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

  For the time being, the locations highlighted cyan are open construction sites – information is missing there or needs further discussion and investigation to be determined. Any contributions in this matter are highly appreciated.

HP is a registered trade mark of *Hewlett-Packard*.

The pictures on p. 159 and bottom of p. 160 were kindly supplied by *SwissMicros* as well as the drawing on p. 213, the picture on p. 209 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, MENUS, PREDEFINED VARIABLES and SYSTEM FLAGS are generally called by their *names*, printed capitalized in running text (*menus* underlined). Quoted text is printed blue (as well as translator's footnotes). *Specific terms, titles, trademarks, names* or *abbreviations* are printed in italics, *hyperlinks* in blue underlined italics. The latter will beam you to its target in the `.pdf` file – it cannot work in a printed copy for obvious reasons; thus such a link generally refers to a page number, to the *Table of Contents*

, or to a fully specified external address.

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

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

All this holds unless stated otherwise locally.

The following abbreviations are used throughout this manual:

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

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

BCD = binary coded decimal.

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

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*).

SRS = subroutine return stack (see *App. B* on pp. 163ff).

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 displayed and printed in routines) in column 1 and the keystrokes necessary to call them. Most *items* shall be picked from *menus* (see pp. 120ff). For such *items*, we list the keys calling the respective *menu*, the *prefix* of the respective *menu* row (if applicable), and the label as shown therein; we are confident you will find the corresponding *softkey*. *Items* stored in CONST are listed with their *names* only, however, since they are sorted alphabetically and will be explained in detail in a separate chapter below.

> There is an important difference between the *names* of *items* and their labels as printed on the bezel 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). These labels are not required to be unique.[1]

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

␣ 0…9 10 11…16 Aa…Zz Aα…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

---

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

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

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.

Operations working with the accumulated statistical data are marked light blue. 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 are most useful 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*;

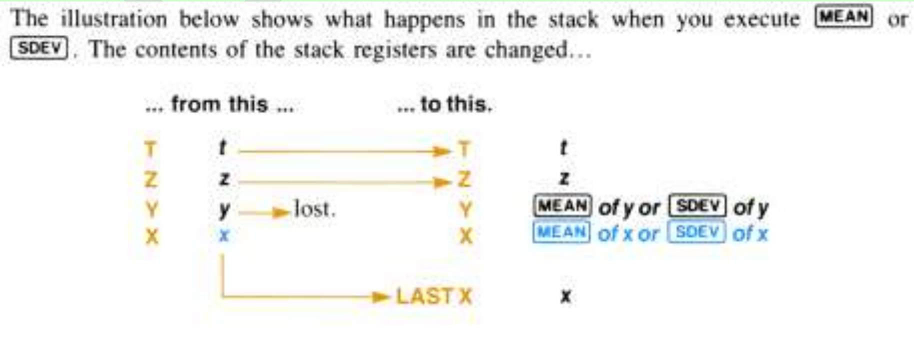
(−1) stands for functions pushing one object on the *stack* and

(−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.[3]

---

[3] 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:



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

   <u>n</u> 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* <u>address</u> or variable *name* which must be keyed in directly or picked from a *menu*, while

   <u>r</u> (or ***<u>s</u>***) 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

   <u>labl</u> 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.

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).

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.

The *data types* a particular function operates on are listed in { } under "remarks" if there are restrictions – cf. *App. B* on pp. 163ff. 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.[4]

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

Some 300 functions featured in your *WP 43S* are new compared to *HP's RPN* pocket calculators. Operations carrying familiar *names* but deviat-

[4] This applies for °C→°F, for instance: For *real* number input, output will stay *real*. For integer input, however, output will be *real*.

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

[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.

ing in their functionality from previous *HP RPN* calculators or the *WP 34S* are marked light red. [6]

Below, the functions checked already are highlighted green, those which didn't work yet (for whatever reason) are marked red. Green highlighting doesn't necessarily mean the function works correctly but its results look like in the right ballpark. What wasn't checked 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	f U→ f °C→°F etc.	(1) {2}; {1} → {2} Convert temperatures. See pp. 144ff.
10^x	g 10^x	(1) {1, 2, 3, 8*, 9*, 10} Returns 10^x, the inverse of $\lg(x)$.
1COMPL	f BITS ▼ 1COMPL g INTS ▲ 1COMPL	(0) Sets 1's complement mode for operations on *short integers*. Indicated in the *status bar*. See *Sect. 2* of the *OM*.
$^1/_x$	$^1/_x$	(1) {2, 3, 8*, 9*}; {1} → {2} Inverts the number x or all elements of the matrix x.
2COMPL	f BITS ▼ 2COMPL g INTS ▲ 2COMPL	(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} Returns 2^x.
$\sqrt[3]{x}$	g EXP f $\sqrt[3]{x}$	(1) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}) Returns the cube root of x. Roots of non-cube *long integers* will return *reals*.

[6] We did not compare the *RPL* calculators of the last three decades nor the *HP Prime*. They are exceeding the realm of shirt pocket calculators.

Item	Keystrokes	Remarks (see pp. 12ff for general information)	
a	f CONST a	(−1) { } → {2} *Gregorian* year in *days* and *Bohr* radius in *meter*.	
a_0	f CONST a_0		
ABS	f CATALOG FCNS ABS	Points to |x| on p. 99.	Maintained for backward compatibility only.
ACOS	f CATALOG FCNS ACOS	Points to arccos on p. 18.	
ac→m^2	f U→ f A: acre → m^2	(1) {2}; {1} → {2} Convert areas. See pp. 144ff.	
ac_{US}→m^2	f U→ f A: $acre_{US}$ → m^2		
ADV	f ADV	*Menu*. See p. 127.	
AGM	g X.FN AGM	(2) {2, 3}; {1} → {2} Returns the *arithmetic-geometric mean* of ***x*** and ***y***. Will throw an error for ***x*** or ***y*** being negative. See p. 247 for more.	
AGRAPH	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 pp. xxxiii ff). 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.	
ALL	g DISP ALL ***n***	(0) Sets the numeric display format to show all decimals of *real* or *complex* numbers whenever displayable (trailing decimal zeros will not be shown). ALL **0** works like ALL in *HP-42S* almost.	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		For $x \geq 10^{16}$ (or earlier for *complex* numbers), display will switch to SCI or ENG with the maximum number of necessary decimals displayable using the large font (see ALLSCI). 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. 63.
a_{Moon}	f CONST 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, Section 2*).
		(2) {1, 2} → {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. 121f.
arccos	TRI arccos	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $arccos(x)$.[7]
arcosh	g EXP g arcosh TRI g arcosh	(1) {2, 3, 8*, 9*} Returns $arcosh(x)$.

[7] 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*}; {1, 2} → {4}; Returns the tagged angle $arcsin(x)$.[8]
arctan	TRI arctan	(1) {3, 8*, 9*}; {1, 2} → {4}; Returns the tagged angle $arctan(x)$.[9]
arsinh	g EXP g arsinh TRI g arsinh	(1) {2, 3, 8*, 9*} Returns $arsinh(x)$.
artanh	g EXP g artanh TRI g artanh	(1) {2, 3, 8*, 9*} 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 *Section 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*.

[8] 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*.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ATAN	f CATALOG FCNS ATAN	Points to ARCTAN above. Maintained for backward compatibility only.
atm→Pa	f U→ F&p: ▼ atm→Pa	(1) {2}; {1} → {2} Convert pressures and distances. See pp. 144ff.
au→m	f U→ x: au→m	
A:	f U→ f A:	*Submenu*. See p. 144.
$a_⊕$	f CONST $a_⊕$	(−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	f U→ F&p: bar→Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 144ff.
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. Measurement resolution is 1 mV
bbl→m^3	f U→ f V: f barrel → m^3	(1) {2}; {1} → {2} Converts volumes. See pp. 144ff.
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	f I/O BEEP	(0) Sounds a sequence of four tones. See also TONE.
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}, $\hat{x}$, and $\hat{y}$. You can accelerate computation of these functions significantly by excluding fit models making no sense for your data (e.g. for physical or technical reasons). The parameter ***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. Note ORTHOF is <u>not</u> part of the set of models under investigation. See pp. 217ff for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Binom Binom_e Binom_p Binom^{-1}	g PROB g Binom: Binom etc.	(1) {2} *Binomial distribution* with the number of successes ***g*** in **X**, the probability of a success ***p₀*** in **I**, and the sample size ***n*** in **J**. See p. 217 for more. Binom^{-1} returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, ***p₀*** in **I** and ***n*** in **J**.
Binom:	g PROB g Binom:	*Submenu*. See p. 130.
BITS	f BITS	*Menu*. See p. 124.
B_n	g X.FN B_n	(1) {1, 2} B_n and B_n^* return the Bernoulli number for an integer ***n*** > 0 given in **X**, working with different definitions (see both formulas on p. 215).
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	f U→ E: Btu→J	(1) {2}; {1} → {2} Converts energies. See pp. 144ff.
c c_1 c_2	f CONST c etc.	(−1) { } → {2} Speed of light in vacuum in *meter/second*; first and second radiation constants in *Planck's* Law (see p. 137).
cal→J	f U→ E: cal→J	(1) {2}; {1} → {2} Converts energies. See pp. 144ff.

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. 121ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Cauch Cauch_e Cauch_p Cauch^{-1}	g PROB f Cauch: Cauch etc.	(1) {2} *Cauchy-Lorentz* (a.k.a. *Lorentz* or *Breit-Wigner*) *distribution* with the location x_0 specified in **I** and the shape γ in **J**. See p. 220 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. 130.
CauchF	f STAT ▼ f CauchF	(0) Selects the Cauchy (a.k.a. Lorentz) peak fit model. Relevant for CORR, COV, L.R., s_{XY}, $\hat{x}$, and $\hat{y}$. See pp. 226ff 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} → {1} Returns the smallest integer ≥ x. Cf. FLOOR.
CF	g FLAGS CF *n* g MODE 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. 121ff.
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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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 **0.**
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. 120.
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 $\geq x$ and $\geq y$ 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. 127.
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 f 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 *stack register* **X**, disabling *automatic stack lift*. The shortcut works for closed ***x*** only. Cf. CLREGS.
	←	
CLΣ	g CLR CLΣ	(0) Clears the statistical summation *registers* and releases the memory allocated for them (see p. 166).
	f STAT g CLΣ	
CNST	g P.FN f CNST *n*	(−1) { } → {2} Returns the constant stored at position ***n*** in CONST (see below and pp. 135ff). Allows for indirectly addressing these constants, also beyond ∞.
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} See pp. 215f for the formula.
CONJ	g CPX conj	(1) {3, 9*} Returns the *complex conjugate* of ***x***.
CONST	f CONST	*Menu*. See CNST above and pp. 135ff.
CONVG?	g TEST g CONVG? *r*	(0) {2} Checks for convergence by comparing ***x*** and ***y*** as determined by the lowest five bits of ***r***. a) The very lowest 2 bits set the tolerance limit: 0 = 10^{-14}, 1 = 10^{-24}, 2 = 10^{-32}.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the numbers x and y relatively, 1 = compare them absolutely. c) The top bit tells how special numbers are treated: 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. 226ff for more.
cos	TRI cos	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the cosine of the angle in **X** (see *Section 2* of the *OM* for details).
cosh	g EXP g cosh TRI g cosh	(1) {2, 3, 8*, 9*} Returns the hyperbolic cosine of x.
COV	f STAT ▲ cov	(−1) { } → {2} Returns the population covariance for the two data sets entered via Σ+, depending on the curve fit model selected. See s_{XY} for the sample covariance and pp. 217ff for more.
CPX	g CPX	*Menu*. See p. 127.
CPXS	f CATALOG VARS CPXS	*Submenu* of *complex* variables defined at execution time. See pp. 121f.
CPX?	g TEST ▲ CPX?	(0) Checks if x is *complex*. Returns true if **X** contains data of type 3 or 9 with nonzero *imaginary* part.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
CROSS	f MATX f cross	(2) {8} Requires two *real* 2D or 3D vectors in **X** and **Y** and returns their cross product. Crossing of 2D vectors works as for *complex* numbers.
	g CPX cross	(2) {3} → {2} When two *complex* numbers are crossed, your *WP 43S* simply returns a *real* number that is equal to the signed *magnitude* of the resulting moment vector.
ct→kg	f U→ m: f carat → kg	(1) {2}; {1} → {2} Convert masses. See pp. 144ff.
cwt→kg	f U→ m: cwt→kg	
CX→RE	CC (works in run mode only)	(−1) {3} → {2}; {9} → {8} Cuts a closed *complex* object ***x***, putting either • (for ∟) its *real* part in **Y** and its *imaginary* part in **X** or • (for ⊙) *magnitude* in **Y** and *phase* in **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. 121f.
DATE→	g CLK DATE→	(−2) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real* number in corresponding format) and pushes its three components as integers on the *stack*. Reversible by →DATE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DAY	g CLK f DAY	(1) {2, 6} → {1} Assumes x containing a *date* in the format selected (or a *real* number in corresponding format) and extracts the day.
DBLR	g INTS g DBLR etc.	{10} Double *word* length commands for remainder, multiplication and division.[10] DBLR and DBL / accept a double size dividend in **Y** and **Z** (most significant bits in **Y**), the divisor in **X** as usual, and return the result in **X**. DBL× takes x and y as factors as usual but returns the product in **X** and **Y** (most significant bits in **X**).
DBL×		
DBL/		
dB→fr	f U→ ▲ dB → field ratio	(1) {2}; {1} → {2} Convert ratios. See pp. 144ff.
dB→pr	f U→ ▲ dB → power ratio	
DEC	f LOOP f DEC *r*	(0) {1, 2, 10} Decrements r by 1. Does not load **L** even for target address **X**.
DECOMP	f PARTS DECOMP	(−1) {1, 2} → {1} Decomposes x (after converting it to an *improper fraction*, if applicable), returning a *stack* [***denominator(x), numerator(x),*** …]. **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→	g L→ f DEG→	(1) {1, 2} → {4} Converts angles as described on p. 150.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
DENMAX	g MODE f DENMAX	(1) Works like /c on *HP-35S*, but the maximum legal denominator is 9 999. For $x < 1$ or $x > 9\,999$, DENMAX will be set to 9 999. 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. 99. Maintained for backward compatibility only.
DIGITS	f CAT. f DIGITS	*Submenu* of digits defined. See pp. 121ff.
DISP	g DISP	*Menu*. See p. 127.
DOT	g CPX dot	(2) {3} → {2} 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} 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} 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 DSTACK parameter set. In any case, command input will be echoed directly below the *status bar*. This command is for old-school calculator users who may feel distracted by a multitude of *stack registers* displayed changing simultaneously while only the lowest ones are really relevant.
DSZ	f LOOP DSZ *r*	(0) {1, 2, 10} 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→	g L→ f D.MS→	(1) {1, 2} → {4} Convert angles as described on pp. 150f.
D.MS→D	g L→ g D.MS→D	
D.MY	g CLK ▲ D.MY	(0) Sets the format `dd.mm.yyyy` for *dates*.
D→D.MS	g L→ g D→D.MS	(1) {1, 2} → {4} Converts angles as described on p. 150.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
D→J	g CLK f D→J	(1) {2, 6} → {1} Assumes *x* containing a *date* in the format selected (or a *real* number in corresponding format) and converts it to a *Julian day number*[11] according to the J/G setting.
D→R	g L→ g D→R	(1) {1, 2} → {4} Converts angles as described on p. 150.
e	f CONST e	(−1) { } → {2} Elementary charge in *coulomb* and *Euler's e*.
e_E	f CONST 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*).

[11] 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. 99.
ENTER↑	ENTER↑	(−1) Separates two entries in input. Copies *x* into **Y**, disabling *automatic stack lift*. See p. 117 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. 128.
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}; {1} → {2} Returns the error function or its complement. See pp. 247ff 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. 177ff 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*}; {1, 10} → {2} Returns e^x.
EXITALL	g P.FN P.FN2 EXITall	(0) Exits all *menus*.
EXP	g EXP	*Menu*. See p. 128.
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. 217ff for more.
Expon Expon_e Expon_p Expon^{-1}	g PROB f Expon: Expon etc.	(1) {2} *Exponential distribution* with the rate λ in **J**. See pp. 217ff for more. Expon^{-1} returns the survival time t_s for a given probability p in **X**, with λ in **J**.
Expon:	g PROB f Expon:	*Submenu*. See p. 130.
EXPT	f PARTS EXPT	(1) {1, 2} → {1} Returns the exponent h of the number $x = m \cdot 10^h$ displayed. Cf. MANT.
e^x-1	g EXP f e^x-1	(1) {2, 8*} For $x \approx 0$, this returns a more accurate result for the fractional part than e^x does.
E:	f U→ E:	*Submenu*. See p. 143.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
F	f CONST F	(−1) { } → {2} *Faraday* constant in *coulomb per mol*.
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 fonts designed for your *WP 43S*.
FCNS	f CATALOG FCNS	*Submenu* of provided functions. See pp. 121ff.
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} *Fisher's F distribution*. $F_e(x)$ equals Q(F) on *HP-21S*. The degrees of freedom are specified in **I** and **J**. See pp. 217ff for more.
FF	g FLAGS FF *n*	(0) Flips the *flag* specified.
FIB	g X.FN f FIB	(1) {1} Returns the Fibonacci number (see pp. 215f).
		(1) {2, 3} Returns the extended Fibonacci number.
FILL	f FILL	Copies ***x*** to all *stack registers*.
FIN	g FIN	*Menu*. See p. 128.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
FIX	g DISP FIX *n*	(0) Sets fixed point display format (see the *OM*, *Sect. 2*).
FLAGS	g FLAGS	*Menu*. See p. 128.
FLASH	f CATALOG PROGS FLASH	*Submenu* of global labels defined at execution time. See pp. 121f.
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} → {1} Returns the greatest integer ≤ x. Cf. CEIL.
fm.→m	f U→ x: ▲ fathom → m	(1) {2, 11; {1} → {2} Converts distances. See pp. 144ff.
FP	f PARTS FP	(1) {1, 2, 8*, 10} 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	f U→ ▲ field ratio→dB	(1) {2}; {1} → {2} Converts ratios. See pp. 144ff.
FS?	g FLAGS FS? *n*	(0) Tests if the specified *flag* is set.
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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ft.→m	f U→ x: f ft.→m	(1) {2}; {1} → {2} Convert distances and volumes, respectively. See pp. 144ff.
ft$_{US}$→m	f U→ x: ▲ survey foot$_{US}$ → m	
fz$_{UK}$→m^3	f U→ f V: f floz$_{UK}$ → m^3	
fz$_{US}$→m^3	f U→ f V: f floz$_{US}$ → m^3	
F$_\alpha$	f CONST F$_\alpha$ etc.	(−1) { } → {2} *Feigenbaum's* α and δ.
F$_\delta$		
F:	g PROB F:	*Submenu*. See p. 130.
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} → {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:	f U→ F&p:	*Submenu*. See p. 144.
G	f CONST G	(−1) { } → {2} *Newtonian* constant of gravitation in $m^3/kg\ s^2$; also called γ by other authors.
G$_0$	f CONST G$_0$	Conductance quantum in *siemens*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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 **2**, **1**, or GAP **0**, <u>no</u> group separators will be displayed in any numbers 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. 226ff for more.
G_C	f CONST 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.[12] This will always be positive.
g_d g_d^{-1}	g X.FN f g_d etc.	(1) {2, 3} ; {1} → {2} Returns the Gudermannian function or its inverse. See p. 248 for details.
g_e	f CONST g_e	(−1) { } → {2} *Landé's* electron g-factor.

[12] 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)
Geom $Geom_e$ $Geom_p$ $Geom^{-1}$	g PROB g Geom: Geom etc.	(1) {2} *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. 217ff 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. 130.
gl_{UK}→m^3 gl_{US}→m^3	f U→ f V: gl_{UK}→m^3 etc.	(1) {2}; {1} → {2} Convert volumes. See pp. 144ff.
$GM_\oplus$	f CONST $GM_\oplus$	(−1) { } → {2} *Newtonian* constant of gravitation times the Earth's mass with its atmosphere included according to *WGS84*.[13] Displayed in m^3/s^2
GRAD	g MODE GRAD	(0) Sets the *ADM* to *grad* (a.k.a. *gradian* or *gon*). Indicated in the *status bar*.
GRAD→	g L→ f GRAD→	(1) {1, 2} → {4} Converts angles as described on pp. 150f.
GTO	f GTO ***labl***	(0) In *PEM*, inserts an unconditional branch to ***labl***. Else positions the program pointer to ***labl***.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)	
GTO.	f GTO . *n*	(0) Puts the program pointer …	to step ***n*** (specify up to four digits until becoming unambiguous in used program memory).
	f GTO . *labl*		to the global label specified.
	f GTO . ▲		directly after previous END, going to the top of *current program* (see *Sect. 3* of the *OM*).
	f GTO . ▼		directly after next END, going to the top of next program.
	f GTO . .		to the end of used program memory in *RAM* (i.e. to the final END statement).
$g_\oplus$	f CONST $g_\oplus$	(−1) { } → {2} Standard earth acceleration in $^{m}/_{s^2}$.	
h	f CONST h	(−1) { } → {2} *Planck* constant in *joule-second*.	
ha→m²	f U→ f A: ha → m²	(1) {2}; {1} → {2} Converts areas. See pp. 144ff.	
H_n	g X.FN Orthog H_n	(2) {2}; {1} → {2} *Hermite polynomials* for probability (H_n) and physics (H_{np}). See p. 242 for details.	
H_{nP}	g X.FN Orthog f H_{nP}		
hp_E→W	f U→ P: hp_E→W etc.	(1) {2}; {1} → {2} Convert powers. See pp. 144ff.	
hp_M→W			
hp_{UK}→W			

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Hyper Hyper_e Hyper_p Hyper^{-1}	g PROB g Hyper: Hyper etc.	(1) {2} *Hypergeometric distribution* with the number of successes ***g*** in $\mathbf{X}$, the probability of a success ***p₀*** in $\mathbf{I}$, the sample size ***n*** in $\mathbf{J}$, and the batch size ***n₀*** in $\mathbf{K}$. See pp. 217ff for the formula. Hyper^{-1} returns the maximum number of successes ***m*** for a given probability ***p*** in $\mathbf{X}$, ***p₀*** in $\mathbf{I}$, ***n*** in $\mathbf{J}$, and ***n₀*** in $\mathbf{K}$.
Hyper:	g PROB g Hyper:	*Submenu*. See p. 130.
HypF	f STAT ▼ f HypF	(0) Selects the hyperbolic fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 226ff for more.
ℏ	f CONST ℏ	(−1) { } → {2} $= h/2\pi$, reduced *Planck* constant in *joule-second* (a.k.a. *Dirac* constant).
IDIV	g INTS f IDIV	(2) {1, 10}; {2} → {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} Like IDIV but also returns the remainder in $\mathbf{Y}$. See the *OM*, *Section 2*, for the resulting *data types* of quotient and remainder.
iHg→Pa	f U→ F&p: f in.Hg →Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 144ff.
Im	g CPX f Im f PARTS g Im	(1) {3} → {2}; {9} → {8}; Returns the imaginary part of x. Cf. RE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
INC	f LOOP f INC *r*	(0) {1, 2, 10} 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. 128.
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. 128.
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. 99. Maintained for backward compatibility only.
in.→m	f U→ x: g in.→m	(1) {2}; {1} → {2} Converts distances. See pp. 144ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
IP	f PARTS IP	(1) {1, 8*, 10}; {2} → {1} Returns the integer part of ***x***. Cf. FP.
ISE	f LOOP ISE s	(0) {1, 2, 10} Given `cccccc.fffii` in the source ***s***, ISE increments ***s*** by `ii`, skipping next program step if `cccccc` ≥ `fff` then. If ***s*** 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 s	(0) {1, 2, 10} Works like ISE but skips if `cccccc` > `fff`.
ISZ	f LOOP ISZ s	(0) {1, 2, 10} Increments ***s*** by 1, skipping next program step if then $\lvert s \rvert < 1$. 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} Returns the *regularized Beta function*. See p. 248 for more.
$I\Gamma_p$	g X.FN f $I\Gamma_p$ etc.	(2) {1, 2} Returns the *regularized Gamma function* (one of two kinds).
$I\Gamma_q$		
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	f I/O	*Menu*. See p. 128.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$J_y(x)$	g X.FN g $J_y(x)$	(2) {2}; {1} → {2} $J_y(x)$ returns the *Bessel function of first kind* and order y. See p. 249 for details.
J+	f MATX ▲ J+	(1) Increments or decrements the column index of the indexed matrix. If 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 J→cal	f U→ E: J→Btu etc.	(1) {2}; {1} → {2} Convert energies. See pp. 144ff.
J→D	g CLK f J→D	(1) {1} → {6} Takes x as a *Julian day number*[14] and converts it to a common *date* according to J/G (see above) and the date format selected.
J→Wh	f U→ E: J→Wh	(1) {2}; {1} → {2} Converts energies. See pp. 144ff.
k	f CONST k	(−1) { } → {2} *Boltzmann* constant in *joule per kelvin*.
KEY		See KEYG and KEYX below.

[14] 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	f U→ m: f kg → carat	(1) {2}; {1} → {2} Convert masses. See pp. 144ff.
kg→cwt	f U→ m: kg→cwt etc.	
kg→lb.		
kg→oz		
kg→scw	f U→ m: f kg → sh.cwt	
kg→sto	f U→ m: f kg → stone	
kg→s.t	f U→ m: ▲ kg → short ton	
kg→ton	f U→ m: ▲ kg→ton	
kg→trz	f U→ m: f kg → tr.oz	
K_J	f CONST 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	f U→ F&p: lbf→N	(1) {2}; {1} → {2} Converts forces. See pp. 144ff.
LBL	g LBL *labl*	(0) Identifies programs and routines for execution and branching. Read more about labels and specifying them in *Section 3* of the *OM*.
LBL?	g TEST g LBL? *labl*	(0) Tests for existence of the label specified, anywhere in program memory. See LBL for more.
lb.→kg	f U→ m: lb.→kg	(1) {2}; {1} → {2} Converts masses. See pp. 144ff.
LCM	g INTS f LCM	(2) {1; 10} Returns the *Least Common Multiple* of x and y.[15] This will always be positive.

[15] See also GCD. Remember school?

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LEAP?	g TEST f LEAP?	(0) {2, 6} Assumes ***x*** containing a *date* in the format selected (or a *real* number in corresponding format), extracts the year, and tests for a leap year.
LgNrm LgNrm_e LgNrm_p LgNrm^{-1}	g PROB f LgNrm: LgNrm etc.	(1) {2ß} *Log-normal distribution* with $\mu = ln\,\bar{x}_g$ specified in **I** and $\sigma = ln\,\varepsilon$ in **J**. See $\bar{x}_g$ and ε below and pp. 217ff for more. LgNrm^{-1} returns ***x*** for a given probability ***p*** in **X**, with ***μ*** in **I** and ***σ*** in **J**.
LgNrm:	g PROB f LgNrm:	*Submenu*. See p. 130.
LinF	f STAT ▼ LinF	(0) Selects the linear fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 217ff 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 **Y** and the count of bit-shifts necessary to left justify the *word* in **X**. **Example** for *word* size 8: $1\,0110_2$ LJ returns ***x*** = 3 and ***y*** = $1011\,0000_2$
L_m $L_{m\alpha}$	g X.FN Orthog L_m etc.	(2) {2}; {1} → {2} *Laguerre polynomials* and *Laguerre's generalized polynomials*. See pp. 232f for more.
LN	ln	(1) {2, 3, 8*, 9*}; {1, 10} → {2} Returns the natural logarithm of ***x***.
LNβ	g X.FN g lnβ	(2) {2, 3}; {1} → {2} Returns the natural logarithm of *Euler's* Beta function (see p. 90).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LNΓ	g X.FN g lnΓ g PROB ▲ lnΓ	(1) {2, 3}; {1} → {2} Returns the natural logarithm of $\Gamma(x)$ (see p. 91). Allows also for calculating really great factorials (see an example in the *OM*).
LN(1+x)	g EXP f ln 1+x	(1) {2, 8*} For $x \approx 0$, this returns a more accurate result for the fractional part than $\ln(x)$ does.
LOAD	f I/O LOAD	Restores the entire backup from *FM* and returns `Backup restored`. Thus, LOAD = LOADP + LOADR + LOADSS + LOADΣ.[16]
LOADP	f I/O LOADP	(0) Loads the entire program memory from backup and appends it to the programs already in *RAM* (if there is sufficient space – else an error will be thrown).[16]
LOADR	f I/O LOADR	(0) Recovers the numbered general purpose *registers* from backup. Lettered *registers* will not be recalled.[16]
LOADSS	f I/O LOADSS	(0) Recovers the system state from backup.[16]
LOADΣ	f I/O LOADΣ	(0) Recovers the statistical summation *registers* from backup. Throws an error if there are none.[16]
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*.
LocR?	g INFO f LocR?	(−1) { } → {1} Returns the number of *local registers* currently allocated.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
LOG_{10}	g lg	(1) {1, 2, 3, 8*, 9*, 10} ({1} → {2}) Returns the logarithm of ***x*** for base 10.
LOG_2	g EXP lb x	(1) {1, 2, 3, 8*, 9*, 10} ({1} → {2}) Returns the logarithm of ***x*** for base 2.
LogF	f STAT ▼ LogF	(0) Selects the logarithmic curve fit model. Relevant for CORR, COV, L.R., s_{XY}, x̂, and ŷ. See pp. 217ff for more.
Logis $Logis_e$ $Logis_p$ $Logis^{-1}$	g PROB f Logis: Logis etc.	(1) {2} *Logistic distribution* with ***μ*** given in **I** and ***s*** in **J**. See pp. 217ff for details.
Logis:	g PROB f Logis:	*Submenu*. See p. 130.
LOG_xy	g EXP log_xy	(2) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}) Returns the logarithm of ***y*** for the base ***x*** .
LOOP	f LOOP	*Menu*. See p. 129.
l_{PL}	f CONST l_{PL}	(−1) { } → {2} *Planck* length in *meter*.
ly→m	f U→ x: ly→m	(1) {2}; {1} → {2} Converts distances. See pp. 144ff.
L.INTS	f CATALOG VARS L.INTS	*Submenu* of *long integer* variables defined at execution time. See pp. 121ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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. 217ff for more.
m²→ac	f U→ f A: m² → acre	(1) {2}; {1} → {2} Convert areas. See pp. 143ff.
m²→ac$_{US}$	f U→ f A: m² → acre$_{US}$	
m²→ha	f U→ f A: m² → ha	
m³→bbl	f U→ f V: f m³ → barrel	(1) {2}; {1} → {2} Convert volumes. See pp. 143ff.
m³→fz$_{UK}$	f U→ f V: f m³ → floz$_{UK}$ etc.	
m³→fz$_{US}$		
m³→gl$_{UK}$	f U→ f V: m³→gl$_{UK}$ etc.	
m³→gl$_{US}$		
MANT	f PARTS MANT	(1) {2}; {1} → {2} Returns the mantissa ***m*** of the number $x = m \cdot 10^h$ displayed. Cf. EXPT.
MASKL	f BITS MASKL ***n***	(−1) { } → {10} Work like MASKL and MASKR on *HP-16C*, but with the mask length (or its address) following the command instead of being taken from **X**. Thus, the mask is pushed on the *stack*. MASKL **0** and MASKR **0** return 0.
MASKR	f BITS MASKR ***n***	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		Example: For WSIZE **8**, MASKL **3** returns a mask *word* $1110\ 0000_2$. Use it e.g. for extracting the top three bits of an arbitrary *byte* via AND.
MATRS	f CATALOG VARS MATRS	*Submenu* of matrix variables defined at execution time. See pp. 121f.
MATR?	g TEST ▲ MATR?	(0) Checks if *x* is a *real* or *complex* matrix.
MATX	f MATX	*Menu*. See p. 129.
Mat_X	f MATX SIM EQ Mat X	(−1) Returns the solution vector of a system of linear equations (see the *OM*, *Sect. 2*).
max	g X.FN g max	(2) {1, 2, 4, 5, 6, 7, 10} Returns the maximum of ***x*** and ***y***.
m_e	f CONST m_e	(−1) { } → {2} Electron mass in *kilogram*.
MEM?	g INFO MEM?	(−1) { } → {1} Returns the number of free *bytes* in program memory, 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. 121ff.
min	g X.FN g min	(2) {1, 2, 4, 5, 6, 7, 10} Returns the minimum of ***x*** and ***y***.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MIRROR	f BITS f MIRROR	(1) {10} Reflects the bit pattern in ***x*** (e.g. $0001\ 0111_2$ would become $1110\ 1000_2$ for *word* size 8).
mi.→m	f U→ x: f mi.→m	(1) {2}; {1} → {2} Converts distances. See pp. 143ff.
M_{Moon}	f CONST M_{Moon} etc.	(−1) { } → {2} Mass of the Moon in *kilogram*; neutron mass in *kilogram*; neutron to proton mass ratio.
m_n		
m_n/m_p		
MOD	g MOD g INTS f MOD	(2) {1, 2, 10} Returns $y \bmod x$ (modulo, see *Section 2* of the *OM* for examples). Cf. RMD.
MODE	g MODE	*Menu*. See p. 129.
MONTH	g CLK f MONTH	(1) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real* number in corresponding format) and extracts the month.
m_p	f CONST m_p etc.	(−1) { } → {2} Proton mass and *Planck* mass in *kilogram*; proton to electron mass ratio.
m_{PL}		
m_p/m_e		
MSG	g P.FN P.FN2 f MSG	(0) {1, 2} Throws the (*temporary*) *error message* specified by the integer part of ***x***. Cf. ERR. See *App. C* on pp. 177ff for the respective error codes.
m_u	f CONST m_u	(−1) { } → {2} Atomic mass constant in *kilogram* and its energy equivalent in *joule*.
m_uc^2	f CONST m_uc^2	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
MULπ	g MODE MULπ	(0) Sets the *ADM* to *multiples of π*. Indicated in the *status bar*.
MULπ→	g L→ f MULπ→	(1) {1, 2} → {4} Converts angles as described on pp. 150f.
MVAR	g P.FN 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, Section 6.*
Myα	f CAT. CHARS Myα	User *menu* in *AIM*. See the *OM, Section 6.*
m_μ	f CONST 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.
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*).[17] See *Section 2* of the *OM*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
M.EDIN	f MATX f EDITN ***name***	(2) Opens a named matrix using the *Matrix Editor* (like MATRIX EDITN in *HP-42S*).[17] See *Section 2* of the *OM*.
M.EDIT		*Submenu* for matrix editing, called by M.EDI or M.EDIN. See p. 129.
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.
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.

[17] 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.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 the *OM*, *Sect. 2*.
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:	f U→ m:	*Submenu*. See p. 143.
m→au	f U→ x: m→au	(1) {2}; {1} → {2} Convert distances or heights. See pp. 143ff.
m→fm.	f U→ x: ▲ m → fathom	
m→ft.	f U→ x: f m→ft.	
m→ft$_{US}$	f U→ x: ▲ m → survey foot$_{US}$	
m→in.	f U→ x: g m→in.	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
m→ly	f U→ x: m→ly	
m→mi.	f U→ x: f m→mi.	
m→nmi.	f U→ x: f m→nmi.	
m→pc	f U→ x: m→pc	
m→pt.	f U→ x: g m → point	
m→yd.	f U→ x: g m→yd.	
m☉ m⊕	f CONST m☉ etc.	(−1) { } → {2} Masses of the Sun and Earth in *kilogram*.
N_A	f CONST 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	f CONST 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} *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. 217ff for more information.
NBin:	g PROB g NBin:	*Submenu*. See p. 130.
NEIGHB	g INFO f NEIGHB	(2) {1} Returns … • $x+1$ for $x<y$; • x for $x=y$; • $x-1$ for $x>y$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		(2) {2} Returns the nearest machine-representable number to x in the direction towards y in the mode set. For • … $x < y$, it is the machine successor of x; • … $x = y$, it is y; • … $x > y$, it is the machine predecessor of x. NEIGHB may be useful investigating numeric stability (see NEIGHBOR in the *HP-71 Math Pac*).
NEXTP	g X.FN g NEXTP	(1) {1, 10}; {2} → {1} Returns the next prime number greater than $\lvert IP(x) \rvert$. See also PRIME? on p. 61.
nmi.→m	f U→ x: f nmi.→m	(1) {2}; {1} → {2} Converts distances. See pp. 143ff.
NOP	g P.FN P.FN2 f NOP	'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} *Normal distribution* with an arbitrary mean $\boldsymbol{\mu}$ given in **I** and a *standard deviation* $\boldsymbol{\sigma}$ in **J**. See *Sect. 2* of the *OM* for an application example and pp. 217ff for more. Norml^{-1} returns x for a given probability p in **X**, with $\boldsymbol{\mu}$ in **I** and $\boldsymbol{\sigma}$ in **J**.
Norml:	g PROB Norml:	*Submenu*. See p. 130.
NOT	f BITS NOT	(1) {10} Inverts x bit-wise as on *HP-16C*. (1) {1, 2} → {1} Returns 1 for $x = 0$, and 0 for $x \neq 0$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
nΣ	g Σ n	(−1) { } → {1} Recalls the number of accumulated data points.
N→lbf	f U→ F&p: N→lbf	(1) {2}; {1} → {2} Converts forces. See pp. 143ff.
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}, x̂, and ŷ. See pp. 226ff for more.
ORTHOG	g X.FN Orthog	*Submenu*. See p. 132.
oz→kg	f U→ m: oz→kg	(1) {2}; {1} → {2} Converts masses. See pp. 143ff.
P_0	f CONST P_0	(−1) { } → {2} Standard atmospheric pressure in *pascal*.
ParabF	f STAT ▼ f ParabF	(0) Selects the parabolic fit model. Relevant for COV, CORR, L.R., s_{XY}, x̂, and ŷ. See pp. 226ff for more.
PARTS	f PARTS	*Menu*. See p. 129.
PAUSE	g P.FN PAUSE *n*	(0) Within a routine running, refreshes the display and pauses program execution for ***n*** ticks (s. TICKS), with 0 ≤ ***n*** ≤ 99. The pause will terminate early when you press a key.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Pa→atm	f U→ F&p: ▼ Pa→atm	(1) {2}; {1} → {2} Convert pressures. See pp. 144ff.
Pa→bar	f U→ F&p: Pa→bar	
Pa→iHg	f U→ F&p: f Pa → in.Hg	
Pa→psi	f U→ F&p: Pa→psi	
Pa→tor	f U→ F&p: f Pa → torr	
pc→m	f U→ x: pc→m	(1) {2}; {1} → {2} Converts distances. See pp. 144ff.
PERM	g PROB P_{yx}	(2) {1} Returns the number of possible arrangements (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 different orders of the same ***x*** *items* are counted separately. Cf. COMB.
		(2) {2, 3} See pp. 215f 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 numbers in the **X**- and **Y**-*registers*. See AGRAPH on p. 17 for more.
PLOT	f STAT g PLOT	(0) Plots the ***n*** data points given by the ***n***×2 matrix ***x***. See p. xxiii for more.
P_n	g X.FN Orthog P_n	(1) {2}; {1} → {2} *Legendre polynomials*. See pp. 232f for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
POINT	g P.FN P.FN2 g POINT	(1) {1, 2} 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 Poiss^{-1}	g PROB g Poiss: Poiss etc.	(1) {2}; {1} → {2} *Poisson distribution* with the number of successes ***g*** in **X** and the *Poisson* parameter ***λ*** in **I**. See pp. 217ff for details. Poiss^{-1} 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. 130.
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. 217ff for more).
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).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
PRIME?	g TEST f PRIME?	(0) {1, 2, 10} Checks if |IP(x)| is a prime. Returns true for prime and false for composite. For x > 3.3×10^{32}, true means 'probably prime' with infinitesimal probability for being composite (see p. 171 for more).
PRINT	g PRINT	*Menus*. See p. 130.
PROB	g PROB	
PROG		*Submenu* of global labels defined when calling XEQ etc. See *Section* 3 of the *OM*.
PROGS	f CATALOG PROGS	*Submenu* of global labels defined at execution time. See pp. 121f.
pr→dB	f U→ ▲ power ratio → dB	(1) {2}; {1} → {2} Convert ratios or pressures. See pp. 143ff.
psi→Pa	f U→ F&p: psi→Pa	
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	f U→ x: g point → m	(1) {2}; {1} → {2} Converts print heights. See pp. 143ff.

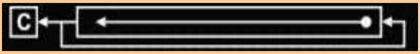
Item	Keystrokes	Remarks (see pp. 12ff for general information)
PUTK	g P.FN f PUTK *r*	(0) Assumes a key code in the address specified. Stops program execution, takes said code and puts it in the keyboard buffer resulting in immediate execution of the corresponding call. R/S is required to resume program execution then. May help in user interaction with routines (see the *OM*, *Section 3*).
P.FN	g P.FN	*Menu*. See p. 130.
P.FN2	g P.FN P.FN2	*Submenu*. See p. 130.
P:	f U→ P:	*Submenu*. See p. 143.
R	f CONST 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→	g L→ f RAD→	(1) {1, 2} → {4} Converts angles as described on pp. 150f.
RAM	f CAT. PROGS RAM	*Submenu* of global labels defined at execution time. See pp. 121f.
RANGE	g DISP ▲ f RANGE	(0) {1, 2} Limits the range of displayable real numbers to $\pm 10^{\pm n}$ with $n = \mathrm{IP}(x)$. Startup default is 6145. $99 \leq n \leq 6145$. For greater input, n will be set to 6145; for less it will be set to 99.
RANGE?	g DISP ▲ f RANGE? g INFO ▲ RANGE?	(-1) { } → {1} Returns the current range setting, cf. RANGE.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
RANI#	g PROB ▲ RANI#	(−1) { } → {1} Returns an integer random number n with $\mathrm{IP}[\min(x, y)] \leq n \leq \mathrm{IP}[\max(x, y)]$. After execution of RANI#, the stack looks like [n, x, y, …], so you can drop or roll down n and call RANI# again to get another random integer with the same parameters. Use RANI# e.g. for throwing dices.
RAN#	g PROB ▲ RAN#	(−1) { } → {2} Returns a random number between 0 and 1 like RAN does in *HP-42S*. See also SEED.
RBR	g RBR	Calls the *register browser* – see the *OM*, *Section 5*. You may call RBR also in *PEM* but it is not programmable. Within RBR, *real* and *complex* numbers are generally displayed in the format chosen within the screen space available. Since RBR uses the small font all the way, more digits are displayable here in ALL **0** than in a numeric row using the large font. For *reals* and ALL **0**, RBR display will turn over 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 *Sect. 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. [18] **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} Replaces ***x*** with the maximum of ***r*** and ***x***. [18]
RCL↓	RCL f Min *r* RCL ▼ *r*	(1) {1, 2, 4, 5, 6, 10} Replaces ***x*** with the minimum of ***r*** and ***x***.[18]
RDP	g DISP f RDP *n*	(1) {2, 3, 5, 8*, 9*} Rounds ***x*** to ***n*** decimal places (0 ≤ ***n*** ≤ 99, think of FIX format), taking the RM setting into account. See RM and compare RSD.
Re	g CPX Re f PARTS g Re	(1) {3} → {2}; {9} → {8} Returns the *real* part of ***x***. Cf. IM.
r_e	f CONST r_e	(−1) { } → {2} Classical electron radius in *meter*.

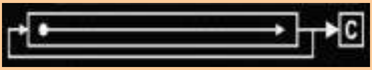
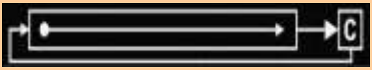
[18] Only legal operations according to the matrices in *Section 2* of the *OM* will work. See also the examples given there.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
REALS	f CAT. VARS REALS	Submenu of *real* variables defined at execution time. See pp. 121f.
REAL?	g TEST ▲ REAL?	(0) Checks if x is a *real* number or matrix.
RECV	f 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 everything to *startup default*, i.e. 2COMPL, ALL **0**, DEG, DENMAX **0**, DSTACK **4**, GAP **3**, J/G **1752-01-01**, LinF, LocR **0**, RM **0**, TDISP **−1**, WSIZE **64**, and Y.MD. RANGE is set to 6145. RESET sets ALLSCI, DECIM**.**, DENANY, MULT×, PROPFR, RECTN, and TDM. It clears all others. See these commands and flags for more.
RE→CX	CC (works in run mode only)	(2) {2} → {3} Composes a *complex* number out of two *reals* or integers x and y, setting ℂ and taking either • (for ∟) the *real* part from **Y** and *imaginary* part from **X**, or • (for ⊙) the *magnitude* from **Y** and *phase* from **X**.
	g CPX f RE→CX	(2) {8} → {9} Works in analogy for two *real* matrices x and y.
Re⇄Im	g CPX Re⇄Im	(1) {3, 9*} Swaps *real* and *imaginary* parts of *complex* objects.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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	f CONST R_K	(−1) { } → {2} *Von Klitzing* constant in *ohm*.
RL	f BITS ▲ RL ***n***	(1) {10}  Works like ***n*** consecutive RLs on *HP-16C*, similar to RLn there (0 ≤ ***n*** ≤ 63). RL **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RLC	f BITS ▲ RLC ***n***	(1) {10}  Works like ***n*** consecutive RLCs on *HP-16C*, similar to RLCn there (0 ≤ ***n*** ≤ 64). RLC **0** executes as NOP, but loads **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 <u>not</u> 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, used in science).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
		1: round half up: ½↑ 0.5 rounds up ('business-man's rounding' [19]). 2: round half down: ½↓ 0.5 rounds down. 3: round up: ←0→ rounds away from 0. 4: round down: →0← rounds towards 0. 5: ceiling: ⌈x⌉ rounds towards +∞. 6: floor: ⌊x⌋ rounds towards –∞. The abbreviations printed on grey are used in STATUS for indicating the respective rounding modes (see *Section 5* of the *OM*).
RMD	f RMD g INTS f RMD	(2) {1, 2, 10} Returns the remainder of a division. Equals RMD on *HP-16C* but works for *reals* as well. See the *OM*, *Sect. 2*, for examples. Cf. MOD.
R_{Moon}	f CONST 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.

[19] 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)
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. 226ff for more.
ROUND	g DISP ROUND	(1) {2, 3, 4, 5, 6, 8*, 9*} Rounds ***x*** using the current display format like RND on *HP-42S*.
ROUNDI	g DISP ROUNDI	(1) {8*}; {2} → {1} Rounds ***x*** to next integer. ½ rounds to 1.
RR	f BITS ▲ RR ***n***	(1) {10}  Works like ***n*** consecutive RRs on *HP-16C*, similar to RRn there (0 ≤ ***n*** ≤ 63). RR **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RRC	f BITS ▲ RRC ***n***	(1) {10}  Works like ***n*** consecutive RRCs on *HP-16C*, similar to RRCn there (0 ≤ ***n*** ≤ 64). RRC **0** executes as NOP, but loads **L**. See the *OM, Sect. 2,* for more.
RSD	g DISP f RSD ***n***	(1) {2, 3, 4, 8*, 9*} Rounds ***x*** to ***n*** significant digits (1 ≤ ***n*** ≤ 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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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} Interprets x in the form `sss.nn`. Clears ***nn*** *registers* starting with address ***sss***. **Example:** For $x = 34.567$, R-CLR will clear **R34** through **R89**. **ATTENTION:** For ***nn* = 0** , clearing will cover the maximum available: • For $sss \in [0; 99]$, it will stop at **R99**. • For $sss \in [100; 111]$, it will stop at **K**. • For $sss \geq 112$, it will stop at the highest currently allocated local *register*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R-COPY	g P.FN g R-COPY	(0) {2} Interprets ***x*** in the form `sss.nnddd`. Takes ***nn*** *registers* starting with address ***sss*** and copies their contents to ***ddd*** etc. **Example:** For ***x*** = 7.0304567 , ***r07, r08, r09*** will be copied into **R45, R46, R47**, respectively. For ***x*** < 0 , R-COPY will take ***nn*** *registers* from *FM* instead, starting with *register* number |***sss***| . Destination will be in *RAM* always. **ATTENTION:** For ***nn* = 0** , copying will cover the maximum available as explained with R-CLR. Then ***x*** must be negative.
R-SORT	g P.FN g R-SORT	(0) {2} Interprets ***x*** in the form `sss.nn`. Sorts the contents of ***nn*** *registers* starting with address ***sss*** . **Example:** Assume ***x*** = 49.036 9, ***r49*** = 1.2, ***r50*** = −3.4, and ***r51*** = 0; then R-SORT will return ***r49*** = −3.4, ***r50*** = 0, and ***r51*** = 1.2. **ATTENTION:** For ***nn* = 0** , sorting will cover the maximum available as explained with R-CLR.
R-SWAP	g P.FN g R-SWAP	(0) {2} Works like R-COPY but swaps the contents of source and destination *registers*.
R→D	g L→ g R→D	(1) {1, 2} → {4} Converts angles as described on pp. 150f.
R↑	f R↑	Rotates the *stack* contents one level up or down, respectively. See *Section 1* of the *OM* for details.
R↓	R↓	

Item	Keystrokes	Remarks (see pp. 12ff for general information)
R∞ R⊙ R⊕	f CONST R∞ etc.	(−1) { } → {2} *Rydberg* constant (see p. 140); mean radii of the Sun and Earth in *meter*.
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 *Section 2* of the *OM* for the output format and pp. 217ff for the formula.
Sa	f CONST Sa	(−1) { } → {2} Semi-major axis in *meter* of the Earth model *WGS84*.[20]
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	f CONST Sb	(−1) { } → {2} Semi-minor axis in *meter* of *WGS84*.[20]
SCI	g DISP SCI *n*	(0) Sets scientific display format (see *Section 2* of the *OM*).
scw→kg	f U→ m: f short cwt → kg	(1) {2}; {1} → {2} Converts masses. See pp. 143ff.

[20] 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

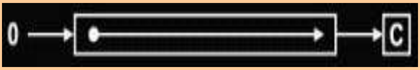
Item	Keystrokes	Remarks (see pp. 12ff for general information)
SDIGS?	g INFO f SDIGS?	(−1) { } → {1} Returns the number of significant digits set by SETSIG. Also returned by STATUS.
SDL SDR	g P.FN P.FN2 f SDL *n* etc.	(1) {2} Shifts digits left (right) by ***n*** decimal positions, equivalent to multiplying (dividing) ***x*** by 10^n. Cf. SL and SR for binary integers.
Se^2	f CONST Se^2	(−1) { } → {2} First eccentricity squared of the Earth model *WGS84* (see footnote 20 on p. 71).
SEED	g PROB ▲ SEED	(0) {2} Stores a seed for random number generation. If ***x*** = 0 , the seed is taken from the real-time clock.
SEND	f I/O f SEND	(0) Sends all *RAM* data to the device connected via serial I/O. See RECV and *Section 3* in the *OM* for more.
SETCHN	g DISP ▲ CHINA	(0) Sets regional format preferences.[21]
SETDAT	g CLK ▲ SETDAT	(0) Sets the date for the real-time clock (the emulator takes this information from the PC clock).
SETEUR	g DISP ▲ EUROPE	(0) Set regional format preferences.[21]
SETIND	g DISP ▲ INDIA	
SETJPN	g DISP ▲ JAPAN	
SETSIG	g MODE f SETSIG	(0) {1} Sets the number of significant digits (0 ... 34) for rounding after each operation. SETSIG 0 sets maximum precision.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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 ▲ UK etc.	(0) Set regional format preferences.[21]
SETUSA		
Se' ^{2}	f CONST Se' ^{2}	(−1) { } → {2} Second eccentricity squared of the Earth model *WGS84* (see footnote 20 on p. 71).
SF	g FLAGS SF *n* g MODE SF *n*	(0) Sets the *flag* specified.
Sf^{-1}	f CONST Sf^{-1}	(−1) { } → {2} Flattening parameter of the Earth model *WGS84* (see footnote 20 on p. 71).
SHOW	f SHOW	(0) {1, 2, 3, 4} Shows all digits stored in **X** until next keystroke in top numeric row. Small font will be used. For *complex* numbers, either part will take one display row. Up to 294 (experimentally: 299) 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.
SIGN	f PARTS sign	(1) {8}; {1, 2, 10} → {1} Returns 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to the mathematical function $\mathrm{signum}(x)$.
		(1) {3} Returns the unit vector of the *complex* number x (cf. UNITV). Maintained for backward compatibility only.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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*}; {1, 4} → {2} Returns the sine of the angle in $\mathbf{X}$.
sinc	g X.FN ▲ sinc	(1) {2, 3, 8*, 9*} Returns $^{sin(x)}/_{x}$ for ***x*** ≠ 0 and **1** for ***x*** = 0. Note input shall be supplied in *radians*.
sinh	g EXP g sinh TRI g sinh	(1) {2, 3, 8*, 9*} Returns the hyperbolic sine of $\boldsymbol{x}$.
SKIP	g P.FN P.FN2 g SKIP *n*	(0) Skips ***n*** program steps forwards (0 ≤ ***n*** ≤ 255). So e.g. SKIP **2** skips over the next two steps, going e.g. from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. **ATTENTION:** If you edit a section of your routine crossed by one or more BACK, SKIP, or CASE jumps, this may well result in a need to **manually maintain all those statements individually.**
SL	f BITS ▲ SL *n*	(1) {10}  Works like ***n*** (≤ 63) consecutive SLs on *HP-16C*. SL **0** executes as NOP, but loads $\mathbf{L}$. See *Sect. 2* of the *OM* for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SLVQ	f ADV SLVQ	{1, 2, 3} → {1 or 2 or 3} Solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input *stack* [***c***, ***b***, ***a***, …], and tests the result. The following holds for *real* parameters: • If $r := b^2 - 4ac \geq 0$, SLVQ returns $-\frac{b \pm \sqrt{r}}{2a}$ in **Y** and **X**. In a routine, the step after SLVQ will be executed. • Else, SLVQ returns the first *complex* root in **X** and the second in **Y** (the *complex conjugate* of the first). In a routine, the step after SLVQ will be skipped. In either case and also for *complex* parameters, 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*).
SMODE?	g INFO SMODE?	(−1) { } → {1} Returns the *integer sign mode* set for *short integers*: true 2 for 2's complement, true 1 for 1's complement, false 0 for unsigned, or true -1 for sign & mantissa mode.
s_{mw}	f STAT f s_{mw}	(−1) { } → {2} Returns the *standard error* for weighted data, i.e. the *standard deviation* of the mean $\bar{x}_w$.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
SNAP	g SNAP	(0) Stores a snapshot of the screen in a `BMP` file on the calculator's USB flash drive in the `/SCREENS` directory. The file name comprises date and time of storage.
SOLVE	f ADV SOLVE ***var***	{2, 3} Solves the equation $f(var) = 0$, with f calculated by the equation specified (in *PEM* by PGMSLV). Two initial estimates of the root must be supplied in **X** and **Y** when calling SOLVE. It returns ***var**_{root}* in **X**, the second last ***var***-value tested in **Y**, then $f(var_{root})$ in **Z**, and 0 in **T**. Additionally, SOLVE acts as binary test in programs, so the next program step will be skipped if SOLVE fails to find a root. See *Section 4* of the *OM* for more. **ATTENTION:** SOLVE fills all *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' (i.e. ±∞ 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.
SSIZE?	g INFO SSIZE?	(−1) { } → {1} Returns the number of *stack registers* currently allocated, 4 or 8.
STAT	f STAT	*Menu*. See p. 131.
STATUS	f STATUS g FLAGS STATUS	*Flag browser*. See *Section 5* of the *OM*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
STK	g STK	*Menu*. See p. 131.
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	(1) {1, 2, 3} Stores a copy of x into the indexed matrix at the current element, a_{ij}. Cf. RCLEL.
	STO g ...EL	
STOIJ	f MATX ▲ STOIJ	(1) {1} Sets the index pointers to IP(x) (= column number) and IP(y) (= row number). Cf. RCLIJ.
	STO g ...IJ	
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.
STO+	STO + *r*	(0) Executes the specified operation on r and stores the result (e.g. $r - x$) at the address specified.[22]
STO–	STO – *r*	
STO×	STO × *r*	
STO/	STO / *r*	
sto→kg	f U→ m: f stone → kg	(1) {2}; {1} → {2} Converts masses. See pp. 143ff.

[22] 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)
STO↑	STO f Max *r* STO ▲ *r*	(0) {1, 2, 4, 5, 6, 10} Stores the maximum of ***r*** and ***x*** in the address specified.[22]
STO↓	STO f Min *r* STO ▼ *r*	(0) {1, 2, 4, 5, 6, 10} Stores the minimum of ***r*** and ***x*** in the address specified.[22]
STRI?	g TEST g STRI?	(0) True if ***x*** is an alphanumeric string (like STR? in *HP-42S*).
STRING	f CATALOG VARS STRING	*Submenu* of *alpha string* variables defined at execution time. See pp. 121f.
SUM	f STAT SUM	(−2) { } → {2} Recalls the linear sums ***Σy*** and ***Σx***. Useful in basic 2D vector algebra. Output is labeled in analogy to s.
s_w	f STAT f s_w	(−1) { } → {2} Calculates the *standard deviation* for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See pp. 217ff for the formula.
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. 217ff for the formula and COV for the *population covariance*.
SYSTEM	g MODE g SYSTEM	(0) Returns to *DMCP*. Works on the calculator only (not on the simulator). See *App F*.
SYS.FL	f CATALOG SYS.FL	*Submenu* of system flags defined. See pp. 121f.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
S.INTS	f CATALOG VARS S.INTS	*Submenu* of *short integer* variables defined at execution time. See pp. 121f.
s.t→kg	f U→ m: ▲ short ton → t	(1) {2}; {1} → {2} Convert masses and times. See pp. 143ff.
s→year	f U→ f s→year	
T_0	f CONST T_0	(−1) { } → {2} Standard temperature (0°C) in *kelvin*.
tan	TRI tan	(1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the tangent of the angle in **X**. Returns "*Not a Number*" for ***x*** = ±90° or equivalents if SPCRES is set.
tanh	g EXP g tanh TRI g tanh	(1) {2, 3, 8*, 9*} 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*** ≥ 3 also for ***n*** − 2 digits showing decimal fractions of *seconds*. TDISP **−1** allows for displaying all digits.
TEST	g TEST	*Menu*. See p. 131.
$t_e(x)$	g PROB t: $t_e(x)$ etc.	(1) {2} *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. 217ff for more mathematical details.
$t_p(x)$		
t(x)		
$t^{-1}(p)$		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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 the *OM, Sect. 5* for a detailed description.
TIMES	f CATALOG VARS f TIMES	*Submenu* of *time* variables defined at execution time. See pp. 121f.
T_n	g X.FN Orthog T_n	(2) {2}; {1} → {2} *Chebyshev polynomials of first kind*. See pp. 232f for details.
TONE	f I/O f TONE ***n***	(0) Sounds a tone according to ***n*** (= 1 … 9) .
ton→kg	f U→ m: ▲ ton→kg	(1) {2}; {1} → {2} Converts masses. See pp. 143ff.
TOP?	g TEST g TOP?	(0) Returns ... • false if called with the program pointer being in a subroutine; • true if called in the top routine (i.e. if the program-running *flag* is set and the *SRS* pointer is clear).
tor→Pa	f U→ F&p: f torr → Pa	(1) {2}; {1} → {2} Converts pressures. See pp. 143ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
T_p	f CONST T_p	(−1) { } → {2} *Planck* temperature in *Kelvin*; *Planck* time in *seconds*.
t_{PL}	f CONST t_{PL}	
TRANS	f CATALOG FCNS TRANS	Works like [M]^T on p. 99. Maintained for backward compatibility only.
TRI	TRI	*Menu*. See p. 131.
trz→kg	f U→ m: f tr.oz → kg	(1) {2}; {1} → {2} Converts masses. See pp. 143ff.
TVM	g FIN TVM	Application. See *Section 5* of the *OM*.
t:	g PROB t:	*Submenu*. See p. 130.
t⇆	g STK t⇆ *r*	Swaps ***t*** and ***r***, in analogy to ***x***⇆
ULP?	g INFO f ULP?	(1) {1, 2} Returns 1 times the smallest power of ten which can be added to ***x*** or subtracted from ***x*** to actually change the 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}; {1} → {2} *Chebyshev polynomials of second kind*. See pp. 232f for details.
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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
	g CPX UNITV	(1) {3} Returns a *complex* number with *magnitude* |*r*| = 1 in direction of *x*.
UNSIGN	f BITS ▼ UNSIGN g INTS ▲ …	(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*.
U→	f U→	*Menu*. See p. 131.
VAR		*Submenu* of variables defined at execution time when calling STO, RCL, etc.
VARMNU	g P.FN 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 CATALOG VARS	*Submenu* of variables defined at execution time. See pp. 121f.
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 a variable called **Test12** containing **−123.45**, VIEW α **Test12** ENTER↑ will display Test12 = -123.45
V_m	f CONST V_m	(−1) { } → {2} Molar volume of an ideal gas at standard conditions in *cubic meter per mol*.
V:	f U→ f V:	*Submenu*. See p. 143.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
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} → {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).[23]
Weibl $Weibl_e$ $Weibl_p$ $Weibl^{-1}$	g PROB f Weibl: Weibl etc.	(1) {2} *Weibull distribution* with its shape parameter ***b*** in **I** and its characteristic lifetime ***T*** in **J**. See pp. 217ff for details. $Weibl^{-1}$ returns the survival time $\boldsymbol{t_s}$ for a given probability ***p*** in **X**, with ***b*** in **I** and ***T*** in **J**.
Weibl:	g PROB f Weibl:	*Submenu*. See p. 130.
WHO?	g INFO g WHO?	(0) Displays credits to the brave men who made this project work.
Wh→J	f U→ E: Wh→J	(1) {2}; {1} → {2} Converts energies. See pp. 143ff.
W_m W_p W^{-1}	g X.FN ▲ W_m etc.	(1) {2, 3}; {1} → {2} W_p returns the principal branch of *Lambert's* W for given $x \geq -1/e$. W_m returns its negative branch (working for $x \in \mathbb{R}$ only). W^{-1} returns ***x*** for a given W_p (≥ -1). See pp. 247ff for more.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
WSIZE	f BITS ▼ WSIZE *n* g INTS ▲ WSIZE *n*	(0) Works almost like on *HP-16C*, but with the parameter $1 \le n \le 64$ 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. 163ff). 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*.
WSIZE?	g INFO g WSIZE?	(−1) { } → {1} Recalls the *word* size set.
W→hp$_E$ W→hp$_M$ W→hp$_{UK}$	f U→ P: W→hp$_E$ etc.	(1) {2}; {1} → {2} Convert powers. See pp. 143ff.
$\hat{x}$	f STAT ▲ $\hat{x}$	(1) {2}; {1} → {2} Returns a forecast *x* for a given *y* (in **X**) according to the curve fit model chosen. See L.R. for more.
$\bar{x}$	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.
x^2	x^2	(1) {1, 2, 3, 8*, 9*, 10} Return the square of *x*.
x^3	g EXP x^3	(1) {1, 2, 3, 8*, 9*, 10} Returns 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. 226ff 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. set SSIZE8). xIN is the recommended way to start an *XROM routine*. Thereafter, SSIZE8 is clear. **Note xIN cannot nest** and *XROM routines* using xIN cannot call user code.
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 ***l*** and the *stack*. Typically, ***way*** = xOUT_NORMAL . Generally, xOUT shall be the last command of an *XROM routine*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
$\bar{x}_{RMS}$	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*.
$\bar{x}_w$	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. 226ff 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* will return *complex* numbers if CPXRES is set.
X.FN	g X.FN	*Menu*. See p. 132.
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} Returns *Γ(**x** + 1).* *2 123.549 956 662 463 236 31* is the maximum ***x*** for *reals*.
x:	f U→ x:	*Submenu*. See p. 143.
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*.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
x→α	g α.FN x→α	(1) {1, 2, 10} → {7} Interprets the integer part of ***x*** as a character code and converts it to the respective character ***x***, similar to XTOA in the *HP-42S*.
x⪥	g STK x⪥ *r*	Swaps ***x*** and ***r***, analogous to ***x***⇆***y***. Will be listed like x⪥ J, x⪥ .12, x⪥ →12, etc.
x⪥y	x⪥y	Swaps the contents of *stack registers* **X** and **Y**.
x = ?	g TEST x= ? *r* etc.	(0) Compare ***x*** with ***r***. See x< ? for more.
x ≠ ?		
x = +0?	g TEST ▲ x=+0? etc.	(0) {1, 2, 3, 10} These tests are for comparing *short integers* in modes 1COMPL and SIGNMT, and for *long integers*, *real* or *complex* numbers if SPCRES is set. Then e.g. 0/(−7) will display −0.
x = -0?		
x ≈ ?	g TEST ▲ x≈ ? *r*	(0) {2, 3, 4, 5, 8, 9} Will be true if the rounded values of ***x*** and ***r*** are equal (see ROUND). See x< ? for more.
x < ?	g TEST x< ? *r* etc.	(0) {1, 2, 4, 5, 6, 7, 10} Compare ***x*** with ***r***. Strings are compared according to their sorting order. **Example:** TEST x< ? K compares ***x*** with ***k***, and will be listed as x< ? K in a routine. It will return true if ***x*** < ***k*** at execution time. See examples in *Sect. 1* of the *OM* for more.
x ≤ ?		
x ≥ ?		
x > ?		

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ŷ	f STAT ▲ ŷ	(1) {2}; {1} → {2} Returns a forecast ***y*** (in **X**) for a given ***x*** according to the curve fit model chosen. See L.R. for more.
yd.→m	f U→ x: g yd.→m	(1) {2}; {1} → {2} Converts distances. See pp. 143ff.
YEAR	g CLK f YEAR	(1) {2, 6} → {1} Assumes ***x*** containing a *date* in the format selected (or a *real* number in corresponding format) and extracts the year.
year→s	f U→ f year→s	(1) {2}; {1} → {2} Converts times. See pp. 143ff.
y^x	y^x	(2) {1, 2, 3, 10} 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 *reals*. Exceeding these boundaries may produce *complex* results if CPXRES is 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	f CONST Z_0	(−1) { } → {2} Characteristic impedance of vacuum in *ohm*.
z⇆	g STK z⇆ r	Swaps ***z*** and ***r***, in analogy to ***x***⇆.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
α	f CONST α	(−1) { } → {2} Fine-structure constant.
αINTL	f CAT. CHARS αINTL	*Submenu*. See pp. 121ff.
	g +	*Menu* in *AIM* (see p. 133).
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*.[24]
αMATH	f CAT. CHARS αMATH	*Submenu*. See pp. 121ff.
	g −	*Menu* in *AIM*. See p. 134.
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.[24] 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**.[24]
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**.[24]

[24] 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)
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. [24]
α.FN	g α.FN	*Menu*. See p. 134.
A…Ω	f CAT. CHARS A…Ω	*Submenu* of Greek letters, see pp. 121ff.
α·	f CAT. CHARS α·	*Submenu*. See pp. 121ff.
	g .	*Menu* in *AIM*. See p. 134.
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*. [24]
β(x,y)	g X.FN ▲ β(x,y)	(2) {2, 3}; {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	f CONST γ	(−1) { } → {2} *Newtonian* constant of gravitation (also called G by other authors) in $m^3/_{kg\ s^2}$;
γ_{EM}	f CONST γ_{EM}	*Euler-Mascheroni* constant (for mathematics);
γ_p	f CONST γ_p	proton gyromagnetic ratio (see p. 141).
Γ_{xy}	g X.FN ▲ Γ_{xy}	(2) {2}; {1} → {2} Returns the *lower incomplete Gamma function*. See pp. 247ff for more.
γ_{xy}	g X.FN ▲ f γ_{xy}	(2) {2}; {1} → {2} Returns the *upper incomplete Gamma function*. See pp. 247ff for more.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Γ(x)	g PROB ▲ Γ(x)	(1) {2, 3}; {1} → {2} Returns $\Gamma(x)$. Note x! calls $\Gamma(x+1)$. See also LNΓ.
δx	f CAT. PROGS δx	Predefined global label for f'(x) and f"(x) – see *Section 4* of the *OM*.
$\Delta\nu_{Cs}$	f CONST $\Delta\nu_{Cs}$	(−1) { } → {2} Hyperfine transition frequency of ^{133}Cs in *hertz*.
Δ%	f Δ%	(1) {2}; {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 multi-plicative instead of additive. See pp. 217ff for more information.
ε_0	f CONST 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*).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
ε_p	f STAT g ε_p	(−2) { } → {2} Works like ε but returns the *scattering factors* of the two populations.
$\zeta(x)$	g X.FN ▲ f $\zeta(x)$	(1) {2, 3} Returns *Riemann's Zeta*. See p. 250 for more.
λ_C λ_{Cn} λ_{Cp}	f CONST 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	f CONST 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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Σ	g Σ	*Menu*. See p. 135.
σ	f STAT σ	(−2) { } → {2} Works like s but returns the *standard deviations* of the two *populations* instead. See pp. 217ff.
σ_B	f CONST σ_B	(−1) { } → {2} *Stefan-Boltzmann* constant (see p. 142).
$\Sigma ^1/x$	g Σ ▲ $\Sigma ^1/x$ 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. 163ff.). **ATTENTION:** Depending on your input data, logarithmic or inverted sums may become infinite or even non-numeric. If this happens no error will be thrown, regardless of the status of SPCRES. For space reasons, two sums are abbreviated: **Σlnxy** denotes $\sum ln(x)ln(y)$ and **Σlny/x** denotes $\sum \frac{ln(y)}{x}$.
$\Sigma ^1/x^2$		
$\Sigma ^1/y$		
$\Sigma ^1/y^2$		
Σln^2x	g Σ f Σln^2x etc.	
Σln^2y		
Σlnx		
Σlnxy		
Σlny		
Σlny/x	g Σ 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. 217ff.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Σx	g Σ Σx etc.	(−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see Σ ¹/x above for more).
Σx^2		
$\Sigma x^2 y$	g Σ g $\Sigma x^2 y$	
$\Sigma x^2/y$	g Σ ▲ $\Sigma x^2/y$	
Σx^3	g Σ ▲ f Σx^3 etc.	
Σx^4		
Σxlny	g Σ g Σxlny	
Σxy	g Σ Σxy	
Σx/y	g Σ ▲ Σx/y	
Σy	g Σ Σy etc.	
Σy^2		
Σylnx	g Σ g Σylnx	
Σ+ [25]	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} Adds one 2D data point to the statistical sums.

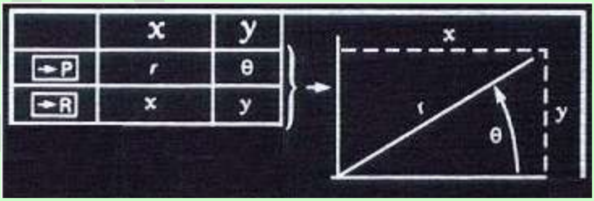
[25] Σ+ and Σ− return *temporary information* as shown in *Section 2* of the *OM* and disable *automatic stack lift*. Both commands may also be used for 2D vector adding and subtracting (see SUM and the corresponding example in *Section 2* of the *OM*).

Item	Keystrokes	Remarks (see pp. 12ff for general information)
Σ– [25]	f STAT f Σ-	{1, 2} Subtracts one 2D data point from the statistical sums.
Φ	f CONST Φ	(−1) { } → {2} Golden ratio; magnetic flux quantum in *volt-second*.
Φ_0	f CONST Φ_0	
$\chi^2_e(x)$	g PROB χ^2: $\chi^2(x)$ etc.	(1) {2} *Chi-square distribution* (with its degrees of freedom given in **I**). $\chi^2_e(x)$ equals $Q(\chi^2)$ on *HP-21S*. See *Section 2* of the *OM* for an application example and pp. 217ff for more.
$\chi^2_p(x)$		
$\chi^2(x)$		
$(\chi^2)^{-1}$		
χ^2:	g PROB χ^2:	*Submenu*. See p. 130.
ω	f CONST ω	(−1) { } → {2} Angular velocity of the Earth in *radians per second* according to *WGS84* (see footnote 20 on p. 71).
$(-1)^x$	g X.FN ▲ f $(-1)^x$	(1) {1, 2, 3, 8*, 9*, 10} 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 A is an $n \times m$ matrix and a_{ij} is an element of it then A^T will be an $m \times n$ matrix B with $b_{ij} = a_{ji}$. The transpose is done in-situ and does not require any additional memory.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
[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.[26]
+/-	+/- (for closed input)	(1) 'Unary minus', returns $x \times (-1)$.[26]
±∞?	g INFO g ±∞?	(0) {2} → {1} Tests x for infinity. Returns true 1 for $x = +\infty$, true -1 for $x = -\infty$, and false 0 else.
-	−	(2) Returns $y - x$ for compatible numeric objects.[26]
-∞	f CONST -∞	(−1) { } → {2} Minus infinity. See p. 143.
×	×	(2) Returns $y \times x$ for compatible numeric objects.[26]
×MOD	g INTS f ×MOD	(3) {1, 2, 10} 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} Returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$. See MOD.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→		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	g L→ →DEG	(1) {1, 2, 4} → {4} Convert angles as described on pp. 150f.
→D.MS	g L→ →D.MS	
→GRAD	g L→ →GRAD	
→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} → {5} Converts ***x*** to a sexagesimal *time* – cf. p. 116.
→INT	f # ***base*** (for closed input)	(1) {1, 2, 10} → {10} Converts the integer part of ***x*** to a *short integer* of the base specified. Conversion to decimal may be abbreviated by # D, to hexadecimal by # H. Cf. p. 116.
→MULπ	g L→ →MULπ	(1) {1, 2, 4} → {4} Converts angles as described on pp. 150f.
→POL	g →P	{2}; {1} → {2} Assumes **X** and **Y** containing 2D *Cartesian* coordinates of a point or components of a vector (***x*** , ***y***). Converts them to the respective polar coordinates or components (***r***, ϑ). See the picture at →REC. For switching the display format of *complex* numbers, see RECTN .

Item	Keystrokes	Remarks (see pp. 12ff for general information)
→RAD	g L→ →RAD	(1) {1, 2, 4} → {4} Converts angles as described on pp. 150f
→REAL	f .d (for closed input)	(1) {1, 2, 4, 5, 6, 10} → {2} Converts x to a *real* number. Any object (e.g. a *time*) tagged sexagesimal will be converted in a decimal number. For *dates*, the date format chosen is taken into account. Numbers shown as fractions will be displayed as decimal numbers (cf. FRACT and PROPFR). For returning the *real* part of a *complex* number, take RE. For cutting a *complex* number into its parts, use CC or CX→RE.
→REC	f R←	{2}; {1, 4} → {2} Assumes **X** and **Y** containing 2D polar coordinates of a point or components of a vector (***r***, ***ϑ***). Converts them to the respective Cartesian coordinates or components (***x***, ***y***). See the picture and cf. →POL.  For switching the display format of *complex* numbers, see RECTN.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
⥮	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. **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.
|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} Returns the absolute (unsigned) value of *x*.
		(1) {8*} Returns a matrix with the absolute values of all input matrix elements. Cf. ENORM.
	f |x| or g CPX f |x|	(1) {3} → {2} 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.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
||	g X.FN ▲ f ||	(2) {2, 3}; {1} → {2} Returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{-1}$; useful in electrical engineering especially. Returns 0. for $x \times y = 0$.
%	g FIN %	(1) {2}; {1} → {2} Returns $\frac{xy}{100}$, leaving y unchanged.
%MRR	g FIN %MRR	(3) {2}; {1} → {2} Returns the mean rate of return in % per period, i.e. $100 \cdot \left(\sqrt[z]{x/y} - 1\right)$ with x = FV = future value after z periods, y = PV = present value. For $z = 1$, Δ% returns the same result easier.
%T	g FIN %T	(1) {2}; {1} → {2} Returns $\frac{100\,x}{y}$, interpreted as % of total. Leaves y unchanged.
%Σ	g FIN %Σ	(1) {2}; {1} → {2} Returns $\frac{100\,x}{\sum x}$.
%+MG	g FIN %+MG	(2) {2}; {1} → {2} Calculates a sales price by adding a margin of x % to the cost y, as %MU-Price in *HP-17B*. Formula: $p_{sale} = \frac{y}{1-\frac{x}{100}}$ 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	g √x	(1) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}, {1, 2} → {3}) Returns the square root of *x*. Square roots of non-square *long integers* will return *reals*. Square roots of negative *long integers* or *reals* will return *complex* numbers if CPXRES is set.
	g EXP √x	
∞	f CONST ∞	(−1) { } → {2} Infinity. See p. 143.
∫	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**.[27] **ATTENTION:** ∫ fills all *stack registers* with *x* before calling the routine specified in PGMINT.
	g EQN ∫f ∫	Integrates the current equation.[27]
∫f	g EQN ∫f	*Submenus*. See pp. 127f.
∫fdx	f ADV ∫fdx	
∡	g ∡ or g CPX f ∡ or f PARTS f ∡	(1) {3} → {2} Returns the *phase* or argument $arg(x) = arctan\left(\frac{Im(x)}{Re(x)}\right)$. Cf. |x|.
		(1) {9*} → {8} Returns a matrix with the *phases* of all input matrix elements. Cf. |x|.

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

Item	Keystrokes	Remarks (see pp. 12ff for general information)
∡→	g L→	*Menu* of angular conversions. See p. 135.
🖶ADV	g PRINT 🖶ADV	(0) Prints the current contents of the print buffer and a linefeed. **ATTENTION:** The printer will actually print only when a linefeed is sent to it.
🖶CHAR	g PRINT f 🖶CHAR *n*	(0) Sends a single character (with the code specified) to the printer. Character codes ***n*** > 127 can only be specified indirectly. 🖶MODE setting will be honored. See 🖶ADV.
🖶DLAY	g PRINT g 🖶DLAY *n*	(0) Sets a delay of ***n*** ticks (see TICKS) to be used with each linefeed on the printer.
🖶LCD	g PRINT 🖶LCD	(0) Sends the contents of the entire *LCD* to the printer, so you get a hardcopy of the screen.
🖶MODE	g PRINT g 🖶MODE *n*	(0) Sets print mode. Legal print modes are: 0: Use the printer font and character set wherever possible (default). All characters feature the same width (5 columns + 2 columns spacing). 1: Use the variable pitch display font, resulting in some jitter on the printout but packing more characters in a row. 2: Use the small display font, which allows for packing even more info in a row. 3: Send the output to the serial line. Works for plain ASCII only – no characters will be translated. Line setup is the same as for serial communication: 9600 baud, 8 bits, no parity.
🖶PROG	g PRINT 🖶PROG	(0) Prints the listing of the *current program* (see *Sect. 3* of the *OM*), one row per step.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
🖶r	g PRINT 🖶r *r*	(0) Prints the *register* specified, right adjusted, without labeling the output (note 🖶r **X** takes a separate label 🖶x in this *menu*). If you want a heading label, compose the string in **X** first or use 🖶REGS. See 🖶ADV.
🖶REGS	g PRINT 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 its label. **ATTENTION for *nn* = 0 :** • For ***sss*** $\in [0;99]$, printing will stop at **R99**. • For ***sss*** $\in [100;111]$, printing stops at **K**. For ***sss*** ≥ 112 , printing stops at the highest allocated local *register*.
🖶STK	g PRINT f 🖶STK	(0) Prints the entire *stack* contents. Each of the 4 or 8 *registers* prints in a separate row starting with its label.
🖶TAB	g PRINT 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 PRINT 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 PRINT g 🖶WIDTH	(−1) Returns the number of print columns that ***x*** would take in the print mode set. See 🖶MODE. Second use: in 🖶MODE 1 or 2, 🖶WIDTH returns the width of ***x*** in px (including the last column being always blank) in the specified font.

Item	Keystrokes	Remarks (see pp. 12ff for general information)
🖶Σ	g PRINT 🖶Σ	(0) Prints the summation *registers*. Each *register* prints in one row starting with its label.
🖶#	g PRINT f 🖶# ***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 ≤ ***n*** ≤ 255 in a single program step. Kept for backward compatibility to *WP 34S* only.
#B	f BITS g #B	(1) {10} Counts the bits set in x (like on *HP-16C*).

Names of Variables and System Flags Provided

There is a *name* overlap between some constants and commands on one side and predefined variables and system flags on the other. Thus, the latter set is kept separate from the other *items*. As required for the *items* above, these *names* must be unique. The current status of the items printed on grey can be seen on the screen directly (e.g. in the *status bar*), for those printed on orange it is returned by STATUS:

Name	Keystrokes if applicable	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*).
ADM		Reserved integer variable for the *ADM*: 0: DEG, 1: D.MS, 2: RAD, 3: MULπ, 4: GRAD.

Name	Keystrokes if applicable	Remarks (see pp. 12ff for general information)
ALLSCI		*System flags* – see next chapter.
ALPHA		
ALP.IN		
ASL.BLK		
AUTOFF		
AUTXEQ		
B		Reserved variables for *registers* **B** and **C**.
C		
CARRY		*System flags* – see next chapter.
CPXj		
CPXRES		
D		Reserved variable for *register* **D**.
DECIM.		*System flags* – see next chapter.
DENANY		
DENFIX		
DENMAX		Reserved integer variable for the maximum denominator, filled by the command DENMAX.
DMY		*System flags* – see next chapter.
FRACT		
FV	g FIN TVM FV	Reserved variable for the future value of your investment or loan in *TVM*.[28]

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

Name	Keystrokes if applicable	Remarks (see pp. 12ff for general information)
GRAMOD		Reserved integer variable determining how the AGRAPH image is displayed: [29] 0: It is merged (OR-ed) with the existing display. 1: It overwrites all pixels in that portion of the display. 2: Duplicate "on" pixels get turned "off". 3: It is XOR-ed with the existing display.
GROW		*System flag* – see next chapter.
I		Reserved variable for *register* **I**.
IGN1ER INTING		*System flags* – see next chapter.
i%/a	g FIN TVM i%/a	Reserved variable for the annual interest rate of your investment or loan in *TVM*. [28]
J K L		Reserved variables for *registers* **J**, **K**, and **L**.
LEAD.0 LOWBAT		*System flags* – see next chapter.
Mat_A	f MATX SIM EQ Mat A	Reserved variables for solving systems of linear equations (see *Sect. 2* of the *OM*).
Mat_B	f MATX SIM EQ Mat B	
Mat_X	f MATX SIM EQ Mat X	
MDY MULT×		*System flags* – see next chapter.

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

Name	Keystrokes if applicable	Remarks (see pp. 12ff for general information)
NPER	g FIN TVM nPER	Reserved variable for the <u>total</u> number of • payment periods for your loan or • compounding periods for your investment.
NUM.IN		*System flags* – see next chapter.
OVERFL		
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]
PRINT		*System flags* – see next chapter.
PROPFR		
PRTACT		
PV	g FIN TVM PV	Reserved variable for the present value of your investment or loan in *TVM*. [28]
QUIET		*System flag* – see next chapter.
REALDF		Reserved integer variable for real number display format: 0: ALL, 1: FIX, 2: SCI, 3: ENG.
RECTN		*System flag* – see next chapter.
REGS		Reserved variable for the 100×1 matrix of registers – if required.
RUNIO		*System flags* – see next chapter.
RUNTIM		
SLOW		

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

System Flags

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

Purpose	*SBI*	*Flag*[30]	Remarks
Time display	Time string	TDM (set)	Set for international 24h time display. Expands the date display in the *status bar* to four digits for the year. Clear for 12h time display: e.g. 1:23 will become 1:23am, 23:45 will become 11:45pm. Shortens the date display in the *status bar* to two digits for the year.
Date display	Date string	YMD, DMY, MDY (YMD set)	Set for the respective date format chosen. The commands Y.MD, D.MY, and M.DY set the corresponding *flag* and clear the two others.
Complex results	ℂ / ℝ	CPXRES	Set for allowing *complex* results also for *real* input (e.g. $\sqrt{-1}$). Else an error will be thrown in such cases.
Complex letter	—	CPXj	Set for the letter $\boldsymbol{j}$ representing the *imaginary* number $\boldsymbol{i}$, clear for $\boldsymbol{i}$.
Rectangular notation	∟ / ⊙	RECTN (set)	Set for rectangular display of *complex* numbers, clear for polar.

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

Purpose	*SBI*	*Flag*[30]	Remarks
Fraction display	—	FRACT	Set if fraction display is chosen, clear for decimal display (`a b/c` sets FRACT, `.d` clears it). See next entries.
Fraction kind 1	—	PROPFR	Set for *proper fractions*, clear for *improper fractions* (`a b/c` toggles PROPFR: coming from decimal display, it sets it). PROPFR set allows only *proper fractions* in display (e.g. $1\,^2/_3$ instead of $^5/_3$); any *reals* (with $\lvert x \rvert < 10^6$) will be displayed according to the settings of DENANY, DENFIX, and DENMAX as *proper fractions*.
Fraction kind 2	see *OM*	DENANY (set)	Set if <u>any</u> denominator up to DENMAX may appear (DENMAX is indicated in the *status bar*).[31] For given DENMAX, this is the most precise way of displaying a decimal number as a fraction. See also next entry.
Fraction kind 3	see *OM*	DENFIX[32]	Set if the value set by DENMAX is the one and only denominator allowed. Clear if the denominator may be an integer factor of DENMAX.[33]
Carry	c or o c	CARRY	Reflects the status of the carry bit.
Overflow	o or o c	OVERFL	Reflects the status of the overflow bit.
Leading zeros	—	LEAD.0	Set for leading zeros turned on in *short integers* of bases 2, 4, 8, and 16.[34]
Alpha	A or α	ALPHA	Set for *AIM*, else clear.

[31] If e.g. DENMAX = 5 and DENANY is set then this allows denominators 1, 2, 3, 4, and 5.

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

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

[34] Works like *flag* 3 in *HP-16C.*

Purpose	*SBI*	*Flag*[30]	Remarks
Upper case	A / α	αCAP (set)	Set for capital letters, clear for lower case.
Running timer	🕓	RUNTIM	Set if the timer is running.
Running I/O	⇕	RUNIO	Set if I/O is in progress.
Printing	🖶	PRINT	Set if your *WP 43S* is sending data to the printer.
Tracing	—	TRACE	Set if the print line is in tracing mode. Else prints must be triggered explicitly.
User mode	U	USER	Set if your *WP 43S* is in user mode.
Low battery	🔋	LOWBAT	Set if the battery voltage is low (see *Section 2* of the *OM*).
Processor speed	—	SLOW	Kept clear for fresh batteries unless the user intervenes, set for battery voltage < 2.5V. Speed and power consumption will be reduced to ~50% with SLOW set.
Special results	—	SPCRES	Set for allowing special results of calculations (i.e. ±∞ and NaN).
Stack size	—	SSIZE8	Set for eight, clear for four *stack registers*. Note all *register* contents will remain unchanged if SSIZE8 is modified (as well as if *stack* size is changed by any other operation – e.g. by RCLCFG).
Beeper	—	QUIET	Set for disabled beeper.
Radix mark	—	DECIM. (set)	Set for a decimal point, clear for a comma.
Multiplication symbol	—	MULT× (set)	Set for multiplication symbol ×, clear for · .

Purpose	*SBI*	*Flag*[30]	Remarks
Overflow from ALL	—	ALLSCI (set)	Set if numbers exceeding the range displayable in ALL or FIX will be shown in SCIentific, clear for ENGineer's format.
Matrix grow mode	—	GROW	Set (e.g. by M.GROW) if matrices may grow, else clear (e.g. by M.WRAP). See the command J+ above and *Section 2* of the *OM*; see also GROW in the *HP-42S Owner's Manual*, p. 213.
Automatic OFF	—	AUTOFF (set)	Set if automatic shutdown is enabled. Else your *WP 34S* will remain ON until you will turn it off manually or battery voltage will drop below the limit.
Automatic execution	—	AUTXEQ	Like *flag* 11 of *HP-42S*
Printer activated	—	PRTACT	Like *flag* 21 of *HP-42S*
Data entry	—	NUM.IN, ALP.IN	Set for numeric or alphanumeric entry (like *flags* 22 and 23 of *HP-42S*).
Disable automatic stack lift	—	ASL.BLK	Set by ENTER, CLX, Σ+, and Σ−, cleared by all other functions (see the *OM*, *S. 1*)
Error handling	—	IGN1ER	If set, your *WP 43S* ignores just 1 arbitrary error and clears IGN1ER then. The operation causing the error will not be executed. This works like *flag* 25 of *HP-42S*
Integrating	—	INTING	Set while the integrator is running.
Solving	—	SOLVING	Set while the solver is computing a root.
Variable menu	—	VMDISP	Set while the variable menu is displayed.

Nonprogrammable Commands and Keys

The commands 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, 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* (on pp. 120ff) for more about this topic.

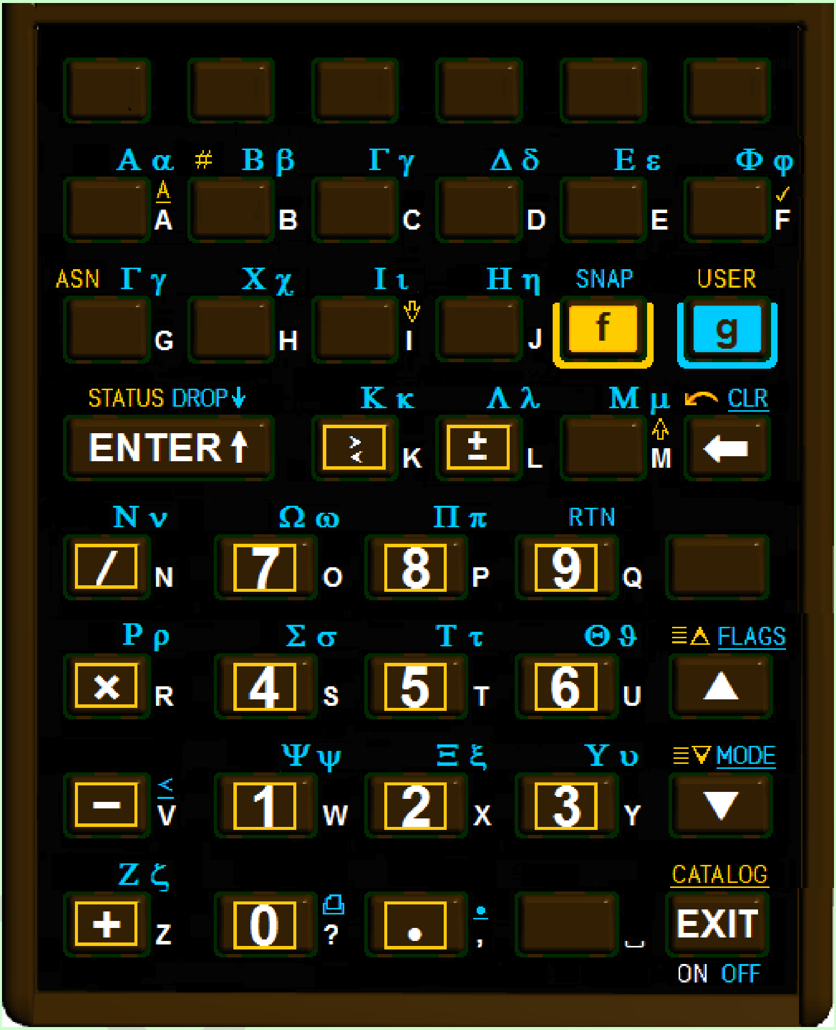
The *browsers* RBR and STATUS as well as the *application* TIMER use some keys for particular control purposes (e.g. STO, RCL, ., and numeric keys – see the *OM, Section 5*).

Command Parameter Input and Closing It

The following table shows what will happen when particular keys are pressed while command parameter input is not finished yet (see pp. 116ff for input in **X** instead). Note that a "character" may be a letter, digit, punctuation mark, etc. The table below lists the respective keys beginning top left on the keyboard:

Keystrokes	Situation	Meaning
A … D, I … L, T, X … Z	addressing	Enters the address of a *global general purpose register* or *user flag*.
A … Z	entering a label, a *system flag* or variable *name*	Appends the corresponding Latin or Greek letter or digit to the label, *system flag* or variable *name* pending. Use ▼ and ▲ to switch cases for letters. See the virtual keyboard on p. 115 (and cf. the *OM*, *Sect. 2*).
g A … g O	entering a label, a *system flag* or variable *name*	Appends the corresponding Latin or Greek letter or digit to the label, *system flag* or variable *name* pending. Use ▼ and ▲ to switch cases for letters. See the virtual keyboard on p. 115 (and cf. the *OM*, *Sect. 2*).
f 0 … f 9	entering a label, a *system flag* or variable *name*	Appends the corresponding Latin or Greek letter or digit to the label, *system flag* or variable *name* pending. Use ▼ and ▲ to switch cases for letters. See the virtual keyboard on p. 115 (and cf. the *OM*, *Sect. 2*).

Keystrokes	Situation	Meaning
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 *user flag* address, a *system flag* or 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 or specifying	Enters a numeric parameter, an address, or a local label. See *Sections 1* and *3* of the *OM* for valid number ranges.
.	addressing	Header for *local registers* or *user flags*.
EXIT	arbitrary parameter input pending	If there is an open *menu*, closes it. Else cancels pending command input, returning to the status of your *WP 43S* as it was before the current command was called.



Virtual keyboard in *alpha input mode* (*AIM*). *AIM* is also active when a catalog is entered, so you can use all accessible characters for alphabetic searching (see pp. 124f).

Alphanumeric Input in X and Closing It

The following table shows what will happen when particular keys are pressed with alphanumeric (incl. numeric) input in X being open still (turn to pp. 113f for command parameter input instead). The table lists the respective keys top left to bottom right on the keyboard:

Keystrokes	in mode(s)	Meaning
A … Z g A … g O	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*.
f #	A, α	Appends # to the *alpha string* ***x***.
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**.[35]
f x^2 (✓)	A, α	Appends a checkmark ✓ to the *alpha string* ***x***.
CC	¬ (A, α)	Closes input of the first part (i.e. *real* part or *magnitude*) of a *complex* number in **X** and waits for input of its second part (i.e. *imaginary* part or *phase*, see the *Key Response Table* and *Section 2* of the *OM*).
f R↓ (⇩)	A, α	Prefix making the next character a subscript, if applicable.

[35] 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)	Meaning

ENTER↑	arbitrary input pending	If there was input expected but not entered, cancels entry.
		Else closes input (in **X**) and checks the following conditions top-down:
		• If this input is alphanumeric (i.e. if it contains at least one non-numeric character except [.]), takes it as 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 35 and interprets it. Finally, unless an error had to be thrown, copies ***x*** into **Y**.

f x≷y	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*).
f E (⇧)	A, α	Prefix making the next character a superscript, if applicable.

Keystrokes	in mode(s)	Meaning
←	arbitrary input pending	Deletes the last (rightmost) character keyed in. If there is nothing left, cancels the pending input, returning to the status of your *WP 43S* as it was before that input was started.
f /	A, α	Appends / to the *alpha string* **x**.
f ×	A, α	If MULT× then appends ×, else · to the *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**.
▲	α	Turns to upper case for the following letters.
▼	A	Turns to lower case for the following letters.
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).
+ (?)	A, α	Appends ? to the *alpha string* **x**.
f +	A, α	Appends + to the *alpha string* **x**.
g 0 (🖶)	A, α	Appends 🖶 to the *alpha string* **x**.

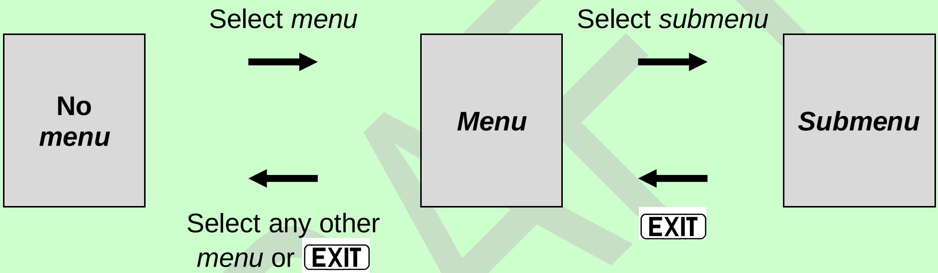
Keystrokes	in mode(s)	Meaning
.	¬ (A, α)	Inserts a radix mark as selected.
		Separates *degrees* from *minutes*, *seconds*, and *hundredths of seconds* in angular input, so input format is ddddd.mmsshh d.ms for sexagesimal angles (cf. p. 116 and *Section 2* of the *OM*).
		Separates *hours* from *minutes*, *seconds*, and fractions of *seconds* in *time* input, so input format is hhhhh.mmssfffff h.ms for sexagesimal *times* (cf. p. 116 and *Section 2* of the *OM*).
Second .	¬ (A, α)	A 2nd . in input indicates a fraction. See the *OM*, *Sect. 2* for examples. The 2nd . just separates the nominator and the denominator in input. Note you cannot enter E after you entered . twice – but you may delete the 2nd dot while editing the input.
.	A, α	Appends , to the *alpha string* **x**.
f .	A, α	Appends . to the *alpha string* **x**.
R/S	❶, program waits f. input	Closes input and starts its checks and interpretation like ENTER↑ above. Resumes program execution.
	A, α	Appends a blank space 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. 121 and 133ff 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).

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



Catalogs are a special kind of *menus* with their contents sorted alphabetically. Your *WP 43S* provides the following 16 *catalogs*:

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

Within *catalogs*, some special operations ease your path accessing the *items* stored therein (as shown on pp. 124f).

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. 124f.

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

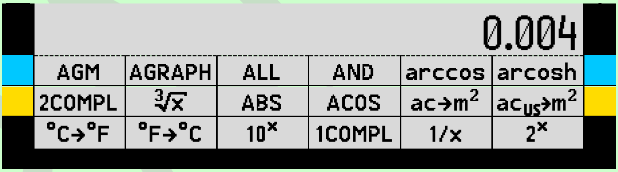
							Remarks
CATALOG:	FCNS	SYS.FL	CHARS	PROGS	VARS	MENUS	top branches
			DIGITS				
FCNS:	°C→°F	°F→°C	10^x	1COMPL	$^1/x$	2^x	contains some 700 functions provided
	2COMPL	$\sqrt[3]{x}$	ABS	ACOS	ac→m²	ac$_{US}$→m²	
	AGM	AGRAPH	ALL	AND	arccos	arcosh	
	…						
	…	#B					
SYS.FL:	ALLSCI	ALPHA	ALP.IN	…			system flags (cf. pp. 104ff)
	…						
CHARS:	αINTL	A…Ω		αMATH	Myα	α•	character branches
αINTL:	A	Ā	Á	Ă	À	…	international Latin letters, see p. 133
αMATH:	<	≤	=	…			mathematical operators and symbols, see p. 134
A…Ω:	Α	Β	Γ	Δ	…		Greek letters, see p. 134
α•:	!	;	…				punctuation marks, see p. 135

	▭	▭	▭	▭	▭	▭	Remarks
PROGS:	RAM					FLASH	global labels currently defined
RAM:	...						both branches are empty at startup; they will be filled with your creations
FLASH:	...						
VARS:	L.INTS	S.INTS	REALS	CPXS	STRING	MATRS	branches for various types of variables
	DATES	TIMES	ANGLES				
ANGLES:	...						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:	BITS	Binom:	Cauch:	CHARS	*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
	CLK	CLR	CONST	CPX	CPXS	DATES	
	DIGITS	DISP	EQN	EXP	Expon:	E:	
	...						
	...	∡→					
A:	...						*(sub-) menus* provided unless mentioned above already, see pp. 126ff for predefined contents – here your creations will be inserted as new entries
Binom:	...						
BITS:	...						
...							
...							
DIGITS:	0	1	2	3	4	5	digits defined
	6	7	8	9	A	B	
	C	D	E	F	i		

Three branches of CATALOG are expandable (MENUS and the *submenus* of PROGS and VARS) since you may create *items* of these kinds (cf. the *OM, Sect. 6*); the other ones are fixed size (FCNS, DIGITS, and CHARS) since all functions, digits, and characters are predefined.

Calling CATALOG will display its top level branches. The seven labels shown are pointers to the *submenus* containing all the functions, *system flags*, digits, characters, programs, variables, and *menus* defined at execution time.

Choosing one of these branches will display its first *view* of *items* (primary, f- and g-shifted, as applicable). Pressing the leftmost *softkey*, for instance, will call the *submenu* 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).[36] Select an *item* by pressing the corresponding top row key (headed by f or g if applicable); e.g. call a function by pressing the corresponding *softkey*. EXIT will just leave this branch without doing anything.

All the over 700 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* (see below). Remember that each and every command featured on your *WP 43S*, the keystrokes calling it, and the necessary particular explanations are listed for your reference in the *IOI* on pp. 12ff.

---

36 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.

As mentioned in the *OM*, 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 the chapter after 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.

Accessing Cataloged Items Rapidly

You can browse a *catalog* like any other *menu* just using ▲ and ▼ as explained in previous chapter. In CONST and major parts of CATALOG (FCNS, SYS.FL, MENUS, CHARS αINTL, and the *submenus* of PROGS and 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

Return — **Your *WP 43S* displays the first *view* in this *catalog*;[37] *AIM* is on.**

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

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

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

FLOOR	FP	FP?	$F_p(x)$	FS?	FS?C
$F_e(x)$	FF	FIB	FILL	FIX	FLASH?
FB	FBR	FC?	FC?C	FC?F	FC?S

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

$fz_{US}\to m^3$	f'(x)	f"(x)	GAP	GaussF	GCD
fm.→m	fr→dB	ft.→m	$ft_{US}\to m$	ft.→m	$fz_{UK}\to m^3$
FS?	FS?C	FS?F	FS?S	F(x)	$F^{-1}(p)$

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

[38] This search is case independent (i.e. specifying **A** will find **a** as well). Note, however, that **A** and **a** remain different letters nevertheless. Remember you can 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.

Return	**Your *WP 43S* executes or inserts the command, recalls the constant, or inserts the letter selected. *AIM* remains on until this *catalog* is exited – then your *WP 43S* returns to the mode as set before entering this catalog.** **Result** (in this example after specifying the *flag* number): true 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 700);
- any constant provided can be recalled by f CONST + 3 keystrokes maximum if you know its first character;
- any letter provided can be inserted by f CATALOG CHARS αINTL (or in *AIM* by f A) + 3 keystrokes maximum.

Further Menus and Their Contents

In the table 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	F1	F2	F3	F4	F5	F6	Remarks
ADV	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx	advanced operations, see *Sect. 4 of the OM*
	PGMSLV		f"(x)			PGMINT	
∫fdx			ACC	↓Lim	↑Lim	∫	
BITS	AND	OR	XOR	NOT	MASKL	MASKR	contains all the Boole's and bit operations (first two views) and settings (third view) of *HP-16C* and *WP 34S*
	NAND	NOR	XNOR	MIRROR		ASR	
	SB	BS?	#B	FB	BC?	CB	
	SL	RL	RLC	RRC	RR	SR	
	LJ					RJ	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
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	
						J/G	
CLR	CLΣ	CLP	CF	CLMENU	CLCVAR	CLX	almost as in *HP-42S*
	CLREGS	CLPall	CLFall		CLLCD	CLSTK	
	CLall					RESET	
CONST							*catalog* of constants, see pp. 135ff
CPX	dot	cross	UNITV	Re	conj	Re⇄Im	special *complex* functions
	CX→RE	RE→CX		Im	|x|	∡	
DISP	FIX	SCI	ENG	ALL	ROUNDI	ROUND	display rounding and shifts, formats and settings, mostly for *reals*
	SDL	SDR			RDP	RSD	
	CHINA	EUROPE	INDIA	JAPAN	UK	USA	
	GAP		RANGE	RANGE?		DSTACK	

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

Menu							Remarks
LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG	
	DEC					INC	
MATX	NEW	$[M]^{-1}$	$\lvert M\rvert$	$[M]^T$	SIM EQ	EDIT	matrix operations (the items sorted almost as in *HP-42S*)
	dot	cross	UNITV	DIM	INDEX	EDITN	
	ENORM		STOEL	RCLEL	PUTM	GETM	
	I+	I-	STOIJ	RCLIJ	J-	J+	
	RSUM	RNORM	M.LU	DIM?		R⇄R	
	EIGVAL					EIGVEC	
M.EDIT	←	↑	OLD	GOTO	↓	→	Matrix Editor as in *HP-42S*
	INSR		DELR		WRAP	GROW	
M.SIMQ	Mat A	Mat B				Mat X	solver for systems of linear equations
MODE	SF	DEG	RAD	GRAD	MULπ	CF	mode settings; SYSTEM is not available on the simulator
			RM		SETSIG	DENMAX	
	SYSTEM						
MyMenu							will show up out of *AIM* [39]
Myα							will show up in *AIM* [39]
PARTS	IP	FP	MANT	EXPT	sign	DECOMP	some overlaps with *HP-42S* CONVERT
					$\lvert x\rvert$	∡	
					Re	Im	
PRINT	⎙x	⎙r	⎙Σ	⎙ADV	⎙LCD	⎙PROG	the PRINT commands of the *HP-42S*
	⎙STK	⎙REGS	⎙USER	⎙TAB	⎙#	⎙CHAR	
				⎙WIDTH	⎙DLAY	⎙MODE	

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

Menu	▭	▭	▭	▭	▭	▭	Remarks
PROB	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:	combinations, permutations, random number generators and 14 probability distributions. Selecting one (e.g. Norml) opens a submenu featuring 4 entries for its *PDF* (or *PMF*), *CDF*, error probability, and quantile function
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:	
		NBin:	Geom:	Hyper:	Binom:	Poiss:	
	RAN#	SEED	RANI#		lnΓ	Γ(x)	
Binom:	Binom$_p$	Binom			Binom$_e$	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:	$\chi^2{}_p(x)$	$\chi^2(x)$			$\chi^2{}_e(x)$	$(\chi^2)^{-1}$	
P.FN	INPUT	END	ERR	TICKS	PAUSE	P.FN2	additional programming functions (avoided a *multi-view menu* here).
	PSTO	PRCL	VARMNU	MVAR	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			
	BACK	CASE	SKIP	AGRAPH	PIXEL	POINT	

Menu	▭	▭	▭	▭	▭	▭	Remarks
STAT	Σ+	$\bar{x}$	s	σ	s_m	SUM	for sample statistics.
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}		
	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	PLOT	
	L.R.	r	s_{xy}	cov	$\hat{x}$	$\hat{y}$	for curve fitting and 2d sample statistics.
		$\bar{x}_H$					
		$\bar{x}_{RMS}$					
	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.
TEST	x< ?	x≤ ?	x= ?	x≠ ?	x≥ ?	x> ?	binary tests.
	INT?	EVEN?	ODD?	PRIME?	LEAP?	FP?	
	ENTRY?	KEY?	LBL?	STRI?	CONVG?	TOP?	
	x=+0?	x=-0?	x≈ ?	MATR?	CPX?	REAL?	
	SPEC?	NaN?		M.SQR?			
TRI	sin	arcsin	cos	arccos	tan	arctan	trigonometric & hyperbolic functions (cf. EXP).
	sinh	arsinh	cosh	arcosh	tanh	artanh	
U→	E:	P:	year→s	F&p:	m:	x:	unit conversions (see pp. 144ff).
	°C→°F	°F→°C	s→year		V:	A:	
	power ratio → dB	dB → power ratio			field ratio → dB	dB → field ratio	
A:	acre → m²	m² → acre	ha → m²	m² → ha	$acre_{US}$ → m²	m² → $acre_{US}$	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	

Menu							Remarks
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 $foot_{US}$	

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)	$(-1)^x$			||	
Orthog	H_n	L_m	$L_{m\alpha}$	P_n	T_n	U_n	orthogonal polynomials
	H_{np}						

Menu							Remarks
αINTL	A a	Ā ā	Á á	Ă ă	À à	Ä ä	[α] *catalog* of all Latin letters provided.[40] All letters but one in this *menu* will change when case is switched in *AIM* – note you will see the individual letters displayed in either case only at one time. You will reach each and every letter by 3 keystrokes maximum since letters can be accessed alphabetically (cf. pp. 124f).
	Ã ã	Â â	Å å	Æ æ	Ą ą	B b	
	C c	Ć ć	Č č	Ç ç	D d	Đ đ	
	Ď ď	E e	Ē ē	É é	Ĕ ĕ	È è	
	Ë ë	Ê ê	Ė ė	Ě ě	Ę ę	F f	
	G g	Ğ ğ	H h	I i	Ī ī	Í í	
	Ĭ ĭ	Ì ì	Ï ï	Î î	Į į	İ i	
	I ı	J j	K k	L l	Ł ł	Ĺ ĺ	
	Ľ ľ	M m	N n	Ń ń	Ň ň	Ñ ñ	
	O o	Ō ō	Ó ó	Ŏ ŏ	Ò ò	Ö ö	
	Õ õ	Ô ô	Ø ø	Œ œ	P p	Q q	
	R r	Ř ř	Ŕ ŕ	S s	ß	Ś ś	
	Š š	Ş ş	T t	Ť ť	Ţ ţ	U u	
	Ū ū	Ú ú	Ŭ ŭ	Ù ù	Ü ü	Ũ ũ	
	Û û	Ů ů	Ų ų	V v	W w	Ŵ ŵ	
	X x	Y y	Ŷ ŷ	Ý ý	Ÿ ÿ	Z z	
	Ź ź	Ž ž	Ż ż				

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

Menu	▭	▭	▭	▭	▭	▭	Remarks
αMATH	<	≤	=	≈	≥	>	[α] for comparison symbols, parentheses & brackets, as well as more mathematical and related symbols. You can reach every character by 3 keystrokes maximum.
	{	[	(	)	]	}	
	× / · [41]	÷ / :	∫	$_{\infty}$	$^{\infty}$	∞	
	¬	∧	∨	≠	|	&	
	∡	⊾	⊥	∛	√	ˣ√	
	x̄	ȳ	x̂	ŷ	$\overline{x}$	$\overline{y}$	
	:=	≙	≘	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.[42] Note the two kinds of lower case Σ. See αINTL for more.
	Η η	Θ ϑ	Ι ι	Κ κ	Λ λ	Μ μ	
	Ν ν	Ξ ξ	Ο ο	Π π	Ρ ρ	Σ σ	
	ς	Τ τ	Υ υ	Φ φ	Χ χ	Ψ ψ	
	Ω ω	ά	έ	ή	ί	ΐ	
	Ϊ ϊ	ό	ύ	Ϋ ϋ	ΰ	ώ	

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

[42] The Greek alphabet (sic!) goes alpha, beta, gamma, delta, e-psilon, zeta, eta, theta, iota, kappa, lambda, my, ny, xi, o-mikron, pi, rho, sigma, tau, y-psilon, phi, chi, psi, o-mega. 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 – these suffice: there is no goose named Seitsemän, Kahdeksan, Yhdeksän, nor Kymmenen in that novel).

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

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

Menu							Remarks
∡→	→DEG	→RAD	→GRAD		→D.MS	→MULπ	angular conversions, cf. pp. 150f.
	DEG→	RAD→	GRAD→		D.MS→	MULπ→	
	D→R	R→D		D→D.MS	D.MS→D		

Constants

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

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

Names of astronomical and mathematical constants are printed on colored background in the table starting overleaf. Values of physical constants (including their relative standard deviations in red print below) are printed on light background if they are exactly defined or almost exactly known – the darker the background, the less precisely the particular value is known.[43] We use commas as radix marks for better visibility and multiplication dots for space reasons. Formulas are printed where applicable.

Name	Numeric value and *rel. SD*	Remarks
a (0)[44]	365,242 5 d (*per definition*)	*Gregorian* year
a_0	5,291 772 109 03·10^{-11} m *(1,5·10^{-10})*	*Bohr* radius $a_0 = {}^{\alpha}/_{4\pi\, R_\infty}$
a_{Moon}	3,844·10^{8} m *(1·10^{-3})*	Semi-major axis of the Moon's orbit around the earth ≈ 1,3 *light seconds*.

---

[43] 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 the fundamental constants (printed **bold** in the table) of physics all feature less than 16 significant digits.

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

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.

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

Name	Numeric value and *rel. SD*	Remarks
$a_{\oplus}$	1,495 979·10^{11} m *(1·10^{-6})*	Semi-major axis of the Earth's orbit around the sun. Within the uncertainty stated here, it equals 1 *astronomic unit* ≈ 499 *light seconds* ≈ 8 *light minutes*.
c	**2,997 924 58·10^{8} $^{\mathbf{m}}/_{\mathbf{s}}$** ***(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 (5)	3,741 771 852…·10^{-16} W m^2 *(exact)*	First radiation constant $c_1 = 2\pi\, h\, c^2$
c_2	0,014 387 768 77… m·K *(exact)*	Second radiation constant $c_2 = {}^{hc}/_{k}$
e	**1,602 176 634·10^{-19} 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… $^{\mathrm{A\,s}}/_{\mathrm{mol}}$ *(exact)*	*Faraday* constant $F = e\, N_A$
F_α (10)	2,502 907 875 095 892 8...	*Feigenbaum's* α and δ
F_δ	4,669 201 609 102 990 6...	
G	6,674 30·10^{-11} $^{\mathrm{m^3}}/_{\mathrm{kg\,s^2}}$ *(2,2·10^{-5})*	*Newtonian* constant of gravitation; also known as γ from other authors. See $GM_{\oplus}$ below for a more precise value.
G_0	7,748 091 729…·10^{-5} / Ω *(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

Name	Numeric value and *rel. SD*	Remarks
g_e (15)	−2,002 319 304 362 56 *(1,7·10^{-13})*	*Landé's* electron g-factor
$GM_\oplus$	3,986 004 418·10^{14} $^{\mathrm{m}^3}/_{\mathrm{s}^2}$ *(2,0·10^{-9})*	*Newtonian* constant of gravitation times the Earth's mass with its atmosphere included (according to *WGS84* [45])
$g_\oplus$	9,806 65 $^{m}/_{s^2}$ *(per def.)*	Standard earth acceleration
h	**6,626 070 15·10^{-34} J s *(exact)***	***Planck* constant**
$\hbar$	1,054 571 817…·10^{-34} J s *(exact)*	Reduced *Planck* constant $\hbar = {}^{h}/_{2\pi}$
k (20)	**1,380 649·10^{-23} $^{\mathrm{J}}/_{\mathrm{K}}$ *(exact)***	***Boltzmann* constant** $k = {}^{R}/_{N_A}$
K_J	4,835 978 484…·10^{14} $^{\mathrm{Hz}}/_{\mathrm{V}}$ *(exact)*	*Josephson* constant $K_j = {}^{2e}/_{h}$
l_{PL}	1,616 255·10^{-35} m *(1,1·10^{-5})*	*Planck* length $l_{PL} = t_{PL}c$
m_e	9,109 383 701 5·10^{-31} kg *(3,0·10^{-10})*	Electron mass $\triangleq$ 511,00 keV
M_{Moon}	7,349·10^{22} kg *(5·10^{-4})*	Mass of the Moon
m_n (25)	1,674 927 498 04·10^{-27} kg *(5,7·10^{-10})*	Neutron mass $\triangleq$ 939,57 MeV
m_n/m_p	1,001 378 419 31 *(4,9·10^{-10})*	Neutron to proton mass ratio

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

Name	Numeric value and *rel. SD*	Remarks
m_p	$1{,}672\ 621\ 923\ 69 \cdot 10^{-27}$ kg *($3{,}1 \cdot 10^{-10}$)*	Proton mass $\triangleq 938{,}27$ MeV
m_{PL}	$2{,}176\ 435 \cdot 10^{-8}$ kg *($1{,}1 \cdot 10^{-5}$)*	*Planck* mass $m_{PL} = \sqrt{\hbar c / G} \approx 22\ \mu g$
m_p/m_e	$1\ 836{,}152\ 673\ 43$ *($6{,}0 \cdot 10^{-11}$)*	Proton to electron mass ratio
m_u (30)	$1{,}660\ 539\ 066\ 60 \cdot 10^{-27}$ kg *($3{,}0 \cdot 10^{-10}$)*	Atomic mass constant $\approx 10^{-3}\ \mathrm{kg}/N_A$
$m_u c^2$	$1{,}492\ 418\ 085\ 60 \cdot 10^{-10}$ J *($3{,}0 \cdot 10^{-10}$)*	Energy equivalent of the atomic mass constant $\approx 931{,}49$ MeV
m_μ	$1{,}883\ 531\ 627 \cdot 10^{-28}$ kg *($2{,}2 \cdot 10^{-8}$)*	Muon mass $\triangleq 105{,}66$ MeV
$M_\odot$	$1{,}989\ 1 \cdot 10^{30}$ kg *($5 \cdot 10^{-5}$)*	Mass of the Sun
$M_\oplus$	$5{,}973\ 6 \cdot 10^{24}$ kg *($5 \cdot 10^{-5}$)*	Mass of the Earth. See $GM_\oplus$ above for a more precise value.
N_A (35)	**$6{,}022\ 140\ 76 \cdot 10^{23}$ / mol** ***(exact)***	***Avogadro's* number**
NaN	*Not a Number*	See p. 170 and the corresponding entry in *Section 5* of the *OM.*
p_0	101 325 Pa *(per def.)*	Standard atmospheric pressure
R	$8{,}314\ 462\ 618\ldots\ \mathrm{J}/\mathrm{mol\ K}$ *(exact)*	Molar gas constant
r_e	$2{,}817\ 940\ 326\ 2 \cdot 10^{-15}$ m *($4{,}5 \cdot 10^{-10}$)*	Classical electron radius $r_e = \alpha^2\, a_0$

Name	Numeric value and *rel. SD*	Remarks	
R_K (40)	25 812,807 45… Ω *(exact)*	*Von Klitzing* constant $R_K = {}^{h}\!/_{e^2}$	
R_{Moon}	1,737 530·10^{6} m *(5·10^{-7})*	Mean radius of the Moon	
R_∞	10 973 731,568 160 / m *(1,9·10^{-12})*	*Rydberg* constant $R_\infty = \dfrac{\alpha^2 m_e c}{2h}$	
$R_\odot$	6,96·10^{8} m *(5·10^{-3})*	Mean radius of the sun	
$R_\oplus$	6,371 010·10^{6} m *(5·10^{-7})*	Mean radius of the Earth	
Sa (45)	6,378 137 0·10^{6} m *(p. def.)*	Semi-major axis	… according to *WGS84* (see footnote 45 on p. 138).
Sb	6,356 752 314 2·10^{6} m *(1,6·10^{-11})*	Semi-minor axis	
Se^2	6,694 379 990 14·10^{-3} *(1,5·10^{-12})*	First eccentricity squared	
Se'^{2}	6,739 496 742 28·10^{-3} *(1,5·10^{-12})*	Second eccentricity squared	
Sf^{-1}	298,257 223 563 *(per def.)*	Flattening parameter	
T_0 (50)	273,15 K *(per definition)*	= 0°C, standard temperature	
T_P	1,416 785·10^{32} K *(1,1·10^{-5})*	*Planck* temperature $T_P = \dfrac{c^2}{k}\sqrt{\dfrac{\hbar c}{G}} = \dfrac{M_P c^2}{k} = \dfrac{E_P}{k}$	
t_{PL}	5,391 245·10^{-44} s *(1,1·10^{-5})*	*Planck* time $t_{PL} = {}^{l_{PL}}\!/_{c}$	

Name	Numeric value and *rel. SD*	Remarks
V_m	0,022 413 969 54… $^{m^3}/_{mol}$ *(exact)*	Molar volume of an ideal gas at standard conditions $V_m = \frac{RT_0}{p_0} \approx$ 22,4 l/mol
Z_0	376,730 313 668 Ω *($1{,}5\cdot10^{-10}$)*	Characteristic impedance of vacuum
α (55)	7,297 352 569 3$\cdot10^{-3}$ *($1{,}5\cdot10^{-10}$)*	Fine-structure constant $\alpha = \frac{e^2}{2\varepsilon_0\, h\, c} \approx \frac{1}{137}$
γ	6,674 30$\cdot10^{-11}$ $^{m^3}/_{kg\ s^2}$ *($2{,}2\cdot10^{-5}$)*	*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	2,675 221 874 4$\cdot10^{8}$ $^{Hz}/_{T}$ *($4{,}2\cdot10^{-10}$)*	Proton gyromagnetic ratio $\gamma_p = 4\pi\, ^{\mu_p}/_h$
$\Delta\nu_{Cs}$	**9 192 631 770 Hz *(exact)***	**Hyperfine transition frequency of ^{133}Cs**
ε_0 (60)	8,854 187 812 8$\cdot10^{-12}$ $\frac{A\ s}{V\ m}$ *($1{,}5\cdot10^{-10}$)*	Vacuum electric permittivity $\varepsilon_0 = {}^1/_{\mu_0\, c^2}$ (note the so-called *Coulomb's* constant is just $^1/_{4\pi\varepsilon_0}$)
λ_C	2,426 310 238 67$\cdot10^{-12}$ m *($3{,}0\cdot10^{-10}$)*	*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$\cdot10^{-15}$ m *($5{,}7\cdot10^{-10}$)*	
λ_{Cp}	1,321 409 855 39$\cdot10^{-15}$ m *($3{,}1\cdot10^{-10}$)*	

Name	Numeric value and *rel. SD*	Remarks
μ_0	$1{,}256\,637\,062\,12 \cdot 10^{-6}\ \frac{\mathrm{V\,s}}{\mathrm{A\,m}}$ *($1{,}5 \cdot 10^{-10}$)*	Vacuum magnetic permeability
μ_B (65)	$9{,}274\,010\,078\,3 \cdot 10^{-24}\ \mathrm{J}/\mathrm{T}$ *($3{,}0 \cdot 10^{-10}$)*	*Bohr* magneton $\mu_B = e\hbar/2m_e$
μ_e	$-9{,}284\,764\,704\,3 \cdot 10^{-24}\ \mathrm{J}/\mathrm{T}$ *($3{,}0 \cdot 10^{-10}$)*	Electron magnetic moment
μ_e/μ_B	$-1{,}001\,159\,652\,181\,28$ *($1{,}7 \cdot 10^{-13}$)*	Ratio of electron magnetic moment to *Bohr's* magneton
μ_n	$-9{,}662\,365\,1 \cdot 10^{-27}\ \mathrm{J}/\mathrm{T}$ *($2{,}4 \cdot 10^{-7}$)*	Neutron magnetic moment
μ_p	$1{,}410\,606\,797\,36 \cdot 10^{-26}\ \mathrm{J}/\mathrm{T}$ *($4{,}2 \cdot 10^{-10}$)*	Proton magnetic moment
μ_u (70)	$5{,}050\,783\,746\,1 \cdot 10^{-27}\ \mathrm{J}/\mathrm{T}$ *($3{,}1 \cdot 10^{-10}$)*	Nuclear magneton $\mu_u = e\hbar/2m_p$
μ_μ	$-4{,}490\,448\,30 \cdot 10^{-26}\ \mathrm{J}/\mathrm{T}$ *($2{,}2 \cdot 10^{-8}$)*	Muon magnetic moment
σ_B	$5{,}670\,374\,419\ldots \cdot 10^{-8}\ \frac{\mathrm{W}}{\mathrm{m^2\,K^4}}$ *(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 $\Phi = \frac{1}{2}(1+\sqrt{5})$
Φ_0	$2{,}067\,833\,848\ldots \cdot 10^{-15}\ \mathrm{V\,s}$ *(exact)*	Magnetic flux quantum $\Phi_0 = \frac{h}{2e}$

Name	Numeric value and *rel. SD*	Remarks
ω (75)	$7{,}292\,115 \cdot 10^{-5}\ \mathrm{rad}/_{s}$ *($2 \cdot 10^{-8}$)*	Angular velocity of the Earth according to *WGS84* (see footnote 45 on p. 138).
-∞	$-\infty$	Note both these 'constants' are counted as numeric values in your *WP 43S*. They can be recalled and used with SPCRES set, else an error will be thrown.
∞	∞	

Some values found in nature are known with up to one or two digits precision only – thus, they are not stored in CONST but just printed here since they may be helpful for quick estimates:

Radius of an atomic nucleus	$\sim 10^{-15}$ m
Radius of an atom [46]	$\sim 10^{-10}$ m
Radius of the observable universe [47]	$\approx 45 \times 10^{9}\ l.y. \approx 4.3 \times 10^{26}$ m
Number of stars in our home galaxy	$\sim 2 \times 10^{11}$
Number of neuron connections in human brain	$\sim 10^{14}$
Number of stars in observable universe	$\sim 10^{23}$
Amount of atoms in the Sun [48]	$\sim 10^{57}$

---

[46] So the nucleus takes far less than a billionth of the volume of an atom. Electrons are even smaller. Thus, an atom is almost completely empty space. Our world as we know and see it every day is built of atoms. You can even touch these objects, though not all. Think about it!

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

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

[48] You can take this number for the amount of atoms in our solar system as well.

Amount of atoms in observable universe $\sim 10^{80}$

Baryonic mass in observable universe $\sim 10^{53}$ kg

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

Unit Conversions

Your *WP 43S* features 14 angular conversions provided in ∡→ (cf. p. 135) 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 p. 131 for more details of the structure.

Conversions contained in U→ either begin or end in one of the seven 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*	$1\ \mathrm{W} = 1\ \mathrm{V\ A} = 1\ {}^{\mathrm{J}}/_{\mathrm{s}}$
Electric potential	*volt*	$1\ \mathrm{V} = 1\ {}^{\mathrm{W}}/_{\mathrm{A}}$
Charge	*coulomb*	$1\ \mathrm{C} = 1\ \mathrm{A\ s}$
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 below is beneficial, covering 36 orders of magnitude:

Prefix	**Name**	**Factor**	**Prefix**	**Name**	**Factor**
			d	*deci-*	10^{-1}
h	*hecto-*	10^{2}	c	*centi-*	10^{-2}
k	*kilo-*	10^{3}	m	*milli-*	10^{-3}
M	*mega-*	10^{6}	µ	*micro-*	10^{-6}
G	*giga-*	10^{9}	n	*nano-*	10^{-9}
T	*tera-*	10^{12}	p	*pico-*	10^{-12}
P	*peta-*	10^{15}	f	*femto-*	10^{-15}
E	*exa-*	10^{18}	a	*atto-*	10^{-18}

All conversions featured in U→ and ∡→ are explained in alphabetical order below. Numeric values 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 here for better visibility.

Softkey	**Calculation**	**Remarks**	**Branch**
°C→°F	× 1,8 + 32		U→
°F→°C	– 32) / 1,8		U→
acre→m^2	× 4 046,86	These *acres* are based on the *'international feet'*, see below	U→ f A:
acre$_{US}$→m^2	× 4 046,87	These *acres* are based on the *U.S. survey feet*, see below	U→ f A:
atm→Pa	× 1,013 25×10^{5}	*Atmospheres*	U→ F&p:
au→m	× 1,495 98×10^{11}	*Astronomic units*	U→ x:

Softkey	**Calculation**	**Remarks**	**Branch**
barrel→m³	× 0,158 987	*(U.S.) barrels* of oil, abbr. *bbl*	U→ f V:
bar→Pa	× 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* := 2 *yards* = 6 *feet*	U→ x:
field ratio→dB	$20\ lg\left({}^{a_1}/_{a_2}\right)$	Also known as *amplitude ratio*	U→ ▼
$floz_{UK}$→m³	× 2,841 31×10^{-5}	*Fluid ounces*	U→ f V:
$floz_{US}$→m³	× 2,957 35×10^{-5}		
ft.→m	× 0,304 8	These are the so-called *'international feet'* of 1959 1 *foot* := 12 *inches*	U→ x:
gl_{UK}→m³	× 4,546 09×10^{-3}	*Gallons*	U→ f V:
gl_{US}→m³	× 3,785 42×10^{-3}		
ha→m²	× 10 000	*Hectares*	U→ f A:
hp_E→W	× 746	*Electric horsepower*	U→ P:
hp_M→W	× 735,498 8	So-called *'metric' horsepower* (equivalent to PS in German)	
hp_{UK}→W	× 745,699 9	*British Imperial horsepower*	
in.→m	× 0,025 4	1 *inch* := 1000 *mil*	U→ x:
in.Hg→Pa	× 3 386,39	*Inches of mercury*	U→ F&p:

Softkey	Calculation	Remarks	Branch
J→Btu	/ 1 055,06	*Joules*	U→ E:
J→cal	/ 4,186 8		
J→Wh	/ 3 600		
kg→carat	/ 0,000 2	1 t [(*metric*) *ton*] = 1000 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:
ly→m	× $9,460\ 73 \times 10^{15}$	*Light years*	U→ x:
m^2→acre	/ 4 046,86	*Square meters*; 1 a [*are*] = 100 m^2, 1 ha [*hectare*] = 10 000 m^2, 1 km^2 = 100 ha = 1 000 000 m^2	U→ f A:
m^2→$acre_{US}$	/ 4 046,87		
m^2→ha	/ 10 000		
m^3→barrel	/ 0,158 987	*Cubic meters*; 1 l [*liter*] = 1 dm^3 = 10^{-3} m^3, 1 ml [*milliliter*] = 1 cm^3	U→ f V:
m^3→$floz_{UK}$	/ $2,841\ 31 \times 10^{-5}$		
m^3→$floz_{US}$	/ $2,957\ 35 \times 10^{-5}$		
m^3→gl_{UK}	/ $4,546\ 09 \times 10^{-3}$		
m^3→gl_{US}	/ $3,785\ 42 \times 10^{-3}$		
m^3→qt.	/ $1,1365 \times 10^{-3}$		

Softkey	Calculation	Remarks	Branch
mi.→m	× 1 609,344	1 *mile* = 1 760 *yards*	
m→au	/ $1{,}495\ 98\times10^{11}$	*Meters*	U→ x:
m→fathom	/ 1,828 8		
m→ft.	/ 0,304 8		
m→in.	/ 0,025 4		
m→ly	/ $9{,}460\ 73\times10^{15}$		
m→mi.	/ 1 609,344		
m→nmi.	/ 1 852		
m→pc	/ $3{,}085\ 68\times10^{16}$		
m→point	/ $352{,}778\times10^{-6}$		
m→survey foot$_{US}$	/ 0,304 801		
m→yd.	/ 0,914 4		
nmi.→m	× 1 852	*Nautical miles*	
N→lbf	/ 4,448 22	*Newtons*	U→ F&p:
oz→kg	× 0,028 349 5	*Ounces*	U→ m:
Pa→atm	/ $1{,}013\ 25\times10^{5}$	*Pascals*; 1 hPa = 1 mbar	U→ F&p:
Pa→bar	/ 10^{5}		
Pa→in.Hg	/ 3 386,39		
Pa→psi	/ 6 894,76		
Pa→torr	/ 133,322		
pc→m	× $3{,}085\ 68\times10^{16}$	*Parsecs*	U→ x:
point→m	× $352{,}778\times10^{-6}$	*(Typographical) points*	
power ratio→dB	$10\ lg\left({}^{p_1}/_{p_2}\right)$		U→ ▼

Softkey	Calculation	Remarks	Branch
psi→Pa	× 6 894,76	*Pounds per square inch*	U→ F&p:
qt.→m³	× 1,1365×10⁻³	*(Imperial) quarts*	U→ f V:
short cwt→kg	× 45,359 2	1 *short hundredweight* = 100 lbs	U→ m:
short ton→kg	× 907,185	1 *short ton* = 2000 lbs	
stone→kg	× 6,350 29		
survey foot$_{US}$→m	× 0,304 801	1 U.S. *survey foot* $:= \frac{1200}{3937}\ m$	U→ x:
s→year	/ 31 556 952		U→
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:
W→hp$_E$	/ 746	*Watts*	U→ P:
W→hp$_M$	/ 735,498 8		
W→hp$_{UK}$	/ 745,699 9		
yd.→m	× 0,914 4	1 *yard* := 3 *feet*	U→ x:
year→s	× 31 556 952	= 365,242 5 × 24 × 60^2	U→

L→ …	Remarks
DEG→	Takes an integer or *real* [49] x as an angular input in *decimal* or *sexagesimal degrees*, respectively, and converts it to the current *ADM*.
D.MS→	

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

…	Remarks
D.MS→D	Takes an integer or *real*[49] ***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*[49] ***x*** as an angular input in *decimal degrees* and converts it to … *radians*.[50]
D→D.MS	Takes an integer or *real*[49] ***x*** as an angular input in *decimal degrees* and converts it to … *sexagesimal degrees* (corresponding to the old command H→H.MS).
GRAD→	Takes an integer or *real*[49] ***x*** as an angular input in … *grad/gon* … and converts it to the current *ADM.*
MULπ→	Takes an integer or *real*[49] ***x*** as an angular input in … *multiples of π* … and converts it to the current *ADM.*
RAD→	Takes an integer or *real*[49] ***x*** as an angular input in … *radians*[50] … and converts it to the current *ADM.*
R→D	Takes an integer or *real*[49] ***x*** as an angular input in *radians* and converts it to *decimal degrees* (equaling the old command R→D).[50]
→DEG →D.MS →GRAD →MULπ →RAD	Takes an integer or *real* (*data type* 2) ***x*** as an angular input in the current *ADM* and converts it to *decimal degrees*, *sexagesimal degrees*, *grad/gon*, *multiples of π*, or *radians*, respectively.[50] If ***x*** is a tagged *real* (*data type* 4), on the other hand, this information is used in conversion (e.g. if ***x*** = **1.5*π*** then →GRAD will return **300^g** regardless of current *ADM*). If ***x*** is neither of *data type* 1, 2, nor 4, error 24 will be thrown ('illegal input data type for this operation').

Angular output is tagged always.

---

[50] Note that *real* angles given in *radians* cannot represent full circles (or simple fractions of π like π/2, π/3, π/4, π/5, etc.) exactly but with an accuracy of 34 digits 'only'. If you want to avoid most rounding errors caused by that, *multiples of **π*** may be a better choice here. Note large numeric inputs in trigonometric functions will be reduced to values between −π and +π before calculating (as mentioned in *Section 2* of the *OM*).

SECTION 3: CALLING AND EXECUTING OPERATIONS

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

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

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

Using XEQ for Executing Operations

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

1. Press XEQ.
2. Press α. You are in *AIM* 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	**Numeric parameter**	**Alpha par.**
AGRAPH CONVG? DEC DSE DSL DSZ INC INPUT ISE ISG ISZ KEY? KTYP? PUTK RCL RCLCFG RCLS RCL+ RCL- RCL× RCL/ RCL↑ RCL↓ STO STOCFG STOS STO+ STO- STO× STO/ STO↑ STO↓ t≷ VIEW x≷ x=? x≠? x≈? x<? x≤? x≥? x>? y≷ z≷ αLENG? αPOS? αRL αRR αSL α→x 🖶r	*Register* number	Variable *name*
ALL ENG FIX GAP RDP RSD SCI SDL SDR	Number of decimals	
ASR MASKL MASKR RL RLC RR RRC SL SR WSIZE	Number of bits	
BACK CASE SKIP	Number of program steps	
BC? BS? CB FB SB	Bit number	
BestF	Fit model code	
CF FC? FC?C FC?F FC?S FF FS? FS?C FS?F FS?S SF	*User flag* number	*System flag name*
CONST	Constant number	
DSTACK	Number of *stack registers*	
ERR MSG	Error number	
f'(x) f''(x) GTO LBL LBL? PGMINT PGMSLV XEQ Π_n Σ_n		Program label
GTO.	Number of program step	Label

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

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

XYZ → ST.X

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

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

Operation requiring four trailing parameters	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} $J_y(x)$ LN LNβ LNΓ LOG_{10} LOG_2 $LOG_x y$ MANT POISS... SCI sin tan W_m W_p W^{-1} $\sqrt[x]{y}$ β(x,y) $Γ_{xy}$ $γ_{xy}$ Γ(x) Δ% →REAL % %MRR %T %Σ %+MG $\sqrt{x}$ || as well as all unit conversions and all orthogonal polynomials	2 [51]	**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**

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

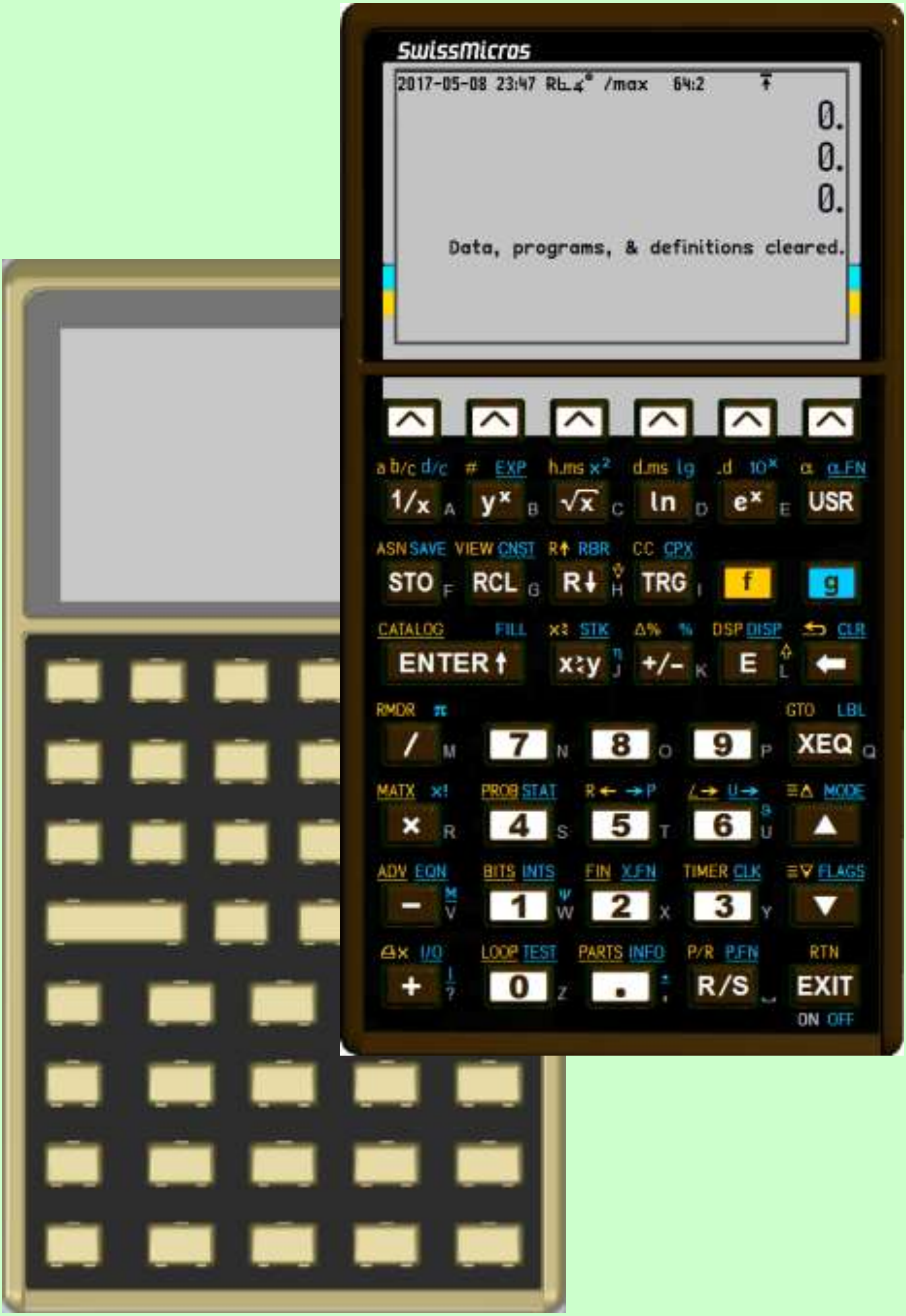
Input *DT*	Operation(s)	Output *DT*	Output *registers* involved
2	AND CEIL DATE→ DAY D→J EXPT FLOOR IDIV IDIVR IP MONTH NAND NEXTP NOR NOT OR ROUNDI SIGN WDAY XNOR XOR YEAR ±∞?	1	**X**
	DECOMP	1	**X, Y**
	SLVQ	2 or 3	**X, Y, Z**
	RE→CX	3	**X**
	arccos arcsin arctan and all angular conversions	4	**X**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

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

APPENDIX A: HARDWARE

Overall dimensions:	wedge-shaped: 77 mm × 144 mm × 13 mm or 8 mm (see p. 159)
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. 163ff), 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. 212f).

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
/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 ten *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*),
- *flag* words for storing 128 (i.e. 112 global plus 16 local) *user flags* and the same amount of *system flags*,[52]
- 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*:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
pointer to the data															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
pointer to the variable *name*					**0**	**0**	data information[53]					*data type* (as specified below) − 1			

[52] Up to 256 *flags* would be possible in total. We need far less *system flags* so far.

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

	***Data type* number and meaning**	**Size** [*bytes*]
1	ℤ *Long integer* [54]	≥ 4 + 2 + 4 = 10…422
2	ℝ *Real* number [55] (*real34*)	4 + 16 = 20
3	ℂ *Complex* number (*complex34*)	4 + 2 × 16 = 36
4	*Angle* [56] (*angle34*)	4 + 16 = 20
5	*Time* [57]	4 + 16 = 20
6	*Date* [58]	4 + 16 = 20
7	*Alpha string* (each character requires 16 *bits*)[59]	4 + 2 + ***n*** × 2
8	*Real* matrix (featuring ***n*** rows and ***m*** columns)	4 + ***n*** × ***m*** × 16
9	*Complex* matrix (featuring ***n*** rows and ***m*** columns)	4 + ***n*** × ***m*** × 32
10	*Short integer* [60]	≤ 4 + 8 = 8 or 12
11 … 13	n/a	n/a
14	Constant [61]	4 + 0 = 4

[54] This *data type* is for number theory kind of problems. 2 *bytes* are for the size (in *bits*) of the integer following. 4 *bytes* following 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. Maximum integer size is 3328 *bits* (= 416 *bytes*) equivalent to 1001 decimal digits. Look up the display limits further below.

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

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

[57] 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. The format allows for expressing intervals of some 100 million years with *femtoseconds* precision.

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

[59] 2 *bytes* are for the size (in *bytes*) of the string following, including the trailing zero. The size must be even.

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

[61] A pointer is sufficient regardless of the precision of the constant itself.

	***Data type* number and meaning**	**Size** [*bytes*]	
15	Extended precision *real* (39 digits, exclusively for internal use, not exposed to the user) [62]	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) [63]	4 + M	
22	Program step, may be stored as *alpha string* (7) [64]	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	

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.

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

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

[64] 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 14 of the *WP 34S* and *WP 31S* 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. 58 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 Real Numbers

Your *WP 43S* could calculate with *real* numbers of more than 12 000 orders of magnitude. Within a range of $10^{-6143} \le |x| < 10^{+6145}$, it computes with 34 digits precision.[65] Results should be accurate within

[65] 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 about thousand digits for proper reduction to the standard range for trigonometric functions).

More elaborate algorithms may be coded as keystroke programs to save flash space (for the cost of execution speed and the loss of a few digits of accuracy). The internal formats used for storing numbers in your *WP 43S* (as shown just above) 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

- 8,999 999 999 999 999 999 999 999 999 937 535 = 9 − 6,246 5·10^{-29} .

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 net works as follows: Enter 1,000 000 1 and then press **x^2** just 27 times. Your *WP 43s* will return

$\pm 1\times10^{-33}$ (e.g. ***n*** [1/x] ***2n*** [×] returns a plain 2 for all primes < 500 – just [7] [1/x] [1] [4] [×] [2] [−] returns 1×10^{-33}, and for 67 and 83 results are -1×10^{-33}). Note that also rounding errors due to calculating with a finite number of digits will accumulate as demonstrated by two examples in the footnote here.

Results $|x| < 10^{-999}$ are set to zero. For results $|x| \geq 10^{999}$, error 4 or 5 will appear unless SPCRES is set (see *App. C*).

All these effects are caused by the **internal representation of *reals***: *Standard floating point numbers* are stored on your *WP 43S* in sixteen bytes using an internal format coarsely following *decimal128* packed coding,[66] though with some exceptions:

- *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 as pure integer in eleven groups of three digits. Each such group is packed into 10 bits straight forward, meaning e.g. 555_{10} = $10\ 0010\ 1100_2$ = $22B_{16}$ or 999_{10} = $11\ 1110\ 0111_2$ = $3E7_{16}$. So the 33 rightmost decimal digits of the *significand* take the least significant 110 bits. Trailing zeroes are omitted, so the *significand* will be right adjusted.
- The most significant (128th) bit takes the sign of the mantissa.
- The remaining 17 bits are used for the exponent and the leftmost digit of the mantissa. Of those 17, the lowest 12 are reserved for the

---

- 674 530,470 741 084 559 382 689 1*84 727 772 2*.

This is the most precise result known of a pocket calculator so far (*WP 34S* in *Double Precision* and *Free42* concur, computing with 34 digits as well). Nevertheless, only the first 25 digits of this output are correct! Calculating with unlimited precision returns

- 674 530,470 741 084 559 382 689 178 029 746 812 844 444 143 410 34

instead for the first 50 of the 10^9 digits of the complete result (note you get over 1000 decimals after 8 presses of [x^2] already).

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

[66] It comes close to what is called quadruple (precision) in this text about floating point arithmetic: https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF . Find out about *decimal128* in https://en.wikipedia.org/wiki/Decimal128_floating-point_format .

exponent (< 4096). For the top 5 bits (below the sign bit) it becomes complicated.[67] 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 (see below).

The maximum absolute value of the stored exponent is $10\ 1111\ 1111\ 1111_2 = 12\ 287_{10}$. For reasons becoming obvious below, 6176 must be subtracted from the stored value to get the true exponent of the floating point number represented. Thus and since $12287 - 6176 + 34 = 6145$ as well as $-6176 + 34 = -6142$, *data type* 2 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. 144 for the reasons).

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

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	*SE*
1.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0010 0000 1000 00	6176
−1.	a2 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		1010 0010 0000 1000 00	6176
111.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 6f	06f	0010 0010 0000 1000 00	6176

[67] Don’t blame us – this part follows the standard *IEEE 754*.

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	*SE*
111.111 (111 111×10^{-3})	22 07 40 00 00 00 00 00 00 00 00 00 00 01 bc 6f	06f 06f	0010 0010 0000 0111 01	6173
−123.000 123 (−123 000 123 ×10^{-6})	a2 06 80 00 00 00 00 00 00 00 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0000 0110 10	6170
9.99×10^{99} (999×10^{97})	22 20 40 00 00 00 00 00 00 00 00 00 00 00 03 e7	3e7	0010 0010 0010 0000 01	6273
1×10^{-99}	21 ef 40 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0001 1110 1111 01	6077
−1×10^{-999}	a1 0e 40 00 00 00 00 00 00 00 00 00 00 00 00 01		1010 0001 0000 1110 01	5177

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} in default rounding mode (and in RM 1, 2, 3, and 5, see the command RM). Divide it by 2 instead and the result will become zero.

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

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	*SE*
9.999 999 999 999 999 999 999 999 999 999 999×10^{998} (= 99…999×10^{965})	6e f9 7e 7f 9f e7 f9 fe 7f 9f e7 f9 fe 7f 9f e7	9 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7	0110 1110 1111 1001 01	7141

This *real* number (featuring 36 times the digit 9) is the maximum which can be keyed in directly. In startup default format, it will be displayed on your *WP 43S* as 1.000 000 000 000 000×10[999] – its full content 9.999 999 999 999 999 999 999 999 999 999 999×10[998] will be only unveiled by SHOW (see p. 73). The greatest legal *significand* is 9 999 999 999 999 999 999 999 999 999 999 999 $= 10^{34} - 1$.

Additionally, your *WP 43S* features three 'special reals':

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

An exponent is not applicable here. **These three 'special reals' may be legal results of your *WP 43S* <u>if SPCRES is set</u>** – no error will be thrown then. '*Not a Number*' (NaN) covers poles as well as regions where a function result is not defined at all – see the corresponding entry in *Section 5* of the *OM* and examples in next chapter. Note that **+∞ and −∞ may be also legal numeric inputs on your *WP 43S*.**

Remember not every 34-digit number displayed will be true to 34 digits – cf. footnote 65 on pp. 166f. And errors accumulate as explained in footnote 43 on p. 136.

As mentioned above, some calculations are executed in "*internal high precision*". This means even more digits than 34 – it may go up to some hundred digits in special cases.

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

Numeric Limitations

Maximum numeric input for *data type* …

- **1:** $\pm 10^{1001}$ (if your patience will suffice for the input), 2^{3326} (if you are less patient)
- **2:** $\pm 9.999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999\ 999 \times 10^{998} \approx 10^{999}$
- **3:** $\pm 9.999\ldots \times 10^{998}$ for either part
- **4:** $\pm 9.999\ldots \times 10^{998}$ in arbitrary *ADM,* so the actual absolute maximum is $9.999\ldots \times 10^{998}\ \boldsymbol{\pi}$
- **5:** times xxx
- **6:** dates xxx
- **8:** $\pm 9.999\ldots \times 10^{998}$ for each matrix element
- **9:** $\pm 9.999\ldots \times 10^{998}$ for either part of each matrix element
- **10:** FF FF FF FF FF FF FF FF_{16} in unsigned mode, equivalent to some 1.8×10^{19}.

Internal limits:

- PRIME? works like the other binary tests described in the *OM*. Above 3 317 044 064 679 887 385 961 981, however, NEXTP cannot find primes anymore but 'probable primes' only (cf. p. 61). If you want to apply NEXTP or PRIME? above this limit, consult appropriate reference literature.[68]
- Trigonometric functions actually operate between $+\boldsymbol{\pi}$ and $-\boldsymbol{\pi}$ (or their equivalents) exclusively. The necessary modulo reduction ensures that these functions return correct 34-digit results for the entire legal input range ($-10^{999} < x < 10^{999}$).
- Any intermediate result exceeding $-10^{999} < x < 10^{999}$ (or $-10^{999} \leq x \leq 10^{999}$ for *long integers*) will be rated as −∞ or +∞, respectively,

[68] See https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test for a start (also available in other languages). With a bit of care, NEXTP can find 'probable primes' up to some 6.3×10^{1001} on your *WP 43S*, and PRIME? will check them and their neighbouring numbers but you cannot see all their 1002 digits.

and will then be treated according to the actual system settings at calculation time (cf. SPCRES in particular).

- Any intermediate result within $-10^{-999} < x < 10^{-999}$ will be set to 0.

Maximum numeric output for *data type* …

- **1:** ±10^1001 with full 1001-digit precision (though you can see and read a maximum of 294 digits "only" of such a number (cf. SHOW on p. 73).
- **2, 3, 8, 9, and 10:** The maxima are as specified for input above.
- **4:** For angular conversions, the maxima are as specified for input above. The functions ARCSIN, ARCCOS, and ARCTAN return values between $-\pi$ and π (or their equivalents) only.
- **5:** xxx
- **6:** xxx

Special Results (as of 2020-03-30)

Within this chapter, SPCRES 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 (cf. the end of previous chapter). In this chapter, results were crosschecked against the *WP 34S* wherever possible. Deviations are highlighted. Additionally, *Wolfram Alpha* was used for checking results with finite arguments.

The following monadic functions, if called with ℝ lit (i.e. CPXRES clear), return either ∞, -∞, or NaN under the conditions stated below:

Input x	Operation(s)	Output for ℝ lit
-1.	artanh	-∞
0 or 0.	ln, lg, lb x	
0.	1/x	∞
1.	artanh	

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

And the following monadic functions operate also on infinities:

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

For dyadic functions, we combined the respective tables:

Input y	x	Op.(s)	Output for ℝ lit
∞	arbitrary $x \neq -\infty$	[+]	∞ [69]
−∞	arbitrary $x \neq \infty$		−∞ [69]
−∞	∞	[+]	NaN [69]
∞	arbitrary $x \neq \infty$	[−]	∞ [70]
−∞	arbitrary $x \neq -\infty$		−∞ [70]
−∞	−∞	[−]	NaN
∞	∞		
∞	arbitrary $x > 0$	[×]	∞ [69]
−∞	arbitrary $x < 0$		
∞	arbitrary $x < 0$	[×]	−∞ [69]
−∞	arbitrary $x > 0$		
0 or 0.	−∞ or ∞	[×]	NaN [69] (xxx ℂ!)
$0 < y \leq \infty$	0.	[/]	∞
$-\infty \leq y < 0$			−∞
−∞ or ∞	−∞ or ∞	[/]	NaN
0 or 0.	0.	[/], [y^x]	NaN
−∞ or ∞	0. or 0	[y^x]	NaN
$-\infty < y < 0$	non-integer x	[y^x]	NaN
−∞	odd $x > 0$	[y^x]	−∞
−∞	even $x > 0$		∞
∞	arbitrary $x > 0$	[y^x]	∞
arbitrary $y \neq 0$	−∞	[y^x]	0.
	∞		∞
0.	$0 < x < \infty$	$\log_x y$	−∞

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

[70] Swapping x and y will return this result times -1.

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

Some particular returns of elementary transient functions operating near ±∞ are listed in the table below:[71]

<table>
<tr><th colspan="2">Input[72]
Re(x) Im(x)</th><th colspan="2">r(x) φ(x)</th><th>Op.</th><th>Output for ℂ lit</th></tr>
<tr><td>-∞</td><td>—</td><td colspan="2">—</td><td rowspan="2">√x</td><td rowspan="2">∞∡90° = 0.+i×∞</td></tr>
<tr><td>-∞</td><td>0</td><td>∞</td><td>180°</td></tr>
<tr><td>0.</td><td>∞</td><td>∞</td><td>90°</td><td>x²</td><td>∞∡180° = -∞+i×0.</td></tr>
<tr><td>-∞</td><td>—</td><td colspan="2">—</td><td rowspan="3">∛x</td><td>-∞</td></tr>
<tr><td>-∞</td><td rowspan="2">0</td><td>∞</td><td rowspan="2">180°</td><td>∞∡45° = ∞+i×∞ (34S: NaN+i×NaN)</td></tr>
<tr><td>-10^999</td><td>10^999</td><td>1.×10^333 ∡ 60° =
5.×10^332 + i×8.660 254 037×10^332
$= 5 \times 10^{332} \left(1 + i \times \sqrt{3}\right)$</td></tr>
<tr><td colspan="2">—</td><td>10^333</td><td>60°</td><td>x³</td><td>-1.×10^999 + i×0. $\rightarrow -\infty + i \times 0$</td></tr>
<tr><td>-∞</td><td>—</td><td colspan="2">—</td><td rowspan="2">x³</td><td>-∞</td></tr>
<tr><td>-∞</td><td>0</td><td>∞</td><td>180°</td><td>-∞+i×0.</td></tr>
<tr><td>∞</td><td>—</td><td colspan="2">—</td><td rowspan="3">ln</td><td>∞</td></tr>
<tr><td>∞</td><td rowspan="2">0</td><td>∞</td><td rowspan="2">0°</td><td>∞+i×∞ (34S returns ∞+i×0.)</td></tr>
<tr><td>10^999</td><td>10^999</td><td>$\rightarrow \infty + i \times 0$</td></tr>
<tr><td>-10^999</td><td rowspan="2">0</td><td>10^999</td><td rowspan="2">180°</td><td rowspan="3">ln</td><td>$\rightarrow \infty + i\pi$</td></tr>
<tr><td>-∞</td><td>∞</td><td>∞+i×∞ (WP 34S $= \infty + i\pi$)</td></tr>
<tr><td>-∞</td><td>—</td><td colspan="2">—</td><td>NaN</td></tr>
</table>

[71] Red results in the tables are considered wrong although they may concur with the *WP 34S*.

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

Input[72] Re(x)	Im(x)	r(x)	φ(x)	Op.	Output for ℂ lit
∞	∞	∞	45°	ln	∞+i×∞
10^{999}	10^{999}	10^{999}			$\to \infty + i\,{}^{\pi}/_{4}$ (confirm. by *34S* & *WA*)
∞	−∞	∞	−45°	ln	∞−i×∞
10^{999}	-10^{999}	10^{999}			$\to \infty - i\,{}^{\pi}/_{4}$ (conf. by *34S* & *WA*)
0.	∞	∞	90°	ln	∞+i×∞
	10^{999}	10^{999}			$\to \infty + i\,{}^{\pi}/_{2}$ (conf. by *34S* & *WA*)
0.	−∞	∞	−90°	ln	∞−i×∞
	-10^{999}	10^{999}			$\to \infty - i\,{}^{\pi}/_{2}$ (confirm. by *34S* & *WA*)
−∞	∞	∞	135°	ln	∞+i×∞
-10^{999}	10^{999}	10^{999}			$\to \infty + i\,{}^{3\pi}/_{4}$ (conf. by *34S* & *WA*)
−∞	−∞	∞	−135°	ln	∞−i×∞
-10^{999}	-10^{999}	10^{999}			$\to \infty - i\,{}^{3\pi}/_{4}$ (conf. by *34S* & *WA*)
0.	0.	0.	0.	ln	NaN+i×NaN
10^{-999}	0.	10^{-999}	0.		$\to -\infty + i \times 0$
0.	—	—			−∞
0.	∞	∞	90°	e^x	NaN+i×NaN
	10^{999}	10^{999}			-0.926 663+i×0.375 893 (*34S*: NaN+i×NaN)
0.	−∞	∞	−90°	e^x	NaN+i×NaN
	-10^{999}	10^{999}			-0.926 663−i×0.375 893 (*34S*: NaN+i×NaN)
−∞	0	∞	180°	e^x	0.+i×0.
-10^{999}	10^{-999}	10^{999}			0.+i×0.
−∞		∞			NaN+i×NaN

Input[72] Re(x)	Im(x)	r(x)	φ(x)	Op.	Output for ℂ lit
−∞	∞	∞	135°	e^x	NaN+i×NaN
-10^{999}	10^{999}	10^{999}			0.+i×0.
−∞	−∞	∞	-135°	e^x	NaN+i×NaN
-10^{999}	-10^{999}	10^{999}			0.+i×0.

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

At the bottom line, we hope confusion is limited (and 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. 165.

APPENDIX C: MESSAGES AND ERROR CODES

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

Furthermore, there are a number of error messages issued by the operating system. Depending on conditions, the following messages will be displayed. They are listed below in alphabetical order (*EC* means *error code* here):

	EC	Explanations, countermeasures and examples
An argument exceeds the function domain	1	{1, 2, 3, 4, 10} An argument exceeds the domain of the mathematical function called. May be caused by roots of negative numbers or logs of $x \leq 0$ (unless CPXRES is set), by 0^0, $x/0$, $0/0$, $\Gamma(0)$, $\tan(\pm 90°)$ and equivalents, by $\operatorname{artanh}(x)$ for $\lvert \operatorname{Re}(x) \rvert \geq 1$, by $\operatorname{arcosh}(x)$ for $\operatorname{Re}(x) < 1$, etc.[73]
Bad time or date input	2	{2, 5, 6} 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	Predefined variables or menus cannot be deleted.
Distribution parameter out of valid range	16	{1, 2} A parameter specified in **I**, **J**, or **K** is out of valid range for the distribution function called (e.g. if LGNRM is called with $j < 0$).
Flash memory is full	23	Delete a program from *FM* to regain space.

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

	EC	Explanations, countermeasures and examples
`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 data types do not match`	31	Attempt to operate on different data types (e.g. for Boole’s operations: real AND short integer).
`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} The *Solver* did not converge.

	EC	Explanations, countermeasures and examples
`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.
`No such label found`	6	Attempt to address an undefined label.
`No summation data`	28	Attempt to address an un-allocated summation register.
`Operation is undefined in this mode`	13	Caused e.g. by calling a *real*-number operation in *AIM*. Cf. "*illegal input data type for this operation*".
`Out of range`	8	{1, 2, 3, 10} • A number exceeds the valid range. This can be caused by specifying decimals > 16, *word size* > 64, negative *flag* numbers, short integers ≥ 2^{64}, *hours* or *degrees* > 9 000, invalid *dates* or *times*, denominators ≥ 9 999, etc. • A *register* or *flag* address exceeds the valid range of currently allocated *registers* or *flags*. May also happen in indirect addressing or when calling nonexistent local addresses. • An R-operation (e.g. R-COPY) attempts accessing invalid *register* addresses.
`Overflow at +∞`	4	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a number > 0 by 0. • Divergent sum or product or integral. • Positive overflow (see p. 166).

	EC	Explanations, countermeasures and examples
Overflow at -∞	5	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a number < 0 by 0. • Divergent sum or product or integral. • Negative overflow (see p. 166). • Logarithm of 0 (note a logarithm of -0 will return NaN).
Please enter a NEW name	26	Trying to define a new variable or user *menu* with a *name* already in use.
RAM is full	11	May be caused by attempts to write too large routines, allocate too many variables, and the like (see pp. 163ff 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 set 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 SPCRES is set, errors 4 and 5 will not occur at all, and error 1 will happen less frequently, since ±∞ and NaN are legal results then (cf. the corresponding entries in CONST on pp. 135ff and the tables on pp. 170ff). 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
345 3.0 2310
PAULI WALTE


HEWLETT PACKARD
42S
RPN SCIENTIFIC
X: 812.6000
FIX SCI ENG ALL RDX. RDX,


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. 190, one for the *HP-21S* on p. 192, and another one for the *WP 34S* on p. 194. Functions newly introduced with *WP* calculators are compiled on pp. 198ff.

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

Corresponding Operations on *HP-42S*

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

HP-42S	*WP 43S*	Remarks
ACOSH	arcosh	In EXP
ADV	🖶ADV	In PRINT
AIP	Dispensable	You can merge text and numeric data easily 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*.
AOFF	CF ALPHA	

HP-42S	*WP 43S*	**Remarks**
AON	SF ALPHA	
ARCL	Dispensable	Any *register* or variable can take an *alpha string*. Simply press RCL instead.
AROT	αRL or αRR	In α.FN
ASHF	αSL	
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−		
BASE×		
BASE÷		
BASE+/−		
BINM	Dispensable	Press # 2 for converting any closed integer number or integer part in *x* to binary.
BIT?	BS?	In BITS
BST	(▲)	Shortcut works if no *multi-view menu* is open.
CLA	0 STO K	
CLD	Dispensable	Any keystroke will clear *temporary information*.
CLEAR	CLR	
CLKEYS	n/a	See *Section 6* of the *OM*.
CLRG	CLREGS	In CLR
CLST	CLSTK	Press 0 FILL in run mode.

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

HP-42S	*WP 43S*	**Remarks**
INSR	M.INSR	In MATX
INTEG	∫	In ADV
INVRT	$[M]^{-1}$	In MATX
KEYASN	Dispensable	Not needed since no CUSTOM *menu* is featured (see CUSTOM).
LASTx	RCL L	
LBL		Press LBL.
LCLBL	Dispensable	Obsolete since no CUSTOM *menu* is featured (see CUSTOM). Nevertheless, your *WP 43S* provides local labels (see *Section 3* of the *OM*).
LINΣ	Dispensable	Your *WP 43S* runs in ALLΣ mode always.
LIST	n/a	Use 🖶PROG instead.
LOG	LOG_{10}	Press lg.
MAN	CF T	Manual print mode is *startup default* here.
MAT?	MATR?	In TEST
MEAN	$\bar{x}$	In STAT
MOD		Press MOD.
MODES	MODE	
N!	x!	
NEWMAT	M.NEW	In MATX
NORM	n/a	Not featured.
OCTM	Dispensable	Press # 8 for converting any closed integer number or integer part in x to octal.
OLD	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

HP-42S	*WP 43S*	**Remarks**
PRA	🖶r K	In PRINT
PRGM	P/R	
PRLCD	🖶LCD	In PRINT
PROFF	CF T	
PROMPT	Dispensable	Use VIEW , STOP instead.
PRON	SF T	
PRP	🖶PROG	In PRINT
PRSTK	🖶STK	
PRUSR	🖶USER	
PRV	🖶r	
PRX	🖶r X	Press 🖶x.
PRΣ	🖶Σ	In PRINT
PUTM	M.PUT	In MATX
PWRF	PowerF	In STAT
RAN	RAN#	In PROB
RND	ROUND	In PARTS
RNRM	RNORM	In MATX
ROTXY	RL, RLC, RR, and RRC	In BITS
RTN		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 ΣREG?	Dispensable	There are 100 global general purpose *registers* always. Statistical registers are separate.
→DEC	→INT 10	Press # 1 0 ... for closed input.
→HR		Press .d ... for closed input.
→H.MS		Press h.ms ... for closed input.
→OCT	→INT 8	Press # 8 ... for closed input.
→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.

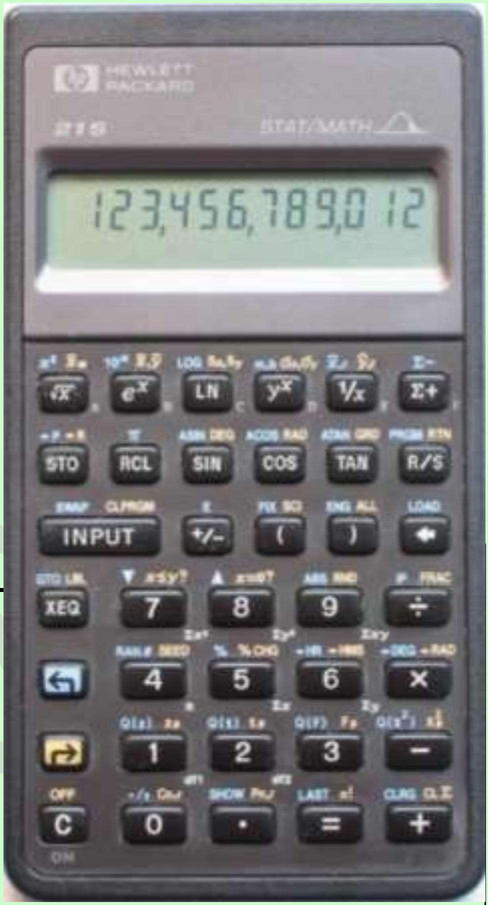
HP-16C	*WP 43S*	Remarks
RL , RLn	RL	In BITS
RR , RRn	RR	In BITS
RLC , RLCn	RLC	In BITS
RRC , RRCn	RRC	In BITS
÷	/	(see also *ISO 80000-2*: “The symbol ÷ should not be used.”)
DBL÷	DBL/	In INTS (see also *ISO 80000-2*: “The symbol ÷ should not be used.”)
x≷(i)	Superfluous	Any *register* may be used for 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	SF LEAD.0s	In FLAGS. Control display of leading zeros.
CF 3	CF LEAD.0s	In FLAGS. 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
INPUT	Dispensable	This functionality is contained in ENTER. Also your *WP 43S* features a command called INPUT but this works in programs.
(,)	Dispensable	You can forget these keys in *RPN.*

HP-21S	*WP 43S*	**Remarks**
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	
FRAC	FP	
÷	/	Cf. *ISO 80000-2*: "The symbol ÷ should not be used."
SEED	SEED	In PROB
%CHG	Δ%	
Q(z)	Norml_e	In *submenus* of PROB. Note your *WP 43S* features the normal distribution for arbitrary μ and σ instead of the standardized one (μ=0, σ=1). And the implementation of 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.
zp	Norml^{-1}	
Q(t)	$t_e(x)$	
tp	$t^{-1}(p)$	
Q(F)	$F_e(x)$	
Fp	$F^{-1}(p)$	
Q(χ²)	$\chi^2{}_e(x)$	
χ²p	$(\chi^2)^{-1}$	
Cn,r	COMB	In PROB
Pn,r	PERM	
LAST	RCL L	
=	Dispensable	You can forget this key in *RPN.*
n!	x!	
CLRG	CLREGS	In CLR

Corresponding Operations on *WP 34S*

The *WP 34S* and *WP 43S* share over 90% of their function sets. It was our objective that your *WP 43S* is equal or better than the *WP 34S* in every aspect. Most of the discrepancies between both calculators are caused by their different displays. 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.
CONV	U→	
DBLOFF	Dispensable	Your *WP 43S* features *34-digit data types* – it does neither need nor feature a *double precision* mode.
DBLON		
Expon$_u$	Expon$_e$	
F$_u$(x)	F$_e$(x)	
dRCL	Dispensable	Your *WP 43S* features various *data types*.

WP 34S	*WP 43S*	**Remarks**
gCLR, gDIM, gDIM?, gFLP, gPIX?, gPLOT, gSET	n/a	The *LCD* of your *WP 43S* features 240 × 400 px rows compared to 6 × 43 px of *HP-30b* – the graphic paradigm of *WP 34S* makes no sense on your *WP 43S*. On the other hand, it was not our objective designing a graphing calculator. Thus, we include just the basic graphic support of *HP-42S* (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.
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.

WP 34S	*WP 43S*	**Remarks**
M×	Dispensable	Your *WP 43S* features two dedicated *data types* for matrices. Thus you can simply multiply two matrices using [×] and copy matrices like any other objects.
M.COPY		
M.IJ, M.REG	Dispensable	Obsolete matrix commands.
nBITS	**#B**	
nCOL, nROW	Dispensable	Obsolete matrix commands.
Norml$_u$	**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.*
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	Use [VIEW] instead; *alpha strings* are just another *data type*. Combine text and numeric data easily using [+] as shown in the OM, *Sect. 2.*
Weibl$_u$	**Weibl$_e$**	
XEQα	Dispensable	Use [XEQ] with an appropriate parameter instead.

WP 34S	*WP 43S*	**Remarks**
XTAL?	Dispensable	A quartz crystal is installed by default.
YDOFF, YDON	Dispensable	Your *WP 43S* displays ***y*** whenever possible and wanted. See DSTACK.
αDATE, αDAY	Dispensable	You can combine text and numeric data easily using + as shown in *Section 2* of the *OM.*
αGTO	Dispensable	Use GTO w/ an appropriate parameter instead.
αIP, αMONTH	Dispensable	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*).
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 μ=0 and σ=1 instead.
$\chi^2{}_u$(x)	$\chi^2{}_e$(x)	
→H	→HR	
🖶PLOT	n/a	See gCLR.
🖶$^C r_{XY}$	Dispensable	Use 🖶r instead.
🖶α, 🖶α+, 🖶+α	Dispensable	Combine text and numeric data easily using + as shown in *Section 2* of the *OM.* Then use 🖶r.
🖶?	Dispensable	A quartz crystal and the proper firmware for printing are installed by default.

New Commands on your *WP 43S*

The following table lists the commands and pseudo-commands created for your *WP 43S* (and for preceding *WP* calculators, if applicable), offering new or extended functionality compared to earlier *HP RPN* and algebraic pocket calculators. In total, these are more than 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 CLPall CONJ CONVG? COV	●	—	new
CX→RE RE→CX	**new**	—	—
DATE TIME	●	—	(●)
DATE→ DAY MONTH YEAR →DATE	●	—	new
DEC DSL INC ISE	●	—	new
DECOMP	●	●	new
DEG→ D.MS→ GRAD→ RAD→	●	—	new
DROP	●	—	new

Command	***WP 43S***	*WP 31S*	*WP 34S*
DROPy DSTACK	**new**	—	—
D→J J→D	●	—	new
EIGVAL EIGVEC	**new**	—	—
ENTRY?	●	—	new
EQ.DEL EQ.EDI EQ.NEW	**new**	—	—
erf erfc ERR MSG	●	—	new
EVEN? ODD?	●	—	new
Expon $Expon_p$ $Expon_e$ $Expon^{-1}$	●	●	new
EXPT MANT	●	—	new
FBR	**new**	—	—
FC?F FC?S FF FS?F FS?S	●	—	new
FIB	●	—	new
FILL	●	●	new
FLASH? FP?	●	—	new
$F_p(x)$ F(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**	—	—
INT? I_{xyz} $I\Gamma_p$ $I\Gamma_q$	●	—	new
$J_y(x)$	**new**	—	—

Command	***WP 43S***	*WP 31S*	*WP 34S*
J/G	**extended**	●	new
KEY? KTYP? LBL? LEAP?	●	—	new
LgNrm LgNrm$_p$ LgNrm$_e$ LgNrm^{-1}	●	—	new
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
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
RANGE RANGE?	**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

Command	***WP 43S***	*WP 31S*	*WP 34S*
RMD	●	●	extended
RM RM? ROUNDI RSD RTN+1 R-CLR R-COPY R-SORT R-SWAP	●	—	new
SDIGS? SETSIG	**new**	—	—
SDL SDR SETCHN SETEUR SETIND SETJPN SETUK SETUSA	●	—	new
SETDAT SETTIM	●	—	(●)
SIGNMT sinc	●	—	new
SL SR	●	—	extended
SLVQ SMODE? SPEC?	●	—	new
s_m s_{mw} s_w	●	●	new
SNAP	**new**	—	—
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_4	**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
$\bar{x}_G$	●	●	new
$\bar{x}_H$ $\bar{x}_{RMS}$ x→DATE	**new**	—	—
x< ? x≤ ? x= ? x≠ ? x≥ ? x> ?	**extended**	—	extended
x= +0? x= -0? x≈ ?	●	—	new

Command	***WP 43S***	*WP 31S*	*WP 34S*
Y.MD	●	●	new
αLENG?	**extended**	—	●
αPOS?	**extended**	—	—
αRL αRR αSL αSR	●	—	extended
β(x,y) Γ_{xy} γ_{xy} ε ε_m ε_p ζ(x) Π Σ σ_w	●	—	new
Σ^1/x Σ^1/x^2 Σ^1/y Σ^1/y^2 Σlny/x $\Sigma x^2/y$ Σx^3 Σx^4 Σx/y	**new**	—	—
$\Sigma \ln^2 x$ $\Sigma \ln^2 y$ Σlnx Σlnxy Σlny Σx Σx^2 $\Sigma x^2 y$ Σxlny Σxy Σy Σylnx 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
→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 (please see its manual).

Reference Literature

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

Topic	**Recommended literature**
General calculation examples and applications	All vintage *HP* calculator manuals can be recom-mended
Root finding and numeric integration	*HP-34C Owner's Handbook and Programming Guide*[74] *HP-15C Owner's Handbook*[75] *HP-15C Advanced Functions Handbook*[76] *HP-42S Programming Examples & Techniques*[77]
Accuracy of numerical calculations	*HP-15C Advanced Functions Handbook,* pp. 172 – 211.[76]
Statistical distributions and their application	*HP-21S Owner's Manual*[78]
Financial calculations	*HP-17BII+ User's Guide*[79]

[74] Read here: https://www.yumpu.com/en/document/read/19323790/hp34c-slide-rule-museum

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

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

[77] Download from http://www.hp41.net/forum/fileshp41net/hp42s-programming-examples.pdf

[78] Download from https://www.manualslib.com/manual/940232/Hp-Hp-21s.html#manual

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

Topic	Recommended literature
Manipulating *short integers*	*HP-16C Owner's Handbook* [80]
Programming	*HP-42S Owner's Manual* [81] *HP-42S Programming Examples & Techniques* [77]

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 88 on p. 226 below as well as the last paragraph on p. 16 of the *OM*).

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

[80] Download from *http://www.hp41.net/forum/fileshp41net/hp16c.pdf*

[81] Download from *http://www.hp41.net/forum/fileshp41net/manuel-hp42s-us.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 https://www.msys2.org. Install it following the on-site instructtions until beginning of step 6. Then enter in the black window:

pacman -S mingw-w64-x86_64-gcc git base-devel mingw-w64-x86_64-gtk3

cd wp43s for changing to the proper directory.

git pull for pulling the changed files from gitlab repository.[82]

make for building a new `wp43s.exe`.[83]

rm backup.bin for starting with the simulator reset to default.

./wp43s for starting the simulator. Continue with c).

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`.

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

---

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

[83] 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** (this can not, however, overcome real errors in the software).

(These pictures of the simulator surface 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. The ten digits as well as [.], [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 overleaf.

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* thereto.

… **Z** copies the full contents of all 112 global *registers* thereto.

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

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

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*

1) Under *Windows*:

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

SETUP

5	System
2	Enter system menu
4	Reset to *DMCP* menu
	Now connect your computer to the calculator *USB* socket using a proper data transfer cable. The flash disk of your *DM42* should show up as an external mass storage volume after you pressed …
6	Activate *USB* Disk

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

Copy `WP43S.pgm` to the *DM42* flash disk. (Option: Copy also `keymap.bin` to the *DM42* flash disk. This will reassign keys to match the *WP 43S* layout also when leaving *WP 43S*.)

Then press from *Free42*: SETUP 5 2 4 3 `WP43S.pgm` ENTER ENTER . Wait for flashing completed. Then press EXIT EXIT 1 EXIT. Your *WP 43S* is up and waiting for your orders..

The two files `Key_stickers.xcf` and `WP43S_overlay.xcf` are *GIMP* images to make your life easier. Print them, cut, and apply (see this picture of an earlier layout) as long as you converted a *DM42 to get your WP 43S.*

To leave the *WP 43S* program, enter g MODE ▲ g SYSTEM to return to the *DMCP* system. If you chose the option above, the key assignments will stay as they were in *WP 43S* when navigating therein (due to `keymap.bin`).

To retrieve the original *DM42* keyboard layout (cf. p. 159), 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/

2) Under *OSX* (*Mac*):

Here is *Harald Overbeek's* step by step solution if you want the *WP 43S* running on *OSX*:

a) Install *XCode* from the *App Store*.

b) Clone the source code of the *WP 43S* project by opening *Xcode* and clone it using https://gitlab.com/Over_score/wp43s.git. Cloning the project has some advantages over downloading the code. It makes sure *XCode* tracks the changes, for example.

c) Install *MacPorts* using https://guide.macports.org/chunked/installing.macports.html

d) Thereafter install the following *MacPorts* one by one:

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

e) Then run the *makefile* form the root directory of the project (probably **wp43s**)

f) When the project has successfully compiled, run the program, for example like this: **./wp43s**

   In the terminal window the following warning may appear:

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

   Solve this by entering:

```
export LC_ALL="en_US"
export LANG="en_US"
export LANGUAGE="en_US"
export C_CTYPE="en_US"
export LC_NUMERIC=
export LC_TIME=en"en_US"
```

... or similar locale settings.

If you get this error:

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

use this:

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

After that I get no more warnings or error messages.

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

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

add:

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

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

```
cp wp43s_pre.css ~
```

Let me know if this does not work.

How to Update Your *WP 43S*

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

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

- Start your computer.
- Take your calculator and turn it on.

  Leave *WP 43S* pressing g MODE ▲ g SYSTEM to return to the *DMCP* system. If you had copied `keymap.bin` of *WP 43S* before last flashing, key assignments will stay as they were when navigating in the *DMCP* system – else the *Free42* assignments will become valid (cf. p. 159).
- Connect your computer to the calculator *USB* socket using a proper data transfer cable. The flash disk of your calculator should show up as an external mass storage volume after you pressed …

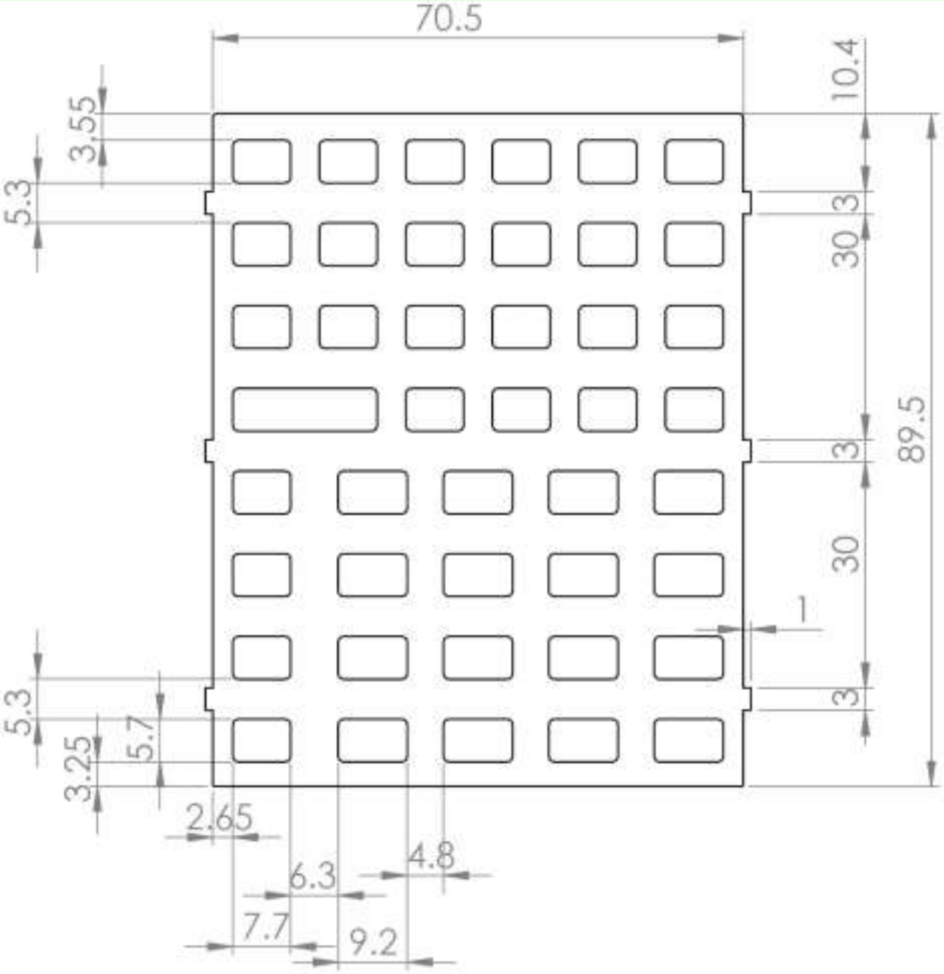
  6 Activate *USB* Disk
- Start the internet browser on your computer and go to https://gitlab.com/Over_score/wp43s/tree/master/DM42%20binary .

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

  Option: Copy `keymap.bin` to the *DM42* flash disk. This will reassign keys to match the *WP 43S* layout also when leaving *WP 43S*.
- Then press EXIT ENTER 3 (Load Program), select `WP43S.pgm`, an press ENTER ENTER.
- Wait for flashing completed. Then press EXIT EXIT 1 EXIT. Your updated *WP 43S* is up and waiting for your orders.

Overlays

See here the drawing for a blank overlay – all dimensions are given in mm. Note the overall width is 72.5 mm.



Find the original print of your keys and keyplate on the last page of this manual, displayed to scale.

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. 162 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.[85]

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

---

[85] 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* and *complex* domain functions.

Ensure you understand the respective fundamental mathematical concepts; else leave these functions aside. By experience, it is only beneficial to use something you overview and know the background of – 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 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*	B_n* works with the old definition instead: $$B_n^* = 2 \cdot \frac{(2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$$ See p. 250 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 59, 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. 217): In a *Galton box*[86] (a.k.a. *bean machine*) featuring y rows of pins and fed with 2^y balls, $C_{y,x}$ is the number of balls expected in column x of that box (start column counting with zero). Generally, $P_{y,x} = \frac{\Gamma(y+1)}{\Gamma(y-x+1)}$ and $C_{y,x} = \frac{\Gamma(y+1)}{\Gamma(x+1) \cdot \Gamma(y-x+1)}$ work also for non-integer numbers and in *complex* domain.
FIB	For integers, FIB returns the *Fibonacci* number f_n with $n = x$. The *Fibonacci* numbers are defined as $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$. With UNSIGN, f_{93} is the maximum before an overflow occurs. 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.

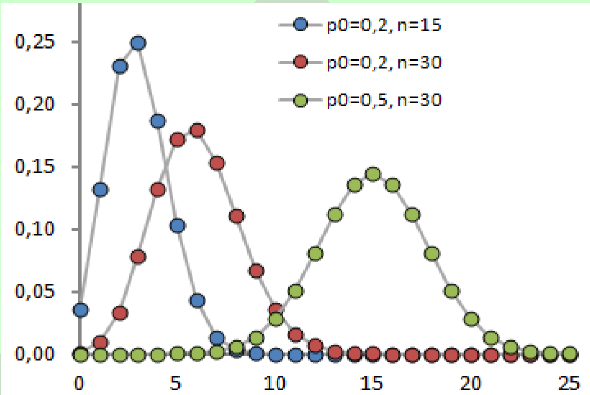
[86] Translator's note: This is called «Planche de Galton» in French, "Galtonbrett" in German, and "macchina di Galton" in Italian. Note the subtle differences in naming. Galton invented his box in 1889.

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 eight continuous ones thereafter. Typical plots are shown for the *PMF*'s or *PDF*'s.

Binom: *Binomial distribution* with the *number of successes* ***g*** in **X**, the *gross probability of a success* $\boldsymbol{p_0}$ 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. 216 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.

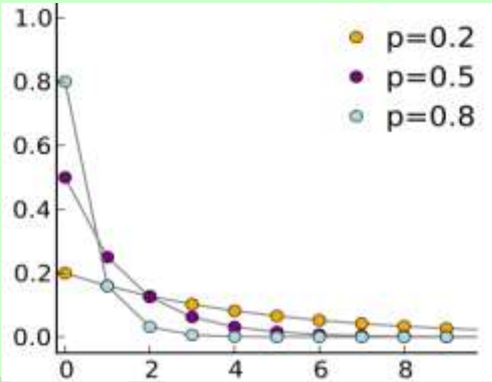
Example: What is the probability for finding no faulty item in a sample of 15 items drawn from a batch of 300 wherein you expect 3% defective items overall? This will tell you:

.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! [87]

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

Geom: *Geometric distribution*:



GEOM$_P$ returns

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

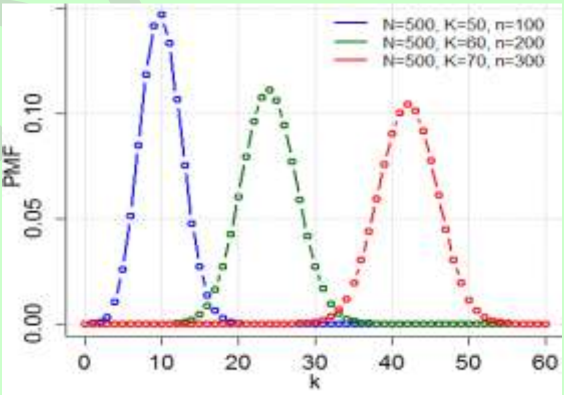
GEOM returns

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

being the probability for a first success after ***m*** = ***x*** *Bernoulli* experiments. The probability ***p₀*** for a success in each such experiment must be specified in **I**.

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

Hyper: *Hypergeometric distribution* with the *number of successes* ***g*** in **X**, *gross probability of a success* ***p₀*** in **I**, *sample size* ***n*** in **J**, and batch size ***n₀*** in **K** (in the diagram, ***g*** = k, ***p₀*** = K / N, and ***n₀*** = N).



[87] 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 $p_H(g;n;p_0;n_0) = \dfrac{\begin{pmatrix} n_0 p_0 \\ g \end{pmatrix} \cdot \begin{pmatrix} n_0(1-p_0) \\ n-g \end{pmatrix}}{\begin{pmatrix} n_0 \\ n \end{pmatrix}}$ (see COMB on p. 216 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 **X**, the *gross probability of a success in a single draw* ***p₀*** in **I**, and ***n*** in **J**.

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

$= C_{n-1,f-1} \cdot p_0^f \cdot (1-p_0)^{n-f}$ (see COMB on p. 216 and cf. BINOM).

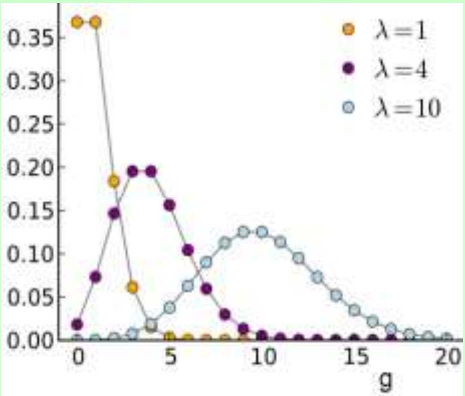
Start reading here for more: *http://en.wikipedia.org/wiki/Negative_binomial_distribution* .

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

POISS$_P$ computes

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

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

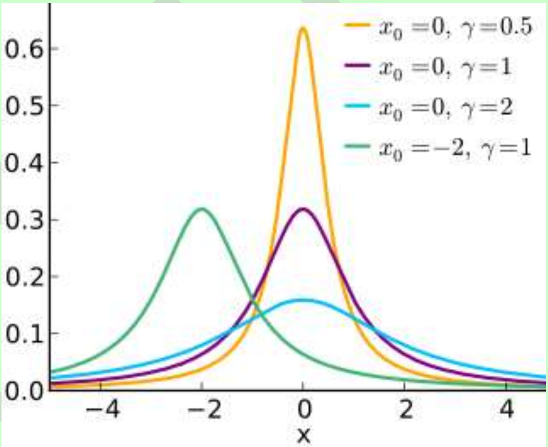


The *Poisson distribution* provides the mathematically simplest model for industrial sampling tests – use $\lambda = np_0$ with the gross error probability $\boldsymbol{p_0}$ and the sample size $\boldsymbol{n}$ (cf. BINOM). For the example introduced with BINOM above, POISS returns 0.638.

Read here for more information: http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm .

Continuous distributions:

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



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

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

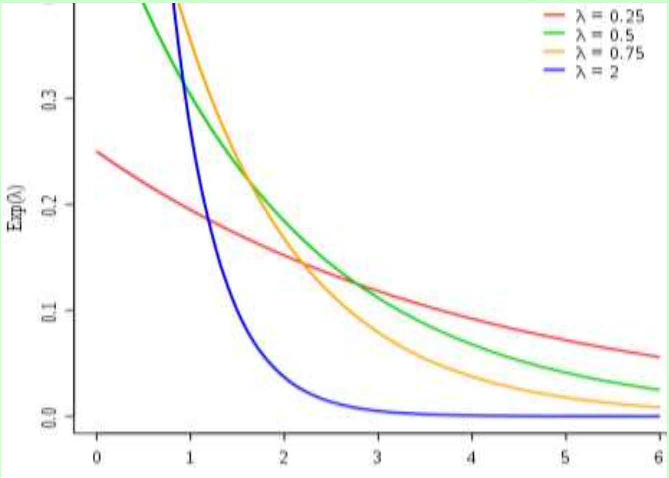
CAUCH^{-1} returns $F_{Ca}^{-1}(p) = x_0 + \gamma \tan\left[\pi \cdot \left(p - \frac{1}{2} \right) \right]$.

This distribution is quite popular in physics. It is a special case of *Student's t distribution*. Start reading here for more: http://en.wikipedia.org/wiki/Cauchy_distribution .

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

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

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

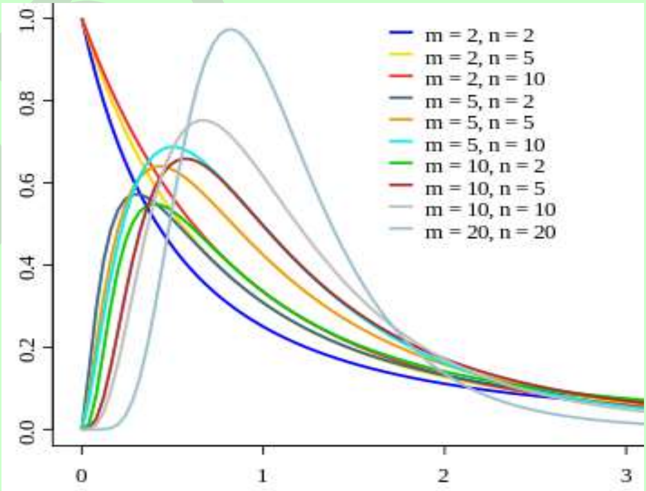


Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3667.htm

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

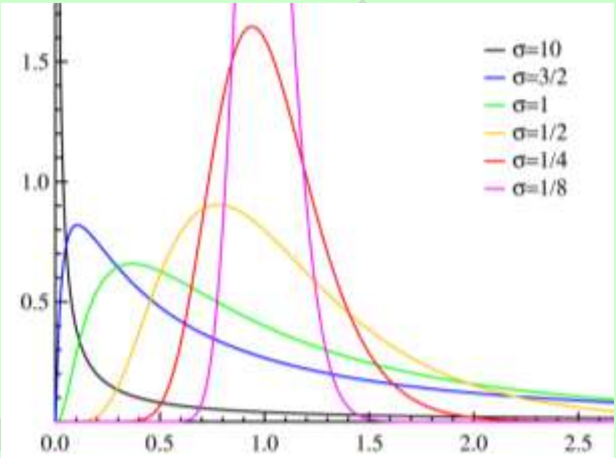


Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3665.htm

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. 225.



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

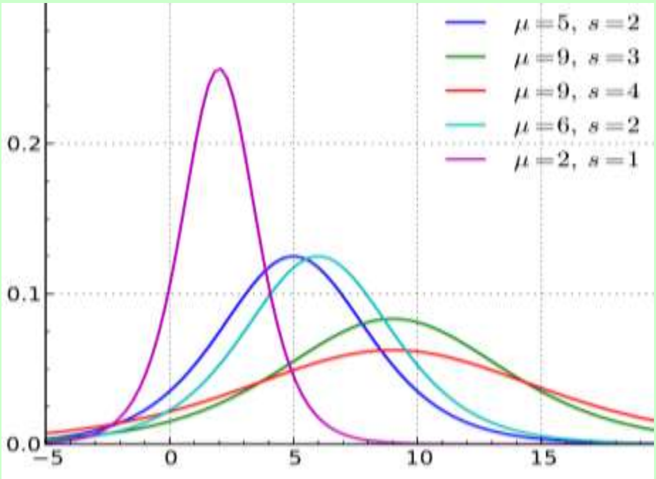
Logis: *Logistic distribution* with an arbitrary *mean* $\boldsymbol{\mu}$ given in $\mathbf{I}$ and a *scale parameter* ***s*** in $\mathbf{J}$.

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

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

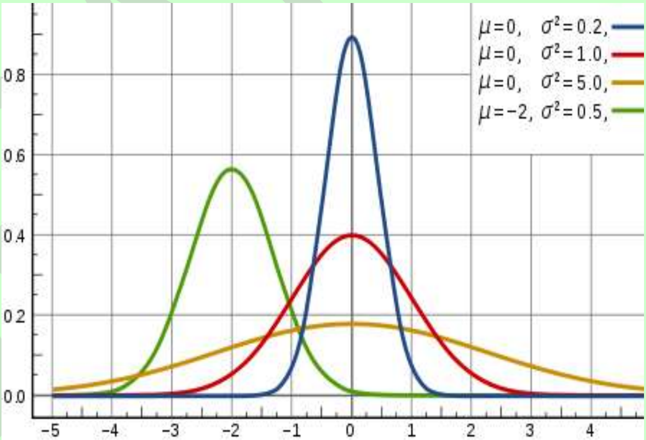
LOGIS returns $F_{Lg}(x) = \frac{1}{1+e^{-\xi}}$.

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* ***μ*** given in **I** and an arbitrary *standard deviation* ***σ*** in **J**. The red curve (for ***μ*** = 0 and ***σ*** = 1) is the *standardized normal* (a.k.a. *Gaussian*) *distribution* φ.



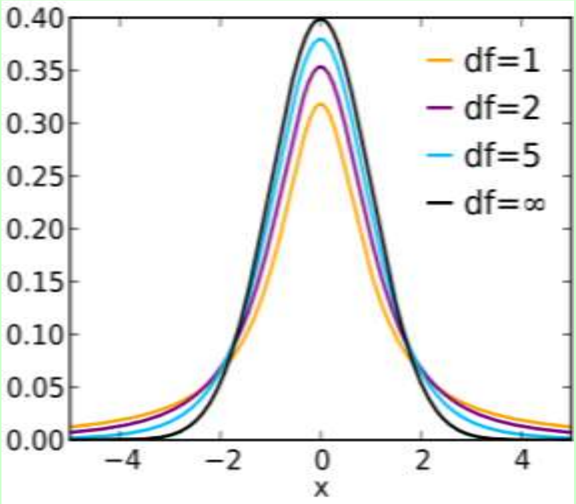
NORML$_P$ returns $f_N(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\,\sigma^2}} = \varphi\left(\frac{x-\mu}{\sigma}\right)$ and

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. 225.

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*** $\rightarrow \infty$, the shoulders of *t(x)* shrink and it approaches the *PDF* of the *standard normal distribution* (compare the red *Gaussian* curve at NORML on p. 223).



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**.

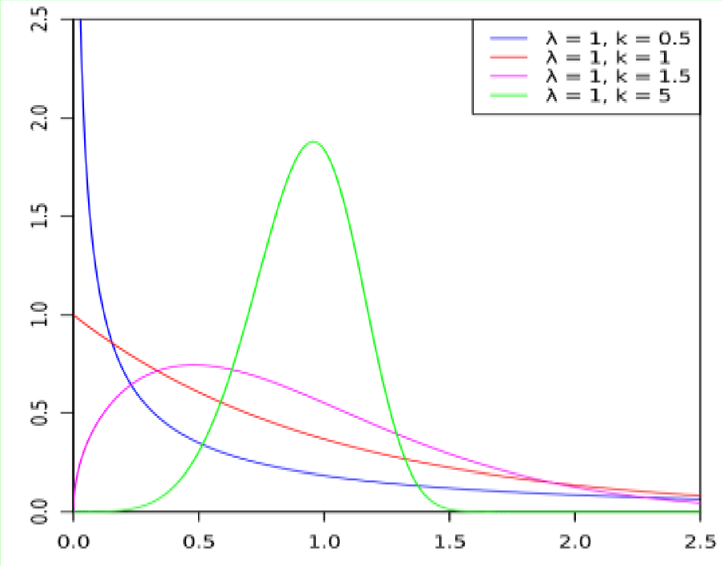
WEIBL_P returns $f_W(t) = \frac{b}{T} \cdot \left(\frac{t}{T}\right)^{b-1} e^{-\left(t/T\right)^{b}}$ for $t \geq 0$, else 0. This is a very flexible function – see the curves plotted overleaf.

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.

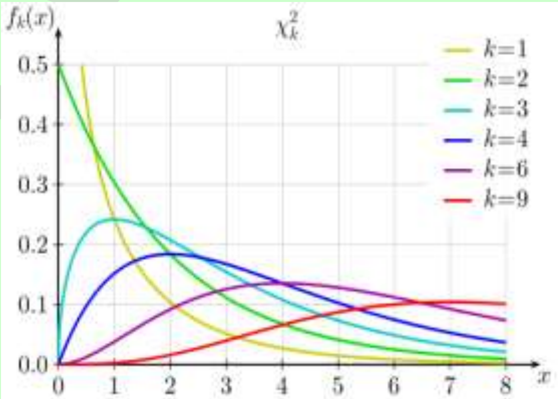
Read here for more information:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm .

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. 223), while $\Phi(x) = \int_{-\infty}^{x} \varphi(\tau) d\tau$ is the corresponding *CDF* (cf. the *error function* on p. 247). 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![88] 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.

---

[88] 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 often found – then you cannot expect the calculated consequences matching reality any better.

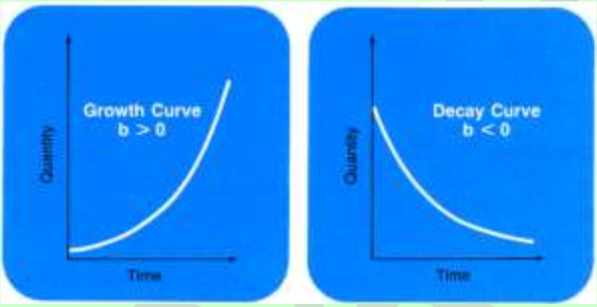
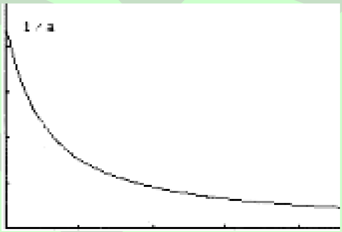
As mentioned in the main text, we recommend you look deeply into statistics textbooks to ensure you fully understand what you do with the functions provided in your *WP 43S*. The real world shows lots of sad examples where people full of good will caused large damages by applying tools they did not know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

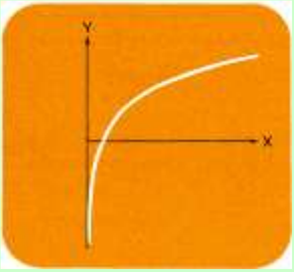
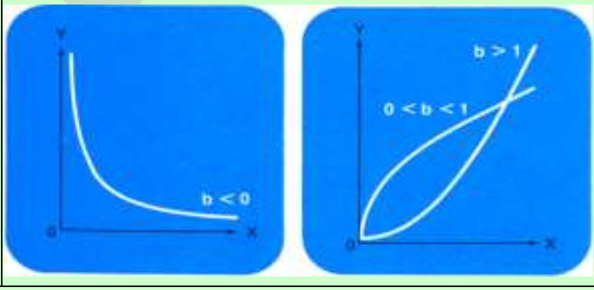
The following functions as named in the left column (sorted alphabetically) are all found in STAT:

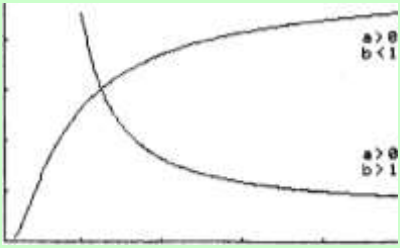
Name	Remarks (see pp. 12ff for general information)
CauchF	Selects a Cauchy (a.k.a. Lorentz, Breit-Wigner) peak fit model $R(x) = {}^{1}/_{[a_0\ (x+a_1)^2 + a_2]}$ for least squares regression.[89] See p. 220 for shapes of such peaks.
CORR	For any set of data points (x_i, y_i), the *coefficient of correlation* 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. 224).

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

Name	Remarks (see pp. 12ff for general information)
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.
ExpF	Selects the exponential curve fit model $R(x) = a_0 e^{a_1 x}$ for least squares regression. [89] 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).[90]
GaussF	Selects a Gauss peak fit model $R(x) = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$ for least squares regression. [89] See p. 223 for the shapes of such peaks.
HypF	 Selects the hyperbolic fit model $R(x) = {}^1/_{(a_0 + a_1 x)}$ for least squares regression. [89]
LinF	Selects the linear fit model $R(x) = a_0 + a_1 x$ for least squares regression. [89] Generally, this will be a good choice if the measured data follow a straight line, raising or falling (but compare ORTHOF).

[90] 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)
LogF	 Selects the logarithmic curve fit model $R(x) = a_0 + a_1 ln(x)$ for least squares regression. [89] Generally, this will be a good choice if the measured data follow a curve looking like drawn at left.
L.R.	Uses the fit model selected and computes the two or three parameters of the regression for the data accumulated. For all curve fit models provided on your *WP 43S*, a regression parameter is (statistically) *significant* 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. 224).
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. 232ff 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. [89]
PowerF	Selects the power curve fit model $R(x) = a_0\, x^{a_1}$ for least squares regression. [89] 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*).[90]

Name	Remarks (see pp. 12ff for general information)
RootF	 Selects the root curve fit model $R(x) = a\, b^{1/x} = a_0 a_1^{1/x}$ for a least squares regression. [89]
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 <u>weighted</u> data (where the weight $\boldsymbol{y_i}$ of each data point $\boldsymbol{x_i}$ was entered via Σ+) is $$s_w = \sqrt{\frac{\sum y_i \sum y_i x_i^2 - \left(\sum y_i x_i\right)^2}{\sum y_i \left(\sum y_i - 1\right)}}$$ And the corresponding *standard error* (the *SD* of the *mean* $\bar{x}_w$) is $$s_{mw} = \frac{1}{\sum y_i} \sqrt{\frac{\sum y_i \sum y_i x_i^2 - \left(\sum y_i x_i\right)^2}{\sum y_i - 1}}$$

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

Name	Remarks (see pp. 12ff for general information)
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 weighted data (where the weight $\boldsymbol{y_i}$ of each data point $\boldsymbol{x_i}$ was entered via **Σ+**) is $$\sigma_w = \sqrt{\frac{\sum y_i\,(x_i - \bar{x}_w)^2}{\sum y_i}}$$

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. 228ff):
   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. 233 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. 233 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. 233 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 $\mathrm{R(x)} = \frac{1}{\mathrm{a_0 + a_1 x}}$ 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

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

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

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

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,

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

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

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. from LINF).

About Error Propagation

Experimental data are always attended with errors (cf. footnote 43), caused by e.g. the uncertainty of the measuring method, the instrument used, and/or environmental variations. Even under controlled environmental and measuring conditions, random errors remain. These errors must be taken into account for a proper estimation of the uncertainty of your results computed using those experimental data. For about 200 years, *Gauß' least squares method* can be employed for this task.

Assume you know that your result ***R*** depends on several experimental parameters $\boldsymbol{x_1}$ 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 its error

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

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

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

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

And with your *WP 43S*, the following algorithm will do for computing $\boldsymbol{\Delta R}$, even if $\boldsymbol{f}$ 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 of ***f***, the following simpler algorithm will do, requiring down to half as many steps:

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

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

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

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

Let us drop the restriction for *b* now. Replacing *a* in the previous set of equations by the right side of the differential equation on p. 239, we will get the following new set:

$$\frac{df}{dt_{1/2}} \approx \frac{df}{dt_0} + \left[a - b\left(\frac{df}{dt}\right)_0^2\right]\frac{\Delta t}{2}$$

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

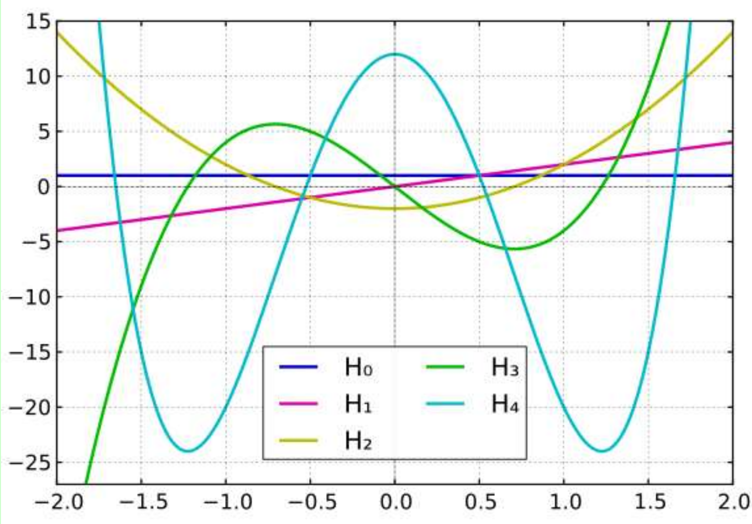
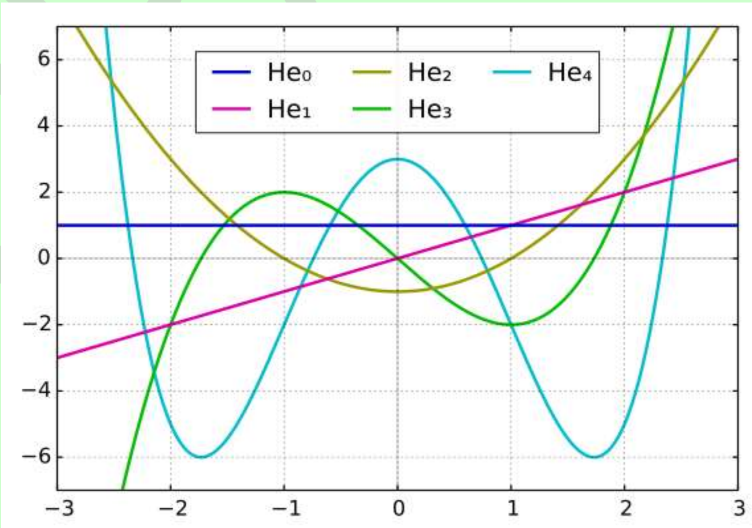
$$\left(\frac{dx}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dx}{dt}\right)_{i-\frac{1}{2}} + K_x\Delta t \qquad \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dy}{dt}\right)_{i-\frac{1}{2}} + K_y\Delta t$$

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

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 **Y**, solving the differential equation $f''(x) - 2x \cdot f'(x) + 2n \cdot f(x) = 0$. See the first five polynomials plotted overleaf.

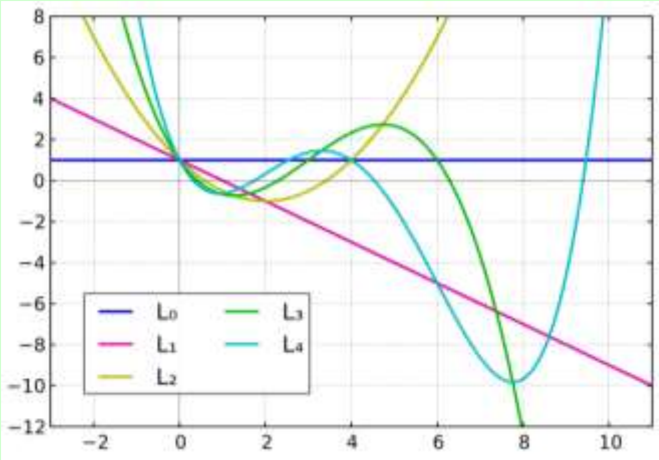
Name	Remarks (see pp. 12ff for general information)
	
H_{np}	***Hermite** polynomials* for <u>physics</u>: $H_{np}(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in $\mathbf{Y}$, solving the same differential equation. See the first five polynomials plotted below. 

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

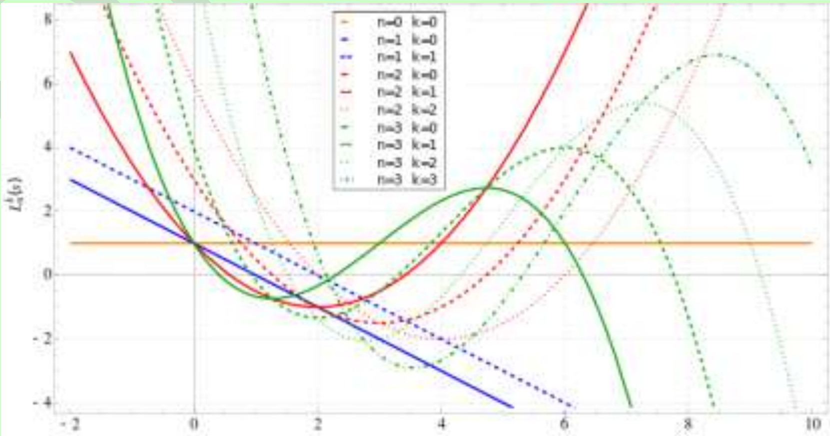
See the first five *Laguerre polynomials* plotted here.



</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 ($k = \alpha$).



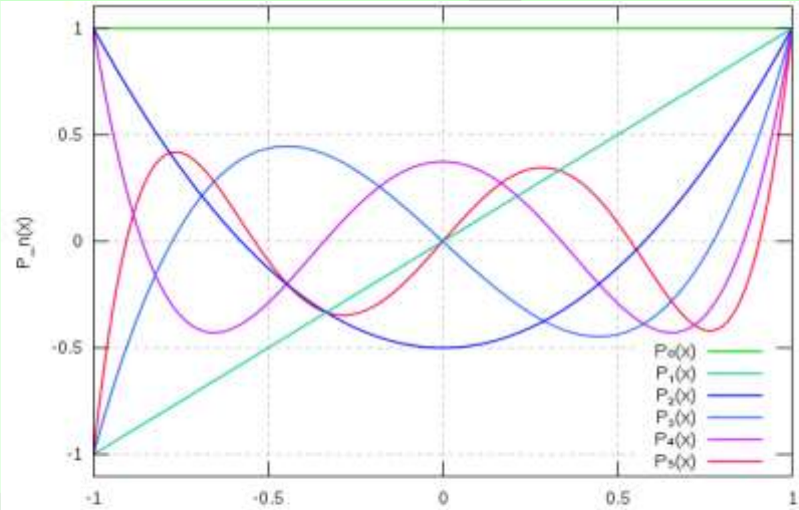
</td></tr>
</table>

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

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

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

See the first six polynomials plotted here:



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

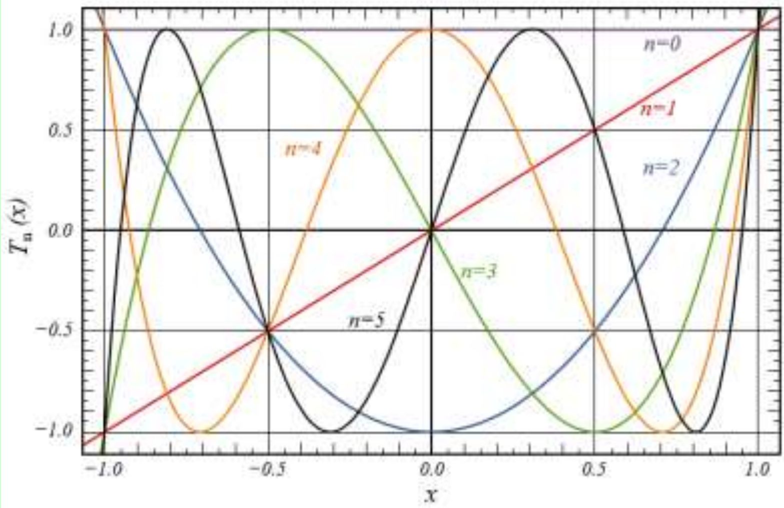
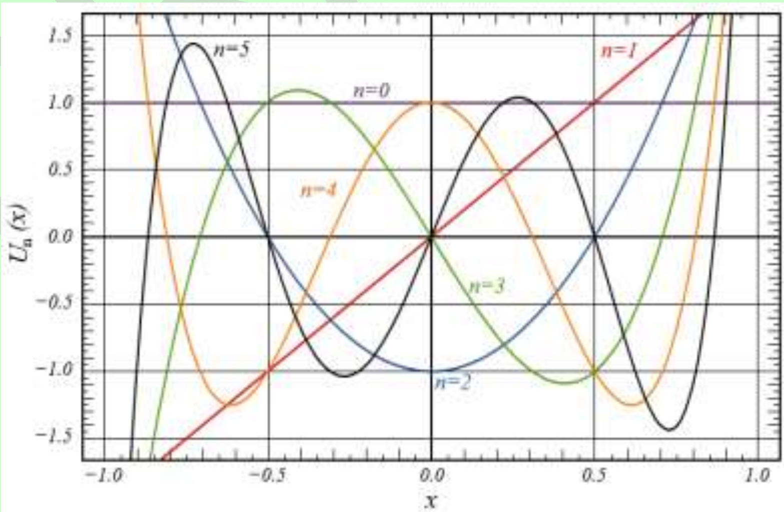
Chebyshev (a.k.a. *Čebyšev, Tschebyschow, Tschebyscheff*) *polynomials of **first** kind*

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

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

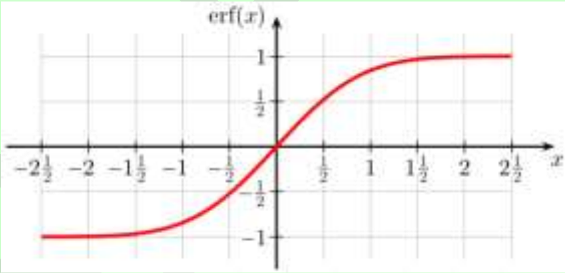
The plot overleaf shows $T_0(x)$ ... $T_5(x)$.

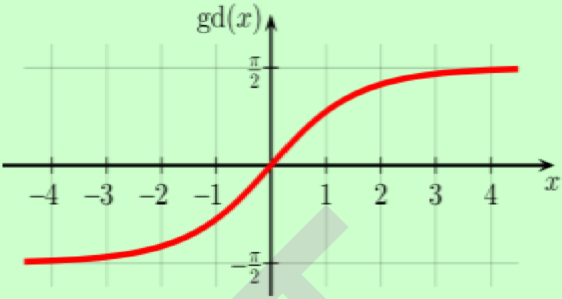
</td></tr>
</table>

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

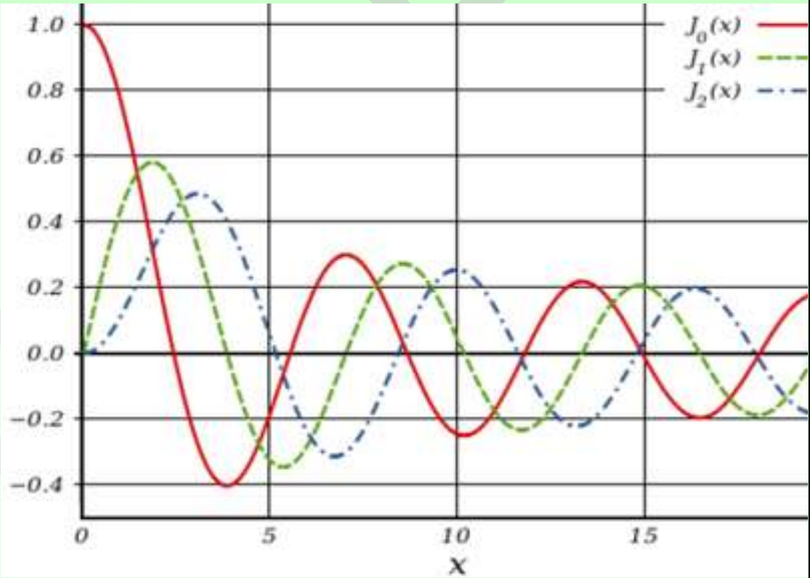
Even More Mathematical Functions

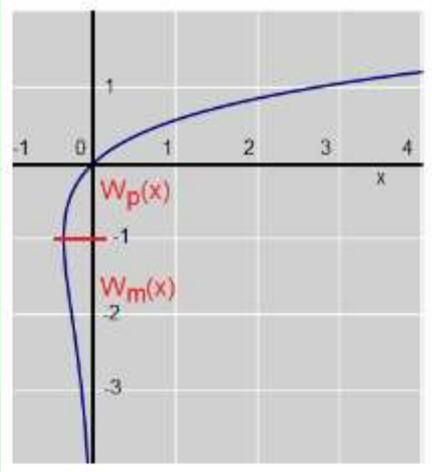
All the following functions are found in X.FN. Some of them are for pure mathematics only but were useful at some stages of the *WP 34S* or *WP 43S* projects, so we made them accessible for the public.

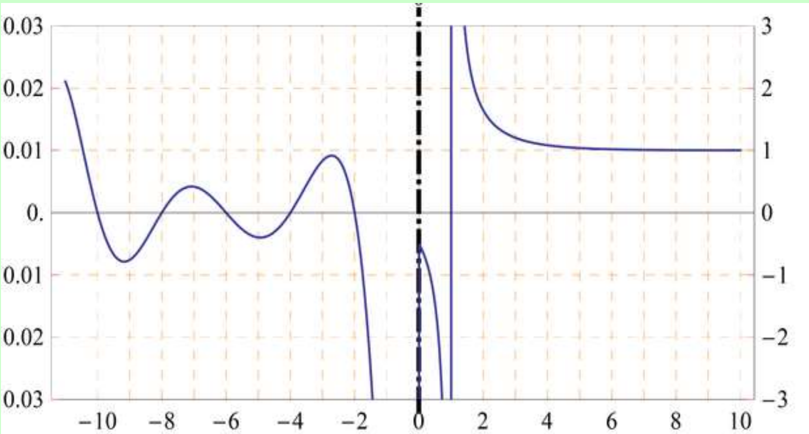
Name	Remarks (see pp. 12ff for general information)
AGM	Returns the *arithmetic-geometric mean*. Find more about it here: http://mathworld.wolfram.com/Arithmetic-GeometricMean.html .
erf	Returns the *error function* $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau$.  Note that $erf\left(\frac{x}{\sqrt{2}}\right) = 2 \cdot \Phi(x) - 1$ with $\Phi(x)$ representing the *standard normal CDF* as described on p. 225. 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 , g_d^{-1}	Returns the *Gudermannian function* $g_d(x) = \int_0^x \frac{d\xi}{\cosh \xi}$ linking hyperbolic and trigonometric functions. See the plot for its *real* values. The *inverse 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. 91 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. 91 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. 217ff) is a competent source. And *Mathworld* (quoted on pp. 247ff) 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 $\boldsymbol{x} = 4$ when you call FACT. Then it will allocate one local register (**R.00**) and store **4** therein. After decrementing $\boldsymbol{x}$, FACT will call itself.

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

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

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

Building *WP 43S* Almost from Scratch

How to build the simulator on Windows:

1. Navigate to https://www.msys2.org/. Download and run `msys2-x86_64-yyyymmdd.exe`
2. Click **Next**
3. Enter the installation folder **C:\msys2_64** and click **Next**
4. Tick **Run MSYS2 now** and click **Finish**

The following **commands printed blue** must be executed in the black *MSYS2* window:

5. **pacman -Syu** . Confirm each question with ENTER and wait until finished.
6. Close the black window by Alt-F4.
7. Reopen *MSYS2 MinGW 64-bits* (every time you need to open *MSYS2*, open the 64-bits version).
8. **pacman -Syu** . Confirm each question with ENTER.
9. **pacman -S mingw-w64-x86_64-gcc git base-devel mingw-w64-x86_64-gtk3** . Confirm each question with ENTER.
10. **cd ~** to navigate to your home directory.
11. **git clone https://gitlab.com/Over_score/wp43s.git** to get a local copy of the gitlab source repository on your PC.
12. **cd ~/wp43s** to navigate to the *WP 43S* program sources.
13. **git pull** to get the latest version from gitlab.com.
14. **make mrproper** to clean the build environment (this is not required but recommended).
15. **make** to build the *WP 43S* simulator.
16. **./wp43s** to run the simulator.
17. Return to step 13. (or to *App. F*) to get a new version of the sources.

How to build the WP43S.pgm for the *DM42* hardware:

1. Navigate to http://gnutoolchains.com/arm-eabi/, download and run **arm-eabi-gcc9.2.1.exe** or a more recent version.
2. Installation directory **C:\msys2_64\arm-eabi** shall be the same base directory as in step 3 on previous page.
3. Select **Current user**
4. Tick **Hardlink duplicate files**
5. Untick **Add binary directory to %PATH%**
6. Tick **I accept the terms of the license agreement**
7. Click **Install**
8. Click **OK** on the Installation succeeded dialog.
9. Start *MSYS2 64 bits* if not yet started.
10. **cd ~/wp43s/DMCP_build** to navigate to the *DMCP* build directory.
11. **./build_GMP_static_ARM_library** this is a long process: about 12 *minutes* on my PC.
12. **cd ~/wp43s/DMCP_build** to navigate to the *DMCP* build directory if not yet there.
13. **git pull** to get the latest version from **gitlab.com**
14. **./build_WP43S.pgm_for_DM42_hardware** to build `WP43S.pgm` for the *DM42*. Maybe the first time the process ends with an error – then run the same command a second time.
15. Copy the file `~/wp43s/DMCP_build/build/WP43S.pgm` via *USB* to your *DM42* and flash it (cf. *App. F*).
16. Loop to step 12.

Index of Everything Provided

This index lists each and every item provided, be it a command, constant (C), menu or submenu (M unless trailed by a colon), predefined label (L) or variable (V), reserved character (c), or system flag (S). Note that the sorting order in your *WP 43S* deviates for good reasons from the order *MS Word* applied here:

The same items sorted as they are in your *WP 43S*:

APPENDIX J: RELEASE NOTES

	Date	Release notes
0	29.11.12	**Official project start with first publication of the 43S concept and a layout on one of the forums of the *Museum of HP Calculators*** (https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685)**.** Though there are found far older traces of a '*43S*' denoting a '*Super HP-42S*', though in various more or less fictional cases – pure vapourware™.
0.1	2.2.14 23.5.15	Manual setup based on the one of *WP 34S*. Passed to *Jake Schwartz*, *Eric Smith*, and *Richard Ottosen* for first information.
0.2	3.10.15	Update based on Jake's feedback and further thoughts, distributed to *Eric*, *Jake*, *Marcus*, and *Pauli*.
0.3	21.3.16	Split the manual in three; moved LBL onto the keyboard, renamed STOM to STOCFG, RCLM to RCLCFG, SERR to s_m, and $SERR_w$ to s_{mw}; refined the *Key Response Table*. Passed to *Michael Steinmann* for information.
0.4	28.3.16	Renamed LOGS to EXP and EEX to E. Added hardware information from 2nd manufacturer.
0.5	29.10.16	Returned EEX. Changed keyboard layout.
0.6	22.8.17	Merged the Applications and Owner's Manual. Changed the input order of complex number parts on Pauli's request. Changed keyboard layout introducing D.MS, SST, BST, and % while removing ŷ, RAN#, 'FRC, and 'CFIT. Put 'CFIT into 'STAT and 'FRC into 'MODE. Placed OFF below EXIT for easier customizing. Renamed cc to C5, EEX to E, STOPW to TIMER, SHOW to REGS, 'SOLVE to 'ADV, DLINES to DSTACK, 12h to CLK12, and 24h to CLK24. Replaced IND by →. Deleted %MG since covered by Δ%, added EIGVAL and EIGVEC. Swapped CNST and CONST. Defined the echo rows for alphanumeric and command input. Expanded and modified the character sets for better use of display space. Added the *QRG*.
0.7		Changed keyboard layout. Replaced the labels BST by ≡▲, SST by ≡▼, and UNDO by ↶; added some *alpha input mode* reminders on the keyboard. Added AGRAPH, CLLCD, EQ.xxx, 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

	Date	Release notes
	2.4.18	*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	Changed keyboard layout: introduced TRG containing trigonometric functions, removed HYP into EXP and [π] to g-shifted [/], swapped some shifted labels. Refined the chapters about register arithmetic, *Command Parameter Input*, *Alphanumeric Input*, *Matrix Calculations*, and *Orthogonal Polynomials*. Introduced CLCVAR and more vintage examples. Rearranged *temporary information* on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE.
	20.9.18	Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9	3.1.19	Removed *angle data type*. Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded *App. B*. Added SHOW for displaying full precision of *DP* numbers and FBR for browsing our two fonts. Split a chapter. Expanded some titles. Added the overlay drawing. Modified functionalities of [EXIT] and [1/x] to match *HP-42S*. Added a chapter about curve fitting. Modified functionalities of [ENTER↑] and [↰]. Expanded *App. K*. Renamed DOUBLE to →DP. Added →SP and conversions of *quarts*. Rearranged X.FN. Replaced [USR] by [UM]. Changed keyboard moving [UM], [√x], and [TRI]. Moved 🖶 to [f] [R/S]. Added XIN and XOUT. Added a chapter in *App. E* and information about infinite integers Extended the domain of GCD and LCM. Refined and corrected.
0.10	3.3.19	Returned *angle data type* and αSR. Added IDIVR and VANGLE. Refined FP, IP, IMPFRC, PROFRC, SDIGS?, →DP, →HR, →INT, →REAL, →SP, explanation of ALL, the summary of integer functions, and handling of long alpha strings. Modified contents of CPX, MATX, and α•. Added a summary of matrix functions. Removed the [ON]-key combinations. Modified MEM?. Rewrote the angular conversions. Renamed infinite and finite integers to *long* and *short integers*. Added a chapter about ±∞ and NaN. Modified RBR and the menu

	Date	Release notes
		for STO and RCL. Removed 🖶 from the keyboard. Renamed X_u to X_e for the distributions. .
0.11	8.5.19	Changed keyboard making CC primary and user mode shifted, removing x^2, x≹, and DSP, adding |x|, DROP, and SHOW, and moving some shifted labels. Modified BITS, CLREGS, CNST, CPX, DISP, EXP, INTS, MODE, PARTS, SHOW, STAT, U→, αMATH, the division matrix, *data type* conversions, and the *Quick Reference Guide*. Added conversions of *barrels*, *carats*, and *fathoms*. Deleted DSP. – Separated predefined variables. Refined *Sect. 6*. Added $\bar{x}_H$, $\bar{x}_{RMS}$, nine statistical sums and five curve fit models. Split STAT in STAT and SUMS; renamed RMDR to RMD, L_n to L_m, $L_{n\alpha}$ to $L_{m\alpha}$, Π to Π_n, Σ to Σ_n, and some constants to avoid search ambiguities. Refined *App. J*, *Sect. 3 and 4*, →INT, CLR, and the functions of ▲ and ▼. Put SUMS instead of RMD on the keyboard, moved ADV, BITS, CATALOG, EQN, FILL, INTS, MATX, MODE, PROB, RTN, SHOW, STAT, and α.FN. Rearranged A…Ω and *Sect. 2* of the *OM*.
0.12	16.10.19	Rearranged the appendices of the *ReM* from *App. D* on. Expanded *App. A* of the *OM* and *App. K*. Deleted the standardized normal distribution Φ and rearranged PROB. Updated CNST following *CODATA 2018*. Renamed the angular conversions. Changed the composing and cutting functionality of CC. Refined exiting *short integer* input. Expanded *App. D*. Specified maximum size of *long integers*. Changed keyboard adding ∡, moving CPX, FIN, RBR, R↑, and SHOW, removing %. Renamed VANGLE to V∡. Modified CPX, MATX, TRI, and X.FN. Rearranged *Section 1* of the *OM*. Added some internal *data types* to *App. B*; reduced the range of *long integer* results and *DP* real inputs to $10^{\pm 999}$. Defined the domains of e^x-1, IDIVR, LN(1+x), MOD, and RMD according to the *HP-42S*; modified PLOT and Σ+. Refined the *Addressing Tables*. Added a *data type* matrix for IDIVR. Refined the *Special Results* in *App. B*.
0.13	30.11.19	Expanded the alpha keyboard and *App. I*. Modified CPX, INTS, MODE, PROB, STK, TEST, α●, SHOW, and STATUS. Refined the sorting order of *items*, ALL, CX→RE, MEM?, RE→CX, RBR, RM, SLVQ, and U→. Started filling *App. F* and *G*. Refined *App. 2*. Added a *long integer* example, CPXR?, LZ?, $\Delta\nu_{Cs}$, conversions of *hectares*, and a proposal for system status information.
0.14	7.3.20	Introduced *system flags* for status information. Split I/O. Added CATALOG'SYS.FL, PRINT, PROG, RANI#, VAR, auxiliary constants, some predefined variables, and an index in *App. I*. Changed keyboard swapping MODE and FLAGS, U→ and ∡→, moving CPX, FILL, RBR, R↑, USER, α.FN, αINTL, √x, and 🖶, displaying PRINT, RMD, STATUS, x^2, and ':', and removing c/d, 🖶x,

	Date	Release notes
		→SP, and →DP. Renamed DISP to DSP and SUMS to Σ, changed ↰ to ↶. Refined the addressing tables and catalog access, a b/c, ADV, BATT?, BITS, CATALOG'CHARS and 'MENUS, CLALL, CLFALL, CPX, EXP, GAP, INTS, I/O, MODE, NEIGHB, PARTS, PRIME?, P.FN, SHOW, STAT, STK, X.FN, αINTL, and α●. Deleted all 16-digit (i.e. *SP*) *data types* as well as A…Z and the commands CLK12, CLK24, CPXi, CPXj, CPXRES, CPXR?, DBL?, DENANY, DENFAC, DENFIX, ENGOVR, FAST, IMPFRC, LZOFF, LZON, LZ?, MULT×, MULT·, POLAR, PROFRC, QUIET, RDX., RDX,, REALRE, RECT, SCIOVR, SLOW, SSIZE4, SSIZE8, →DP, and →SP. Corrected.
0.15	31.3.20	Changed keyboard shifting N, O, P, and Q, swapping ? and Z, moving CNST, CPX, FLAGS, RBR, RTN, R↑, and 🖶, removing :, and adding MOD, ✓, and SNAP. Renamed DSP to DISP, CNST to CONST, CONST to CNST, and SSIZE to SSIZE8. Added RANGE, RANGE?, and SNAP as well as errors 28 and 31. Refined CNST, NEXTP, PRIME?, and RESET. 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*. and some 35 named *system flags*.

Programs consist of ≥4 program steps: LBL with a global label, at least one action step, RTN, and END. Each program may contain subroutines (up to 8 levels deep). See p. 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], [CC], or [#] is taken as a *long integer* of base 10.

Real numbers: Any number you enter using [.] and/or [E] is a real number.

Complex numbers: A complex number consists of two real numbers combined to represent its real and imaginary part like 1.23-i×4.56 in rectangular mode (SF RECTN and press **1.23** CC **4.56** +/- ENTER↑) or its magnitude and phase like -7.89 ∡ 120° in polar mode (CF RECTN 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 decimals of *seconds* as needed.

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

Matrices: see pp. vii f.

Short integers: Any purely 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

SYSTEM					
		RM		SETSIG	DENMAX
SF	DEG	RAD	GRAD	MULπ	CF

SF, CF Set (or clear) a flag.

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*.

RM Sets rounding mode.

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

DENMAX Sets the maximum denominator for calculating with fractions.

SYSTEM Returns the calculator to the *DMCP* system for updating.

DISPLAY FORMATS

DISP

GAP					DSTACK
CHINA	EUROPE	INDIA	JAPAN	UK	USA
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.

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

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

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

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 PROG 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 <u>or</u> 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. xf) 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.

To edit the matrix in X, use MATX EDIT .

To edit a named matrix, use MATX EDITN ***name***.

When a matrix is being edited it is said to be *indexed* (to index a named matrix without editing it, use INDEX). Whenever there is an indexed matrix, two pointers are used to indicate the row and column of the current element: they are stored in **I** and **J**, respectively. If **I** and **J** are pointing to the last element (bottom right) in a matrix and you press → then ...

- …the pointers wrap around to the first element of the matrix (**Wrap mode**, automatically set whenever you enter or exit the *Matrix Editor*) or ...
- …the matrix grows by one complete row and the pointers move to the 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

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

Binom:	d	*Binomial d.*	($i = p_0$ = gross probability of a success, $j = n$ = 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 1 (dof_1), $j = dof_2$)
Geom:	d	*Geometric d.*	($i = p_0$)
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.*	($i = n\ p_0$ = *Poisson* parameter)
t:	c	*Student's t d.*	(i = *dof*)
Weibl:	c	*Weibull d.*	(i = shape, j = characteristic lifetime)
χ^2:	c	*Chi-square d.*	(i = *dof*)

Following naming convention holds for most distributions, e.g. for the *normal d.*: $\textbf{Norml}_\textbf{p}$ denotes the *probability density function*, **Norml** the *cumulated d. function*, $\textbf{Norml}_\textbf{e}$ the *error probability*, and $\textbf{Norml}^{-1}$ 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
		$\bar{x}_{RMS}$				
		$\bar{x}_H$				
	L.R.	r	s_{xy}	cov	$\hat{x}$	$\hat{y}$
STAT	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	PLOT
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	
	Σ+	$\bar{x}$	s	σ	s_m	SUM

$\bar{x}$, s, σ, s_m	Arithmetic mean value, sample standard deviation (*SD*), population *SD*, standard error (a.k.a. *SD* of the mean).
$\bar{x}_w$, s_w, σ_w, s_{mw}	Same for weighted data.
$\bar{x}_G$, ε, ε_p, ε_m	Geometric mean value, sample scattering factor (*SF*), population *SF*, *SF* of the mean.
$\bar{x}_H$, $\bar{x}_{RMS}$	Harmonic and quadratic mean values.
SUM	Recalls Σx and Σy.
PLOT	See the *ReM*.

L.R. Computes the parameters a_0 and a_1 (and a_2, if applicable) of the fit model selected (see below).

r	Returns the correlation coefficient.	
s_{xy}, cov	Return the sample or population covariance.	
$\hat{x}$, $\hat{y}$	Return the forecast for x or y according to the fit model selected.	
LinF	Linear fit model:	$y = a_0 + a_1 x$.
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.	

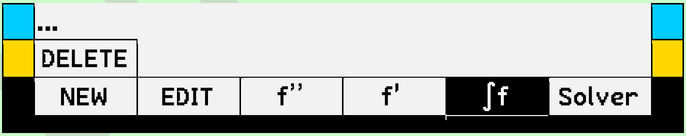
ADVANCED OPERATIONS

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

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

Interactive Operations on Equations

EQN



For creating a new equation, press NEW. The *Equation Editor menu* will open, and the blue row will display the current equation. Press EXIT when finished.

For browsing existing equations, press ▲ or ▼. The equation displayed in g-shifted row is called the *current equation*.

For editing (or deleting) the current equation, press EDIT (or DELETE).

For operating on the current equation, press the respective *softkey*. A *menu* will pop up displaying the *names* of all variables used and more.

Using Advanced Operations in Programs

ADV

PGMSLV		f"(x)			
SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx

SLVQ solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input *stack* [***c, b, a***, …]. It returns two real or complex solutions.

Π_n *label* calculates the product of the terms given by the routine specified, using the loop control number given in **X** (cf. p. vi).

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

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

- **AB** must begin with a global label.
- The body of **AB** shall evaluate the expression. For an expression to be solved, it must be coded that *$f(x_1, x_2, ...) = 0$* is fulfilled. Recall the variables of the expression as they are required and calculate *f*.
- **AB** must logically end with RTN.

Then write a program calling the *Solver* (let's call it **CD**, for example). At the position where you need the expression solved, press ADV :

1. Press PGMSLV and specify **AB**.
2. Store a value into each known variable, e.g. using STO. Optionally store a guess into the unknown variable to direct the *Solver* to a solution.
3. Press SOLVE and specify the unknown variable.

When running **CD** later on, SOLVE will solve for the unknown.

f'(x) (or **f"(x)**) calculates the first (or second) derivative of *f(x)* at location ***x***. The function *f(x)* shall be written as a program (e.g. called **EF**); it must begin with a global label, take care of all variables used, and evaluate *f(x).*

Then write a program calling the derivator (let's call it **GH**, for example).

1. Store a value into each of the variables that shall remain constant under derivation.
2. At the position where you need the derivative, put the respective location into **X**, then press ADV f'(x) (or f"(x)) specifying **EF**.

When running **GH** later on, the derivative will be returned in **X**.

∫fd *var* numerically computes a definite integral. The integrand *f(x)* shall be written as a program (e.g. called **IJ**); it must begin with a global label, recall all integration constants used, and evaluate *f(x).*

Then write a program calling the integrator (let's call it **KL**, for example). At the position where you need the integral, press ADV :

1. Press ∫fdx . A *submenu* will open.
2. Press PGMINT and specify **IJ**.
3. Store a value into each of the variables that shall remain constant under integration, e.g. using STO.
4. Store the lower limit (↓LIM), the upper limit (↑LIM), and the accuracy factor (ACC).
5. Press ∫ and specify the variable of integration.

When running **KL** later on, the integral will be returned in **X** and the uncertainty of computation will be returned in **Y**.

OPERATIONS ON SHORT INTEGERS

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

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

NOT Inverts every bit in **X**.

MASKL, MASKR 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).

INTS

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

A ... F — Digits for short integers of 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 [+]. 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 substring ***x*** begins in the string in ***s***.
FBR	Displays all characters defined in both fonts.

BACKGROUND CONSIDERATIONS AND FACTS

This section is for recording and explaining some of the boundary conditions considered and settings chosen for the *WP 43S* in the course of this project. It is not necessary for operating the *WP 43S* but may foster understanding.

Accessing Items

The hardware offers 43 keys for some 750 *items*. Subtract six for the *softkeys*. Space for primary functions is quickly occupied. These functions are mostly set. For the remaining set, secondary functions compete with *menus*.

Obvious primary functions are 0 … 9, ., ENTER↑, x≷y, +/−, E, ←, +, −, ×, /, STO, RCL, XEQ, R/S, ▲ and ▼, EXIT, f and g, taking 29 key tops. So eight locations are left. Once you want to deal with complex numbers seriously, you need a primary CC.

Other functions may be debated: R↓, 1/x, y^x, x^2, $\sqrt{x}$, e^x, ln, 10^x, lg, sin, cos, and tan are most popular – twelve candidates for seven key tops left. Either x^2 or $\sqrt{x}$ shall be primary (we chose x^2); and we can ditch 10^x and lg when we have e^x and ln. So R↓, 1/x, y^x, sin, cos, and tan compete for the remaining four key tops. We chose the first three and put the latter three (and their inverses) under one primary *menu* key: TRI; thus, you can access each of sin, cos, tan, arcsin, arccos, and arctan with two keystrokes maximum.

The losing candidates $\sqrt{x}$, 10^x, and lg shall become secondary functions. But is it better having them as shifted keyboard functions or unshifted *softkeys*? No definite answer can be given here since it depends on the time the respective *menu* will stay on screen. For these particular functions, we made all three g-shifted and put $\sqrt{x}$ also in the unshifted row of EXP.

The *WP 43S* features 33 *menus* on its keyboard. Of these, CATALOG, CONST, U→, and the three alpha character *menus* must be separated since they follow special rules. Each of the remaining 27 *menus* offers six unshifted locations for its most popular *items*. Selecting these can be easy (like in ADV, LOOP, STK, or TRI) or more difficult (like in P.FN or X.FN).

Unshifted *softkeys* are (cf. pp. 127ff):

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

TRI	sin	arcsin	cos	arccos	tan	arctan
U→	E:	P:	year→s	F&p:	m:	x:
X.FN	AGM	B_n	B_n^*	erf	erfc	Orthog
α.FN	x→α	αRL	αRR	αSL	αPOS?	α→x
Σ	n	Σx	Σx^2	Σxy	Σy^2	Σy
∡→	→DEG	→RAD	→GRAD		→D.MS	→MULπ

Repeating any unshifted *softkeys* on the keyboard is of limited value. It makes sense just for occasional use of popular functions – √x needs two keystrokes while √x needs three keystrokes unless EXP is open already. |x| and ∡ are also featured as shifted *softkeys*: accessing |x| or ∡ needs two keystrokes while |x| and ∡ require four maximum (and two if the *menu* is open). In consequence, |x| and ∡ are actually not needed but left in CPX and PARTS since sufficient space is available there.

Alpha Register

For long I thought we could do without a dedicated *alpha register* since each and every *register* is capable holding an alpha string. Some special programming functions like KEYG and KEYX, however, seem to require such a *register* – else handling these functions would become more complicated than it was on the *HP-42S*.

Especially direct entry of alphanumeric constants in programs is easier when the destination is automatically defined, and people became used to this method in decades since the *HP-42S* was launched. Thus, I introduced this *register* in v0.7, taking **K** for 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*). They were represented internally in a fixed format of 1296 units per turn – similar to *short integers* where a fixed bit pattern may be displayed differently depending on *integer sign modes* and bases selected. *Radians*, however, did not fit into this concept due to the necessity for high precision storage of **π** for modulo calculations and reduction of 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 2**π** suffice to warrant 16 digits accuracy of respective function results for the number range of *single precision* reals.[91] *WP 43S* uses 1065 digits for 2**π** to warrant 34 digits accuracy of respective function results within $\pm 10^{999}$.

Backward Compatibility

Compatibility to *WP 34S* and *HP-42S* was kept for many years in a way that programs written for both calculators could run on the *WP 43S* as well (except matrix operations and some flag allocations). It became difficult with the full implementation of the *data type* concept and was eventually abandoned officially when introducing named *system flags* with v0.14.

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).

---

[91] See https://forum.swissmicros.com/viewtopic.php?f=2&t=350#p4349 .

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 CONST.

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

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

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

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

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

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

Afrikaans, Aymara, Bahasa Indonesia, Bahasa Melayu, 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, Polskì, Português, Quechua, Română, Shqip, Slovenščina, Slovensky, Srpski, Suomi, Svenska, Tagalog, Tatarça, Türkçe, Türkmençe, Vlaams, Wallon, and Zhōngwén (hànyǔ pīnyīn).

	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 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.[92] See here some sample strings in either font, approximately to a common realistic scale:

−1.602 22×10⁻¹⁹ C   −1,602 22·10⁻¹⁹ C

−1.602 22×10⁻¹⁹ As   −1,602 22·10⁻¹⁹ As

[92] Some characters displayed by FBR are not found in any other *menu* of your *WP 43S*. They are not required for any *item* provided so far and may be for future use.

Complex Infinities

Since infinities are counted as numeric data being part of the real number range (see p. 170), also complex infinities are part of the complex number plane the *WP 43S* operates on. Coming from rectangular notation, 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 with $\varphi = 0$ at positive real axis.

<u>Calculating</u> with infinities, any finite numbers may be neglected in comparison; so for $|x| \neq \infty$ any <u>inputs</u> like e.g. $x + i \times \infty$ may be replaced by $0 + i \times \infty$, easing calculations significantly. Actually, you will have to deal with the eight cases listed above only.[93]

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

Complex Notation and Storage

Like with angles or short integers, there are different ways a complex number can be written: either in Cartesian or polar notation, the latter with all kinds of angular units. As long as you stay away from infinities, any notation will do (cf. previous chapter).

For reasons of mathematical tradition, rectangular notation is found most frequently. We used it for storing complex numbers in the *WP 34S*. Thus, it is used for the *WP 43S* as well, but care must be taken at complex infinities as explained in previous chapter.

Display Limits

Due to the character sizes and their design (cf. pp. xviii f), the screen could take inputs of up to 23 digits, a sign, and an 8-px radix mark:

-4.2345678901234567890123 ,

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 ,

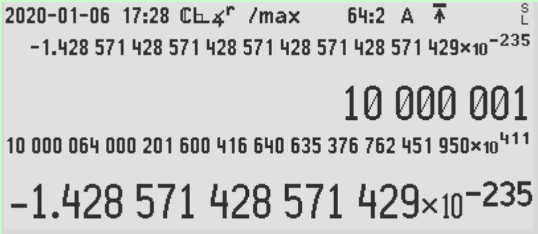
taking 15 + 20 × 16 + 7 × 8 = 391 px again. This maximum precision is independent of the position of the radix mark. Scientific or engineering notation allows for a 16-digit mantissa

$-4.234\ 567\ 890\ 123\ 456 \times 10^{-925}$,

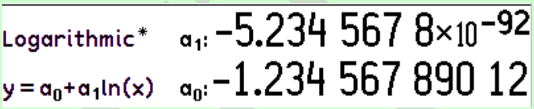
taking 395 px (= 15 + 16 × 16 + 5 × 8 + 15 + 16 + 4 × 13 + 1) for displaying this number this way.[94]

With SHOW, any real number can be displayed with 34-digits precision in a single row:

[94] 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.

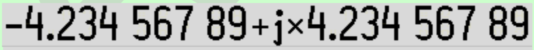


Some *temporary information* may limit output precision, though without limiting its use for real-world applications. E.g. for linear regression, up to 8 digits are viable allowing for 2-digit exponents in SCI or ENG and up to 12 digits in FIX:



.

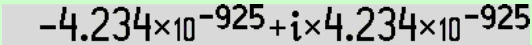
Complex numbers in Cartesian notation require 1 + 15 + 12 + 15 + 1 = 44 px for +j× in addition to the space for two reals. Only the real part may need extra space for a 15-px sign. This allows for 8 decimals per part in worst case



,

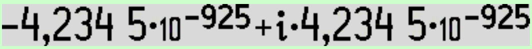
since 44 + 15 + 2 × (16 + 8 + 48 + 8 + 48 + 8 + 32) = 397 px in total. It applies if both real and imaginary parts are in the same order of magnitude and the multiplication cross is chosen.

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):



.

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:



.

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

$$-6.234\,5\times10^{-925}+i\times6.234\,5\times10^{-925},$$
$$-6.234\,56\cdot10^{-925}+i\cdot6.234\,56\cdot10^{-925}.$$

With SHOW, any complex number can be displayed with 34 digits precision:

```
2020-01-06  17:15  CL∡r  /max      64:2  A  ⤒          S L
  -8.403 361 344 537 815 126 050 420 168 067 229×10^-126
-i×5.882 352 941 176 470 588 235 294 117 647 059×10^-158

                                             -1.×10^-33
                   -8.403×10^-126 -i×5.882×10^-158
```
.

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

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

With SCI or ENG, the minimum number of decimals depends on the angular display mode since output is confined to the interval –180° to +180° or its equivalents, e.g. –π to +π in *radians* or –200^g to +200^g in *gon* (see *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*:

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

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

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

Digits in **fractions** are 13 px wide like in exponents. Thus, a 4-digit numerator and denominator take $4 \times 13 + 8 = 60$ px each; the fraction bar takes another 16 px and the trailer 29 (= 16 + 12 + 1). The remaining 235 px would suffice for the optional sign, an 11-digit number, and the 16 px gap between integer and fraction ($15 + 12 \times 16 + 3 \times 8 = 231$ px) in a proper fraction:

-67 890 234 567 2 289/4 567 > .

For ***long integers***, up to 21 digits and a sign may be displayed using the usual large digits:

-123 456 789 012 345 678 901 ,

taking ($1 + 15 + 21 \times 16 + 6 \times 8 = 400$ px). Larger *long integers* employ the small font, allowing for 42 digits and a sign:

-123 456 789 012 345 678 901 234 567 890 123 456 789 012 .

Even larger *long integers* may be displayed with an exponent replacing as many of their least significant digits as necessary:

-123 456 789 012 345 678 901 234 567 890 123 456×10^21 .

With SHOW, one *long integer* may take up to 7 rows meaning 294 digits. *Long integers* $\geq 10^{294}$ shall be trailed by an exponent again taking $9 + 13 + 3 \times 8 = 46$ px in worst case; thus, display precision may be reduced to 288 digits minimum then:

```
2020-01-06  17:20 ℂ⊾∡r /max      64:2  A  ⤒        S L
100 000 320 000 496 000 496 000 359 600 201 376 090 619 2
33 658 570 518 302 804 880 645 122 529 024 502 579 287 47
3 736 471 435 656 572 278 010 804 465 722 767 143 563 473
 736 225 792 852 902 448 645 122 428 048 801 051 830 033
658 560 906 192 020 137 600 359 600 004 960 000 049 600 0
                                          00 320 000 001
```

```
2020-01-06  17:22 ℂ⊾∡r /max      64:2  A  ⤒        S L
10 000 064 000 201 600 416 640 635 376 762 451 949 744 30
1 216 634 616 812 205 996 593 289 175 611 024 560 167 423
 597 795 591 072 087 154 275 016 633 545 248 630 060 887
595 709 345 161 126 819 915 383 185 691 423 735 290 107 2
59 571 090 772 323 598 587 085 564 911 661 593 733 567 97
4 201 161 434 343 572 772 188 085 733 555 689 437 511 105
   995 105 806 963 778 061 809 682 674 493 540 378×10^156
```

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*:

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

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

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

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

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

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

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

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 R∟∡π /3 546f 64:u$^{o}_{c}$ A

containing 45 characters.

Putting the alphabet in a row allows for

abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMN

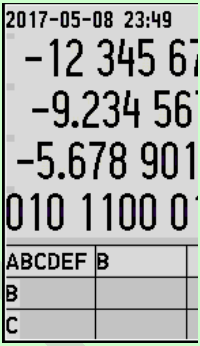
i.e. 41 characters.

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. xviii f). 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→, extra-large labels may appear (cf. p. 135): 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. p. xviii) 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.
- 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 xxv.

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 EQN as written (i.e. following algebraic notation and rules). While editing them, punctuation spaces are automatically inserted after each constant or variable (you know a variable *name* ends when the next operator is entered) as well as after **=** and 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.

Layouting

After drawing a lot of fictional calculators just for fun, we gained our real-world layout experience with the *WP 34S.* Based on it and seeing a forth-coming better display (than the very limited one of the *HP-20b/30b*) allowing for softkeys at the horizon, I conducted a poll on the forum of the *Museum of HP Calcu-lators* about general preferences for the placement of the four arithmetic operators on a then quite hypothetical portrait *RPN* pocket calculator (see

https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234505). I published the basic concept and a first keyboard layout for the *WP 43S* on said forum in November 2012 (see https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685). The picture above shows the last layout before said post, assuming the good old slanted keys of the Seventies.

Everything thereafter was and is just refinement and careful tuning of the basic idea. It even survived an hardware change – we were waiting for *Eric Smith's* and *Richard Ottosen's Reptiles* (see the HPCC meetings until 2015) still when *Michael* mailed me the concept of his and *David's DM42* in March 2016. Actually, the *DM42* development overtook the *WP 43S* – it was launched in the market late in spring 2017 while the *Pauli* and me were looking for willing software engineers and actual support beyond friendly words still in vain. Despite of its popularity in the community, manpower for coding was the bottleneck in our project throughout its life.

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

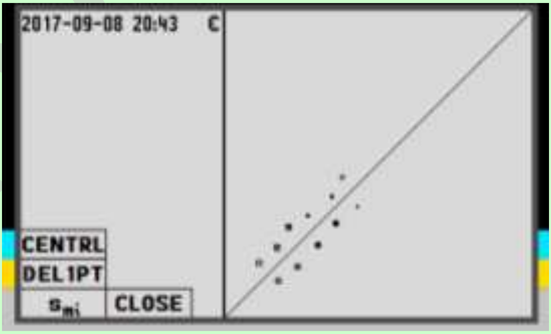
A number range up to 10^{99} is sufficient for almost all real-world problems – else common scientific calculators would feature a larger numeric range generally. So we can conclude that the real number range supported (see p. 166) suffices <u>by far</u> for solving what has to be solved. For sake of 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.

Note the number range determines the precision required for calculating accurately (see *Precision and Accuracy* on p. xxxv).

Plotting?

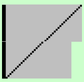
It is mentioned elsewhere we are not out for creating a graphing calculator. There is, however, a very useful application where a basic scatter plot of measured data points would support decision making significantly (see the third industrial statistics application example in *Section 2* of the *OM*). 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 earlier *RPN* calculators.

Plotting could be called by a command PLOT stored in <u>STAT</u> 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 over-written by the diagram. See the sketch (to scale) for general screen layout and various data point symbols for checking visibility.



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

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



Though both axis and line can also be created using AGRAPH and PIXEL.

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, cf. **GRAMOD** on p. 106):

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 orthogonal 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 numeric calculation step, frequently caused by rounding to the internal finite precision the calculator features. Already a simple fraction like $^1/_3$ stored as a real number deviates from the truth by more than 3×10^{-35}. During calculations, such errors accumulate as elaborated e.g. in footnote 65.

In real-world problems, usually the least accurate of all input (real) parameters determines the accuracy of the result. In the standard test mentioned in said footnote starting with 9°, you can nevertheless get 28-digits precision in the result since the input of 9° is exact (but note 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, the *WP 43S* works with 39 digits internally and rounds the results to 34 digits as well. SLVQ calculates using 72 digits internally. Cf. also *Angles* on pp. xviif.

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 (see pp. 135ff).

Prefixes

Prefixes f and g passed without any discussion for over 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, *system flags*, etc. on the *WP 43S* works as listed below, top down and left to right.

Note that 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: [95]

␣ 0020	2003	2004	…	2007	2008	200a
0 0030	▯ 220e	° 00b0	⁰ 2070	₀ 2080		
1 0031	1 2027	½ 00bd	¼ 00bc	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			
8 0038	⁸ 2078	₈ 2088	₈ 2467			
9 0039	⁹ 2079	₉ 2089	₉ 2468			
10 2491	10 2469					
11 246a						
12 246b						

[95] Applying this algorithm, a section of CATALOG'FCNS looks like e.g. s, SAVE, SB, SCI, SCIOVR, scw→kg, …, SLVQ, s_m, SMODE?, s_{mw}, SOLVE, …

The 4-digit number trailing each character in the table is its hexadecimal *Unicode*. Characters printed on grey background are inaccessible for users; those printed on darker grey are not used at all so far.

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

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	Ⓘ	24be	⒤	24a4	Ì	00cc	ì	00ec		
		Í	00cd	í	00ed	Î	00ce	î	00ee	Ï	00cf	ï	00ef
		Ī	012a	ī	012b	Ĭ	012c	ĭ	012d	Į	012e	į	012f
										İ	0130	ı	0131
J	004a	j	006a	Ⓙ	24bf	⒥	24a5						
K	004b	k	006b	Ⓚ	24c0	⒦	24a6	ₖ	2096				

L 004c	l 006c	Ⓛ 24c1	⒧ 24a7	ₗ 2097	Ĺ 0139	ĺ 013a
			Ľ 013d	ľ 013e	Ł 0141	ł 0142
M 004d	m 006d	Ⓜ 24c2	⒨ 24a8	ₘ 2098		
N 004e	n 006e	Ⓝ 24c3	⒩ 24a9	ₙ 2099	Ñ 00d1	ñ 00f1
	Ń 0143	ń 0144	Ň 0147	ň 0148	ℕ 2115	
O 004f	o 006f	º 00ba	© 00a9	Ⓞ 24c4	⒪ 24aa	ₒ 2092
	Ò 00d2	ò 00f2	Ó 00d3	ó 00f3	Ô 00d4	ô 00f4
	Õ 00d5	õ 00f5	Ö 00d6	ö 00f6	Ø 00d8	ø 00f8
	Ō 014c	ō 014d	Ŏ 014e	ŏ 014f	Œ 0152	œ 0153
P 0050	p 0070	Ⓟ 24c5	⒫ 24ab	ₚ 209a		
Q 0051	q 0071	Ⓠ 24c6	⒬ 24ac	ℚ 211a		
R 0052	r 0072	⒭ 24ad	Ⓡ 24c7	Ŕ 0154	ŕ 0155	Ř 0158
					ř 0159	ℝ 211d
S 0053	s 0073	Ⓢ 24c8	⒮ 24ae	ₛ 209b	Ś 015a	ś 015b
	Ş 015e	ş 015f	Š 0160	š 0161	ß 00df	
T 0054	t 0074	⒯ 24af	⊤ 22a4	Ⓣ 24c9	ₜ 209c	
			Ţ 0162	ţ 0163	Ť 0164	ť 0165
U 0055	u 0075	Ⓤ 24ca	⒰ 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	Ⓥ 24cb	⒱ 24b1			
W 0057	w 0077	Ⓦ 24cc	⒲ 24b2	Ŵ 0174	ŵ 0175	
X 0058	x 0078	ᵡ 1d61	Ⓧ 24cd	⒳ 24b3	ₓ 2093	
			x̂ 0379	x̄ 0378	x̅ 037f	∜ 221c
Y 0059	y 0079	Ⓨ 24ce	⒴ 24b4	Ý 00dd	ý 00fd	
	Ŷ 0176	ŷ 0177	Ÿ 0178	ÿ 00ff	ȳ 0233	Ȳ 0232
Z 005a	z 007a	Ⓩ 24cf	⒵ 24b5	Ź 0179	ź 017a	Ż 017b
			ż 017c	Ž 017d	ž 017e	ℤ 2124

Α	0391	α	03b1	α	2065	ά	03ac				
Β	0392	β	03b2								
Γ	0393	γ	03b3								
Δ	0394	δ	03b4	δ	2066						
Ε	0395	ε	03b5	έ	03ad						
Ζ	0396	ζ	03b6								
Η	0397	η	03b7	ή	03ae						
Θ	0398	ϑ	03b8								
Ι	0399	ι	03b9	ί	03af	Ϊ	03aa	ϊ	03ca	ΐ	0390
Κ	039a	κ	03ba								
Λ	039b	λ	03bb								
Μ	039c	μ	03bc	µ	00b5	μ	2067				
Ν	039d	ν	03bd								
Ξ	039e	ξ	03be								
Ο	039f	ο	03bf	ό	03cc						
Π	03a0	∏	220f	π	03c0						
Ρ	03a1	ρ	03c1								
Σ	03a3	σ	03c3	ς	03c2						
Τ	03a4	τ	03c4								
Υ	03a5	υ	03c5	ύ	03cd	Ϋ	03ab	ϋ	03cb	ΰ	03b0
Φ	03a6	φ	03c6								
Χ	03a7	χ	03c7								
Ψ	03a8	ψ	03c8								
Ω	03a9	ω	03c9	ώ	03ce						
(	0028	)	0029								
[	005b	⎡	23a1	⎢	23a2	⎣	23a3				
		]	005d	⎤	23a4	⎥	23a5	⎦	23a6		
{	007b	}	007d								

+ 002b	$^{+}$ 207a	$_{+}$ 208a	± 00b1			
- 002d	$^{-}$ 207b	$^{-1}$ 2072	$_{-}$ 208b	∓ 2213		
× 00d7	· 00b7	∙ 2219	∘ 2218	* 002a	* 208f	
/ 002f	\ 005c					
^ 005e						
, 002c	$_{;}$ 2429					
. 002e	$_{:}$ 2428	… 2026				
! 0021	¡ 00a1					
? 003f	¿ 00bf					
: 003a	∶ 2236	÷ 00f7				
; 003b						
' 0027	‘ 2018	’ 2019	‚ 201a	‛ 201b		
" 0022	“ 201c	” 201d	„ 201e	‟ 201f	« 00ab	» 00bb
@ 0040						
_ 005f	2427					
~ 007e						
→ 2192	› 21c0					
← 2190	↺ 21cd					
↑ 2191	⇧ 21e7	△ 21c9	⤒ 242b			
↓ 2193	⇩ 21e9	▽ 21cb				
⇄ 21c4						
↕ 2195						
≡ 21cc						
¬ 00ac						
∧ 2227	∨ 2228	⊻ 22bb	⊼ 22bc	⊽ 22bd		
& 0026						
| 007c	∣ 2223	∤ 2224	∥ 2225	∦ 2226		

« 226a	< 003c	≤ 2264	≡ 2261	≔ 2254	= 003d	≃ 2243
	≈ 2248	≘ 2258	≙ 2259	≠ 2260	≥ 2265	> 003e
						» 226b
% 0025	$ 0024	€ 20ac	¢ 00a2	£ 00a3	¥ 00a5	§ 00a7
√ 221a	∝ 221d					
∞ 221e	∞ 209e	∞ 209f				
∫ 222b	∬ 222c	∮ 222e	∯ 222f			
⊙ 2299	⊙ 229a	⊙ 2068				
⊕ 2295	⊕ 2069					
∟ 221f	⊥ 22a5					
∠ 2220	∡ 2221					
⌈ 2308	⌉ 2309					
⌊ 230a	⌋ 230b					
⎙ 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			
✓ 2713						

Stack Size

At a very early stage of this project (2013), *stack* size was discussed. An *RPL*-like ‘infinite’ *stack* would allow for saving (pushing) everything thereon before calling a (sub-) routine and popping it after RTN but makes traditional R↓, R↑, and top level repetition obsolete (and FILL as well). In this context I suggested two new commands called CLOSES and OPENS for closing the bottom section (4 or 8 *registers*) of an infinite stack for the time when R↓, R↑, FILL, and top level repetition were required, and opening it thereafter. At the bottom line, eight *stack registers* turn out being sufficient for solving any real-world mathematical, scientific, or engineering problem (cf. *Section 1* of the *OM* as well as field experience with *WP 34S* and *WP 31S* since 2011).

After all, we decided sticking to *RPN* as implemented on the *WP 34S* and *WP 31S*. It covers everything needed most easily. For 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. 42.

Structured Programming

In 2013, I suggested the following control structures:

- IF … THEN … ELSE … END,
- FOR … FROM … TO … END,
- REPEAT … UNTIL, and
- WHILE … END.

Traditional END would need to be called ENDPGM then.

Later, we discussed some *PASCAL*-like structures:

- IF … THEN BEGIN … END ELSE BEGIN … END;
- FOR … DO BEGIN … END; and
- WHILE … DO BEGIN … END;

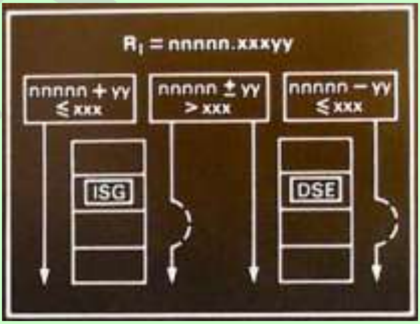
We refrained from implementing such commands since we had doubts about the sensibility of mixing keystroke programming and structured programming features.

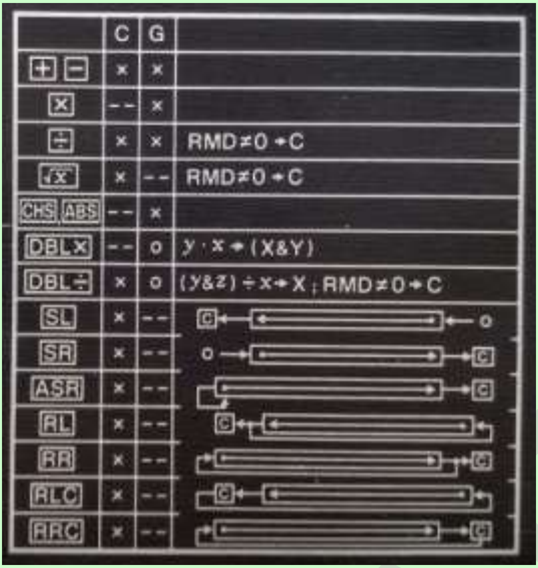
UNDO

In 2013, UNDO was planned as it works in *HP-48*, recalling just the *stack* as it was before executing last command. The *WP 31S*, on the other hand, features an UNDO recalling the entire calculator status as it was before executing last command. It turned out that such a complete UNDO was viable on the *WP 43S*, too, so we implemented it for better user experience.

Please find here two surfaces printed to scale: at right the original keys and keyplate of your *WP 43S* and below its virtual keyboard in alpha input mode (*AIM*). Furthermore there are two pictures taken from the back of an *HP-11C* and *-16C* below.

Choose your favorite, cut it out, and attach it to the back of your *WP 43S*.





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