9	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			

[75] Sorting is a two-step procedure: Step 1 sorts the alphanumeric 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, SB, SCI, scw→kg, …, SLVQ, s_m, SMODE?, s_{mw}, SOLVE, …

The 4-digit number trailing each character in said 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.

Sorting is illustrated for the small font here. It holds also for the large font as far as characters are applicable.

8	0038	^{8}	2078	$_8$	2088	⑧	2467						
9	0039	^{9}	2079	$_9$	2089	⑨	2468						
⒑	2491	⑩	2469										
⑪	246a												
⑫	246b												
⑬	246c												
⑭	246d												
⑮	246e												
⑯	246f												
A	0041	a	0061	ª	00aa	Ⓐ	24b6	$_a$	2090	⒜	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	Ⓑ	24b7	⒝	249d						
C	0043	c	0063	Ⓒ	24b8	⒞	249e	Ç	00c7	ç	00e7		
		Ć	0106	ć	0107	Č	010c	č	010d	ℂ	2102	∁	2201
D	0044	d	0064	Ⓓ	24b9	⒟	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	I 24be	i 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	J 24bf	j 24a5			
K 004b	k 006b	K 24c0	k 24a6	k 2096		
L 004c	l 006c	L 24c1	l 24a7	l 2097	Ĺ 0139	ĺ 013a
			Ľ 013d	ľ 013e	Ł 0141	ł 0142
M 004d	m 006d	M 24c2	m 24a8	m 2098		
N 004e	n 006e	N 24c3	n 24a9	n 2099	Ñ 00d1	ñ 00f1
	Ń 0143	ń 0144	Ň 0147	ň 0148	ℕ 2115	
O 004f	o 006f	º 00ba	© 00a9	O 24c4	o 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	P 24c5	p 24ab	p 209a		
Q 0051	q 0071	Q 24c6	q 24ac	ℚ 211a		
R 0052	r 0072	r 24ad	R 24c7	Ŕ 0154	ŕ 0155	Ř 0158
					ř 0159	ℝ 211d
S 0053	s 0073	S 24c8	s 24ae	s 209b	Ś 015a	ś 015b
	Ş 015e	ş 015f	Š 0160	š 0161	ß 00df	
T 0054	t 0074	T 24af	⊤ 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	ᵧ 24ce	ᵧ 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	∑ 2211	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										
+	002b	⁺	207a	₊	208a	±	00b1						
-	002d	−	2212	⁻	207b	⁻¹	2072	₋	208b	∓	2213		
×	00d7	·	00b7	∙	2219	∘	2218	*	002b	∗	2217	*	208f
/	002f	∕	2215										
\	005c	∖	2216										
^	005e												
,	002c	,	2429										
.	002e	.	2428	…	2026								
!	0021	¡	00a1										
?	003f	¿	00bf										
:	003a	∶	2236	÷	00f7								
;	003b												
'	0027	‘	2018	’	2019	‚	201a	‛	201b				
"	0022	“	201c	”	201d	„	201e	‟	201f	«	00ab	»	00bb
@	0040												
_	005f												
~	007e	∼	0040										
→	2192	⇀	21c0										
←	2190	⇍	21cd										
↑	2191	⇧	21e7	△	21c9								
↓	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	> 226b
						≫ 226b
% 0025	$ 0024	€ 20ac	¢ 00a2	£ 00a3	¥ 00a4	§ 00a7
√ 221a	∝ 221d					
∞ 221e	∞ 209e	∞ 209f				
∫ 222b	∬ 222c	∮ 222e	∯ 222f			
⊙ 2299	⊚ 229a	⊙ 2068				
⊕ 2295	⊕ 2069					
∟ 221f	⊥ 22a5					
∠ 2220	∡ 2221					
⎙ 2399	⌛ 231b	⌚ 231a	242a	242c	242f	
# 0023						
UK 242d	US 242e					
∀ 2200	∂ 2202	∃ 2203	∄ 2204	∅ 2205	∆ 2206	∇ 2207
	∈ 2208	∉ 2209	∋ 220b	∌ 220c	∩ 2229	∪ 222a
2422	2423	2425	2426			

Stack Size

At a very early stage of this project (2013), *stack* size was discussed. At the bottom line, eight *stack registers* turn out being sufficient for solving any real-world mathematical, scientific, or engineering problem (cf. also the *WP 34S* and *WP 31S*).

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.

After all, we decided sticking to *RPN* as implemented on the *WP 34S* and *WP 31S*. It covers everything needed most easily. For special action support, 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 to keep 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 *automatic stack lift*, all other functions enable it. But compare INPUT on p. 43.

Structured Programming

In 2013, I suggested the following control structures:

- IF – THEN – ELSE – END,
- FOR – FROM – TO – END,
- REPEAT – UNTIL,
- 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;
- WHILE – DO BEGIN – END;

We refrained from implementing such commands since we had doubts about mixing keystroke programming and structured programming features.

UNDO

In 2013, UNDO was planned as it works in *HP-48*, recalling the *stack* as it was before executing last command. The *WP 31S*, on the other hand, features an UNDO recalling the entire calculator status as it was. It turned out that such a complete UNDO was viable here, too, so we implemented it.