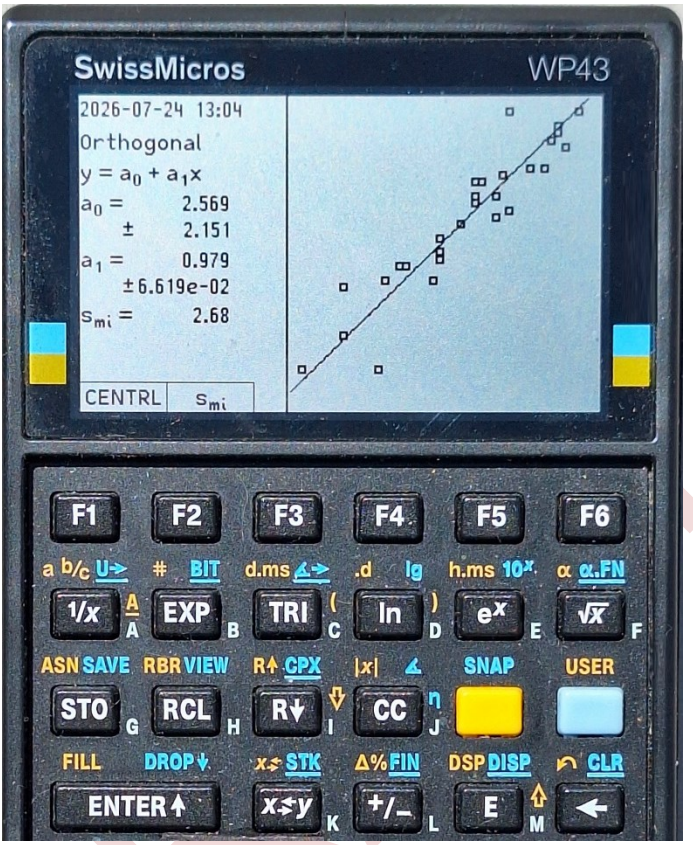


WP43 OWNER'S MANUAL
AND PROBLEM SOLVING GUIDE

This manual explains *WP43*. (*WP43* is a free scientific software for *DM42* calculator hardware built by *SwissMicros*. You can redistribute that software and/or modify it under the terms of the *GNU General Public License* as published by the *Free Software Foundation*, either version 3 of the *License*, or (at your option) any later version. *WP43* is published and distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. Please see *https://www.gnu.org/licenses/* for more details.)

This manual is preliminary still; it will change while we develop *WP43* in the course of this project. We reserve the right to do so any time. The fundamental principles of *WP43* will remain constant, however. You can stay informed by watching *https://gitlab.com/rpncalculators/wp43*.

HP is a registered trade mark of *Hewlett-Packard*. *Excel* and *WordPad* are applications of *Microsoft*.

The photographs on p. 20 were published by *NASA* in 1969 and 2026. All paintings, the drawings on pp. 37, 81, 146-148, 153, 156-173 (top), 229f, 268, 277, 295ff, 348f, and 352-356, as well as the pictures on pp. 57, 215ff, 260, and 368 were taken from various calculator manuals and advertisements published by *HP* between 1974 and 1989 (some were refined by me). The diagrams on p. 98 and the picture on p. 298 are based on material found in *Wikipedia*. The sources of the maps on pp. 92, 312, and 314 are stated locally – I found them by chance browsing the internet, as well as the pictures on pp. 24 and 53. All *WP43* keyboard graphics as well as the other photographs, pictures, graphics, and diagrams printed in this manual were created by the author.

Internet addresses and links are specified as found and verified at 2026-02-03 (just the map on p. 92 is no longer online). Please note internet addresses may change or vanish 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 in Europe) – using it we can reach the maximum number of readers without further translations. This is the only reason. I apologize to the people of other languages and inserted some '*translator's notes*' where applicable.

(It is remarkable that the two most widespread written languages on this planet nowadays are the most ambiguous ones: English and Chinese. Both rely heavily on context since they have a reduced grammar and flexion compared to, e.g., Spanish. Older 'common' languages were more elaborated (compare Latin). What can be learned from that about development of our species? – You may find more barbed remarks below, reflecting the author's impressions watching mankind for 7 decades so far. According to *George Orwell*, *all animals are equal, but some animals are more equal than others*. Fortunately, my ancestors fought successfully for freedom of the press, hence also the latter animals may be criticized. Great ones as well. No personal offense intended.)

Printed in the *USA* just since I got the *ISBN* there most easily.

ISBN-13: 978-172950098-9
ISBN-10: 172950098-6

WP43 would neither have been created without our love for *Classics, Woodstocks, Stings, Spices, Nuts, Voyagers*, and *Pioneers* nor without our ability to imagine better products on a way towards a better world. Thus we want to quote what was printed in *HP* 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

Just in Case You Don't Have Got Your Own *WP43* Calculator yet …

A pilot batch of *WP43* calculators was made on *SwissMicros'* production line in China in the summer of 2022. These were sold out in few weeks – thus, please turn to *https://www.swissmicros.com/* expressing your wishes.

Alternatively, you can create a *WP43* most easily by flashing a *DM42* you own – this way will cost you next to nothing but an overlay will become inevitable. Please consult *Appendix F* of the *WP43 Reference Manual* for what and how to do.

The *WP43* software package includes a free full-size *Simulator* (see *Appendix E* of said manual), so you can check it out on your computer before buying any hardware. And later on, you may test and edit your programs there as well before sending them to your *WP43* calculator. Function sets are 99% identical on both platforms.

For the following, we assume you hold your own *WP43* in your hands – or have at least access to its *Simulator*.

TABLE OF CONTENTS

Welcome!

We congratulate you on your new *WP43.* It will serve you well, being an accurate, reliable, fast, and compact problem solver like you have never owned before – programmable, user-customizable, connecting to your computer, incorporating a high-resolution display and *RPN*[1] – a serious scientific instrument supporting you in your studies and professional activities, be it in mathematics, science, engineering, computing, or business. It readily provides advanced capabilities never before combined so conveniently in a true pocket-size calculator:[2]

- The ***Solver*** (root[3] finder) can solve for any variable in any equation you entered using the ***Equation Editor***. Numeric integration, differentiation, and basic plotting of equations are supported as well.
- A **wealth of functions** operating on integers, fractions, real and complex numbers, vectors, matrices, angles, dates, times, and text strings. Programmable sums, products, integrals, and derivatives.
- A comprehensive set of field proven **statistical operations**: probability distributions, functions for data analysis, curve fitting, interpolation, forecasting, and – for the very first time on a pocket calculator – **measuring system analysis (*MSA*)**.
- **Vector and matrix operations** on real and complex arrays of arbitrary size; dot and cross products, eigenvalues and eigenvectors, and a solver for systems of linear equations, all supported by a **full screen *Matrix Editor***.
- **Long decimal integers** of up to 1001 digits for precise numeric applications. Find and check primes. Factorize integers.
- **Integer arithmetic** in 15 bases from binary to hexadecimal; base

[1] *RPN* stands for *Reverse Polish Notation*, a fast, effective, and coherent method for solving mathematical problems efficiently. It is based on a logic known as *Polish Notation* (since proposed by the Polish logician *Jan Łukasiewicz*). He put operators before operands instead of between them as in usual math notation. *RPN* places operators after operands; → *Section 1* below and the *WP43 Reference Manual* (*ReM*).

[2] Per original definition, a pocket calculator is a device fitting comfortably in the pocket of your shirt or jacket. Marketing people see this feature more elastic – our pockets are not elastic enough for that. Being at it, we recommend not to put your *WP43* in the back pocket of your jeans – beyond frazzling your pocket, it may break or multiply there.

[3] Translator's note for readers speaking German: *Root* bedeutet hier *Nullstelle*.

conversions and extensive **bit manipulation** (shift, rotate, mask, count, flip, test, etc.) in words of up to 64 *bits*.

+ Field proven **keystroke programming** incl. conditional tests and branching, finite and infinite loops, user and system flags, subroutines, recursions, global and local data. Create and refine your programs using a **multi-line editor** on your calculator or computer.
+ An **innovative, convenient menu system** assigning up to 18 operations to the top 6 calculator keys according to your actual needs.
+ **Alphabetically searchable *catalogs*** for easy finding all the items stored in memory, be they provided by us or created by you.
+ A **timer** (a.k.a. stopwatch) based on a real-time quartz clock.
+ Easy system **control by named system flags and variables**.
+ **Battery-fail-safe on-board backup** for all your data and programs; calculator state files you can exchange with your fellow *WP43* users.
+ A ***USB* socket** allowing for auxiliary external power supply as well as for program exchange with your computer – feel free to transmit, edit, and test routines there and return them verified to your *WP43*.
+ A **user interface (*UI*) you can customize yourself**: create your own menus, specify your personal keyboards and calculator states, save all of them on-board and recall them one by one as you need them. Dedicated keyboard overlays are supported.

WP43 provides the amplest function set ever seen in an *RPN* calculator:

+ **786 functions**, including *Euler's Beta* and *Riemann's Zeta*, *Lambert's W*, *Bessel functions of 1st and 2nd kind*, *Bernoulli* and *Fibonacci numbers*, *Jacobi elliptic functions* and *integrals*, as well as *Chebyshev*, *Hermite*, *Laguerre*, and *Legendre polynomials* – get independent of heavy table books and computer software in these matters.
+ **14 probability distributions**: *normal* (*Gaussian*), *Student's t*, *chi-square*, *Fisher's F*, *Poisson*, *binomial*, *geometric*, *hypergeometric*, *Cauchy-Lorentz*, *exponential*, *Weibull*, and more.
+ **10 curve fit models** (linear, orthogonal, exponential, logarithmic, power, root, hyperbolic, parabolic, as well as Cauchy and Gauss peak). Data plots let you assess your fits, draw histograms, and even

analyze and qualify your measuring instruments on site. Just do it! ™

+ **54 fundamental physical constants** as precise as used today by your national standards institutes (following *CODATA 2022*) plus 25 important mathematical, astronomical, and surveying constants.
+ **164 unit conversions** (for temperatures, times, angles, distances, areas, volumes, masses, forces and pressures, torques, energies, powers, and ratios), mainly from *British Imperial* or *US customary* to universal *SI* units and vice versa.
+ A **compact set of financial functions** and applications for the business matters you may experience as a technical manager.

WP43 features lots of space for your data, results, programs, and ideas:

+ A high contrast ultra-low power **240 × 400 pixel *LCD*** allowing for crisp results and menus, plots, natural display of matrices, a wide variety of mathematical symbols, Greek and international Latin letters.
+ Your choice of **workspace comprising 4 or 8 stack registers – calculate sans soucis**™; up to **107 global general purpose registers** and **1000 named variables.** Each register or variable can take any object of arbitrary data type (be it a matrix, vector, text string, or any number of arbitrary kind).
+ **100 global user flags** and some **40 named system flags**.
+ Additionally, **32 local user flags** and up to **99 local registers** per routine, allowing, e.g., for recursive programs.
+ Space for some **20 000 program steps in *RAM***, up to **10 000 per program**. Battery-fail-safe flash memory for saving and exchanging your routines, data, settings, configurations, layouts, and entire calculator states.
+ Last not least, your *WP43* computes with over **34-digit precision** – the most precise pocket calculator available worldwide, second to none.[4]

Your *WP43* is a true pioneer: since 2022, it is the very first entirely community-designed and -built *RPN* pocket calculator on the market. All of its hardware, firmware, *UI*, and layout were thoroughly thought through, discussed, designed and assembled, written and tested by us over and

[4] See the *ReM* for more about precision and accuracy.

over again since 2012. *WP43* arose from a temporary collaboration of two independent international teams: *SwissMicros* (i.e. Czech *David Jedelsky*, Swiss *Michael Steinmann* and *Emy Amstein*) created the hardware and *DMCP*;[5] French *Martin Lorang* and *Didier Lachieze*, *Mihail* from Japan, South African *Jaco Mostert*, British *Benjamin Titmus*, Australian *Paul Dale*, and I designed the *UI* and wrote the firmware.[6] Our overarching goals were the coherence and internationality of the *UI* as well as precision, accuracy, and reliability of the results.

As our *WP 34S* and *WP 31S* calculators before, also *WP43* is just a hobbyist's project. I published a first *WP 43S* draft layout in 2012; it was discussed on two online forums then. Prototypes of *SwissMicros'* *DM42* were presented first at *HHC2016* (*USA*) and the *Allschwil Meeting (AM) 2016* (Switzerland). *Martin* started coding *WP 43S* firmware in 2017. He and I presented the first version of the *WP 43S Simulator* at *AM2018*. Later on, the narrow blank and the *scientific* '*S*' were sacrificed for better alignment with *SwissMicros'* calculator labels, and *AM2022* saw the pilot batch of *WP43* calculators made and distributed.[7] We thank all participants of said meetings and all members of the international calculator community who contributed their ideas, put their votes, and in particular

[5] *DMCP* = *David's* and *Michael's Control Program* = basic firmware layer made for their *DM42* launched in 2017. The *DM42* layout is very closely linked to the *HP-42S* (produced from 1988 to 1995) and its firmware uses *Thomas Okken's* outstanding *Free42* software simulating the *HP-42S*.

[6] … with major contributions of Italian *Gianluca Puggelli*, Dutch *Harald Overbeek*, as well as Germans *Gert Menke* and *Friedrich Mütschele*.

The firmware of your *WP43* is based on the experience *Paul* and I gained with the *WP 34S RPN Scientific Calculator* (this project started in 2008 when *HP* launched its first repurposable pocket calculator, the *HP-20b*). Together with German *Marcus von Cube*, we could launch *WP 34S* in 2011. Find specific information about it and its derivative, the *WP 31S* of 2014 (created by *Sanjeev Visvanatha* (Canada), *Jonathan Cameron* (*USA*), and us) at *https://sourceforge.net/projects/wp34s/*.

[7] There is a video of a talk at the 2022 *AM* describing our path from *WP 34S* to *WP43* in a nutshell (→ *https://www.youtube.com/watch?v=gPzCMss_imE*).

If you are interested in the entire long and winding road how your *WP43* got the features and layout you are facing now, look up *https://www.hpmuseum.org/forum/forum-8.html* until 2016 and *https://forum.swissmicros.com/viewtopic.php?f=41&t=1816* since 2017. See also the *ReM* for the full *Release Notes* as well as *Background Facts and Considerations*. And feel free to check *https://gitlab.com/rpncalculators/wp43* to find out who contributed when and what – there are quite some fluctuations (and, in consequence, ups and downs in activity) in such a hobbyists' project.

those who lent their support in various phases of this project. We greatly appreciate your constructive contributions! Feel free to monitor said forums for further news. And be aware we are always looking for new team members, in particular for our software department.

We baptized our baby in honor of the *HP-42S* of 1988, the most powerful *RPN* scientific pocket calculator made before our activities. May it be a worthy and valiant successor of the *HP-42S*[8] – though we would have preferred *HP* making it (*HP* as many of us remember it controlled by *Bill Hewlett* and *David Packard* until the 1980's).[9] In any way, *WP43* stands in the tradition of over 50 years of *RPN* pocket computing.[10]

We carefully checked the features of *WP43* to the best of our abilities, so we are confident it is free of severe bugs. This cannot be guaranteed, however, so we will improve *WP43* whenever necessary. Should you discover any strange result, please verify its reproducibility and look up the *Troubleshooting Guide* in the *Reference Manual*; if it persists, report it to us (SNAP will take a snapshot of your *WP43* screen, see p. 257). We will investigate, correct if necessary, and provide a free firmware update as soon as possible – you can flash it yourself as explained in *Appendix F* of said manual. Like we have done for our *WP 34S* and *31S* since 2011, we strive for maintaining short response times.

Enjoy your *WP43*, be it the *Simulator* or the real thing!

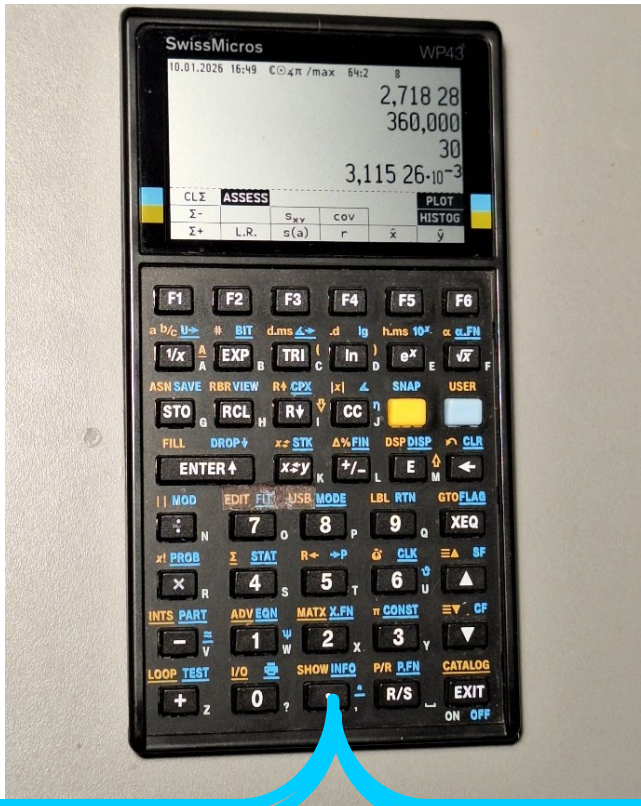
Walter

---

[8] In a nutshell, *WP43* comprises the functionalities of *WP 34S*, *WP 31S*, *HP-42S*, *HP-16C*, *HP-21S*, and more. See overleaf and the *ReM* for details. *HP-42S* is our reference – if in doubt we follow its functionality. *WP43* is not (and was never planned to be) *HP-42S* compatible, however.

[9] Still a recommended read if you can get this vintage book about *HP's* rise and philosophy: *David Packard: The HP Way; Harper Collins*, 1996.

[10] There are even some offsprings of our *WP43*, called *C43, C47*, *R47*, etc. While documented on their own sites (just search for the respective model), they lack dedicated user manuals as far as I know. Since they forked from an earlier stage of *WP 43S* in 2019, the fundamental ideas (e.g., display layout, the 3 menu rows, the overwhelming part of their fonts) and most of their functions are identical. Thus, you may profit from our two *WP43* manuals (conveniently readable also on your smartphone) for those, too. We will not, however, explicitly refer to those in the following text – design philosophy differs and we simply don't know enough about them to be competent in those matters.


SwissMicros
WP43
10.01.2026 16:49
2,718 28
360,000
30
3,115 26·10⁻³
ASSESS
PLOT
HISTOG
ENTER
EXIT
ON OFF


42S
RPN SCIENTIFIC
-2.0676E-13
21S
STAT/MATH
0.0435
4E20 h
HEWLETT-PACKARD
WP 31S
RPN SCIENTIFIC
3.571428571429
WP 34S
RPN SCIENTIFIC
0.0302
1.934235976793

Print Conventions and Common Abbreviations

- Standard text font in our manuals is *Arial*. Emphasis is added by underlining or **bold** printing. **COMMANDS**, MENUS, predefined VARIABLES and SYSTEM FLAGS (SET or ~~CLEAR~~) are generally called by their *names*, printed capitalized in running text.

  Examples are printed green, quoted text blue (as well as translator's notes). Bold italic letters (like ***n***) denote variables, bold regular characters (like **b**) represent constant values (e.g., specific labels, numbers, or symbols).

  Specific *terms, titles, trademarks, names*, or *abbreviations* are printed in italics, *hyperlinks in blue underlined italics*; such links will point to pages, the *Table of Contents*, or an external address fully specified.

- *Times New Roman* regular letters are used for unit symbols and mathematical functions, italics for *unit names* in running text.

  Bold capitals denote **REGISTERS**, lower case bold italics their ***contents***.[11] E.g., *register* **Y** contains the value ***y***, and **R45** contains ***r45***. *Stack* contents are generally quoted in order [***x***, ***y***, ***z***, …].

- This KEY font (created by *Luiz Vieira* of Brazil) denotes calculator KEYS, incl. **softkeys** in general. For shifted operations like GTO or SF, the respective colors are used. **Text** and numeric outputs (like 1.234×10^{-56} or $7{,}089\cdot10^{12}$) are printed as seen on your calculator screen.

- *Courier* is used for `filenames` and describing numeric `formats`.

- Regarding mathematical notation, we follow *ISO 80000-2* to comply with international standards. For the manuals, we chose decimal points, multiplication crosses, and division slashes – you may choose decimal commas and multiplication dots on your *WP43* as well, of course.

All this holds unless stated otherwise locally.

The abbreviations below, sorted alphabetically, are used throughout this manual:

→	=	see (→ ***n*** = see page ***n***).
1D, 2D, 3D	=	1-, 2-, or 3-dimensional.
4LS, 8LS	=	4-*level* or 8-*level stack*.
ADM	=	angular display mode.
AFH	=	(*HP-15C*) *Advanced Function Handbook*
AIM	=	alpha input mode.

[11] We kept the term *register* for the space where an object is stored, although the actual size of such a *register* may vary widely following to the size of the object stored therein.

App.	=	Appendix.
DT	=	data type.
FD	=	the *USB FAT* disk.
FM	=	battery-fail-safe flash memory, encompassing the *FD*.
f.	=	footnote.
GP	=	general purpose.
HP	=	*Hewlett-Packard.*
IOI	=	*Index of all items* provided in *WP43* (see *ReM* below).
MSA	=	measuring system analysis.
OH	=	*Owner's Handbook.*
OHPG	=	*Owner's Handbook and Programming Guide.*
OM	=	*Owner's Manual* (*OMS **n*** = *OM Section **n***).
PC	=	computer of arbitrary kind and brand.
PEM	=	program entry mode (compare *RUM*).
PM	=	program memory (in *RAM* for the calculator).
PP	=	program pointer.
ReM	=	*WP43 Reference Manual*, comprising the *IOI* and more.
ReMA …	=	*ReM App. ...*
*ReMS **n***	=	*ReM Section **n***. Note *ReMS 1* contains the *IOI*.
RoV	=	register or variable.
RPN	=	*Reverse Polish Notation* (→ 9, f. 1).
RUM	=	run mode (compare *PEM*).
SDC	=	startup default configuration (e.g., after **RESET**).
SI	=	*système international d'unités*.[12]
TAM	=	transient alpha mode.
TI	=	temporary information.
TN	=	translator's note (and *TNG* = *TN* for German readers).

A few further abbreviations may be introduced and used locally.

Finally: WARNING indicates risk of severe errors. Only 3 warnings had to be printed in this manual. Resetting your calculator by actuating the RESET button on its back will help in almost all cases – but this will erase all your data in *RAM*, just the *FM* contents will remain. You find a *Troubleshooting Guide* in *ReMA G*.

[12] *SI* is a comprehensive coherent system of rational units of measurement agreed on internationally and adopted by almost all countries on this planet. The last 3 not participating yet are the people of Liberia, Myanmar, and the *USA*, obviously not knowing what they miss. Feel free to look up basic information about *SI* in *https://en.wikipedia.org/wiki/International_System_of_Units*. See also *Constants* and *Unit Conversions*.

SECTION 1: GETTING STARTED

Your *WP43* is an extremely powerful, versatile problem-solving tool. It allows you to solve even very elaborate mathematical and computational problems in either of two ways:

- **Manual problem solving:** Using the *RPN* logic system of your *WP43*, you can manually work step-by-step through the toughest problems while seeing intermediate answers after each and every step of the way. The advantages of *RPN* become particularly apparent when working with exploratory type problems where intermediate answers are an important part of the problem solving process.
- **Programmed problem solving:** Your *WP43* can remember any sequence of keystrokes you entered, and it can then run it repeatedly as often as you need it. This simple programming paradigm is particularly useful in providing answers to repetitive problems that require different inputs. Advanced programs may also be written for solving more elaborate tasks, e.g., iterative computations containing automatic decisions and branching. Thousands of keystrokes can be recorded in your *WP43* and exchanged with your *PC*.

SAFETY WARNING: Your *WP43* is not designed to be used by children under 3 years. Keep toddlers away from it. This is not a toy.[13]

Don't use it before you can tell left from right, read, and do mental arithmetic reliably. It contains small parts which if swallowed are hazardous for health. Don't ingest the battery.[13] **Chemical Burn Hazard: SWALLOWING THE BATTERY CAN BE FATAL WITHIN 2 HOURS – SEEK MEDICAL ATTENTION AND HELP IMMEDIATELY!**

Follow the operating instructions in this manual carefully. Do not use your *WP43* for any other purposes than specified herein (neither

[13] No, we don't think you are stupid, irresponsible, or lacking any life experience at all. Alas, however, such a waste of print space became inevitable due to the legal system, lawyers, and courts of that great nation where such warnings appeared first in user instructions a few decades ago. Other people knew and know coffee being hot, usually, and don't even think of drying pets in a microwave. Good grief!

TN: Battery bays must be secured by a small *Phillips* screw in the *European Union* while extended warnings are printed in the *USA*. Guess what will keep toddlers away from batteries more reliably.

as a hammer nor as a lever nor as a door stop, neither kick nor throw it); else you may destroy your *WP43* and/or other objects easily, even hurt animals or human beings including yourself.[13]

ATTENTION: Your *WP43* shall be operated with care in a clean, dry environment (relative humidity less than 75%, non-condensing) at ambient temperatures from 5 to 40 °C (41 to 104 °F).

Do not drop your *WP43* on solid ground – it may break. Do not leave it lying in the sun; its dark surface may become hot, and hot surfaces may cause burns. Do not leave your *WP43* lying in the cold; humidity may condense on its surface when a cold *WP43* is put in warmer space.[13]

Should your *WP43* become wet, turn it off, remove the battery (see *Disassembly* below), and let your *WP43* dry for sufficient time before reinserting the battery and turning it on again. Do not try to accelerate drying neither by blowing hot air (exceeding 60 °C or 140 °F) into your *WP43* nor by putting it into an oven nor the like – you may destroy it.[13]

Disassembly: Do not disassemble your *WP43* unless you are qualified for such work and have the proper tools handy. You will need 1 small *Phillips* screwdriver and 2 hands for opening it.[13]

Replacing the Battery: Your *WP43* contains a battery (surprise!). When it runs out of power, the symbol 🪫 will appear top right on its screen. Then, open your *WP43* (→ *Disassembly* above) and replace its old battery by a fresh quality cell.[14] Do not operate your *WP43* without a good battery installed – old cells may leak and the acid may oxidize and destroy your *WP43*. Dispose of the old battery responsibly (see next paragraph). Keep it out of kids' reach! Neither take it apart nor throw it in a bin nor in a fire nor in your environment.

Disposal of the calculator: Your *WP43* must not be disposed of along with household waste. You are responsible for deleting all personal data stored on it prior to its disposal. Remove its battery (→ *Replacing the Battery* above) and dispose of it separately. Lithium batteries are only to be handed in the appropriate in-store containers or at your local disposal center in a discharged state. The batteries must always be protected against short circuits by taping off the poles. Then dispose of your *WP43* calculator according to applicable laws and regulations at your official electronics collection point.[15]

[14] → *https://technical.swissmicros.com/dm32/doc/dm32_user_manual.html#_battery*

[15] Life was easier in these matters before electronic devices became abundant.

When you have taken that and **you know already how to deal with an *HP RPN* scientific calculator, feel free to start using your *WP43* right away** now**.** You will find it easy and intuitive. Browse this manual to learn about some fundamental design concepts putting your *WP43* ahead of previous *RPN* scientific pocket calculators. Don't miss *App. 1* (→ 327).

If, on the other hand, *RPN* is new to you, we recommend going through *Sections 1* and *2* thoroughly. This will empower you to easily solve problems manually, benefitting from this unique logic system implemented. Once learned, *RPN* forms a lifelong lasting, reliable basis of your work.

Most commands of your *WP43* work like they did on its antecessors, in particular on *WP 34S* and *HP-42S*. Hence, its manuals focus on the new features of your *WP43*. They may include some formulas and technical explanations; though they are not intended to replace any textbooks on mathematics, statistics, science, engineering, economy, computer science, or programming.

Within this *Owner's Manual*, …

OMS 1 starts introducing the keyboard to you, so you learn where to find what you are looking for. It continues demonstrating basic calculations, shows the *WP43* memory and how to address objects therein.

OMS 2 goes beyond basic operations, presenting output formatting and more advanced functions on *real numbers* and *fractions*.

OMS 3 goes beyond *real numbers*, introducing further *data types* (*angles*, *complex numbers*, *matrices*, *integers*, *dates*, *times*, and *text strings*).

OMS 4 goes beyond *run mode*: it demonstrates how to program your *WP43* for automatic problem solving, following tried and tested concepts (known from successful antecessors up to *HP-42S* and *WP 34S*). It also shows how your *WP43* can communicate with your *PC*.

OMS 5 and ***6*** present advanced functionalities implemented.

OMS 7 covers everything you need for customizing the *user interface* of your *WP43* according to your very personal preferences.

This manual is supplemented by the *WP43 Reference Manual*. Most important may be its *Index of all Items* available (*IOI*): it presents comprehensive information about all the *commands*, *menus*, *system flags*, con-

stants, and *variables* provided, what they do and how to call them. The *ReM* closes with appendices covering special topics (like the free *WP43 Simulator* for your *PC*, advanced mathematical functions implemented, and how to keep your *WP43* up-to-date whenever we will release new firmware revisions).

Before diving into this *OM*, here is something we ask you to remember:

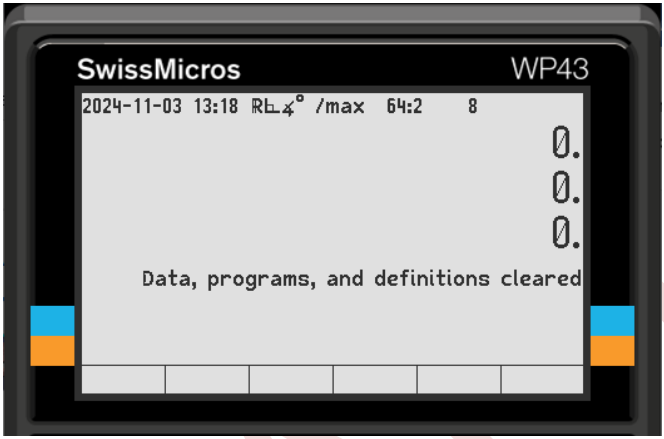
> **Your *WP43*** is designed to support you solving analytical and calculatory problems. But it **is just a tool:** it can neither think for you nor can it check the sensibility of a problem you apply it to**. Gather information, think before and while keying in and calculating, and check your results: these tasks will stay your responsibility always. We will not be liable for any of your results, nor their consequences.**

This iconic photograph of our Earth rising over the Moon was taken by the 'astronauts' of *Apollo 8* in 1968. Remember orbiting the Moon and even landing on it five times was accomplished before just one single scientific pocket calculator was launched on our planet – most engineering results then were achieved by careful slide rule calculations (compare the movie '*Apollo 13*').

Imagine how much more is within your power using precise tools like your *WP43*. Apply it wisely, taking care of our planet – it's our only home we have in this breathtaking, black, and bitter cold huge universe. Distant worlds are just too far away – they most probably exist but definitely out of our reach. Just similar views reappeared in 2026, 58 years later.

Problem Solving, Part 1: First Steps

Start exploring your *WP43*: Press its bottom right key – notice **ON** is printed below that key. Turned on the very first time, it will show a display like this:



To switch it off, press , then **EXIT** (note **OFF** printed below it). Note your *WP43* will shut down automatically after 10 minutes of inactivity for conserving battery power – just turn it on again and you can resume your work right where you left off. Nothing is deleted.

This works as on preceding calculators (like *WP 34S* or *HP-42S*). Your *WP43*, however, looks cleaner than a *WP 34S* while more colorful than an *HP-42S*. This is since your *WP43* features 2 general *prefixes* and – offering you up to 6 functions per key.

To call a *primary* function printed white, just press the corresponding key. For calling a golden or blue function, press or first.

Take the key [×],[16] for **example** (see overleaf). Pressing ...

- [×] alone executes a multiplication.
- + [×] calls the *factorial* [x!] (you will learn more about it in *Part 2*).

[16] For better readability in the manuals, we refer to keys using dark print on light background from here on. And for shifted functions like [x!] or **OFF**, we will omit or most times since redundant by color print.

- [ ] + [×] opens PROB, a *menu* of functions related to probability (therein, [▲] or [▼] will show you more of this *menu*, and [EXIT] will leave it). All labels printed underlined refer to *menus*.[17]
- The grey **R** will become relevant with text data.

Find all labels printed on the keyboard explained in *Appendix 1: Key Response Table*. – Time for a first problem solving **example**:

Let's assume you want to fence a little patch of land,[18] rectangular, 40 *yards* long and 30 wide.[19] You have already set the 1st corner post (**A**), and also the 2nd (**B**) in a distance of 40 *yards* from **A**. Where do you set the posts **C** and **D** to warrant the fence forming a proper rectangle? Just key in:

[3] [0]	0. 30_
[DISP] FIX [0] [1] [20]	0.0 30

Note the input cursor _ vanished and 30 is adjusted to the right now, indicating

---

[17] This clear and compact way of signaling *menus* is also used in these manuals.

[18] Assume it in suburbia. Remember?

[19] Here, we employ old *British Imperial* units since they are still spread in some British colonies and ex-territories. Although dealing with them is far more cumbersome than using rational units, the parochial heirs of the progressive immigrants there will just not get rid of them. Though solving this fencing problem will work with *meters* as well.

[20] Press [ ], then [E] to access [DISP]; then [F1] for FIX and type [0] [1], telling your *WP43* to display only 1 decimal digit (internally, everything is handled with full 34-digit precision always). You will learn more about output formatting in *OMS 2*.

Generally, we will print no more than 1 display row comprising just **0** from here on.

this input is closed; so the next number can be entered without interfering:

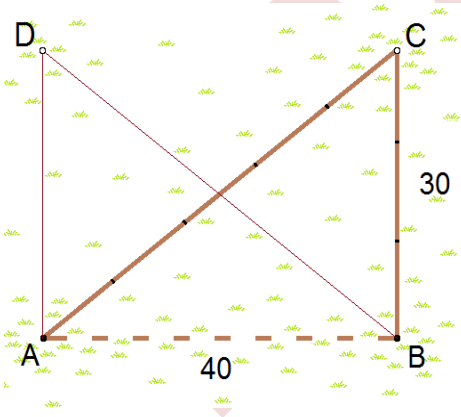
4 0

```
                0.0
               30
40_
```

→P


```
                0.0
ϑ =          36.9°
r =          50.0
```

Now, all you need is the bottom number, a friend, 80 *yards* of rope, and a black marker: Ask your friend to hold both ends of the rope firmly, take the loose loop and walk away as far as you can – when the loop is stretched, mark this position on the rope and return. Ask him or her to hold this marked point of the rope as well, fetch the 2 loose loops and walk again as far as possible: with both loops stretched, mark these positions on the rope. Return again; hand over the 2 new marked points and walk away once more, now with 4 loose loops. After marking as before, your rope will show regular marks every 10 *yards*.[21]



Take this rope now, nail its one end on post **A**, its other end on **B**; fetch the loose loop, a hammer, and a post, and walk 5 marks away as calculated. When both sections of the rope are tightly stretched, place post **C** there. Set **D** the same way on the other side.

This solution will work for arbitrary rectangles (rather take a long tapeline instead of a rope then): As soon as you press →P, your *WP43* will calculate the diagonal automatically for you. You just provide the land, posts, hammer, nails, and tapeline – and it will remain up to you to set the posts.

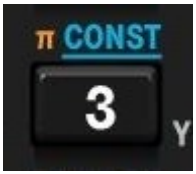
Another **example**, with slightly greater areas:

[21] Something alike, a knotted cord or rope, is known as one of the oldest surveying tools of mankind, evidently used in Egypt some 5000 years ago. Markers were not invented yet.

To calculate the surface area of a sphere, the formula $A = \pi\, d^{\,2}$ can be used, where ***A*** is the surface area, π is the constant 3.141 59..., and ***d*** is the diameter of the sphere. [22]

Ganymede, one of *Jupiter's* 115 moons, has a diameter of 5262 km. To use your *WP43* to manually compute the area of *Ganymede*, you can press the following keys in order:

5 2 6 2	5 262_	
ENTER↑ ×	27 688 644	square of the diameter
π	3.1	the constant π (rounded to 1 decimal as chosen above)
×	86 986 440.6	area of *Ganymede* (in km^2)



If you wanted the surface areas of each of *Jupiter's* spherical moons, you could repeat the above procedure as often as needed. However, you might wish to write a *program* that would calculate the area of a sphere from its diameter, instead of pressing all the keys for each such moon.

To calculate the area of a sphere using a program, you should first

write the program, then you must

record the program into the calculator, and finally you

run the program to calculate the answer.

Writing the program: You have already written it! A *WP43* program is nothing more than the sequence of keystrokes you would execute to solve the same problem manually.

Recording the program: To record the keystrokes of the program into the calculator, press the following keys in order:

---

[22] *TN*: Quoted from the *HP-25 OH* though updated – only 12 moons of *Jupiter* were known in 1975 (not counting the black cuboidal monolith seen in *Stanley Kubrick's* masterpiece *2001 – a Space Odyssey* of 1968, an amazingly realistic science fiction movie shot before even *Neil Armstrong* set foot on the Moon). Emphases were added by me.

Reading *HP*'s vintage calculator manuals may be real fun. *Eric Rechlin* collected them in various languages (and more literature about *HP* and 'almost *HP*' pocket calculators of 1972 to today). He put them at *https://literature.hpcalc.org/* from where you can download them for free. But take care: copyright is not always observed there.

The picture is a clip of a work by *Gerald Eichstädt* based on data sent by *Juno* in 2016.

P/R turns your *WP43* to *program entry mode*. The display will change in this course – simply don't bother for the time being.

GTO . . goes to the location where free *program memory* space begins.

LBL T is the *opening* step of your program (labelled **T** for total surface – for T just press F6 seeing the letter T displayed above it).

ENTER↑
×
π
×

These keys are the same you pressed above to solve this problem manually.

RTN is the *closing* step of your program. Finally,

P/R returns from *program entry mode* to *run mode*.

Any straight *WP43* program will begin with LBL and close with RTN (or END), framing the keystrokes needed for solving the corresponding problem manually.

Running the program: Now all you have to do to calculate the area of any sphere is keying in the value for its diameter and press XEQ T meaning 'execute program **T**'. Then the sequence of keystrokes you recorded is automatically executed by the calculator, giving you the same answer you would have obtained manually:

For instance, to calculate the surface area of *Ganymede*, press

5 2 6 2 — 5 262 — *Ganymede's* diameter

XEQ T — 86 986 440.6 — its surface area as you know it from above; so you have verified your routine works properly.

With the program you have recorded, you can now calculate the respective surface area of any of *Jupiter's* moons – in fact, of any sphere – using its diameter. You have only to leave the calculator in *run mode* and key in the diameter of each sphere that you wish to compute, and then press XEQ T.

For instance, to compute the surface area of *Jupiter's* moon *Io* with a diameter of 3643 km:

3 6 4 3 — 3 643 — *Io's* diameter

XEQ T — 41 693 486.7 — its surface area;

3 1 2 2 — 3 122 — *Europa's* diameter

XEQ T — 30 620 739.2 — its surface area;

4 8 2 1	4 821_	*Callisto's* diameter
XEQ T	73 017 025.3	its surface area; etc.

Programming your *WP43* is *that* easy! It remembers a series of key-strokes and then executes them automatically when you press XEQ ... [23]

There is no need memorizing any formula you keyed in once – your *WP43* can remember it for you (and provides space for hundreds more). You can even define shortcuts to your favorite formulas by customizing its *user interface* (see *Section 7*) following your very personal requirements and wishes.

The early portions of this handbook show you how easy it is to manually use the power of your *WP43*; while in *Section 4: Programming* you will find a complete guide to *WP43* calculator programming. – … you will want to take a good look at this handbook: It explains the unique *HP* logic system that makes simple answers out of complex problems, and *WP43* features that make programming painless. When you see the simple power of your *WP43*, you'll become an apostle just as have some twenty million *RPN* calculator owners before you.

How to Enter Numbers (and How to Edit Them)

Numeric entry is as easy as typing. Press 1 2 3 4 5 . 6 7 8 9 in *SDC* and you will see

12 345.678 9_

Note the narrow gap inserted automatically after each group of 3 digits to ease reading. You may enter up to 43 digits for one number. As long as input is open, any digit mistyped may be erased by ← and replaced (when no digit is left, your *WP43* will return to its *state* as it was before you began this input).

[23] Program **T** as recorded above is very short and simple: its center part consists of 5 keystrokes only (ENTER↑ × f π ×). You may store far, far more keystrokes in *program memory* – the general procedure of recording and running programs will remain unchanged; only the center parts of programs will grow.

For entering a negative number like −7.5, type 7 +/− . 5 or 7 . +/− 5 or 7 . 5 +/−: pressing +/− flips the sign of the open input. Only negative signs will be displayed.

For a huge figure like the age of our universe in *years*, just type 1 3 . 8 E 9. It is echoed

13.8×10^{9}

i.e. 13.8 *billion*[24] *years* in scientific '*mantissa plus exponent*' format. E opens entry for an exponent of 10 – your *WP43* translates this short-cut into a naturally readable display.

Input is closed and released for interpretation by a command – e.g., by ENTER↑. Here, ENTER↑ will change the display to the equivalent

13 800 000 000.

The closed number is shown right adjusted and also duplicated by ENTER↑ (compare p. 22).

Really tiny values such as a typical diameter of an atom (i.e. $^{1}/_{10\,000\,000\,000}$ *meter*) are entered in analogy:

E +/− 1 0 is echoed $1.\times10^{-10}$

Closed, this will be displayed as 0.000 000 000 1

➡ There's no need to type 1 . E +/− 1 0 here: as long as no digit is heading E, **1** is assumed for the mantissa. And pressing +/− after E flips the sign of the exponent – if you want to flip the sign of the mantissa, press +/− before E or after closing the entire input.[25]

The entire bottom row in display may be dropped by DROP↓ at any time;[26]

[24] *TN*: A *billion* for the *USA*. Many other people call this a *milliard* – a *billion* is 10^{12} here. Thus, exponents of 10 (*scientific* format) are actually a better notation since unambiguous. But businesspeople and engineers don't like it for different reasons.

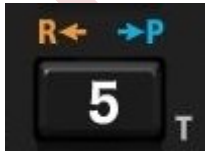
[25] If you prefer output in *scientific* format as above, *Section 2* will show you how to get it.

[26] R↓ will remove the bottom row as well, even by a single keystroke – but it will not be dropped here. See *Looking Closer at the Automatic Stack* below for the differences.

subsequent input will fill this row again, pushing old content up (→ *ReMA H* for reasons). A closed bottom row may be cleared by ⬅ at once, resetting it to 0. – subsequent input will overwrite this zero then.

How to Enter and Execute Commands

For calling any operation you wish executed, just enter the keystrokes to access its label (→ *21*). Pending input will be echoed at left end of the top numeric row on your screen until the command is completed and can be executed. Therein, pending 🟧 will be echoed by ᶠ and 🟦 by ᵍ, if applicable;[27] these symbols will be replaced by the *name* of the command called as soon as it can be decoded.



For many commands, ᶠ or ᵍ will be the only echo you will see since the next keystroke may well terminate entry already, call the command, execute it, and display its result, all done in a fraction of a *second* (as observed with 🟦 →P above).

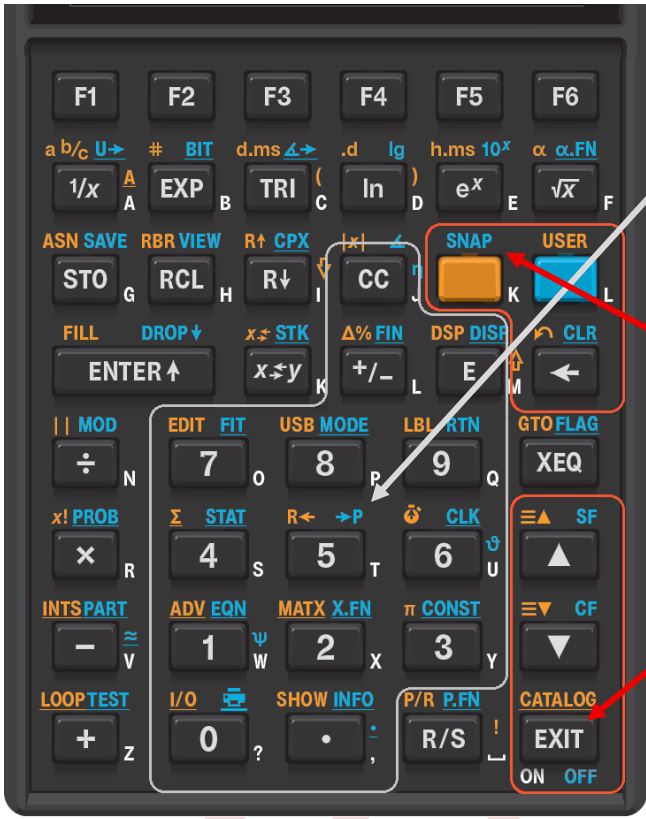
There are, however, also commands requiring parameters trailing – they will hence stay on the screen until these are entered. **STO** and **RCL** are of this kind, and there are many more (→ *64ff*).

🟧, 🟦, and the *menus* in particular allow for easily accessing a multiple of the primary labels the keyboard of your *WP43* can take.

How the Keyboard is Organized

You might have noticed already that the labels on your *WP43* are grouped according to their purposes. Beyond the four basic arithmetic operations [+], [−], [×], and [/], six sets of four or more keys are provided as pictured on the next two pages:[28]

[27] If you pressed 🟧 or 🟦 erroneously, just press the same key once more to cancel it. And if you inadvertently executed a command you did not want to, press 🟧 [↶] to undo this last command immediately (→ *Error Recovery* further below). Mistakes and errors can be undone most easily on your *WP43*.



Keys for numeric input (→ *26ff* and *OMS 3*).

General navigation and control functions like SNAP for screenshots, USER for *user mode*, ⬅ for deleting, ↶ for undoing the last command, ▲ and ▼ for browsing, and EXIT for escaping.

[28] The division key on the *WP43* calculator is labelled ÷ for sake of tradition only – although a printed ÷ looks very similar to + (check vintage program listings!) and may irritate a large group of users initially (actually *ISO 80000-2* recommends not to use ÷ in print at all). Instead, this key could read : as well (visually unambiguous at least) – though this had upset another large group of users forced to learn something new.

As an internationally acceptable compromise, we chose and print / in the manuals whenever we talk about the division key. It is displayed as / in *WP43* programs and equations for sake of clarity as well.

This /, however, dashed against rigid resistance of *SwissMicros* although *HP* had used it together with a multiplication star on the *HP-71B, -38G,* and earlier desktop calculators already. Hence, please excuse the ÷ on your *WP43*.

For multiplication, × is printed on the calculator and in the manuals although this causes ambiguities in *vector* calculus (→ *OMS 3*).

Users preferring • and : over × and ÷ for good reasons can achieve this easily using a black permanent marker (featuring a finer tip than the one taken for the rope in the very first example above).

Softkeys calling *items* from the *menu* displayed above (see next chapter).

Modes, *DTs*, and 'common' trans-cendental functions (the trigonometric func-tions **sin**, **cos**, **tan**, and their inverses are collected in TRI as displayed here).

Stack and register operations.

Functions for programming and calling routines, like XEQ for executing.

Find the basic functionalities of all primary keys listed below, top left to bottom right on the keyboard. See the pages mentioned for details about these functions.

➡ **The lowest numeric row displayed shows x, the next y** (*Part 2* of this *Section* will give the reasons).

1/x Inverts x (→ 36 for an example).

EXP Opens a *menu* containing exponential functions like y^x, 2^x, x^2, x^3, roots, logarithms, and hyperbolics (→ 33).

TRI Opens a *menu* of trigonometric and hyperbolic functions and their inverses (→ above and pp. 95f and 132ff).

[ln]	Returns the natural logarithm
[e^x]	Raises Euler's ***e*** to the power
[$\sqrt{x}$]	Returns the square root

… of x, → 36, 81ff, and 86ff.

[STO]	Stores (copies) x in the destination specified (→ 57ff).
[RCL]	Recalls (copies) an object from the source specified into **X** (→ 49 and 57ff).
[R↓]	Rolls the *stack* contents 1 *level* down (→ 43).
[CC]	Composes, cuts, or converts complex numbers (→ 152ff).
[🟨]	*Prefix* to access golden labels. Press [🟨] twice to cancel it.
[🟦]	*Prefix* to access blue labels. Press [🟦] twice to cancel it.

[ENTER↑]	Context sensitive key, → 26f and 339.
[x⇄y]	Swaps the contents of the bottom 2 rows (→ 43).
[+/−]	If pressed during input of a mantissa or exponent changes its sign (→ 26f). Else multiplies x times **−1**.
[E]	Allows entering an exponent of 10 for convenient input of very big or very small numbers (→ 26f).
[⬅]	Context sensitive key, → 26f and 342.

[/]	If there is an open question like `Are you sure?`, enters N for denial. Else divides y by x (→29, f. 28).
[7], [8], [9], [4], [5], [6], [1], [2], [0]	Enter the respective digit.
[XEQ]	If there is an open question like `Are you sure?`, confirms it; else, if in *RUM*, calls the routine carrying the label specified and starts executing it (→ 218ff); else (in *PEM*) inserts a call to the subroutine carrying the label specified (→ 227f).

[×]	Multiplies y times x.

▲ Context sensitive key, → 34 and 341.

− Subtracts x from y.

3 If there is an open question like `Are you sure?`, enters Y for 'yes'. Else enters the digit **3**.

▼ Context sensitive key, → 34 and 341.

+ Adds x to y.

. Enters a decimal radix mark in numeric input (→ 26).
If pressed twice in numeric input, allows for entering a fraction (→ 75f and 128ff).
In *RoV* addressing, . heads a local address (→ 61ff).

R/S Context sensitive key, → *OMS 4* and pp. 341ff.

EXIT/ON Context sensitive key, → 34f and 340.

And here are the main meanings of the 🟨-shifted labels in the 2nd row of keys, since calling them will lead you to various *data types* your *WP43* will process for you (covered in *OMS 3*):

a b/c	Starts *fraction* display (→ 128ff)
#	Leads *base* input for *short integers* (→ 185ff)
d.ms	Trails sexagesimal input of *angles* (→ 142ff)
.d	Trails input of *dates* (→ 202ff)
h.ms	Trails input of *times* (→ 205ff)
α	Opens alphanumeric input mode (→ 208ff)

There may be further functionalities assigned to above keys not printed here, depending on particular operands: they are covered in App. 1.

Menus – Items à la carte

Your *WP43* features 786 operations, far too many for its keyboard. Hence, most of them live in *menus*. Beyond commands, also applications,

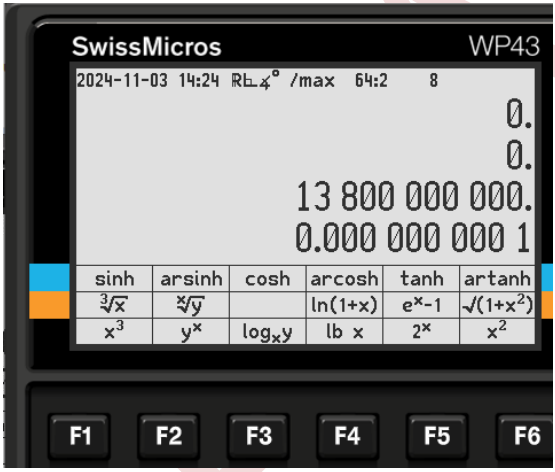
browsers, constants, conversions, routines, *submenus,* digits, symbols (a.k.a. characters), *system flags*, or *variables* defined may be stored in *menus*: collectively, we call them *menu items* or simply *items*.

You find 35 *menus* on the keyboard, keeping it relatively tidy.[29] Open any of them by accessing its label (→ 21) – this will cause the lower third of the screen (called the *menu section*) displaying the respective *menu view*. For a *menu* called the first time, its top *view* will appear, comprising up to 18 *items*:

- up to 6 assigned to the unshifted keys F1 - F6,
- up to 6 to the 🟧-shifted, and
- up to 6 to the 🟦-shifted, all readily accessible for you now.

Example:

Press EXP and EXP will open showing the functions stored therein as pictured. As long as this *menu view* is displayed, simply press, e.g., ...



- F2 for executing y^x,
- 🟧 F2 for $\sqrt[x]{y}$ (note the golden stripes framing this row in display),[30]
- 🟦 F3 for cosh, i.e. the hyperbolic cosine (note the blue stripes).
- If you press 🟧 F3, nothing will happen since no label is displayed there – no operation is linked to this location.

Saving print space in the manuals, we may as well write, e.g., cosh for a function in the blue row, $\sqrt[x]{y}$ for a function in the golden row, and y^x for a function in the unshifted row.

[29] Just EXP and TRI are not underlined, for space reasons, although they call *menus*. All *menu names*, however, are stored without underlines, hence they are also displayed this way. Find an alphabetical list of all menus provided in App. 1.

[30] These 'Ukrainian' stripes are self-explanatory, rendering cursor or display gymnastics obsolete in this matter.

A predefined *menu* may comprise more *views* than just one – this will be indicated by a dashed line above the *menu section*. Whenever such a *multi-view menu* shows up, ▲ will advance to its next *view* and ▼ will return to the previous one, changing the labels displayed. Since *multi-view menus* are circular, also pressing ▲ (or ▼) repeatedly will return to the first *view* after all others were displayed (thus, for a *menu* containing just 2 *views*, both ▲ and ▼ will show the next *view*).

> **Every *menu view* will stay on screen** granting direct access to all its *items* displayed **until you leave it** (by ▼ or ▲, if applicable, by EXIT, or by calling another *menu*).

Catalogs are special *menus* with their *items* sorted alphabetically – they may be quickly searched this way (see *ReMS 2* for how to do this, e.g., in CATALOG'FCNS).[31] Thus, there are various possibilities to call a *cataloged item*, hence we will refer to it using the notation 〔item〕. Picking an *item* from a *catalog* will exit this *catalog* automatically.[32]

We will use the same notation referring to an arbitrary 〔item〕 whose precise location in its *menu* is irrelevant at the moment. In all other cases we will use the background colors introduced above from here on to indicate the access path to a specific *menu item*. Note *submenu* labels are displayed **inverted** in *menu views* (→ f. 29).

Pressing EXIT in any (*sub-*) *menu* will return to the (*sub-*) *menu* you called before, if applicable. The last 4 (*sub-*) *menus* you called are preserved on a *menu stack* therefore.[33]

Before demonstrating the operation of your *WP43* in more detail, let's return to our introductory problem solving examples for 4 general remarks:

1. We presume you have graduated from an *US High School* at minimum, passed *Abitur*, *Matura*, or an equivalent graduation. Thus,

[31] This catalog contains all functions – searching it you will always find any function if you just remember its first characters but forgot in which particular *menu* it lives.

[32] … with 1 exception, → *OMS 3*.

[33] Holding EXIT depressed for > 1 *second* will call **EXITALL**, exiting and closing all open predefined *menus*, so MyMENU will appear usually (→ *OMS 7*).

we will not explain basic mathematical rules and concepts within these manuals. Please turn to respective textbooks.

2. As long as you stay with 1 coherent set of units while calculating reasonably you will get meaningful results within this set.[34] Thus, it were a waste of your time and energy to enter any units and pull them through your calculations. Should you need to convert special inputs into common units or require results expressed in particular or local units, however, U→ and ⍺→ are ready to help (→ *304ff*).

3. Although you entered just *integer numbers* for both edges of the little patch of land on pp. *22f*, your *WP43* calculated the diagonal using *real numbers* (see below). This also allows for decimals (or fractions if needed) in input and output.[35]

4. In five decades of scientific pocket computing, a wealth of examples has been published – more and better than we can ever invent. We did not intend to copy all of them; instead, we recommend the media mentioned in f. 22 (on p. *24*) and 36 here. All computations described there for any scientific calculator can be solved on your *WP43* – distinctly faster and often in a more elegant way. Nevertheless, we included over 200 new and quoted vintage **examples** in this manual to support you getting accustomed to your new power tool.

[34] A quick unit analysis (a.k.a. dimension check) is recommended before starting a calculation. *SI* is the largest coherent set of units available on this planet.

Working with more than one set of units carries risks.

TN: Look at this very expensive (though absolutely unintentional) lithobraking experiment: *https://www.latimes.com/archives/la-xpm-1999-oct-01-mn-17288-story.html*. People worldwide were chuckling about that mishap and its cause – certainly not nice.

[35] Your *WP43* features many *data types* more – they will be introduced in *OMS 3*.

[36] Almost all user guides, handbooks, and manuals printed for *HP* calculators in 2 decades, from the *HP-9100A* of 1968 (*HP*'s first desktop calculator, without a single *IC* but with a small cathode ray tube built in for display, → *ReMA H*) until 1990 are still available at low cost in a 14 GB media package distributed by *David Hicks'* online *Museum of HP Calculators* (→ *http://www.hpmuseum.org/cd/cddesc.htm*).

Problem Solving, Part 2: Elementary Stack Mechanics

Most commands of your *WP43* are mathematical functions or operations taking and returning *real numbers* (a.k.a. *reals*) like 3.141 592 or 0.125 or -5.67×10^{-8} (note *integers* like 3 or 12 345 678 or −12 321, as well as *fractions* like $^{4}/_{5}$ or $^{137}/_{7}$ are mere subsets of *reals*).

Any command you choose may operate on 1, 2, 3, or 4 numbers at once to generate a result. In spite of the hundreds of functions available, you will find them simple to operate by a single, all-encompassing rule:

When you press a function key, your *WP43* will execute the operation assigned to it immediately.[37]

***Monadic* functions** operate on 1 number (x) only. You find the reciprocal [1/x], the logarithms [ln] and [lg], the exponentials [e^x] and [10^x], the square root [$\sqrt{x}$] as well as [|x|] (always returning a positive number), [+/−] (→ *31*), and the factorial [x!] on the keyboard.

Examples to be executed consecutively, starting in *SDC*:

6 4 [1/x]	0.015 625	
[ln]	−4.158 883 083 359 672	,
[|x|]	4.158 883 083 359 672	,
[lg]	0.618 976 711 428 782	,
[10^x]	4.158 883 083 359 672	,
[e^x]	64.	,
[$\sqrt{x}$]	8.	,
[x!]	40 320.	= 8×7×6×5×4×3×2×1.

Each *monadic* function replaces x by the respective result $f(x)$ (=$x!$ in last case). Everything else will stay as it was. – Please check the *IOI* for the many more *monadic* functions provided on your *WP43*.

[37] If it requires trailing parameters it will execute with parameter input completed.

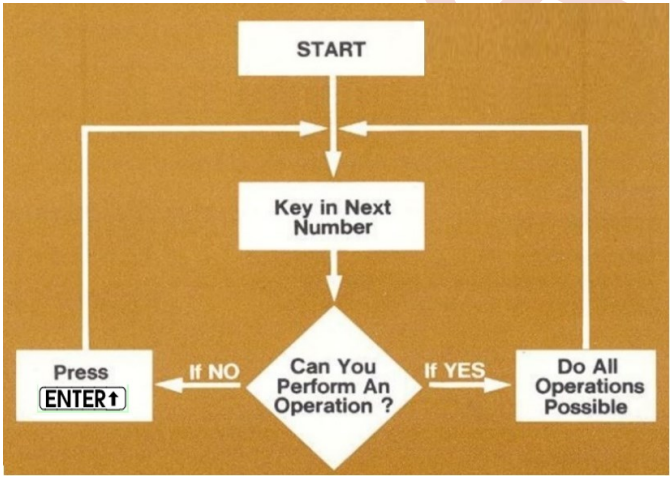
***Dyadic* functions**, on the other hand, operate on 2 numbers (x and y) and return one. Think of the 4 basic arithmetic operators **+** and **−**, **×** and **/**.

Example (in *SDC*):
Assume withdrawing 56.70 $ from an account of 1234 $. What will remain?

Solving such a problem easily may go like this (# abbreviates *number*):

On a piece of paper:		→	On your *WP43*:	
Write down the 1st #	1234.00		Enter the 1st #	[1][2][3][4] 1 234_
Start a new line and write down the 2nd #	56.70		Close input	[ENTER↑]
Draw a line below.			Key in the 2nd #	[5][6][.][7] 56.7_
Subtract:	1 1 1 1177.30		Subtract:	[−] 1 177.3

This is the essence of *RPN*: **Provide the required operands, then execute the requested operation by accessing the corresponding label.**

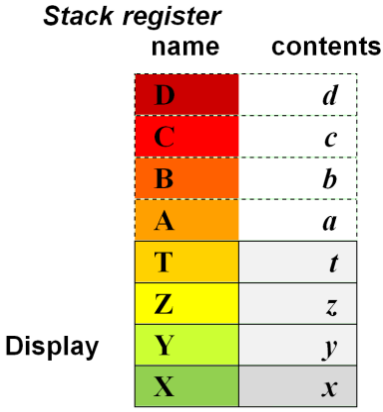


HP explained *RPN* using this scheme. And a major advantage of *RPN* compared to other calculator operating systems we know is it sticks to this basic rule for all functions, regardless of the kind and number of objects they operate on.[38]

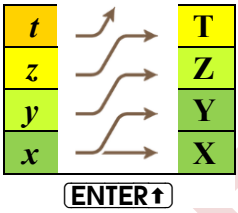
[38] This way of writing operations is called *postfix* notation since the operator is entered after the operands (→ f. 1). It suits electronic calculating very well (and eases work for human brains as well) – see more below. (The picture is taken from *HP's* brochure '[ENTER↑] *vs.* [=]' of 1974.)

TN: Some might claim said rule strictly holds for *RPL* only (*RPL* was developed from *RPN* in the 1980s, → *ReM*). Maybe they are even right. In our opinion, however, *RPL* strains the *postfix* principle beyond the pain barrier – also of many scientists

As paper holds the your operands while you are calculating manually, a space holding them is also required on your *WP43*. The ***stack*** does this job, and this is its *raison d'être*: It is a pile of linked *registers*, workspace for your calculations. Bottom up, these *stack registers* are traditionally called **X**, **Y**, **Z**, and **T** in a *4-level stack* (*4LS*), optionally followed by **A**, **B**, **C**, and **D** on your *WP43*.[39]



Your input is entered in **X** always, and at least ***x*** is always displayed in *RUM* – ***y***, ***z***, and ***t*** may be, so you can see the actual contents of the bottom 4 *stack registers* at a glance if you want.



ENTER↑ separates 2 input numbers by closing ***x*** and copying it into **Y** (→ 22). The contents of the upper *stack registers* are lifted out of the way before. For a *4LS*, ***z*** is moved into **T**, then ***y*** into **Z**, then ***x*** copied into **Y**. Then **X** can take a new input without loss of information.[40]

This is the functionality of ENTER↑ as implemented in the *HP-35* of 1972 (→ *ReMA H* for reasons) and kept until the *HP-42S* ceased production in 1995. ENTER↑ affects all *stack registers*, and the previous content of the top *register* is lost. It is often said ENTER↑ 'pushes ***x*** on the *stack*'.[41]

Let's look at our account example again, putting it in a *stack diagram*:

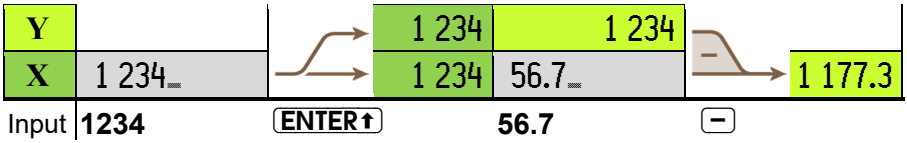
---

and engineers, in particular when using a calculator casually. Thus, we stick to classic *RPN* on *WP43* as we did on *WP 34S* and *31S*.

[39] This optional *8-level stack* was invented by *Paul* and me and launched with *WP 34S* in 2011. *WP 31S* features it as well. See below for its advantages.

[40] Arbitrary data from earlier work may be present in the *registers* **Y**, **Z**, etc. above. As you see, these are pushed out of the way so they will not affect the current calculation. Hence, we will not print such data unless relevant for the calculation performed. We will also use plain bold black print denoting numeric input from here on for space reasons.

[41] *TN:* Actually, it pushes ***x*** under it. The original *stack* model was built with **X** on top of **Y**, **Z**, etc. (cf. German *Kellerspeicher* for *stack*); but *HP* draw it like above always. – There are other dialects of *RPN* but this is the classic standard (→ *ReM*).



After entering the 2nd number (**56.7** = x), pressing − subtracts x from the 1st number (**1234** = y) and puts $f(x, y) = y - x$ = **1177.3** into X. This simple method applies to all *dyadic* functions on your *WP43*:

Put the operands on the *stack*,
then execute the operation $f(x, y)$,
and its results will be displayed.[42]

Also a few ***triadic* functions** are featured. Executing such a function replaces x by $f(x, y, z)$ (→ *44* for an instance). All *triadic* functions provided on your *WP43* are found in *menus*.

SLVC and ⇶ (SHUFFLE) are the only functions operating on all bottom **4** stack registers at once. See *OMS 5* for SLVC, look up the *IOI* for ⇶.

Furthermore, there are some functions operating on 1 number but returning 2 (like DECOMP) or 3 (like DATE→). Other operations don't consume any *stack* input at all but may return 1, 2, or 3 objects (like RCL, SUM, or L.R.); then these objects will be pushed on the *stack*, taking 1 *register* each (→ *49*).

Further ways of parameter handling will be shown in *OMS 2* and *3*.

Problem Solving without any Formula

The following little real-life problem shows how *RPN* is useful and fast in solving step-by-step problems where there is no formula, and where the next calculation depends on the result of the previous one. [43]

[42] This eliminates the need for = entirely – also beyond dyadic operations, since this rule applies by analogy there as well.

[43] This example is courtesy of *Gene Wright*.
TN: It takes place in the *USA* – see the setting and the vertical feet.

Example (in *SDC*):

You need to fix a tile that has fallen off your roof. The roof is 28 *feet* up, and you have a stepladder[44] that is 29 *feet* long. You could also borrow your neighbor's 38 *foot* ladder. Is either ladder good for the job? You could try leaning each one in turn against the roof and seeing which is better, but it's raining heavily, so why not work it out first on your *WP43* by seeing what angle each ladder will make with the vertical when you lean it against the roof?

Solution:

We won't need more than 2 decimals for the following calculation, so enter

DISP FIX 2 .

First try it for your own ladder. Enter its length:

29 ENTER↑ .

Then enter the height and divide so you can get the angle,

28 /

You get 1.04. That can't be right; *sines* and *cosines* should be less than 1.

Use ↶ to undo and get the numbers back, then

x⇄y to swap them. Now press

/ to divide them again, in the right order now, and see 0.97.

(Actually, simply pressing 1/x would get the same result, making the correction more quickly.)

Anyway, this is the sine of the angle between the ladder and the vertical. Or is it the *cosine*? Press

TRI **arcsin** and see the answer, 74.91 *degrees*. No, that is not right, it must be the *arc cosine* that you need. Press

↶ to undo the *arc sine*, then

arccos returning 15.09 .

Your ladder would be only 15 *degrees* away from the vertical. That is uncomfortably steep (and it may even tip over as you stand on it). So try the same calculation for your neighbor's 38 *foot* ladder. Type

29 ENTER↑

38 /

arccos.

[44] *TNG*: *Stepladder* = *Stehleiter* – aber hat *Gene* das wirklich gemeint? S.u.

This time the answer is 40.26 *degrees*. The ladder would be at a rather shallow angle and might slip away as you stand on it.[45]

Neither ladder is really suitable. Maybe you should ask some other neighbors if they would lend you a ladder with a better length. What would be a good length?

30° would probably still be too much; 20° would be about right.

So type the height again and divide by the *cosine* of 20°.

28 ENTER↑

20 cos

/

That gives 29.80. A 30 *foot* long ladder would be almost ideal – if a neighbor has got one.

The argument[46] in school calculators is "*Algebraic lets you enter the equation just as you see it on paper!*" Well, for one thing, a lot of programming work falls outside of actual equations. For another, even in the field of calculators, in real life we don't usually *have* an equation in front of us. I think through it, "Let's see – I need ***A***, and then square that; now take ***B***, multiply it by ***C*** and add that result to the earlier one. Now I need ***D*** raised to the power of ***E***, and divide the earlier result by that. Done. Oops, no, I still need to take the log of that ..." That's the way much of real engineering is.

It's only in school that you get canned problems and have the equation in the book and you're supposed to just drop the numbers in the chute and turn the crank – then we wonder why we get graduates who passed all their classes and yet show a disconnect between that knowledge and any understanding of what the problem is in their circuit on the workbench! I've hired a lot of electronics technicians and a few engineers, and I gave all the applicants a circuit-analysis test with a dozen *simple* problems. *Not one* of them ever got it all correct. It was kind of frustrating.

Algebraic tells you what you get. *RPN* tells you how you get there.

[45] *TN*: No, don't move it closer to the wall! Such maneuvers would ruin this nice example. And the use of a stepladder remains mysterious here as well.

[46] This quotation is from *Garth Wilson*, owner of *Wilson Mines Co.* (→ *WilsonMinesCo.com*).

Looking Closer at the Automatic Stack

Beyond [ENTER↑], there are [x⇄y], [STO], [RCL], [R↓], [R↑], [x⇄], and [VIEW] dealing with *registers* (not only on the *stack*). You find these 8 – together with the new [RBR], [FILL], and [DROP↓], as well as the *menu* STK – all within this small area of the keyboard.

Your *WP43* features even more *stack* and *register* commands: `CLX`, `CLSTK`, `CLREGS`; `STOSTK`, `STOCFG`, `STOIJ`, `STOEL`, and the corresponding recalls; more flavors of `STO` and `RCL`, furthermore `DROPy`, `y⇄`, `z⇄`, `t⇄`, and `⇄`. The first 3 are found in CLR (on 🟦-shifted ⬅), the last 5 in STK.

And it offers you an *8LS* (as *WP 34S* does since 2011 and *WP 31S* since 2014) instead of the *4LS* of 1972. Look at the *status bar* at top of the *LCD* (→ *21*): it indicates an **8** at its right side in *SDC*.[47]

Thus, the fate of *stack* contents will depend on the operation executed and *stack* size at execution time. Operations on the *4LS* work as known from *HP's RPN* calculators since the *HP-45*. On the *8LS* of your *WP43*, everything works in analogy – just granting you more workspace for your intermediate results. Benefits will come for free. They will become evident in the following.

Find on the next 2 pages what [ENTER↑], [FILL], [DROP↓], [DROPy], [x⇄y], [R↓], [R↑], and further representative functions do in detail on *stacks* of either size. Then you will also know why [ENTER↑] and [R↑] show arrows pointing up while [R↓] and [DROP↓] point down.

- [0] [FILL] clears the entire *stack* most easily. Though also `CLSTK` is provided (in CLR) for sake of backward compatibility, program space economy, and speed of execution.
- Depending on the problems you solve and the way you proceed, you may find that x and y should be swapped before executing, e.g., [−], [/], or [y^x] sometimes – [x⇄y] will do this for you.

[47] Should you ever want to go back to the old *4LS*, [CF] `SYS.FL` [SSIZE8] will do.

	Stack register	Assumed initial contents	*Stack* contents after executing … ENTER↑	FILL	DROP↓	STK DROPy	R↓	R↑	x≷y
With 8 *stack registers*	D	$d = 8.$	7.	1.1	8.	8.	1.1	7.	8.
	C	$c = 7.$	6.	1.1	8.	8.	8.	6.	7.
	B	$b = 6.$	5.	1.1	7.	7.	7.	5.	6.
	A	$a = 5.$	4.	1.1	6.	6.	6.	4.	5.
	T	$t = 4.$	3.	1.1	5.	5.	5.	3.	4.
	Z	$z = 3.$	2.	1.1	4.	4.	4.	2.	3.
	Y	$y = 2.$	1.1	1.1	3.	3.	3.	1.1	1.1
	X	$x = 1.1$	1.1	1.1	2.	1.1	2.	8.	2.
With 4 only	T	$t = 4.$	3.	1.1	4.	4.	1.1	3.	4.
	Z	$z = 3.$	2.	1.1	4.	4.	4.	2.	3.
	Y	$y = 2.$	1.1	1.1	3.	3.	3.	1.1	1.1
	X	$x = 1.1$	1.1	1.1	2.	1.1	2.	4.	2.

- R↓ (and R↑) once came handy for reviewing *stack registers* else unseen.[48] On your *WP43*, feel free to use RBR instead, displaying 10 *registers* at once without disturbing the *stack* at all (→ 289ff).

[48] Most pocket calculators until today display only x, hence the pressing need for R↓ for *stack* inspection. Actually, ENTER↑, x≷y, and R↓ are the *stack* operations found on each and every *RPN* pocket calculator made since the *HP-35*. Only the very first *financial pocket problem solver* developed, the *HP-80* (the immediate successor of *HP-35*), showed SAVE↑ instead of ENTER↑, presumably to take businessmen (in 1972!) where *HP* expected them to be.

For a *4LS* on a calculator able to display all 4 *stack registers* permanently like your

Stack register	Assumed initial contents	*Stack* contents after executing …					
		RCL L [49]	STAT s [50]	EXP x² [51]	+ [52]	CLK →DATE [53]	DATE→ [54]
D	$d = 8.$	7.	6.	8.	8.	8.	6.
C	$c = 7.$	6.	5.	7.	8.	8.	5.
B	$b = 6.$	5.	4.	6.	7.	8.	4.
A	$a = 5.$	4.	3.	5.	6.	7.	3.
T	$t = 4.$	3.	2.	4.	5.	6.	2.
Z	$z = 3.$	2.	1.1	3.	4.	5.	11
Y	$y = 2.$	1.1	s_y	2.	3.	4.	12
X	$x = 1.1$	*last x*	s_x	1.21	3.1	0003-02-01	30
T	$t = 4.$	3.	2.	4.	4.	4.	2.
Z	$z = 3.$	2.	1.1	3.	4.	4.	11
Y	$y = 2.$	1.1	s_y	2.	3.	4.	12
X	$x = 1.1$	*last x*	s_x	1.21	3.1	0003-02-01	30

WP43, reviewing ***y***, ***z***, and ***t*** turns pointless actually.

[49] RCL L represents an arbitrary function pushing 1 object on the *stack*.

[50] s represents an arbitrary function pushing 2 objects on the *stack*.

[51] x² represents an arbitrary *monadic* function.

[52] + represents an arbitrary *dyadic* function.

[53] Assume →DATE being called in *SDC* (i.e. YMD). Here, →DATE represents an arbitrary *triadic* function.

[54] Assume **11.123** or **11-12-30** in **X** initially and YMD here (→ last chapter of this *Section* as well as *OMS 3*).

- For most functions, your *WP43* copies x into $\mathbf{L}$ automatically just before the new function is executed. RCL L recalls this ***last x*** (→ 44) – learn about its benefits on p. 53.

Note the old content of the top *stack register* is lost when ENTER↑ is executed (→ 43). The same applies to RCL. Functions like s or DATE→ will even cost the contents of the top 2 *stack registers* (all these 3 functions are shown on next page). We recommend keeping SSIZE8 set to mitigate the effects of such losses. See the *IOI* for more about the commands mentioned above.

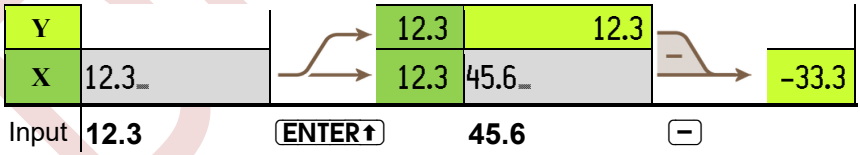
Problem Solving, Part 3: Advanced Stack Mechanics

Using the features and functions introduced above, *RPN* makes chain calculations as easy as can be:[55]

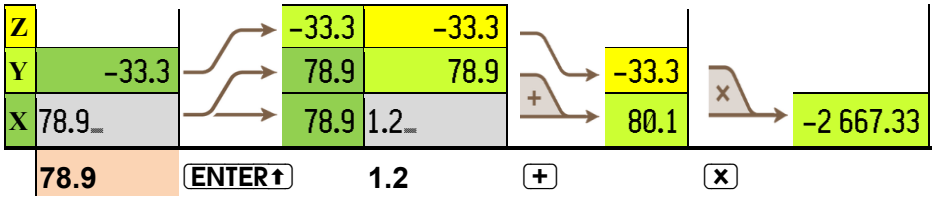
Example:

$$\frac{(12.3 - 45.6)\,(78.9 + 1.2)}{(3.4 - 5.6)^7} = ?$$

This is as a combination of 6 *dyadic* functions: 2 subtractions, 1 addition, 1 multiplication, 1 exponentiation, and 1 division. Starting top left in the formula, this is what will happen on the *stack* of your *WP43* while solving it:



This 1st parenthesis in the numerator is solved exactly as demonstrated in our little account problem above. Proceed to the 2nd parenthesis now:



It is solved like the 1st. But in the 1st step of this sequence, the prior result of 1st parenthesis is lifted automatically into **Y** to avoid overwriting it with the new number keyed in now. This move is called *automatic stack lift* (*ASL*).

> Actually, such an *ASL* (originally called *automatic ENTER*) works as if [ENTER↑] was pressed before **7** here. *ASL* is standard on *RPN* calculators since 1972, saving keystrokes.[56] Remember you need [ENTER↑] just for separating two consecutive input numbers, → 37. In consequence, we need no [(] and can solve problems with a minimum of keystrokes.

After having solved the 2nd parenthesis by pressing [+], the results of both upper parentheses were on the *stack* in **X** and **Y** – well prepared for the multiplication to complete the numerator. Thus, we just did it.

Now start calculating the denominator – once again the intermediate result is lifted automatically in the 1st step:

	–2 667.33	–2 667.33		–2 667.33		
–2 667.33	3.4	3.4	–2 667.33	–2.2	–2 667.33	
3.4	3.4	5.6	–2.2	7	–249.4…	10.693…
3.4	[ENTER↑]	**5.6**	[−]	**7**	[EXP] y^x	[/]

Note the *ASL* when entering **7** affects 2 intermediate results here. Thus, everything is well prepared for the exponentiation in the penultimate step and the final division of the numerator ***y*** by the denominator ***x***. Voilà!

---

[55] Personal side note: Above titles *Elementary Stack Mechanics* and *Advanced Stack Mechanics* go back to a textbook titled *Elementary Quantum Mechanics* I came across in my studies decades ago. It goes without saying that book was as far from being elementary as a theoretical physicist could be.

[56] Due to *ASL*, there is no need for clearing your *WP43* before starting a new calculation – old data are just lifted (pushed) out of the way (as shown above) when new input is pushed on the *stack*. *ASL* affects all *stack registers* (as **ENTER↑** does) and the previous content of the top *register* gets lost again (→ *ReMA H* for background information). Almost all commands enable *ASL* – with 10 exceptions: **CLX**, **ENTER↑**, **Σ+**, **Σ-**, and 6 *matrix* editing commands. Some reasoning for the first 4:

- **CLX** (called by [CLX], or ⬅ for closed ***x***) clears **X** to make room for a corrected value instead of the one deleted – and you don't want a useless zero on the *stack*.
- After [ENTER↑], you usually want to key in the next input number.
- **Σ+** and **Σ-** are designed for entry of measured, counted, or statistical data – they were not meant to be mixed with normal calculating operations. See the chapters about statistical functions in *OMS 2*.

Following this example, you have observed the 5 most popular *dyadic* functions in action: [+], [−], [×], [/], and [y^x]. Your *WP43* provides many more (please see the *IOI*).

➡ The contents of the *stack* drop whenever a *dyadic* (or *triadic*) function is executed. Like *ASL,* also this *automatic stack drop* affects all *stack registers* (→ *44*, and → *54f* for its benefits).

Using the *stack* as demonstrated above, *RPN* completely eliminates the need for any [(], [)], and [=] keys! See the following **example** with a slightly more elaborate formula than above. Find below a way for solving it step by step, and the corresponding *stack* diagrams.

$$2+\sqrt{\frac{1+\left|\left(\frac{30}{7}-7.6\times 0.8\right)^4-\left(\sqrt{5.1}-\frac{6}{5}\right)^2\right|^{0.3}}{\left\{sin\left[\pi\left(\frac{7}{4}-\frac{5}{6}\right)\right]+1.7(6.5+5.9)^{3/7}\right\}^2-3.5}}=\ ?$$

Set [MODE] RAD and start calculating at the red **7** (below, [ENT↑] = [ENTER↑]):

Z						1.75	1.75		
Y		7	7		1.75	5	5	1.75	
X	7_	7	4_	1.75	5_	5	6_	0.83...	0.91...
	7	**ENT↑**	**4**	**/**	**5**	**ENT↑**	**6**	**/**	**−**

								0.25...
				0.25...	0.25...		0.25...	12.4
0.91...			0.25...	6.5	6.5	0.25...	12.4	3
3.14...	2.87...	0.25...	6.5_	6.5	5.9_	12.4	3_	3
π	**×**	**TRI**	**sin**	**6.5**	**ENT↑**	**5.9**	**+**	**3**	**ENT↑**

0.25...								
12.4	0.25...		0.25...					
3	12.4	0.25...	2.94...	0.25...			27.6...	
7_	0.42...	2.94...	1.7_	5.00...	5.25...	27.6...	3.5_	24.1...
7	**/**	**EXP**	**y^x**	**1.7**	**×**	**+**	**x^2**	**3.5**	**−**

This is the solution of the denominator. Let's continue with calculating the numerator now, basically following the same procedure, i.e. calculating inside out (as you would do with pencil and paper):

					24.1...	24.1...			
	24.1...	24.1...		24.1...	6.08	6.08	24.1...		24.1...
24.1...	7.6	7.6	24.1...	6.08	30	30	6.08	24.1...	1.79...
7.6_	7.6	.8_	6.08	30_	30	7_	4.28...	1.79...	4_
7.6	ENT↑	**.8**	×	**30**	ENT↑	**7**	/	−	**4**

		24.1...	24.1...		24.1...	24.1...			
	24.1...	10.3...	10.3...	24.1...	10.3...	10.3...	24.1...	24.1...	
24.1...	10.3...	6	6	10.3...	1.2	1.2	10.3...	10.3...	24.1...
10.3...	6_	6	5_	1.2	5.1_	2.25...	-1.05...	1.12...	9.24...
y^x	**6**	ENT↑	**5**	/	**5.1**	$\sqrt{x}$	−	x^2	−

	24.1...		24.1...						
24.1...	9.24...	24.1...	1.94...	24.1...	2.94...			0.34...	
9.24...	.3_	1.94...	1_	2.94...	24.1...	0.12...	0.34...	2_	2.349...
|x| [57]	**.3**	y^x	**1**	+ [58]	x≷y	/	$\sqrt{x}$	**2**	+

Even solving this formula required only 4 *stack registers*. There are **no pending operations** – each operation is executed individually, one at a time, allowing perfect control of each and every intermediate result. [59]

This is another characteristic advantage of *RPN*. In many real-life applications, intermediate results carry their own value, so further calculations may depend on the numbers you see there (→ *39ff*) – this is called '*exploratory math*' and will definitely occur more frequently in your professional work than evaluating textbook formulas.

[57] You will hardly execute this step manually since you realize immediately x is positive. In an automatic evaluation of such a formula, however, this step may be essential.

[58] **X** contains the result of the numerator here. And the result of the denominator silently traveled into **Y** during all the calculations printed above. In the next step, we swap x and y to arrange the operands for dividing the numerator by the denominator.

[59] Thus, operator precedence is your job. Look up *App. 2* for confirmation or reminder.

Experienced *RPN* calculator users have determined that by starting every problem at its innermost number or parenthesis and working outwards (as you did and do with pencil and paper), you maximize the efficiency and power of your calculator.

Example (continued):
If, instead, you try solving the formula on p. 47 stubbornly from left to right, you will need more keystrokes and more *stack registers*. Try and count yourself.

There may be, however, some problems where 4 *stack registers* will simply not suffice, regardless of the way you tackle with them:

Example (in SDC):

$$\frac{(1+2)\,(9+8)\;+\;(3+4)\,(11-6)}{(5-7)\,(10+12)\;-\;(13+14)\,(15+16)}\;=\;?$$

This highly symmetric formula lacks an unambiguous 'inside', so let's begin with the numerator:

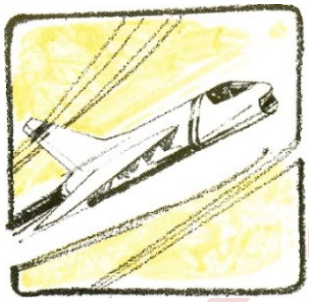
Z						3	3			
Y		1	1		3	9	9	3		51
X	1_	1	2_	3	9_	9	8_	17	51	3_
	1	ENT↑	2	+	9	ENT↑	8	+	×	3

T					51	51				
Z	51	51		51	7	7	51			
Y	3	3	51	7	11	11	7	51		86
X	3	4_	7	11_	11	6_	5	35	86	5_
	ENT↑	4	+	11	ENT↑	6	−	×	+	5

T					86	86				86
Z	86	86		86	−2	−2	86		86	−44
Y	5	5	86	−2	10	10	−2	86	−44	13
X	5	7_	−2	10_	10	12_	22	−44	13_	13
	ENT↑	7	−	10	ENT↑	12	+	×	13	ENT↑

				86	86				
T	86		86	−44	−44	86			
Z	−44	86	−44	27	27	−44	86		
Y	13	−44	27	15	15	27	−44	86	
X	14	27	15_	15	16_	31	837	−881	−0.097 61…
	14	**+**	**15**	**ENT↑**	**16**	**+**	**×**	**−**	**/**

➡ Pressing **ENTER↑** here (in step 4 of this last diagram), a *stack overflow* would have occurred in a *4LS* leading to a loss of data and a completely wrong final result. Focused on calculating, you had hardly noticed that. [60] Overflow probability is low but not negligible in a *4LS* – rather keep SSIZE8 set.



We close with another real-life **example**:

For many years, solving the following formula for the *Mach number* of an airplane as a function of its *calibrated airspeed* (*CAS*) in *knots* (here 350) and *pressure altitude* (*PA*) in *feet* (here 25 500) was used for demonstrating the simplicity and coherence of *RPN*: [61]

$$\sqrt{5\left(\left[\left\{\left(1+0.2\left[\frac{CAS}{661.5}\right]^2\right)^{3.5}-1\right\}\{1-6.875\times10^{-6}\times PA\}^{-5.2656}+1\right]^{0.286}-1\right)}$$

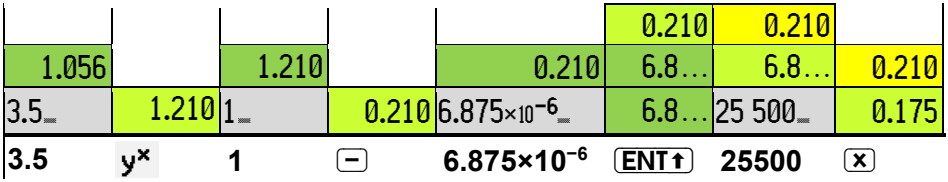
Solve it like this with **DISP** **FIX 3**:

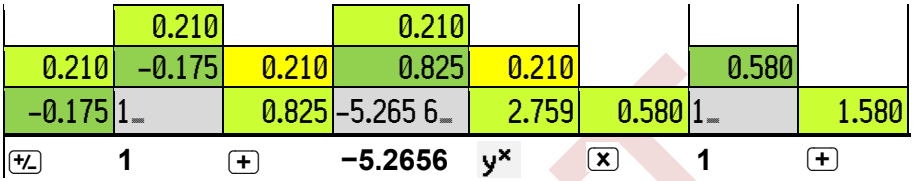
	350	350			0.28		0.056	
350_	350	661.5_	0.529	0.280	0.2_	0.056	1_	1.056
350	**ENT↑**	**661.5**	**/**	**EXP x²**	**.2**	**×**	**1**	**+**

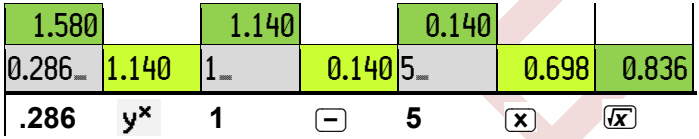
[60] There is (and will be) no warning for stack overflow: even a watchdog on **T** will not help since some special advantages of *RPN* require **T** loaded (→ chapter after next).

[61] *TN*: We quote this *US*-American formula without verification. The medieval *knots* and *feet* survived in western aviation business and navigation, although made obsolete by *SI* for two centuries – one issue Russians did right (→ *OMS 6*).

TN for Europeans: *CAS* does not mean *C·A·S*, nor does *PA* mean *P·A* in that formula.







i.e. 84% of the speed of sound.

As you have seen, the way to solve a problem using *RPN* stays the same regardless of its size and complexity. You are always in control.

While a *stack overflow* may (seldom but silently) happen on a *4LS*, you will be obviously on the safe side with an *8LS*, even dealing with the most advanced mathematical expressions you will meet in your professional life as a scientist or engineer. Promised. The *8LS* warrants absolutely worry-free calculations. Hence it is *startup default* on your *WP43*.[62]

Let's quote a part of the *HP-25 OH* once more, just replacing all strings '*HP-25*' by '*WP43*':

> Now that you've learned how to use the calculator, you can begin to fully appreciate the benefits of the *Hewlett-Packard* logic system. With this system, you enter numbers using a parenthesis-free, unambiguous method called *RPN* (…). It is this unique system that gives you all these calculating advantages whether you're writing keystrokes for a *WP43* program or using the *WP43* under manual control:

[62] Contriving a case leading to *stack overflow* even for an *8LS* is trivial though far off – it will be just a contrived case, no real-world problem. Hence there is no need for more than 8 *stack levels*. A so-called 'infinite' *stack* (as featured in *RPL* and on the *HP Prime*) definitely complicates handling and obstructs some nice shortcuts (→ *54f* and the *ReM*).

- *You never have to work with more than one function at a time.* The *WP43* cuts problems down to size instead of making them more complex.
- *Pressing a function key immediately executes the function.* You work naturally through complicated problems, with fewer key-strokes and less time spent.
- *Intermediate results appear as they are calculated.* There are no "hidden" calculations, and you can check each step as you go.
- *Intermediate results are automatically handled.* You don't have to write down long intermediate answers when you work a problem.
- *Intermediate answers are automatically inserted into the problem on a last-in, first-out basis.* You don't have to remember where they are and then summon them.
- *You can calculate in the same order you do with pencil and paper.* You don't have to think the problem through ahead of time.

RPN takes a few minutes to learn. But you'll be amply rewarded by the ease with which the *WP43* solves the longest, most complex equations. With *RPN*, the investment of a few moments of learning yields a lifetime of mathematical bliss.[63]

And calculations with other objects (to be introduced in *OMS 3*) comply to the same simple rules. Thus, at the bottom line, we recommend:

Keep SSIZE8 set and
let your *WP43* care for the arithmetic
while you care for the mathematics! [64]

[63] For hardcore algebraic calculators: we do support also algebraic equations on the *WP43* – you can enter, edit, and evaluate your favourite equations and expressions – learn how to do this in *OMS 5*.

[64] With the opportunity of an *8LS*, why are there only 4 *stack levels* displayed?

Simple reason: Once accustomed to *RPN*, you know how your data move on the *stack*. Watching *stack* mechanics reliably working exactly as expected carries no valuable information and will become boring or even distracting very soon.

If, however, you feel annoyed by a cluttered screen displaying more than you need, use **DSTACK** (in 2nd view of DISP) to reduce the number of *stack levels* displayed to 3, 2,

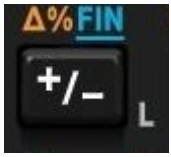
LASTx for Reusing Numbers

Your *WP43* saves x in the special *register* **L** (for **LAST**x) automatically just before a new command is executed.[65] What is your benefit?

Example (from the *HP-15C OH*):
Two close stellar neighbors of Earth are *Rigel Centaurus* [66] (4.3 *light-years* away) and *Sirius* (8.7 *light-years* away). Use the speed of light, c (2.997 925 ×10^8 m/s, or 9.460 536 ×10^{15} m/*year*), to figure the distances to these stars in *meters*.

Solution (with DISP **SCI 01**):

4.3	4.3
ENTER↑	4.3
9.460536 E **15** [67]	9.460 536×10^{15}
×	4.1×10^{16}
8.7	8.7
RCL L [68]	9.5×10^{15}
×	8.2×10^{16}



Result: *Rigel Centaurus* is 4.1×10^{16} m away from us, *Sirius* 8.2×10^{16} m.

---

or even just 1, letting your brains compete with your fellow *RPN* users since 1972. Free space will flow in top down – x will always be shown directly above the *menu section*. See here an exemplary display for **DSTACK 2** – showing just x and y. →

And don't be afraid: multi-line output will be displayed entirely always, regardless of the **DSTACK** setting you choose.

[65] … as all *RPN* calculators did since the *HP-45*. The few commands not loading **L** are mentioned explicitly in the *IOI*. – The picture is a clip of a photo by *Alan Dyer*.

[66] *TN*: This is identical to *Alpha Centauri*. *Rigel* usually means a star in the constellation *Orion*.

[67] *TN*: Comparing the precision of the other inputs, 3 digits had sufficed here (i.e. 9.46×10^{15}). But *HP* wanted to demonstrate the advantage of **LAST**x.

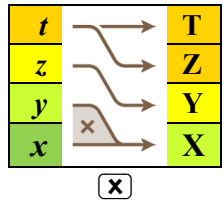
RCL L will often allow for reusing lengthy numbers (mitigating input errors) or recalling intermediate results without the need for storing them explicitly elsewhere.

Top Stack Register Repetition

Whenever a *dyadic* or *triadic* function is executed, the *stack* will drop and the content of its top *register* will be repeated, → 44. You may use this *top stack register repetition* for some nice tricks. See the following compound interest calculation on a *4LS*:[69]

Example:

Assume your bank pays you 3.25% p.a. on an amount of 15 000 US$.[70] What would be your status after 2, 3, 5, and 8 *years*?



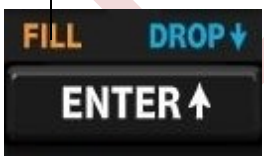
Solution:

Here, you are interested in currency values only, so set the display format to DISP FIX 2, causing the display being rounded to cents:

T		1.03	1.03	1.03	1.03	1.03	1.03
Z		1.03	1.03	1.03	1.03	1.03	1.03
Y		1.03	1.03	1.03	1.03	1.03	1.03
X	1.032 5_	1.03	15 000_	15 990.84	16 510.55	17 601.17	19 373.66

1.0325 FILL **15000** × × × × × × × × ×

after 2 *years* 3 *years* 5 *years* 8 *years*



Each multiplication consumes x and y for the product $x \times y$ put into X, followed by z copied into Y, then t into Z.

---

[68] RCL L is reached by pressing RCL, then +/− – note L printed bottom right of +/−. Allocating a dedicated keyboard label to LASTx (like on *HP-42S* or *DM42*) doesn't pay here since no keystroke will be saved.

[69] *TNG*: *Compound interest* = Zinseszins.

[70] Those were the days, my friend…! Inflation was balancing those interests but saving was definitely more fun then, nevertheless.

Hence, the interest rate is kept as a constant on the *stack*, so the computation of the accumulated capital value becomes a simple series of [×] strokes.[71]

Another application benefitting from *top stack register repetition* is the *Horner scheme* for evaluating polynomials. It tells us:

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + a_nx^n$$
$$= \Big(\big((\ldots(a_nx + a_{n-1})\,x + \cdots + a_3)\,x + a_2\big)\,x + a_1\Big)\,x + a_0$$

Example:
Solve $7 + 6.4x - 2.1x^2 + 5.2x^3 - 3x^4$ for $x = 0.908$.

Solution:
This problem can be rewritten $\{\,[\,(-3x + 5.2\,)\,x - 2.1\,]\,x + 6.4\,\}\,x + 7$ and is easily solved this way (set [DISP] FIX [0][1]):

	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	0.9	0.9	0.9	-2.7	0.9	0.9	2.2	0.9	0.1	0.9	0.9
.908	0.9	-3	-2.7	5.2	2.5	2.2	2.1	0.1	6.4	5.9	12.9

.908 [FILL] **3** [+/−] [×] **5.2** [+] [×] **2.1** [−] [×] **6.4** [+] [×] **7** [+]

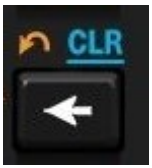
Note how the values automatically float down the *stack* to be used in the multiplications.[72]

Both examples work with the *8LS* as well – just with ***d*** repeated instead of ***t***. [FILL] loads the entire *stack* with ***x*** (regardless of its depth), far easier and more reliable than hitting [ENTER↑] repeatedly.

[71] Debt calculations are significantly more complicated – so avoid debts as long as possible, in your very own interest! In the long run, it is better for you and your economy. When debts become inevitably necessary, however, you and your *WP43* can cope with them as well (→ *OMS 6*).

[72] Try solving such a problem with an algebraic calculator (e.g., a *Casio*, *Sharp*, or *Texas Instruments*) for comparison – it will cost you many more keystrokes

Error Recovery: ⬅, EXIT, and ↶



Nobody is perfect – errors will happen although you are equipped with such a powerful tool. Stay cool – your *WP43* allows for undoing the last command executed, restoring the calculator state as it was before.

1. If you get an **error message** in response to a function call, read it, and when you know what went wrong, press ⬅, EXIT, 🟨 🟨, or 🟦 🟦: this will remove the message and return to the state before that error happened (→ *74f* and *340f*).

2. If you have erroneously called a **wrong function**, just press ↶ to UNDO it immediately. ↶ recalls the state as it was before that wrong operation was executed. So don't worry – be happy!

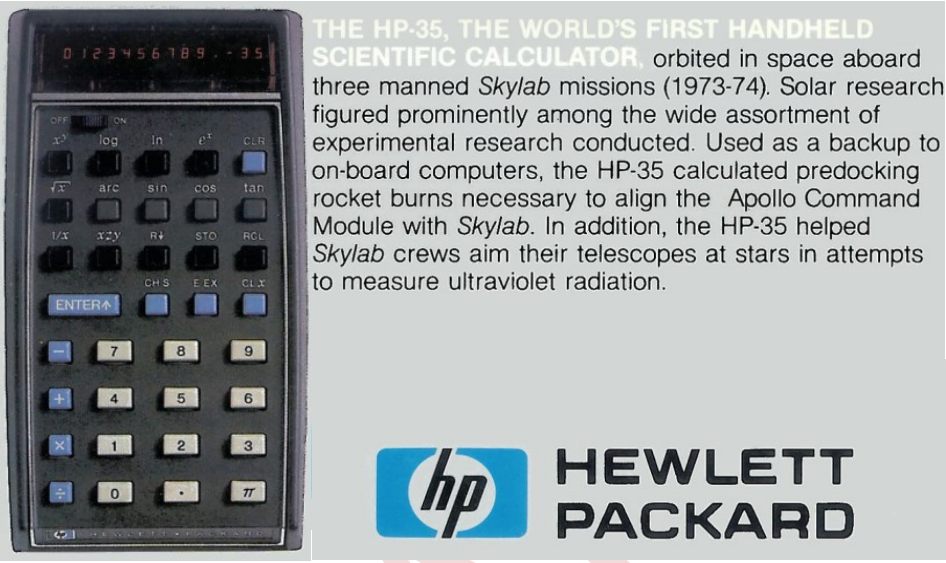
Example:
Assume you – while watching an attractive fellow student or collaborator – pressed × inadvertently instead of / in the fourth last step solving the lengthy formula on p. 47. Shhh… !! Murphy's Law! Luckily, however, there is absolutely no need to start calculating that formula all over again – that user error is easily undone as follows:

Z	...	123	456	123	456	456	123	456
Y	...	2.94...	123	2.94...	123	123	0.34...	123
X	...	24.1...	70.9...	24.1...	0.12...	0.34...	2_	2.349...
		x⇄y	×	↶	/	√x	2	+
		Fine so far	Oops!	Undo	Resume			Correct result

That's all you have to know about calculations with *reals* on the *stack*. ↶ operates on the entire *stack* (be it a *4LS* or *8LS*), statistical storage, and *system flags*. Since your *WP43* features *RPN*, this is all you need for resuming your work as if nothing had happened at all.[73]

[73] ↶ will undo the very last operation before ↶ only, nothing more – ↶ ↶ = ↶. And note operations explicitly confirmed by you can't be undone (→ last chapter of this *Section*).
Previous *RPN* calculators used LAST*x* for error correction – ↶ works easier and more comprehensive.

Such capabilities did suffice for high flying applications.



Note that the *HP-35,* the world's very first electronic pocket calculator featuring transcendental functions (→ *ReMA H*), was absolutely revolutionary at its time – it killed the slide rule – but is a primitive scientific from today's point of view, 54 years later.

There are, however, many more places than just the *stack* where you may store data and results in your *WP43*.

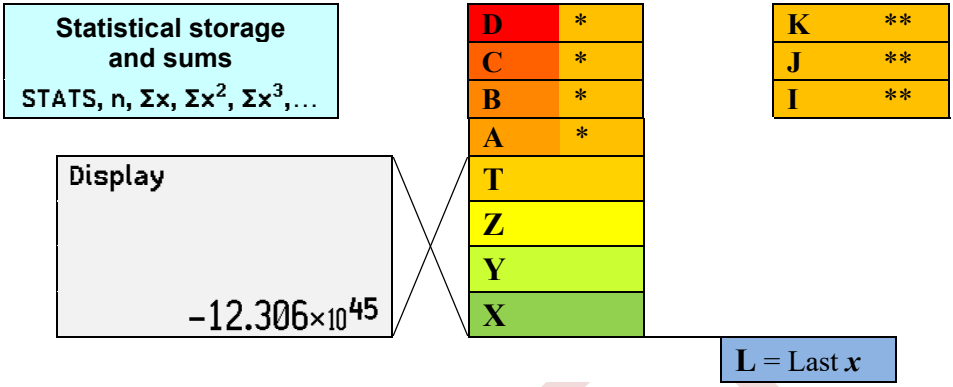
Addressing and Manipulating Objects in RAM

The *stack* provides workspace and temporary storage for your calculations. For longer term storage, feel free to employ additional *registers*, *variables*, and *user flags*. The remainder of *OMS 1* tells how to use them.

Sketches overleaf and on the page following show the address space of your *WP43*. Depending on the way you configure and use its memory, a subset of all these addresses will be accessible for you.

Operations on *general purpose registers* or *variables* (→ next chapter) may need manual undoing.

Special *registers* and *stack*



Statistical data may be stored value by value, point by point, or *en masse* in a *matrix* called STATS. The statistical sums will be accumulated automatically in a set of dedicated *summation registers*, called by their *names* and separated from your other data (→ *OMS 2*).

Special registers **I**, **J**, and **K** (**) may carry parameters of probability distributions; **I** and **J** will also serve as *index pointers* in *matrix* operations (→ *OMS 3*). **K** is also used as default *alpha register* in some special cases (→ *OMS 4*).

Depending on the stack size you choose, either **T** or **D** will be the top *stack register*. **A** - **D** (*) will be allocated to the big *stack*, if applicable.

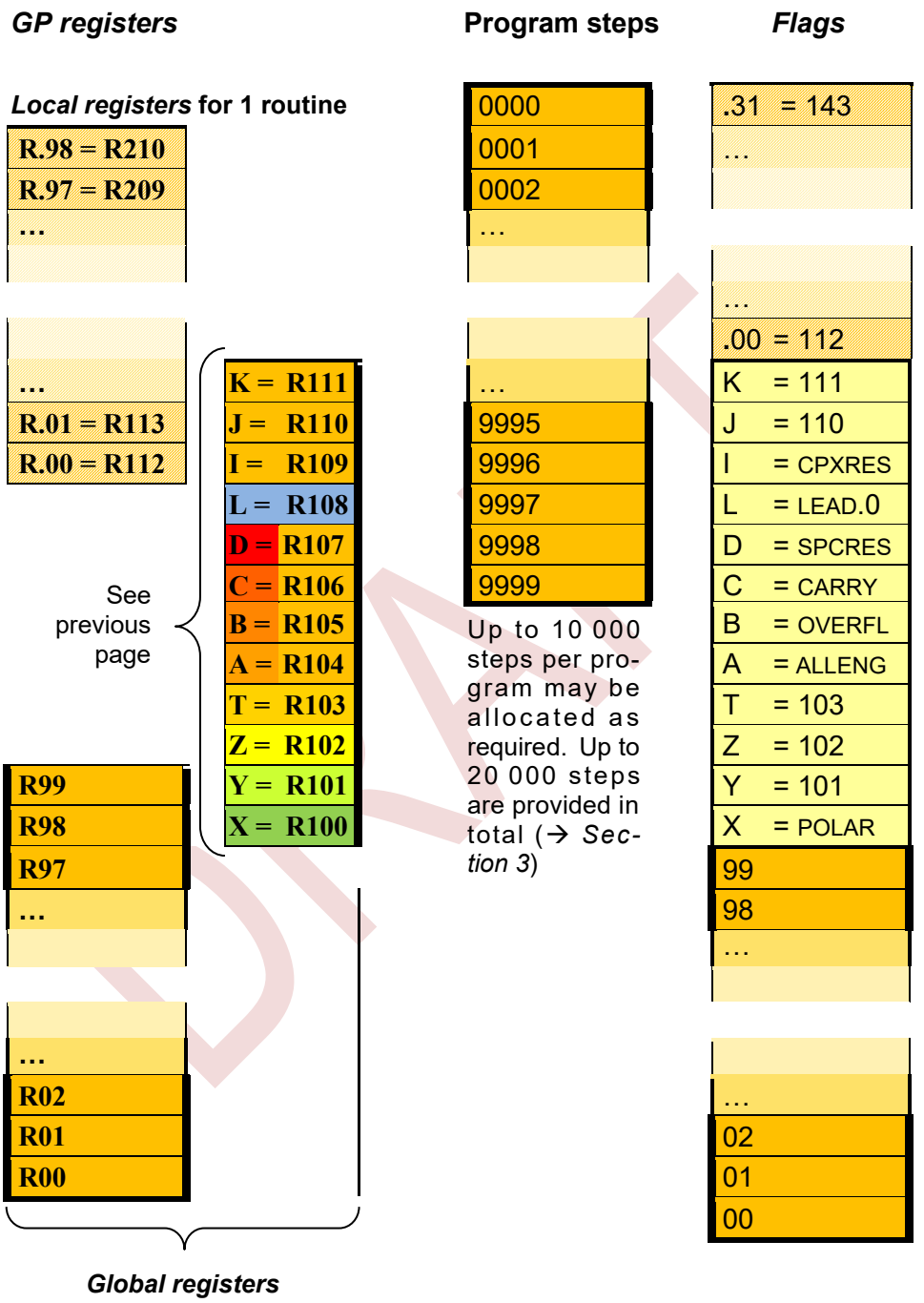
Unless required for said purposes, you may use **A**, **B**, **C**, **D**, **I**, **J**, and **K** as additional global ***general purpose (GP) registers*** (see below).

Please turn to the sketch overleaf:

A ***flag*** is an *item* featuring only 2 states, *set* and *clear* (like a bit). Employ *user flags* for signaling whatever you want.[74] *Flags* are most useful for program control – learn more about them in *OMS 4*.

Program steps will be covered in *OMS 4* as well.

[74] *System flags,* on the other hand, carry *names* and reflect or control specific system *states* (→ *ReMS 1*). Some single-letter *flags* serve as shortcuts to some *system flags* as specified overleaf.


GP registers
Program steps
Flags
Local registers for 1 routine
R.98 = R210
R.97 = R209
…
…
R.01 = R113
R.00 = R112
K = R111
J = R110
I = R109
L = R108
D = R107
C = R106
B = R105
A = R104
T = R103
Z = R102
Y = R101
X = R100
See previous page
R99
R98
R97
…
…
R02
R01
R00
Global registers
0000
0001
0002
…
…
9995
9996
9997
9998
9999
Up to 10 000 steps per program may be allocated as required. Up to 20 000 steps are provided in total (→ Section 3)
.31 = 143
…
…
.00 = 112
K = 111
J = 110
I = CPXRES
L = LEAD.0
D = SPCRES
C = CARRY
B = OVERFL
A = ALLENG
T = 103
Z = 102
Y = 101
X = POLAR
99
98
…
…
02
01
00

Each ***GP register*** (like each *stack register*) can hold any object you store therein (→ *OMS 3*). *GP registers* are beneficial, e.g., keeping intermediate results for repeated use or serving as records.

Example (in *SDC*):

$$\sqrt{3+\left(\frac{1.09}{1.78}\right)^2} \times \frac{ln\left[3+\left(\frac{1.09}{1.78}\right)^2\right]}{4\,cos\left[3+\left(\frac{1.09}{1.78}\right)^2\right]} = ?$$

Solution:

First, calculate the repeating term $3+\left(\frac{1.09}{1.78}\right)^2$ and store it:

1.09 ENTER↑ **1.78** /	6.123 595 505 617 978×10⁻¹
EXP x^2 **3** + STO K	3.374 984 219 164 247

Then solve the entire expression, e.g., like this:

√x	solves the 1st factor of the expression,
RCL K ln	solves the numerator,
×	2.234 647 088 154 349
RCL K TRI cos	solves the 2nd part of the denominator,
/	2.238 529 534 683 649
4 /	5.596 323 836 709 123×10⁻¹

That's it – solving this expression has become really easy this way.

Variables are named global storage locations. Such *names* shall begin with a letter and may consist of up to 7 characters. Creating a new *name*, feel free to use all characters except **+**, **−**, **×**, **/**, **^**, **(**, **)**, **=**, **|**, **!**, **:**, **;**, comma, and space. Note all ***variables*** **(and *menus*) must carry unique *names***. Your *WP43* decodes *names* as follows:

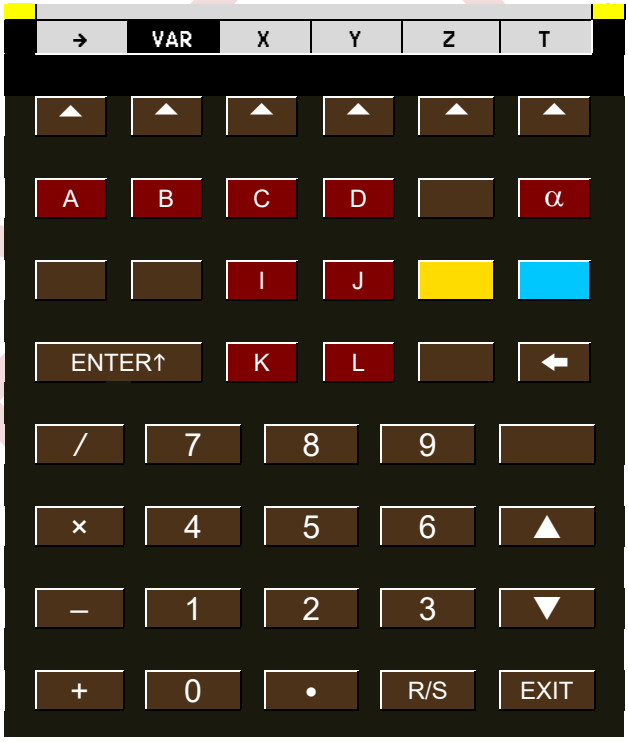
- Upper and lower case letters are interpreted differently (e.g., **X** ≠ **x**, **CHECK** ≠ **Check** ≠ **check** ≠ **cHeCk**).
- Superscript and subscript characters are read as normal characters (e.g., **Var¹** = **Var₁** = **Var1**); nevertheless, *names* will be always displayed as created, for easier reading).

Thus, check the *item names* present already (→ *ReMA N*). In particular, do not attempt to call your *variables* **A**, **B**, **C**, **D**, **I**, **J**, **K**, **L**, **T**, **X**, **Y**, or **Z**,

since these *names* are reserved for (*stack*) *registers* – but feel free to create *variables* named **x**, **y**, **z**, **t**, **X1**, **ab2**, etc. And since *items* may appear in *menus*, no *names* should share their first 6 characters.

Like each *register*, also each *variable* can hold any type of data. Explicitly clearing a *RoV* is dispensable since you can just overwrite its content.

While processing input for memory addressing (i.e. entering parameters for, e.g., comparing, storing, recalling, or copying), you will only need a subset of 30 labels plus 🟨 and 🟦. The calculator mode supporting just these 30 is called *transient alpha mode* (*TAM*). It is automatically set in memory addressing. Within *TAM*, the operational keyboard of your *WP43* is temporarily reassigned as shown here.



Such a picture is called a *virtual keyboard*. Dark red background highlights modified active functions in *TAM*. White print denotes *primary functions* here as well, such as the 1st key in row 2 directly addressing (possibly *stack*) *register* **A**.[75] Also all the other lettered *registers* can be called directly in *TAM* – *stack registers* **X**, **Y**, **Z**, and **T** by unshifted *softkeys*.

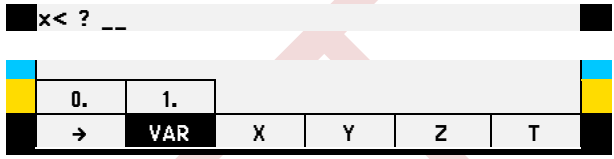
Access to numbered *registers* stays as easy as can be. The softkey → is for indirect addressing (→ 64ff), ⬚ for local addresses (→ 257). What is not printed on this *virtual keyboard* cannot be called in *TAM*.

[75] What is printed in white on your physical *WP43* is called a *default primary function*.

Variables defined at execution time show up in VAR in alphabetical order, so you can easily select the *variable* of your choice by pressing the respective *softkey*. Alternatively, you can address it via [α] (or create a new *variable* this way, → *64f* for how to do this).

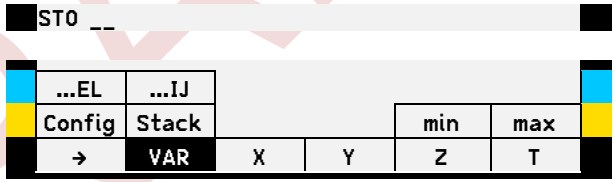
🟧 and 🟦 are just required for *softkeys* in *TAM*.

Menus are context sensitive here. If a **comparison** like [x< ?] is called it will look like this:



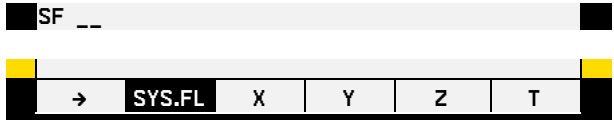
This allows for directly comparing x with **0.** or **1.** as well as with the content of any *RoV* (the latter via VAR).

If [STO] or [RCL] are called, on the other hand, the *menu view* will look like this instead:



Save all your specific calculator settings (a.k.a. its *configuration*) easily in a *RoV* via [STO] Config and recall them via [RCL] Config (→ *78*). Use [STO] Stack to store the entire *stack* in a block of 4 or 8 *registers* (depending on *stack* size at execution time); [RCL] Stack will recall it when you need it again. And max (or min) lets you work with the maximum (or minimum) of x and the source content – you may use ▲ as shortcut for max and ▼ for min here. (For ...EL and ...IJ, → *OMS 3.*)

For commands operating on ***flags***, SYS.FL will grant access to the *system flags* provided:



For commands operating on ***labels***, → *OMS 4*. For all other operations asking for a trailing parameter, the *menu view* will stay as shown on p. *61*.

TAM will be terminated as soon as sufficient characters are entered for the parameter(s) of the respective operation. You may undo pending input keystroke by keystroke using ⬅ and correct it – or just abort the pending command by EXIT; this will leave *TAM* immediately, returning to the mode set and the *menu* displayed before, if applicable.

For just inspecting any *RoV* without disturbing the *stack*, use VIEW.

Example:

DISP FIX 5

VIEW K returns

```
K= 3.374 98
           0.000 00
           0.000 00
           0.559 63
```

… as expected from previous problem.

Note the view into *register* **K** is displayed adjusted to the left just below the *status bar*. This view will vanish with your next keystroke.

For inspecting a set of *registers*, use RBR at any time instead. Call FLAG STATUS for checking the status of all *flags*. Both these commands leave the *stack* untouched, → *289ff*.

➡ You are granted unlimited access to all the *registers* and *user flags* allocated; there is nothing like '*memory protection*' on your *WP43*. You are the sole and undisputed master of its memory. Thus, taking care of it is your responsibility as well – keep suitable records to avoid inadvertently overwriting or deleting your precious data.[76]

At a particular time, a subset of the addressable maximum (20 000 program steps, 211 *registers*, and 144 *flags*) will be available for you, → *ReMA B* for reasons.

[76] You will learn about *local data*, preventing your current program from interfering with data of other programs, in *OMS 4*. This will ease your memory administration work.

Addressing Tables

Comparisons requiring a *RoV* trailing:

1	User input Echo	TEST x< ?, x≤ ?, x= ?, x≈ ?, x≠ ?, x≥ ?, x> ? OP? _ (with *TAM* set), e.g., x< ? __				
2	User input Echo	[0.] or [1.] OP *n*? e.g., x=0.?	*Stack* or *lettered register* (*SoLR*)[77] *or variable defined* [78] OP? *x* e.g., x≥ ? Y	*Register #*[79] (→ 67 for its range) OP? *rnn* e.g., x≠ ? r23	[→] opens indirect addressing OP? → _	[α] turns on *AIM* for typing a (new) *variable name* OP? '_
3	User input Echo	Compares *x* with the real number **0**.	Compares *x* with *y*.	Compares *x* with the content of *register* **R23**.	→ 66ff for more about indirect addressing.	*Variable name* (→ 60)[80] OP? '*xx*' e.g., x> ? 'ST1'

x> ? 'ST1' compares *x* with the content of the *variable* '**ST1**'.
Press TEST x> ? [α] S T [ ] 1 ENTER↑ for this.

[77] i.e. Y, Z, T, A, B, C, D, L, I, J, or K.

[78] We recommend calling an existing *variable* via VAR instead of typing its name.

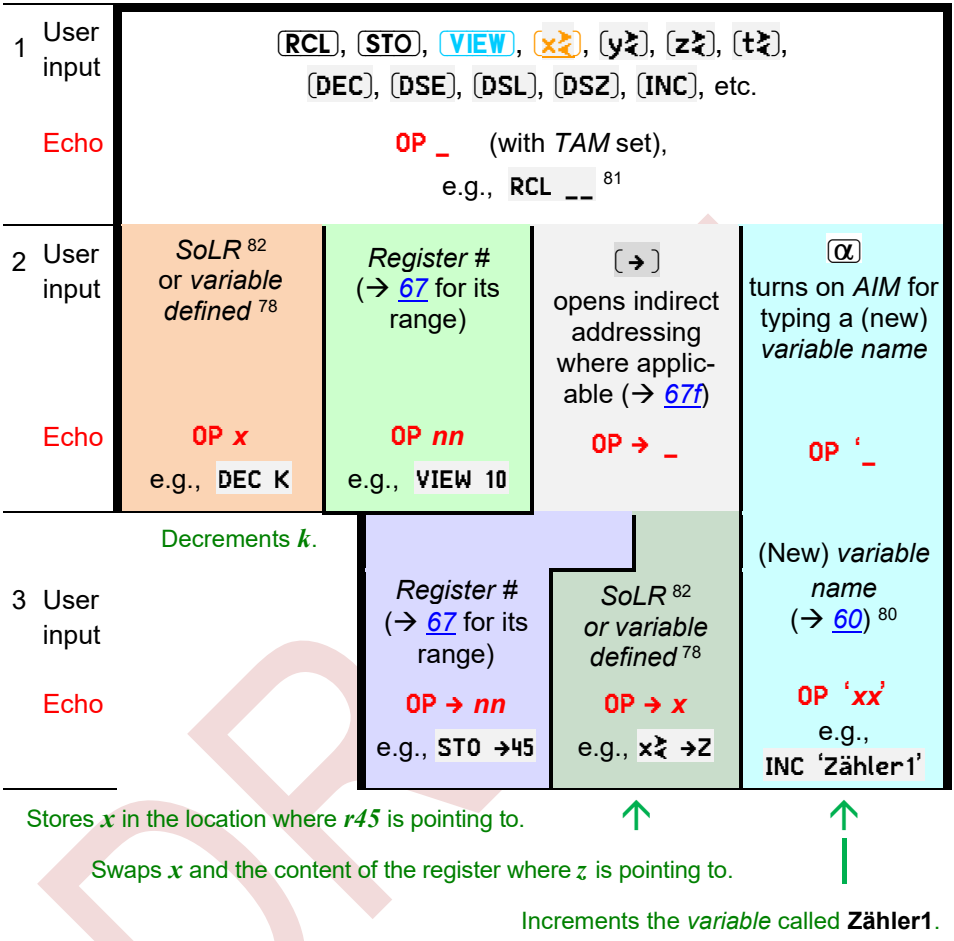
[79] # stands for *number* here.

[80] The 7th input character will terminate entry and close *AIM* – shorter *names* need a closing ENTER↑.

Attempts to write in a *variable* undefined at execution time will create it automatically, containing zero initially. Attempts to read from an undefined *variable* will throw an error.

Examples: Let **UNDEF** be an undefined variable. Then RCL **UNDEF** will throw an error while STO **UNDEF** or INPUT **UNDEF** will create **UNDEF**.

Register operations requiring just 1 *RoV* trailing:



Clearing a single *variable* or *register* is most easily done by storing zero in it. For deleting a *user variable* from memory and freeing the space allocated to it, use **DELITM** instead (→ 323).

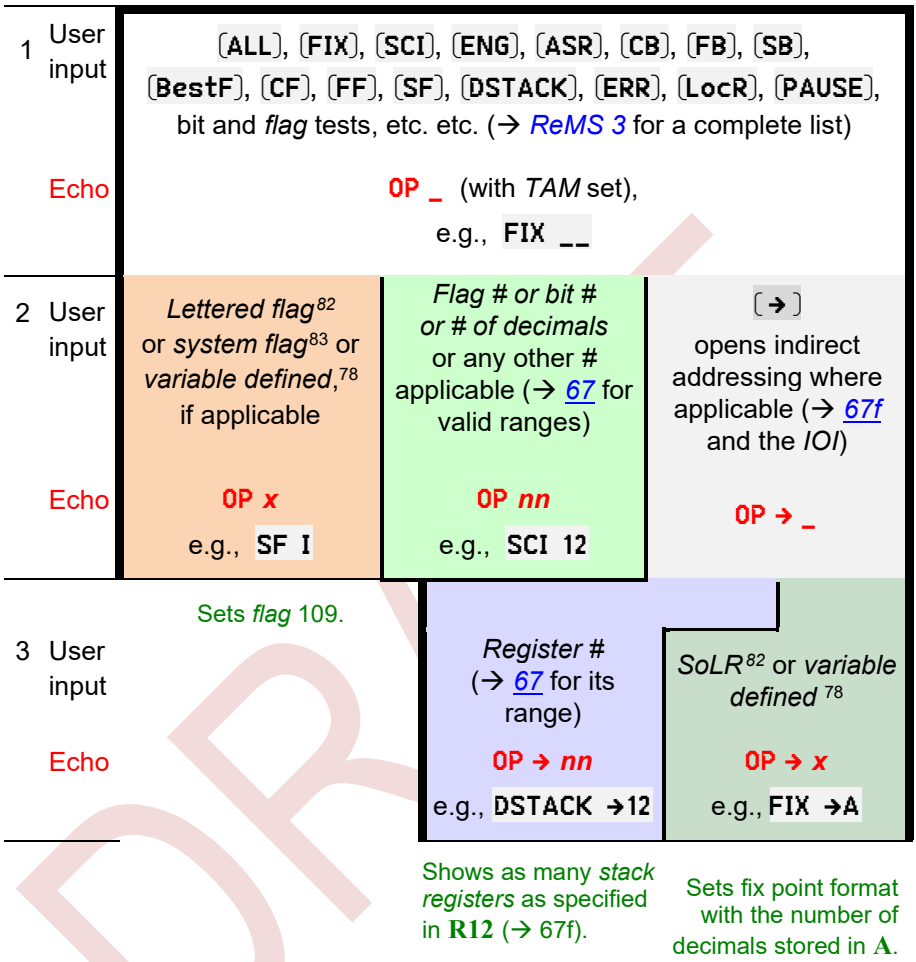
---

81 For RCL and STO exclusively, any of +, −, ×, /, ▲, or ▼ may precede step 2. Typing such a key twice will cancel it (e.g., RCL × × = RCL). See below for this *Store and Recall Arithmetic*.

Note, however, that such operators are disallowed in RCL **Config** (= **RCLCFG**), RCL **Stack** (= **RCLSTK**), **RCLEL**, **RCLIJ**, and the corresponding store operations.

82 i.e. X, Y, Z, T, A, B, C, D, L, I, J, or K.

Other operations requiring 1 trailing parameter:



For…	…the valid number range is…
decimals	0 - 15 (entering any digit except 0 or 1 will terminate input and close it)
integer bases	2 - 16
bit numbers	0 - 63 (note bit 0 is the least significant bit)

[83] → *ReMS 1*. We recommend calling *system flags* via their shortcuts or via **SYS.FL**.

For…	…the valid number range is…
word size	1 - 64 *bits*
registers	0 - 99 for direct addressing of global *registers* .0 - .98 for direct addressing of local *registers* 0 - 210 for indirect addressing (≥ 112 with local *registers* only) } upper limits depend on current allocation
user flags	0 - 99 for direct addressing of global numbered *flags* .0 - .31 for direct addressing of local numbered *flags* if at least 1 local *register* is allocated 0 - 143 for indirect addressing (≥ 112 with local *flags* only)

Registers and flags 100 - 111 can be directly addressed by single letters.

System flags can be called by their *names*; 1-letter shortcuts exist for some of them (→ 59).

Variables are called by their *names* always.

Please see *ReMS 1* for all other command parameters and their valid ranges, as well as for a compilation of all *system flags*.

You may directly address all *registers* incl. the lettered *registers*, in particular also each and every *stack register*. This may be very advantageous. But take care you don't inadvertently interfere with regular stack operation in a way you didn't want.

Indirect Addressing – Working with Pointers

Parameters for many functions can also be specified by supplying a *RoV* pointing to the respective parameter.

Example (assuming *SDC*):
Assume x = **12.34**, j = **45.67**, and $r12$ = **7.891 234 56**. Then ...

STO J

RCL → J will return 7.891 since **J** is containing **12.34** at execution time and thus is pointing to **R12**. And then ...

SF → X will set *flag* 7, and ...

DISP FIX → X will display 7.891 234 6 showing 7 decimals rounded.

Since the content of the *register* specified is not used directly but as a pointer to the *register* wherefrom we want to read (or whereto we want to write), this method is called indirect addressing.

Each and every *register* of your *WP43* can be used for indirect addressing.[84] And each and every *register* can be accessed this way.

Indirect addressing is most beneficial in programs with the parameter for a function calculated automatically (→ examples in *OMS 4*). It may be even combined with *store and recall arithmetic* as explained below, and become mightier yet.

Store and Recall Arithmetic

As mentioned in f. 81 on p. 65, arithmetic operations (and 2 conditional picks, *max* and *min*) can be performed upon the contents of *RoVs* by pressing STO or RCL followed by the respective operator (+, −, ×, /, ▲, or ▼), trailed by the address or *name* of the storage space.

Example for *store arithmetic*:

123.4

STO − K closes input and subtracts **123.4** from ***k***. The difference is stored in **K**. The *stack* and **L** remain unchanged here.

The same result could be achieved by the keystroke sequence

123.4

RCL K

x⇄y

[84] Several vintage calculators, on the opposite, featured just 1 dedicated *register* for indirect addressing, if at all. See the *HP-34C* or *HP-15C*, for instance. Even the *HP-15C CE* of 2024 still sports only 1 register for this purpose.

[−]
STO K
RCL L but this is far clumsier (replacing 1 step by 5) and would cost 1 *stack register* in addition.

General rule for *store arithmetic*:

new content of the *RoV* specified = old content of this *RoV* $\left\{ \begin{matrix} + \\ - \\ \times \\ / \\ max \\ min \end{matrix} \right\}$ current x

Here, *'max'* means the new content of the *RoV* specified will become the maximum of its old content and x.

Example (from the *HP-67 OHPG*):

During harvest, farmer *Flem Snopes* trucks tomatoes to the cannery for three days. On Monday and Tuesday he hauls loads of 25 *tons*, 27 *tons*, 19 *tons*, and 23 *tons*, for which the cannery pays him \$55 per *ton*. On Wednesday the price rises to \$57.50 per *ton*, and *Snopes* ships loads of 26 *tons* and 28 *tons*. If the cannery deducts 2% of the price on Monday and Tuesday because of blight on the tomatoes, and 3% of the price on Wednesday, what is *Snopes'* total net income? (Note the truck used by *Flem* in 1976 looks like *Ford's Model T* of 1910. A frugal farmer.)

Solution:

DISP FIX 2		Standard in financial calculations
25. ENTER↑ 27 +		Total of Monday's and Tuesday's
19 + 23 +	94.00	tonnage
55 ×	5 170	Gross amount for these days
STO J	5 170	Take **J** for accounting
2 FIN %	103.40	Deduction for these 2 days
STO − J	103.40	Subtract from the total in **J**

26. ENTER↑ **28** +	54.00	Wednesday's tonnage
57.5 ×	3 105.00	Gross amount for Wednesday
STO + J	3 105.00	Add to the total in **J**
3 %	93.15	Deduction for Wednesday
STO − J	93.15	Subtract from the total in **J**
RCL J	8078.45	Snopes' total net income from his tomatoes

In analogy to *store arithmetic*, also *recall arithmetic* is provided.

Example for *recall arithmetic*:

78.91

RCL / 2 3 closes numeric input and divides **78.91** by ***r23***. This operation is performed in **X** alone. **L** is loaded with **78.91**. All other *stack* contents and ***r23*** stay unchanged.

Alternatively, the same result could be achieved by the sequence

78.91
ENTER↑
ENTER↑
RCL 2 3
/
STK y≷ L
DROPy but that would replace 1 step by 6 and cost 2 additional *stack registers*.

General rule for *recall arithmetic*:

$$\text{new } x = \text{old } x \left\{ \begin{matrix} + \\ - \\ \times \\ / \\ max \\ min \end{matrix} \right\} \text{ content of the } RoV \text{ specified}$$

Here, *'max'* means the new x will become the maximum of the old x and the content of the *RoV* specified.

Stack-wise, both store and recall arithmetic work like *monadic* functions.

They may operate on each and every *RoV* provided, also on the *stack* and even on **L**.

Example:
What does RCL − Y do?

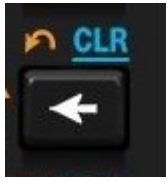
L	old value	1.0

Z	3.2	3.2
Y	2.1	2.1
X	1.0	−1.1
		RCL − Y

Register arithmetic is most beneficial in automatic routines. *Indirect addressing* may be combined with it, then looking like, e.g., STO × → K.

Although both these techniques have been more important in times when *PM* was scarce and processor speed low, they may be still beneficial today, in particular when you are out to create compact routines executing quickly. See pp. 233ff for examples and advantages, and look up the *IOI* for more information.

Clearing, Deleting, and Resetting



Besides ← and ↶, there are various other ways to remove obsolete, erroneous, or unwanted information from your *WP43*. The menu CLR contains the following:

CLX Clears the *stack register* **X** (i.e. resets it to **0.**)[85]

CLΣ Clears all statistical data and returns freed space

CLSTK Clears all *stack registers* allocated (but not **L**)

CLREGS Clears all global and local *GP registers*[86]

[85] You can as well use DROP↓ to get rid of x. → *IOI* for the subtle differences.

[86] *Stack* and *variables* are not touched by **CLREGS**. Learn about *GP registers* in next chapter.
CLREGS (and other far reaching commands explained in red print here) will ask for your confirmation in *RUM*; then they can't be undone by ↶.

CF ***n***	Clears the *flag* specified
CLP ***lab***	Clears the *program* specified and returns freed space
CLCVAR	Clears all user *variables* of the *current program*
CLMyM	Clears MyMENU
CLMyP, **CLMyα**	Resets MyPFN or Myα to their contents in *SDC*
CLFall	Clears all numbered *flags*
CLPall	Clears[87] all programs and returns freed space
CLVall	Clears all user *variables* of any kind
CLMall	Clears all *user menus*
DelVall, **DelMall**	Deletes all *user variables* or *menus* from *RAM* and returns freed space

CLall Clears all your creations in *RAM* (programs, *variables*, *user flags*, *menus*, assignments, and all *registers* incl. the *stack*)[88]

CLBKUP Clears the backup file in *FM*

CLLCD Clears the entire screen northeast of the coordinates specified

RESET Resets your *WP43* to *SDC* (removes all your traces in *RAM* – just the contents of *FM* will stay untouched)[89]

CLKEYS Clears all key assignments (→ *OMS 7*)

DelITM Deletes a user-defined *item* (i.e. a *menu* or *variable*, → *OMS 7*)

CLMENU Clears the *programmable menu* (→ *OMS 4*)

For your reference, the *startup default* settings of your *WP43* are:

> **2COMPL**, **ALL 00**, **DEG**, **DENMAX 9999**, **DSTACK 4**, **GAP 3**, **HIDE 0**, **J/G 1752.0914**, **LinF**, **LocR 0**, **RANGE 999**, **RM 0**, **SETSIG 0**, **TDISP 0**, and **WSIZE 64**.
>
> The *system flags* AUTOFF, DECIM., DENANY, ENDPMT, MULT×, PROPFR, SSIZE8, TDM24, YMD, and αCAP are set, all others are clear in *SDC*.

Turn to *ReMS 1* for more information about the *items* mentioned above.

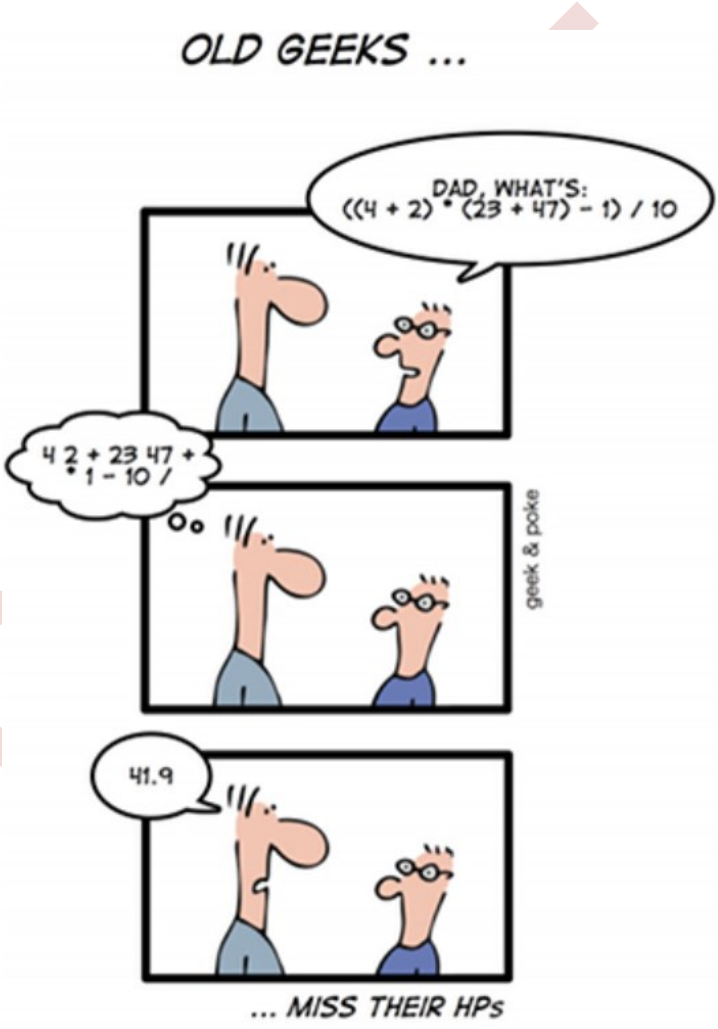
---

[87] Actually, **CLP** and **CLPall** delete programs, but we kept the traditional command names.

[88] Display formats as well as other user settings will remain unchanged (e.g., *system flags*). Only **RESET** clears everything except the contents of *FM* (→ *OMS 4*).

[89] If you can't execute the command **RESET** for any reason whatsoever, a hard reset will do almost the same: use the RESET hole on the calculator backside if necessary.

Up to here you have learned about *RPN* and how to address *registers*, *flags*, and *variables* in your *WP43*. Various methods for accessing these storage places have been presented. Such calculatory capabilities did suffice for orbiting the Earth and controlling spacecraft docking maneuvers, using also functions beyond the basic ones – let's introduce them to you in next *Section*.



Found at https://calc.fjk.ch/old-geeks-miss-their-hps/

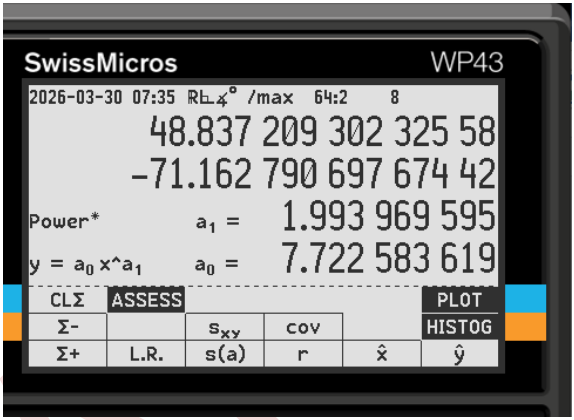
SECTION 2: MORE REAL STUFF

Some Display Basics

The screen is your window to your *WP43* – here you see the latest results and what is going on.

Top down, you find ...

- the *status bar,*
- up to 4 rows of *standard numeric output* (and more), and
- the *menu section* showing up to 3 rows of up to 6 *items* each (as introduced on pp. 32ff).



Let's look at the 4 numeric rows bottom up:

1. The **bottom row shows** x. Its left side is for ...
   a. echoing numeric or text input until it will be closed (→ 26 and 208ff). **X** can take up to 42 digits, a sign, and a radix mark in *SDC* or some 40 alphanumeric characters depending on their widths. You may edit pending input keystroke by keystroke using ⬅. Numeric input will be checked and interpreted according to the calculator settings as soon as it is closed.
   b. ***temporary information*** **(*TI*)** heading x, if applicable:

      TI is displayed information exceeding the plain contents of **X**, **Y**, and **Z** in *RUM* (like messages or explanatory text). *TI* will vanish with next keystroke (thus 🟨 🟨 or 🟦 🟦 will remove it from the screen most easily and free of any side effects).

   c. the output of **SHOW** (displaying full precision of x until next keystroke) if the 3 rows above don't suffice for taking it.

2. For DSTACK **4**, **3**, or **2**,[90] the next **row above presents** y. Alternatively, it may be used by SHOW if the 2 rows above don't suffice.

   Its left side is for *TI* heading y, if applicable.

3. For DSTACK **4** or **3**, the next **row above displays** z. It may be taken by SHOW if the top row doesn't suffice (→ example on p. 186).

   Its left side is for any error message (→ 56 and *ReMA C*) or the output of a binary test (→ *OMS 4*) or for *TI* heading z, if applicable.

4. In *SDC*, the **top (T) row shows** t. This will be replaced by the output of SHOW, if applicable.

   Its left side is for VIEW (→ 63) and for echoing command input until it can be executed (with all required parameters entered); → 28. Feel free to edit a pending command using ← or cancel it by EXIT as explained on p. 63.

Special Displays (Returned by RBR, STATUS, VERS, etc.)

Some applications and commands use the screen in special ways:

1. The *Matrix Editor* is described comprehensively on pp. 162ff.
2. Editing *equations* is covered on pp. 266ff.
3. RBR allows for browsing all *registers* currently allocated (→ 289f).
4. FLAG STATUS returns free pool space available, all *user* and *system flags* set, and more (→ 291).
5. ⏱ calls the *timer* or stopwatch application (→ 292f).
6. a.FN FBR browses all *characters* provided (→ 208ff).

Further commands throw *temporary information* (as defined on p. 74):

[90] If you chose displaying less than 4 *stack levels* (→ DSTACK in the *IOI*), echoes and *TI* will nevertheless appear at the positions specified above, whenever applicable. And operations resulting in multi-row output will always display their entire output regardless of the DSTACK setting.

a. [cov], [r], [s_{xy}], [VIEW], [WDAY], [$\hat{x}$], and [$\hat{y}$] return results with headers.

b. Commands returning 2 or 3 values at once (like [→P], [DATE→], [DECOMP], [$\bar{x}$], [s], [L.R.], [SUM], [?DIM], [RCLIJ], and [Σ+]) tag their output (→ *22* and *99ff* for examples).

c. Tests show their results (`true` or `false`, → *230ff*).

d. [ERR] and [MSG] display error messages (→ *ReMA C*).

e. [VERS] returns the firmware version running on your *WP43* ([WHO] works in a similar way):

```
WP43 v0.30 2026-03-11
```

A few far-reaching commands (e.g., CLall or RESET) will ask you for confirmation before executing. Then answer either

- [Y]es (by pressing [3], [ENTER↑], [XEQ], or YES) or
- [N]o (by pressing [/], [EXIT], [←], or NO).

Other input will be ignored. Note that actions explicitly confirmed can't be undone by [↶] (→ *71f*).

Localizing Numeric Output

You can summon display preferences for *reals*, *times*, and *dates* all at once according to your region's customs using 6 dedicated commands (contained in the 2nd view of DISP):

GAP	HIDE	HIDE?	RANGE	RANGE?	DSTACK
CHINA	EUROPE	INDIA	JAPAN	UK	USA

In the table starting overleaf, …

- *radix mark* denotes the decimal separator; [91]

[91] → *https://en.wikipedia.org/wiki/Decimal_separator* for a world map of radix mark use. Looks like an even score in this matter. Thus, the international standard *ISO 80000-1* allows either a decimal point or a comma as radix mark and requires a narrow blank as unambiguous separator of digit groups (it explicitly states that points or commas shall not be used as group separators to avoid ambiguity).

Note most people using radix commas employ multiplication dots (while those using radix points need crosses to avoid misunderstandings, causing further ambiguities in *vector* multiplication, however).

- GAP states the digit group interval – after ***n*** digits a narrow blank is displayed (→ examples on p. 133); this follows *ISO 80000-1*; [92]
- Date stands for the date format (**Y.MD**, **D.MY**, or **M.DY**);[93]
- *JG* states the year the *Gregorian calendar* was introduced in the particular region, typically replacing the *Julian calendar* (or national calendars in East Asia); check the dates in *Wikipedia* and enter the date applicable at your location using **J/G** (→ *IOI*); [94]
- background colors are again light blue for *SDC*, light red for other widespread settings.

Com-mand	GAP	Radix mark	Time	Date	*JG*	Remarks
SETCHN **CHINA**	4 [95]	point	24h	**Y.MD**	1949	
SETJPN **JAPAN**	3	point	24h	**Y.MD**	1873	

[92] As far as we know, *WP43* was the very first pocket calculator worldwide (in 2022) displaying numbers the way internationally agreed on. Previous calculators had to use, e.g., points or commas as crutches since they could not display narrow blanks.

[93] → https://en.wikipedia.org/wiki/Date_format_by_country for a world map of date formats used. The international standard *ISO 8601* states YMD for dates and 24h for times. This combination carries some inherent logic and is common in East Asia (→ **SETCHN** and **SETJPN**). **D.MY** reverts this sequence. **M.DY** jumps back and forth.

[94] Officially, the *Gregorian* calendar became effective at 1582-10-15 in the catholic world. Many states and territories switched later for various reasons. There are still other calendars widespread, → *Dates* below.

[95] Chinese counting and traditional mathematics work with powers of 10 000 using intervals

一 (1), 十 (10), 百 (100), 千 (1000),
万 (10^4), 十万 (10 万), 百万 (100 万), 千万 (1000 万),
亿 (10^8), 十亿, 百亿, 千亿,
兆 (10^{12}), …

Compare the details of a Chinese calculator keyboard pictured overleaf (note the reverse sequence). **GAP 4** takes care of this notation, while (originally Indian, then Persian, then Arabian, then) European counting and mathematics work with powers of 1000: **GAP 3** formats this way. In the meantime, Indians invented something special.

Com-mand	GAP	Radix mark	Time	Date	*JG*	Remarks
SETIND **INDIA**	3 [96]	point	24h	**D.MY**	1752	Applies to the area of former British India (India, Pakistan, Nepal, Bhutan, Bangladesh, Myanmar, and Sri Lanka).
SETEUR **EUROPE**	3	comma	24h	**D.MY**	1582	Also applies to Latin America and – with other *JG*s – to Indonesia, South Africa, Vietnam, and the area of the former Soviet Union.
SETUK **UK**	3	point	12h	**D.MY**	1752	Also applies to Australia and New Zealand.[97]
SETUSA **USA**	3	point	12h	**M.DY**	1752	

You can save above settings for time and date, all *system flags*, and your choices of formats for *reals*, *fractions*, *angles*, and *integers* (→ respective chapters below and in *OMS 3*) collectively at one single location. **STOCFG** will store this *calculator configuration* in the *RoV* you specify.[98] Use **RCLCFG** to recall it, setting (or resetting) your *WP43* accordingly at once.

[96] Proper South Asian (a.k.a. Indian) formatting nowadays requires separators every 2 digits for numbers > 1000. Think of *lakh* = 10^5, *crore* = 10^7, *arab* = 10^9, and, e.g., *lakh crore* = 10^{12}. Actually, an amount of 50 cr. Rupees reads 50,00,00,000 Rs in Indian newspapers.

Sri Lanka and Nepal turned to GAP **3** recently.

[97] 24h (a.k.a. Indian '*military time*') is taking over in the *UK*, so **SETIND** will work there as well then – but set **GAP 3** thereafter.

[98] Actually, it stores even more, → *OMS 7*.

Changing the Display Format

Any number you enter using one [.] or/and an [E] is taken as a *real* by your *WP43* unless there is additional information given (→ 132). The majority of functions provided operate on *reals*.

A closed *real* will be displayed right adjusted (→ 26f). All its digits will be shown in *SDC* if 16 are sufficient for doing so – your *WP43* will automatically turn to mantissa plus exponent format (*MEF*) if a number is too big or too small; this prevents missing unexpected big or small results.[99] There are 2 flavors of *MEF*:

SCI is called *scientific notation*. E.g., **SCI 2** will show the age of the universe in *years* like 1.38×10^{10} (or $1{,}38\cdot10^{10}$).

ENG displays like **SCI** but the exponent will always be a multiple of 3, corresponding to the *SI* unit prefixes – it is called the *engineer's notation*. **ENG 2** will show 13.8×10^{9} for said age.[100]

Furthermore, **FIX** fixes the radix mark on the screen (thus called *fixed point notation*) while it floats in the other formats – see below.

You can specify the number of decimals you want to see in **FIX** or **SCI**. For **ENG**, the mantissa will (traditionally) show 1 digit more than you specified. Check the following **examples**:

Input \ Format	*SDC* (ALL **0**, GAP **3**, ~~ALLENG~~)	**FIX 2**	**SCI 4**	**ENG 4**
71.2345678 [EXIT]	71.234 567 8	71.23	$7.123\,5\times10^{1}$	71.235
[1/x]	$1.403\,812\,826\,951\,692\times10^{-2}$	0.01	$1.403\,8\times10^{-2}$	14.038×10^{-3}

Now let's continue with the last number divided by **1000**, then multiplied times **10** in every subsequent row, and look at its displays for **FIX 4**, **SCI 4**, and **ENG 4**:

[99] Regardless of the format or notation you select, this affects the display only. Internally, your *WP43* continues using its full precision (use [SHOW] to display it until next keystroke). On the other hand, **SETSIG** can reduce the precision in calculations (→ *ReMS 1*).

[100] Seems engineers prefer *SI* prefixes instead of exponents of 10, but those don't apply everywhere – e.g., not here. Exponents of 10 are also beneficial mitigating confusion caused by identical terms used for different quantities, e.g., 1000 *billions* (*USA*) = 1 *billion* (Continental Europe), 1 *quadrillion* (*US*) = 1 *billiard* (*CE*).

FIX 4	SCI 4	ENG 4
1.403 8×10⁻⁵	1.403 8×10⁻⁵	14.038×10⁻⁶
0.000 1	1.403 8×10⁻⁴	140.38×10⁻⁶
0.001 4	1.403 8×10⁻³	1.403 8×10⁻³
0.014 0	1.403 8×10⁻²	14.038×10⁻³
0.140 4	1.403 8×10⁻¹	140.38×10⁻³
1.403 8	1.403 8	1.403 8
14.038 1	1.403 8×10¹	14.038
140.381 3	1.403 8×10²	140.38
1 403.812 8	1.403 8×10³	1.403 8×10³
14 038.128 3	1.403 8×10⁴	14.038×10³
140 381.282 7	1.403 8×10⁵	140.38×10³
1 403 812.827 0	1.403 8×10⁶	1.403 8×10⁶

Compare p. 133 and note that formatting choices may lead to different screen space requirements.[101]

SCI and **ENG** are boring simple: they display a constant number of (hopefully significant) digits for each number, while **FIX** shows each number precisely to the last decimal specified. Thus, **FIX 2** is the favorite format of financial people.

For **ALL**, its parameter tells your *WP43* how many decimal zeros you allow before the display shall overflow either to **SCI** or **ENG**, determined by the *system flag* ALLENG (or **A**). See the case below.

Once you have chosen a display format, DSP lets you change its parameter (without the need to open DISP and call **FIX**, **SCI**, **ENG**, or **ALL** again), as demonstrated in this **example** starting in *SDC*:

−700	−700
1/x	−1.428 571 428 571 429×10⁻³ turns to **SCI**.
DSP 2	−1.428 571 428 571 429×10⁻³ allowing max. 2 zeros is not sufficient to get a fixed point notation again.

[101] Advancing previous calculators, ×10⁰ is suppressed for **SCI** and **ENG** on your *WP43*.

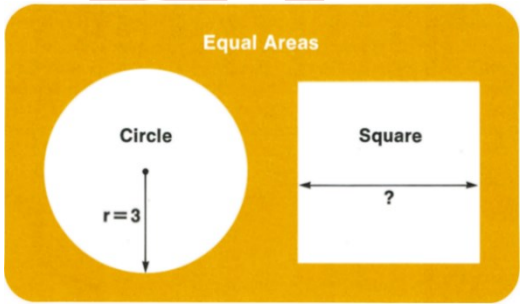
DSP 3	-0.001 428 571 428 571	but 3 zeros suffice.
10 /	$-1.428\,571\,428\,571\,429\times10^{-4}$	this result is too small again.
SF A	$-142.857\,142\,857\,142\,9\times10^{-6}$	turns to **ENG**.
DSP 4	-0.000 142 857 142 857	4 zeros suffice here.
10 /	$-14.285\,714\,285\,714\,29\times10^{-6}$	this is too small again.
CF A	$-1.428\,571\,428\,571\,429\times10^{-5}$	returns to **SCI**.
1000 ×	-0.014 285 742 857 143	

Usually you will be happy with far less than 16 digits shown: just focus on the important (and reliably significant) ones.

FIX, **SCI**, **ENG**, **ALL**, and almost all other functions controlling *real* display format are found in DISP (→ *ReM*).

Squares and Cubes and Their Roots

Let's start looking at the *real* functions: √x is on the keyboard of your *WP43*, while x^2, x^3, $\sqrt{(1+x^2)}$, and $\sqrt[3]{x}$ live in EXP (→ *33*).[102] The following **example** (triggered by the *HP-65 OH*) solves some of the most 'popular' problems of ancient mathematics:



What size square has the same area as a circle whose radius is **3**? And what size cube has the same volume as a sphere whose radius is **3** again? And what can we tell about their surface areas?

Solutions (with **FIX 3**):

The area of a circle is $A_C = \pi r^2$.
The area of a square is $A_{sq} = a^2$.
The volume of a sphere is $V_S = \frac{4}{3}\pi r^3$, while its surface is $A_S = 4\pi r^2$.
The volume of a cube is $V_{cu} = a^3$, while its surface is $A_{cu} = 6a^2$. Thus,

[102] You can also produce a square using ENTER↑ × without opening EXP. But it will cost 1 *stack* level more and take 2 keystrokes always.

3 EXP x^2 π × returns	28.274	for the area of the circle. Then
$\sqrt{x}$ returns	5.317	for the edge length of the square.

Furthermore, 3 x^3

π × 4 × 3 / returns	113.097	for the volume of the sphere.
Then $\sqrt[3]{x}$ returns	4.836	for the edge length of a cube with same volume.
Thus, x^2 6 × returns	140.320	for the cube's surface. Finally,
3 x^2 4 × π × returns	113.097	for the surface of the sphere.

Here is a little winter sports problem of our time:

Example:
Chuck Carver swings down a ski run with moderate 30 km/h. The shape of his skis allows for turns with 12 m radius. He claims having carved this way without any sliding on a short flat part of the run. If true then how many **g** he had to withstand there? Can we believe his story?

Solution (with FIX 1):
Centrifugal acceleration is $a_c = 2\pi\, {v^2}/{r}$. So the total acceleration along *Chuck's* body axis is $a_T = \sqrt{g^2 + a_c^2}$. This means in multiples of **g** :

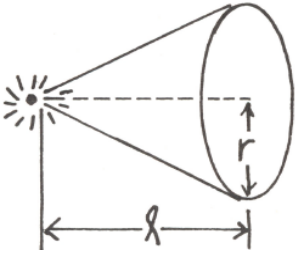
$$ {a_T}/{g} = \sqrt{1 + \left({a_c}/{g}\right)^2}. $$

30 ENTER↑ 3.6 /	8.3	Chuck's speed in $^{m}/_{s}$
EXP x^2	69.4	
2 × π × 12 /	36.4	his centrifugal acceleration
CONST $g_\oplus$ /	3.7	
$\sqrt{(1+x^2)}$	3.8	meaning 3.8 times **g**.

This might be possible to stand shortly for a young sportsman; but the snow under him can't bear the corresponding forces – it will break, so *Chuck* will inevitably slide in a greater radius meaning less acceleration.

Another **example** from an early brochure about *HP-35*:

The solid angle a circular detector covers of a point source is given by



$$\Omega = 2\pi \left[1 - \sqrt{\frac{1}{1 + \left({}^{r}/_{l}\right)^{2}}} \right]$$

How big is it for r = 2.5 cm and l = 9.3 cm?

Solution (with FIX 4):

2.5 ENTER↑ **9.3** / EXP √(1+x²) 1/x +/- **1** + **2** × π ×

returns Ω = 0.215 4. You can solve for annular detectors by analogy.

A technical application found in an *HP* manual of 1988:

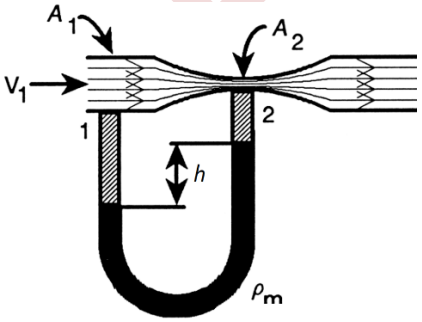
When an incompressible fluid with negligible viscosity flows steadily along a pipe, *Bernoulli*'s equation holds at any point ***i*** along the pipe. It's given by

$$p_i + \tfrac{1}{2}\,\rho\, v_i^2 + \rho\, g\, H_i = K$$

where ***p*** is the pressure, ***ρ*** is the fluid density, ***v*** is the fluid velocity, ***g*** is the acceleration due to gravity, ***H*** is the height of the fluid above some arbitrary reference point, and ***K*** is a constant. … The continuity equation says that the mass of fluid in a pipe is conserved (none enters or exits the pipe). The continuity equation for an incompressible fluid with steady flow is

$$A_1\, v_1 = A_2 v_2$$

where ***A*** is the pipe's cross sectional area and ***v*** is the fluid's velocity at points 1 and 2, respectively. Informally, the equation states that the flow rate in volume per time is constant at any point in the pipe.



Example:

A *Venturi tube* shall be used to measure speed of fluid flow. The U-shaped tube in the figure is called a manometer and is used to measure pressure differences. The manometer equation for pressure difference is $p_2 - p_1 = -\rho_m\, g\, h$, where ***p*** is the pressure at points 1 and 2, ***ρm*** is the density of fluid in the manometer, ***g*** is

the acceleration due to gravity, and ***h*** is the difference in fluid levels in the manometer. When the continuity equation, *Bernoulli's* equation, and the manometer equation are combined, the speed of fluid flow at point 1 in the figure is found to be

$$v_1 = A_2 \sqrt{\frac{2\,(\rho_m - \rho)\,g\,h}{\rho\,(A_1^2 - A_2^2)}}$$

If the manometer is filled with mercury (ρ_m = 13 595 $\mathrm{kg/m^3}$) and the fluid in the pipe is water (ρ = 1000 $\mathrm{kg/m^3}$), find the velocity of fluid flow at point 1. Assume that the cross sectional area of the pipe at point 1 is 0.073 $\mathrm{m^2}$ and at point 2 is 0.0507 $\mathrm{m^2}$. The difference in mercury levels in the manometer is 9.0 cm. Follow the keystrokes below to find v_1.

DISP FIX 2	since the least precise input carries 2 digits.
13595 ENTER↑ E 3 −	12 595.00
2 × CONST $g_\oplus$ × .09 ×	22 232.66
.073 EXP x^2 .0507 STO K	0.51
x^2 − E 3 ×	2.76
/ $\sqrt{x}$ RCL K ×	4.55

The fluid velocity at point 1 is 4.6 $\mathrm{m/s}$.[103]

Here is a high flying problem excavated in a prior manual:

Example:

Finding himself floating dangerously close to the jagged peaks of the Canadian Rockies, intrepid balloonist *Chauncy Donn* frantically cranks open the helium valve on his spherical balloon. Gas from the helium tank increases the balloon's radius from 7.5 *meters* to 8.25 *meters*.[104] *Donn* clears the mountain tops safely. How much did the volume of the balloon increase?

[103] *TN*: We actually helped this problem by assuming the manometer was read with 1 mm precision. The original text just stated **9** cm but printed a 5-digit result. There must have been some magic …

[104] *TN*: The *HP-21 OH* lets the balloonist *Ike Daedalus* increase the radius from 25 to 27 *feet*. 1976 brought some progress, although the numbers look like translated from another system. Since then, the momentum turning to *SI* faded there (idleness?).

Solution:

We know the volume of a sphere from p. 81, thus the difference of two such volumes is $\Delta V = \frac{4}{3}\pi(r_2^3 - r_1^3)$. 1 decimal shall do.

8.25 EXP x^3	561.5
7.5 x^3 −	139.6
π ×	438.7
4 × 3 / returns	584.9 m^3 for the volume increase.

Percent Change

Δ% calculates the percentage of change from y to x.

Example (continued from previous chapter):

This is a volume increase of how many percent?

Solution:

7.5 x^3	421.9
8.25 x^3	561.5
Δ% returns	33.1 % increase of the balloon volume.

Another **example**, more down-to-earth:

Pia Power strives for designing an optimum bicycle gearing for her home in a hilly region. She is free to choose sprockets and gear clusters to her liking.

Solution:

As long as drag can be neglected, an optimum bike gearing will feature constant velocity ratios between subsequent gears (or uniform percental increase of distances per crank revolution). There are several ways *Pia* can achieve this, depending on the number of sprockets chosen at front and rear. One inexpensive and solid way is taking 3 front sprockets of 48, 36, and 24 teeth and a standard 7-gear cluster featuring 13 through 30 teeth. This will result in the following distances per

crank revolution (d/r_c in *meter*) for her 26" MTB: [105]

Gear	1	2	3	4	5	6	7	8	9	10	11	12
Front	24				36			48				
Rear	30	26	23	20	26	23	20	23	20	17	15	13
d/r_c	1.660	1.915	2.165	2.490	2.873	3.247	3.734	4.330	4.979	5.858	6.639	7.660
Δ%		15.4	13.0	15.0	15.4	13.0	15.0	16.0	15.0	17.7	13.3	15.4

Assuming *Pia* pedals steadily with a frequency of 60 rpm, such a gearing will allow her conveniently riding with velocities between 6 and 28 km/h (taking the lowest gears for climbing steep ramps). After some months of continuous training, Pia pedals with 80 rpm, reaching velocities up to 37 km/h this way.

Using 3 functions provided in STAT ([Σ+], [x̄], and [s], explained on pp. 99ff), you can determine an average speed increment per gear step of (14.9 ± 1,4)% here. This gearing turns out being quite uniform and convenient for town and country.[106] Feel free to try and check other configurations.[107]

Logarithms and Powers (a.k.a. Antilogs)

Your *WP43* features 4 logarithmic functions: 2 on its keyboard and 2 more in EXP (→ *33*):

[105] This example was created in 2013.

[106] You get 12 out of 21 theoretically possible gears here only. This is due to gear overlaps; and you will want to avoid extreme chain skew for sake of components life. Conditions change if you plan for a recumbent bike featuring a long chain: Then you may think about 3 sprockets with a 7-gear cluster leading to 17 different, usable gears following a scheme called '*half-step-plus-granny*'. Speed increase in *half-step* range will be 9% per gear step. This gearing will cover velocities from 5 to over 40 km/h for 60 rpm pedaling frequency. Recumbent bikes show significantly less drag than conventional ones.

During the *Tour de France* in 2022, 36 vs. 30 teeth was the *granny gear* (!) for mountain roads in the Alps. Some training was definitely required.

[107] You can obtain up to 13-gear clusters in 2026 – a wide field for considerations and design. Though not every theoretically possible combination may turn out being sufficiently wear-resistant for your bike. Also gear hubs and bottom bracket spur gears featuring up to 18 steps of constant ratio are available. The site *https://ritzelrechner.de/* may support you designing and selecting your optimum bike gearing (→ *159*).

〔ln〕 computes the *natural logarithm* of x, i.e. the logarithm of x to base of Euler's *e*. 〔ln〕 inverts 〔e^x〕.

〔lg〕 returns the *decadic* (a.k.a. *common*) *logarithm*, i.e. the logarithm to base 10. 〔lg〕 inverts 〔10^x〕.[108]

EXP **lb x** computes the *binary logarithm*, i.e. the logarithm to base 2. 〔**lb x**〕 inverts 〔2^x〕.

EXP **$\log_x y$** is the most general of these logarithmic functions, returning the logarithm of y to base x. 〔$\log_x y$〕 inverts 〔y^x〕.

The manual of the world's very first scientific pocket calculator, the *HP-35* (→ *57*), presented just 1 single **example** for this then new class of pocketable functions:

> Although logarithms were originally used to speed multiplication and division, they have particular significance in scientific and engineering problems. There is, for example, a logarithmic relationship between altitude and barometric pressure. Suppose you wish to use an ordinary barometer as an altimeter. After measuring the sea level pressure (30 inches of mercury) you climb until the barometer indicates 9.4 inches of mercury. How high are you? Although the exact relationship of pressure and altitude is a function of many factors, **a reasonable approximation** is given by:[109]
>
> $$\frac{altitude}{[feet]} = 25\,000 \times ln\left(\frac{30}{pressure / [inches\ of\ Hg]}\right)$$
>
> **Solution:**
>
> DISP **FIX** 0 0 should suffice here.
>
> **30** ENTER↑ **9.4** / ln **25** E **3** × returns 29 012.
>
> [We suspect that you may be on *Mt. Everest*[110] (29 028 *feet* in 1972).]

[108] **LOG** or **log** are frequently printed on 'scientific' calculators for the decadic logarithm. But this is mathematically ambiguous, so we avoided it (→ *ISO 80 000*, 2-12.5 and ...6; also → *https://physics.nist.gov/cuu/pdf/sp811.pdf*, p. 33, 10.1.2).

[109] *TN*: Emphases in these quoted examples were added by me.

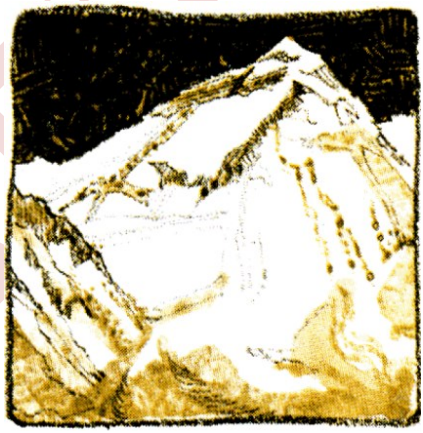
[110] *TN*: Called सगरमाथा (*Sagarmatha*) in Nepali and ཇོ་མོ་གླང་མ (*Qomolangma*) in Tibetan (pronounced like *Dschomolangma* in German and *Chomolungma* in English). Its Chinese name 珠穆朗玛峰 (*Zhūmùlǎngmǎ Fēng*) follows Tibetan pronunciation for centuries. Decolonialize!

Note the concise and factual style of this problem. The *HP-35* was a calculator made by engineers for engineers, and its manual was alike. Above case was reprinted in the *HP-45 OH* one year later. Thereafter, it underwent slight modifications:

Example (from the *HP-21 OH*):

Having lost most of his equipment in a blinding snowstorm, ace explorer *Buford Eugobanks* is using an ordinary barometer as an altimeter. After measuring the sea level pressure (30 *inches of mercury*) he climbs until the barometer indicates 9.4 *inches of mercury*.[111] …

This problem remained in subsequent manuals, just the explorers changed. A picture of the scenery was added in 1976, and not every snowstorm was worth mentioning anymore. Though two years later, more coherent units reached even such expeditions in the *Himalayas* – and also climate and methods changed:



Example (from *Solving Problems with Your HP Calculator*):

With most of his equipment lost in an avalanche, mountaineer *Wallace Quagmire* must use an ordinary barometer as an altimeter. **Knowing** the pressure at sea level is 760 *mm of mercury*, *Quagmire* continues his ascent until the barometer indicates 238 *mm of mercury*. Although the exact relationship of pressure and altitude is a function of many factors, *Quagmire* knows that **an approximation** is given by the formula:

$$\frac{altitude}{[\mathrm{m}]} = 7\,620 \times ln\left(\frac{760}{p/_{[mm\ of\ Hg]}}\right)$$

Where is *Wallace Quagmire*?

[111] *TN:* He lost his equipment, went to the seashore, and then climbed?! Proofreading??

Solution (with **FIX 0**):

760 [ENTER↑] **238** [/]

[ln] **7620** [×] returns 8 847.

Quagmire appears to be near the summit of *Mt. Everest* (8 848 m in 1978).

And it seems neither he nor his barometer returned from this expedition since this case did show up neither in the *HP-41C OHPG* nor later anymore. Perhaps something went wrong with the recalibration of his instrument?

For *SI* units, the formula reads

$$\frac{altitude}{[\mathrm{m}]} = 7620\ ln\left(\frac{1\,013}{p/_{[\mathrm{mbar}]}}\right) = 7620\ ln\left(\frac{101\,300}{p/_{[\mathrm{Pa}]}}\right)$$

Beyond the barometric scale, more logarithmic scales are used in science and engineering:

- in astronomy for the brightness of stars or
- in chemistry for the power of acids (pH); most popular may be
- the *decibel* (dB) in acoustics and electronics and
- the moment magnitude *scale* for earthquakes, for examples.

Example:

One of the strongest earthquakes observed in last decades was the one causing the devastating tsunami in the Indian Ocean near Indonesia in 2004-12, having a magnitude of 9.1. Another one near Japan in 2011-03 (with a magnitude of 9.0) led to another tsunami and the '*Fukushima nuclear accident*'. Compare them with the '*great San Francisco earthquake*' of 1906 with a magnitude of 7.9.

Solution:

The formula for comparing the energies released in 2 different earthquakes of known magnitudes reads

$$E_2/E_1 = 10^{\,1.5(M_2-M_1)}\,.$$

Again, no decimals are needed here – we can continue with the display settings as they are:

9.1 [ENTER↑] **7.9** [−] **1.5** [×] [10^x] returns 63. and

9 [ENTER↑] **7.9** [−] 1.5 [×] [10^x] returns 45.

So the energy released in said Japanese earthquake in 2011 was 45 times greater than the '*great San Francisco earthquake*'. And the earthquake in the Indian Ocean was even 63 times more intense.[112]

Even small numeric differences will gain significance when raised to powers. Human brains are not well equipped for such operations, so we recommend taking good care (and your *WP43*) in such cases:

Example:
What difference in magnitude will cause double destruction?

Solution (with **FIX 2**):

Rewriting the formula above results in $\Delta M = \frac{2}{3}\, lg\left({E_2}/{E_1}\right)$.

Thus, twice the energy released returns a magnitude difference of

2 [lg] **2** [×] **3** [/] equaling just 0.20 .

But there are also friendlier applications of logarithms, e.g., in electronics:

Example:
How many *bits* are required if the *unsigned integer* 3.75×10^9 shall be the maximum number to be handled by a microprocessor?

Solution (with **FIX 1**):

3.75 [E] **9** [EXP] **lb x** returns 31.8 , so 32 *bits* will suffice.

If we had a tri-state logic, however,

[RCL] [L] **3** **log$_x$y** returns 20.1 , so 21 cells would do.

---

[112] Taking into account that published magnitudes of earthquakes never show more than 1 decimal, we did not lose anything real setting *WP43* to **FIX 0** here.

For comparison: The asteroid impact at Chicxulub some 66 million years ago leading to extinction of the dinosaurs is estimated having caused an earthquake of magnitude 11.

Using the inverse of the general logarithm [log$_x$y], i.e. [y^x], your *WP43* allows for raising any positive *real* base to an arbitrary *real* power, as well as any negative *real* base to an arbitrary *integer* power, all returning *real* results (compare the *Mach-number* formula on p. 50).

Let's return to Fukushima once more for a really down-to-earth **example** applying logarithmic functions:

Locations at a distance of 30 km (!) to the nuclear power plant *Fukushima Dai-ichi* being devastated by the tsunami in 2011-03 showed radioactivity in the soil of some 1 ... 3 *megabecquerel* / m^2 corresponding to an annual radiation dose of 4 *millisievert* (mSv) in 2013 (see the map overleaf [113]). This was mainly caused by ^{137}Cs then, having a half-life of 30.2 *years*.[114]

To the best of our knowledge today, an unborn child shall not receive a dose of more than 1 mSv. So when will it be reasonably safe to let the evacuated population (some 150 000) of that area return to their homes finally?

Solution (with **FIX 0**):
Assuming there will be no further nuclear accident in that area, the caesium set free in 2011-03 stubbornly decay following the laws of nature. With a radioactivity a_0 at time zero, the activity a at a later time t will be

$$a = a_0 \times 2^{-(t/T_{1/2})} \Rightarrow t = T_{1/2} \times lb\left(\frac{a_0}{a}\right).$$

1 mSv in 9 months corresponds to an annual dose of $^4/_3$ mSv. The annual dose of 4 in 2013 divided by $^4/_3$ equals 3, and

3 [EXP] **lb** × **30.2** [x] returns **48.** *years*.

So you can recommend reproductive people shall rather not become pregnant living in that area before 2061.

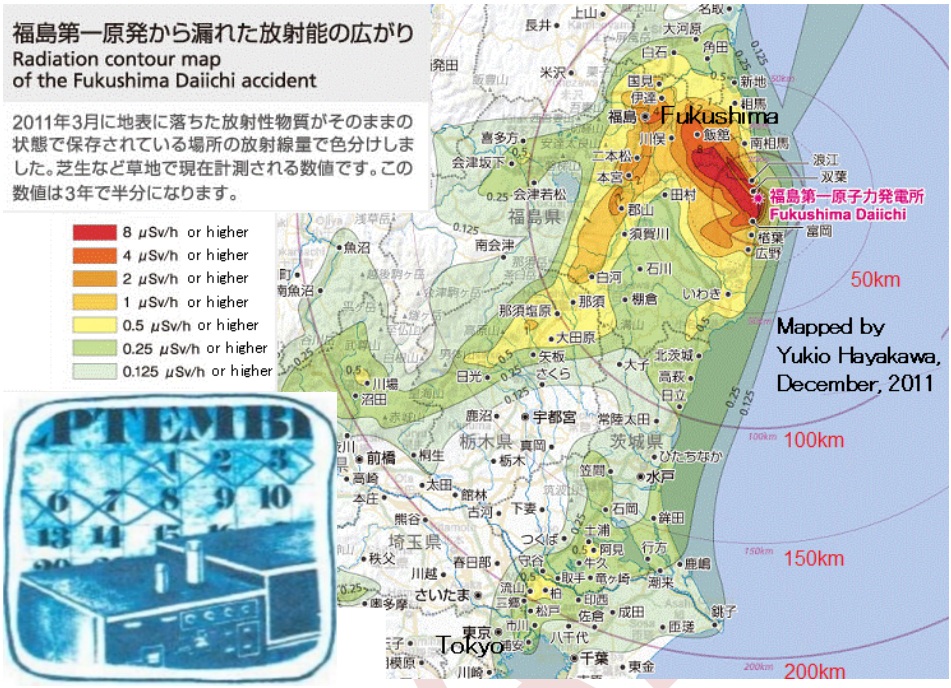
---

[113] Source: http://www.odonata.jp/ico2012/radioactivity/radiation_contour_large_map.gif. Unfortunately it disappeared years ago, though.

[114] In 2011-12, the radioactive iodine blown out of the reactor in 2011-03 had become irrelevant already – ^{131}I has a half-life of only 8 days. The caesium remained.

With a probability of 94.6%, ^{137}Cs decays emitting an electron with kinetic energy of up to 512 keV plus a γ-ray of 662 keV. This is for your information only, it doesn't affect the calculation here.

Natural radioactive background is (0.1 ± 0.08) µSv/h on average, strongly depending on location. (Note this is a nice example of a quantity being obviously not *Gaussian* (why?), but sources specify it this way nevertheless, → *ReMA I*.)



Senior inhabitants may return far sooner.[115]

---

[115] Note different limits are considered 'reasonably safe' for the public by authorities of different nations. By nature, all such limits are arbitrary to some extent since we talk about probabilities here, and there are mostly smooth transitions (→ chapter after next). Furthermore, a major fraction of today's world-wide knowledge about radiation damage in human bodies in the long range is still based on extrapolation of experiences collected since 1945 following 2 really large-scale events in Japan (and 67 more near Bikini until 1958). Another experiment well known by the public was started in the *USSR* in 1986 – also Belarus and the Ukraine have to bear the consequences until today (and for centuries to come, → *311f*). Mankind knows of the physics of radioactivity for some 120 years only so far, that's not long (just 4 half-lifes of ^{137}Cs).

There are further risks linked to agriculture in the Fukushima area – they are beyond the scope of this simple setup though.

Note above problem setting covers a worst case scenario. Actually, radioactivity is washed to deeper layers of soil with rain and time, reducing the activity seen at the surface. And there are mitigation efforts in that area (many km^2): at some places the contamination was washed off houses and trees, and the top layers of contaminated soil were removed, storing them (in big black plastic bags of 1 m^3 each) 'elsewhere'. Some 14 million m^3 are expected by the Japanese authorities – an area of 16 km^2 was planned for 'interim storage'. This radioactive soil shall be buried 15 m under, shielded from ground water. Cost of disposal were estimated to be 1.9×10^{12} ¥ by the Japanese

Above mathematically simple law links science and society quite closely.[116]

Similar considerations apply to nuclear waste: while nuclear power plants don't emit any carbon dioxide in operation, they produce *kilotons* of radioactive material decaying with half-lifes exceeding some thousand years (about 400 *kilotons* are present on our planet already, adding some 12 *kilotons* per year). This means you have to put that waste 'away' safely and reliably for really long times – a task kept under wraps for decades but not solved by waiting so far.[117]

Another (but safer) industrial **example** applying logarithms, dealing with mechanical transducers:

---

administration in 2019 (→ *Frankfurter Allgemeine Zeitung of 2019-03-10*).

Furthermore, 1.3 million m^3 of tritium-contaminated water are stored separately (equivalent to a cube of about 110 m edge length). These waters are mixed with sea water and flow slowly into the Pacific since September 2023 – and they will continue to do so for 30 years following. Tritium is radioactive hydrogen with a half-life of 12.3 *years*, emitting short range radiation (α-particles). Thus, you must not drink or inhale it. Check locally applicable radiation limits.

Success of the mitigation efforts mentioned may reduce the waiting time calculated above; failure will not extend it at least. Today is too early for a definitive assessment – we still know too little about long term effects of radiation in biology.

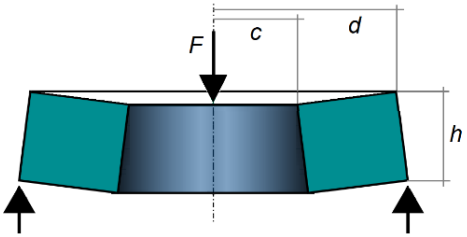
[116] Note the laws of nature apply as well to those who don't believe in them: These are the laws (read like *civil law*, not *common law*) – inevitably! E.g., radioactivity (once set free) decreases slowly: after 10 half-lifes you will still see $^1/_{1024}$ of the original activity (take 137 *gram* of ^{137}Cs, for instance, being 6.022×10^{23} atoms; after 302 *years*, 5.881×10^{20} of them will be present still). Although radioactivity decreases continuously, it may take very long to really vanish – it will just become less relevant with time.

[117] Mankind has absolutely no experience with hiding even smaller amounts of material safely and reliably for just a few millennia (look at the *Giza* pyramids or the *Valley of the Kings*, for instance). Note the final repository sites and the nuclear waste stored therein must also be tagged properly in a way staying comprehensible for millennia – no experience either.

Sad real-life example: A huge concrete coffin holds almost 90 million *liters* (i.e. 90 000 m^3 or a cube of 45 m edge length) of *US* nuclear waste on Eniwetok on the Marshall Islands. Now sea-level rise (caused by *anthropogenic global warming* which is evident today but hardly anybody was aware of 60 *years* ago) is eating away at the dome, and the *USA* are not interested in helping the tiny Pacific Ocean republic to do anything about it (→ *Los Angeles Times of 2019-11-10*). So much about taking responsibility. And 60 << 5000 *years*, so look forward to surprises in forthcoming centuries.

You might meet people talking about 'transmuting' the entire long-living radioactive waste to isotopes with shorter half-lifes by some nuclear reactions. Please ask them if this is physically possible for all that waste, and how much energy is needed for that entire trans-

To measure loads, you can use a bending steel ring supported at its outer radius ***d*** and loaded at its inner radius ***c*** with a force ***F*** (deformation is widely overstated in this sketch). The stress on its top or bottom face is [118]



$$\sigma = \frac{3}{\pi}\ F\ \frac{d-c}{h^2\ \left(\frac{d+c}{2}\right)\ ln\left(d/c\right)}$$

A reasonable stress level for precise measuring is 200 $\mathrm{N/mm^2}$ under full load. Assume you want to design a compact load cell for 5 000 kg featuring an outer radius ***d*** = 25 mm and choose ***c*** = 15 mm, how big shall be ***h*** ?

Solution: $h = \sqrt{\frac{3}{\pi} \times \frac{F}{\sigma} \times \frac{2\,(d-c)}{(d+c)\ ln\left(d/c\right)}}$

DSP 0 1
25 ENTER↑ 15 / ln
25 ENTER↑ 15 + x this is the divisor and …
1/x 25 ENTER↑ 15 −
x 2 x this is the entire 3rd factor.
5 E 3 x CONST $g_\oplus$ x
200 / 3 x π / √x returns 15.1 mm for ***h***. [119]

---

muting process. Compare with the energy 'produced' by nuclear power plants before.

As a matter of fact, the companies which made (and make) profits with those power plants for decades are very reluctant definitely solving the waste problem they created. They would not, certainly, if they expected making money from doing that. – Note that no insurance company on this planet offers any policy covering nuclear power plants.

Now you have got all the facts you need to assess that kind of nuclear (fission) power plants. Consider what you hand down to your descendants.

Nuclear fusion power plants differ. Therein, processes generating the solar fire shall be mimicked. Such plants are predicted being operational in some 40 years (but the same was claimed 50 years ago already). As far as told today, they will not produce any long-lived radioactive material. I will be most happy when this turns out being true.

[118] This formula was derived by the author developing industrial load cells in 1987, a time when first *FEM* software became available for *PCs*. It is field proven since then.

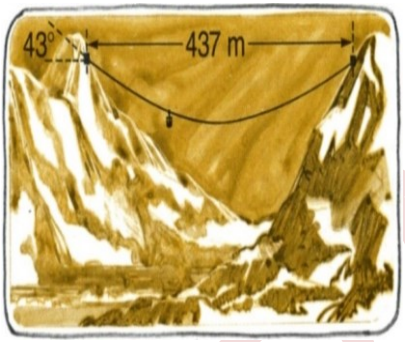
[119] The photo shows a cut through the body of such a load cell but featuring a bending ring of hexagonal cross section. Dimensions differ slightly. Properly heat treated and equipped, such stainless steel springs allow for measuring 5 000 kg with a display resolution (a.k.a. precision) better than 1 kg, legal for trade also in challenging environments.

Hyperbolic Functions

Hyperbolic functions tell us something about free hanging ropes, cables, chains, and the like. Your *WP43* provides 3 hyperbolic functions and their inverses in the 🟦-shifted rows of EXP and TRI:

sinh	*Hyperbolic sine*	**arsinh**	Inverse *hyperbolic sine*
cosh	*Hyperbolic cosine*	**arcosh**	Inverse *hyperbolic cosine*
tanh	*Hyperbolic tangent*	**artanh**	Inverse *hyperbolic tangent*

The following **example** is from the *HP-32 OH* – we modified it a bit:[120]



In *Upper Lagunia*, a tram[121] carries tourists between two peaks in the *Baruvian Alps* that are the same height and 437 *meters* apart. How long does it take the tram to travel from one peak to the other if it moves along its cable at 135 *meters per minute*? Before the tram latches onto the cable, the angle from the horizontal to the cable at its point of attachment is found to be 43°.[122]

Solution (assuming *SDC*):
The travel time is given by $t = \left({}^{d}/_{v}\right)\left[{}^{\tan\alpha}/_{arsinh(\tan\alpha)}\right]$

DISP FIX 2	will suffice here. Then …
43 TRI tan	is a result we need twice.
ENTER↑	duplicates it on the *stack* for numerator and denominator.
arsinh /	
437 ×	489.30 m is the length of the cable.
135 /	3.62, i.e. a bit more than 3 ½ *minutes*.

[120] The *HP-32* was *HP*'s first pocket calculator featuring hyperbolic functions. It was launched in 1978, while the *SR50* of *Texas Instruments* (*HP's* arch rival in those years of the so-called '*calculator wars*') provided hyperbolic functions 4 years earlier already. Actually, these *real* functions aren't needed frequently in real life.

[121] *TN*: British readers might frown here at least.

[122] *TN*: The picture will definitely not look as painted with a tram on the cable – check yourself! It should show the state before instead, with the tram in the station.

Probabilities – Factorials, Combinations, Permutations, and Distributions

Beyond Δ%, you find a lot more operations for probabilities in your *WP43*, going far beyond the *Gaussian* distribution (featuring all the preprogrammed functions implemented in *WP 34S* – and more). These operations are stored in the 3 adjacent *menus* PROB, STAT and FIT.

Example (from the *HP-32 OH*):

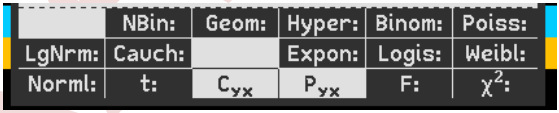
Willie's Widget Works wants to take photographs of its product line for advertising. How many different ways can the photographer arrange their eight widget models?

Solution:

The total number of possible arrangements is given by the *factorial*:

8 x! returns 40 320 for this number.

PROB includes C_{yx} and P_{yx} for *combinations* and *permutations*.



Example (continued):

The photographer looks through his viewfinder (in 1976) and decides that he can show only five widgets if his camera is to capture the intricate details of the widgets … How many different sets of five widgets can he select from the eight?

Solution:

The number of sets equals the number of possible *combinations* (i.e. the number of possible different sets of y different objects taken in quantities of x objects at a time; no object appears more than once in a set, and different orders of the same x objects are not counted separately here):

8 ENTER↑ **5** PROB C_{yx} returns 56 for this number.

Example (continued):
Again, different arrangements are feasible. How many pictures of different widget arrangements are possible within these limits?

Solution: [123]
The number of possible arrangements follows the statement above. Thus,

5 [x!] returns 120 for this number.

Then [×] returns 6 720 for the number of significantly different pictures.

This is the number of possible *permutations* of 5 items out of 8 (i.e. the number of possible different arrangements of y different objects taken in quantities of x objects at a time; no object appears more than once in an arrangement, and different orders of the same x objects are counted separately here). It can be obtained in 1 step by keying in

8 [ENTER↑] **5** [P_{yx}] returning 6 720

Furthermore, PROB contains 9 continuous and 5 discrete distributions for calculating probabilities, confidence intervals, etc.,[124] sharing a few features:

- Discrete (e.g., *Poisson*) distributions operate to *integers*. Summing up a *probability mass function* (*PMF*) $p(n)$ to get a *cumulated distribution function* (*CDF*), $P(m)$ starts at $n = 0$. Thus,

[123] *TN*: These challenging tasks changed the photographer drastically within 1 year.

[124] In a nutshell, discrete statistical distributions deal with (an integer number of) "events" governed by a known mathematical model. Such statistical events may be persons entering a shop, faulty parts appearing, radioactive nuclei decaying, etc. The *PMF* then tells you the probability to observe a certain number of such events, e.g., 7. And the *CDF* gives the probability to observe up to 7 such events then, but not more.

For doing statistics with continuous statistical variables – e.g., the heights of 3-year-old toddlers – similar rules apply: Assume we know the applicable mathematical model; then the respective *CDF* returns the probability for their heights being less than an arbitrary limit, e.g., 1 m. And the corresponding *PDF* tells you how these heights are distributed in a sample of let's say 1000 kids of this age.

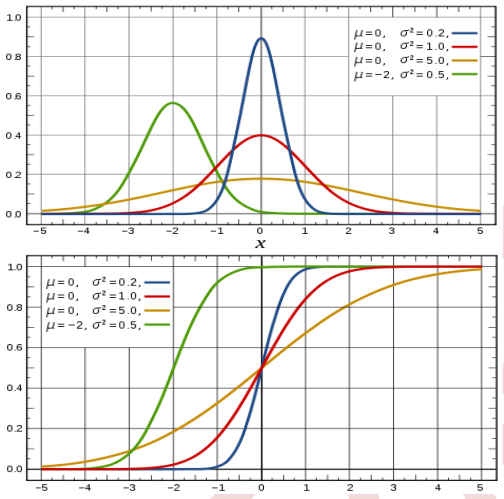
BEWARE: This is a very rudimentary sketch of this topic only – turn to a good textbook to learn dealing with statistics properly.

TNG: *PMF* und *PDF* = *Wahrscheinlichkeitsdichte*, *CDF* = *Verteilungsfunktion* bzw. *Wahrscheinlichkeitsverteilung*.

$$P(m) = \sum_{n=0}^{m} p(n)\,.$$

- Continuous (e.g., *normal*) distributions operate on *reals*. Integrating such a continuous *probability density function* (*PDF*) $f(x)$ to get a *CDF* works as

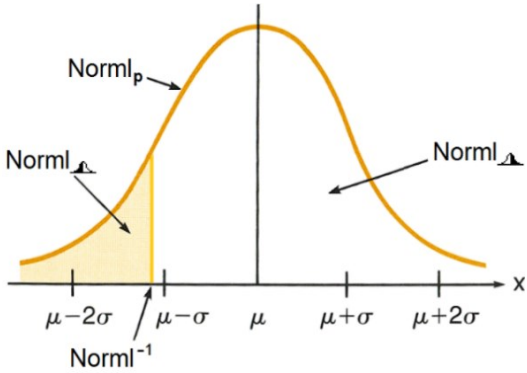
$$P(x) = \int_{-\infty}^{x} f(\xi)d\xi\,.$$



Many 'popular' continuous *PDF*s look more or less like the ones plotted in the upper diagram here. The lower diagram shows their corresponding *CDF*s using the same scale and colors.

Typically, any *CDF* starts at 0 with a slope of almost 0, becomes steeper then, and runs out at 1 with its slope returning to 0. This holds even if the respective *PDFs* don't look as nice and symmetric as the *normal distributions* plotted here (→ *ReMA I*). Thus, you will get the most precise results for the *CDF* on its left side using $\boldsymbol{P}$. On its right side, however, where $\boldsymbol{P}$ approaches **1**, the *error probability* $\boldsymbol{Q} = 1 - \boldsymbol{P}$ will be more precise. Thus, also $\boldsymbol{Q}$ is computed for each distribution on your *WP43*, independently of $\boldsymbol{P}$.

For an arbitrary *PDF* (e.g., **NORML$_{\blacksquare}$**, → picture overleaf), you will find the corresponding *CDF,* called **NORML$_{\blacktriangle}$** (returning $\boldsymbol{P}$), and **NORML$_{\blacktriangle}$** (returning $\boldsymbol{Q}$), as well as the inverse of the *CDF* (the so-called *quantile function*) called **NORML^{-1}**. This naming convention also applies to the *binomial*, *Cauchy* (a.k.a. *Lorentz* or *Breit-Wig-*



ner), *exponential*, *Fisher's F*, *geometric*, *hypergeometric*, *logistic*, *log-normal*, *negative binomial*, *Poisson*, *Student's t*, and *Weibull distributions*. Just the *chi-square distribution* is denoted differently following mathematical tradition. → PROB on p. *115* or *ReMS 2*.

Find applications of the *normal* distribution in the next 5 chapters, of *chi-square* on pp. *102* and *110f*, of *Student's t* on pp. *114ff*, of the *hypergeometric* on p. *119*, and of the *Weibull distribution* on pp. *119f*.

1D Statistics – Sampling Data, Calculating Means, Standard Deviations, and Confidence Limits

A wealth of commands for sample and population statistics is found in STAT, applicable in *1D* or *2D*. After initial clearing all statistical records by [CLΣ], use [Σ+] to accumulate your experimental data. Weighted data require the data value in **X** and its weight in **Y**, pairs of data or coordinates of data points shall be entered in **X** and **Y**. [Σ-] is provided for easy error correction: it will remove the last data point entered.

STAT contains data analysis functions as well: e.g., the *arithmetic mean* $\bar{x}$, *sample* and *population standard deviations (SD)* s and σ, and the *standard error* s_m (a.k.a. *SD of the mean*).

Example (from the *HP-32 OH*):

Norman Numbercruncher, a rising young math professor at *Mammoth University* (note the dress code of 1978), has developed a new test for measuring the mathematical abilities of college freshmen. To evaluate its effectiveness, he administers the test to the 746 students in *Calculus I*. Exhausted after grading the tests, *Numbercruncher* decides to randomly select 8 of the 746 tests and estimate the *SD* of all the scores from the sample of 8. The scores on the tests selected were 79, 94, 68, 86, 82, 78, 83, and 89. What *SD* does *Numbercruncher* calculate?

Solution:

[STAT] [CLΣ]

	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	FIT
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	HISTOG
	Σ+	$\bar{x}$	s	σ	s_m	SUM

0 [ENTER↑] **79** **Σ+** returns

```
001 data point                0.
                             79.
```

Σ+ takes x, stores it in STATS, displays feedback incl. *temporary information*, and disables *ASL* (→ f. 56). – Continue, overwriting x with each new entry:

94 **Σ+** **68** **Σ+** **86** **Σ+** **82** **Σ+** **78** **Σ+** **83** **Σ+**

89 **Σ+** returns

```
008 data points               0.
                             89.
```

[DISP] **FIX** **3**

s returns $s_x =$ 7.836, the *SD* estimated for the 746 students based on a sample of 8.[125]

When your data constitutes not just a sample of a population but rather all of the population, the *SD* of the data is the true population *SD* (denoted σ) … The difference between the values of s and σ is small, and for most applications can be ignored. Nevertheless, if you want to calculate the exact value of the population *SD* for an entire population, you can easily do so with just 1 keystroke on your *WP43*.

Example (continued):

Suppose the data from the previous example represented all the final exam scores from *Numbercruncher's* seminar on transcendental functions. Since this is the first time *Numbercruncher* has given this seminar, he wants to calculate the *SD* of the test scores to determine how good his exam was. *Numbercruncher* takes his calculator in hand, enters the data, then proceeds as follows

σ returns $\sigma_x =$ 7.330 for the *SD* for all scores on final exam. [125]

This is actually a no-brainer with your *WP43*. Thus, let's turn to something completely different:

Example:

Archibald is the undisputed champion of the *Golden Bow*, his archers club. In his standard exercise, aiming at a target disk of 1.5 m diameter at a distance

[125] *TN*: Actually, **Σ+** stores also y. But the *SD* for y data is irrelevant in this case.

of 50 m, his arrows scatter symmetrically around its center showing quite a small variance.[126] Actually, his records tell his arrows scatter with a long-term *SD* of 1 *foot* at said distance. Assuming his shots are distributed *normally* around the center of the disk, how often must he walk to collect an arrow in the green?

Solution (with **FIX 3**):

0 STO I		*Archibald's* mean = center of the disk.
1 U→ x: m ← foot	0.305	= 1 *foot* in *meters*, *Archibald's SD*.
STO J STO K		store this *SD* for later.
1.5 ENTER↑ **2** /	0.750	m = radius of the target disk.
PROB Norml: Norml$_\Delta$	0.007	= the error probability.
2 x 1/x	72.102	so *Archibald* has to collect an arrow in the green only once in 72 shots on long-term average.

Example (continued):

One of his buddies and competitors, *Bill*, also sent his arrows to the same target disk, with his hits scattering symmetrically around its center as well. He, however, has to pick up about 1 out of 15 arrows in the green on average. What is his *SD* in the target plane?

Solution:

15 1/x	0.067	, i.e. ca. 7% of *Bill's* arrows miss the target.
2 /	0.033	~3% misses on either side.
1. STO J x≷y		to get the *standardized normal* distribution.
Norml^{-1}	-1.834	, its corresponding lower limit.
+/- **.75** x≷y /	0.409	m = *Bill's SD*.
STO **0 0**		Just store it since we will need it again.
RCL K		Note that *Archibald's SD* is only ...
Δ%	-25.470	% narrower than *Bill's*, but his rate of misses is more than 5 times less.

[126] Many of our customers live in a country where long range weapons play a significantly greater role in public than in (almost all) civilized societies (feel free to check and assess irregular real-life consequences, e.g., in schools, churches, or shopping malls). Hence this case was chosen – though note I refrained from using their favorite firearms.

There are applications of this method in industry, where the scattering (a.k.a. variation) of a production process is compared with its tolerance limits. This may result in so-called *capability indices*, directly linked to the amount of scrap to be expected in the process investigated. Please consult applicable literature and standards – look for *process capability*.

Let's continue our **example**, guiding to more advanced statistics:

Bill quietly practiced in a *Zen* cloister during his summer vacation. Returning, he went to the *Golden Bow* immediately on next weekend and sent 50 arrows to his club's standard target disk. Only 2 missed, with 1 of them scratching the very edge of the disk. Cheers! But is this just a lucky chance success (within the usual statistical scattering of results to be expected)? Or is it probably correlated to his training efforts?

Solution:

Calculate *Bill's* new *SD*:

1.5 ENTER↑ **50** /	0.030	
2 /	0.015	= 1.5% misses on either side.
Norml^{-1}	−2.170	, the corresponding lower limit of the *standardized normal* distribution.
+/− **.75** x⇄y /	0.346	m = *Bill's* new *SD*.

Now, is this *significantly* better than his old *SD* (s_o)? Statisticians have found it is better (with a *confidence level* of 95%) if it is lower than the 95% *confidence limit* of s_o. Assume s_o was based on 60 shots, then the formula for its *single-sided lower 95% confidence limit* reads:

$$\sigma_L = s_o \times \sqrt{\frac{59}{\left(\chi^2_{59;\,0.95}\right)^{-1}}}$$

The denominator is the *inverse chi-square* for 95% probability and 59 *degrees of freedom*. Calculate inside out as usual:

59 STO I		the *degrees of freedom* must be stored in **I**.
.95 χ^2: $(\chi^2)^{-1}$	77.931	compute the *inverse chi-square*.
/	0.757	
√x	0.870	
RCL x **0** **0**	0.356	m for σ_L.

Looks like *Bill's* training made a difference! – – Wait, … with 95% confidence only … If we had required 99% instead, the lower *confidence limit* had become 0.337 m (you can verify this easily now); then *Bill's* new weekend result had been an insufficient indicator for a *significant* improvement.[127]

Frequencies and Histograms

For the evaluation of *1D* (or 2 independent) samples comprising ≥ 25 data points, you can categorize these data in bins and sensibly analyze their frequencies within these bins.

How to proceed for new data:

1. Press STAT **CLΣ** to clear all prior statistical data.
2. Enter your sample data into STATS using **Σ+** as usual.
3. Open **HISTOG**.
4. Within this *submenu*, choose **HISTOX** if the relevant data are in the 1st column of STATS (e.g., if you entered only x data) or **HISTOY** if they are in the 2nd column. The data will be copied into a new *matrix* called HISTO. Your *WP43* will propose values for the lower limit of the lowest bin, the number of bins, and the upper limit of the top bin.
5. Feel free to modify the values for **↓BIN**, **nBINS**, and/or **↑BIN**.[128]
6. Open **HPLOT** for plotting the histogram.
7. Feel free to fit a *Gauss* curve to your histogram by calling **HNORM**.

---

[127] Applying statistics may cause you having more doubts than without – but such is life: Doubts increase with knowledge – only braindead or lunatic people have never any doubts at all and may therefore feel (or be made feeling) giga gaga great most easily.

➡ Standard *confidence limits* and *levels* (also for indicating *significant differences*) may depend on the country, industry, or science you are working in. And the term '*significant*' is well defined in statistics – it might deviate from common language. → *Some Industrial Problems Solved*.

But *significance* isn't everything, *standard errors* may be a better measure if applicable (→ *ReMA J*).

Just check the pertinent valid standards before blindly (and lazily) copying exemplary calculations printed here.

[128] ***nBINS*** should be $\leq \sqrt{n}$, with ***n*** being the total number of your points. Default is $\sqrt{n}$.

Example (just a quick demonstration, using **FIX 0**):
Plot a histogram for the data points (x, y): 13, 13; 4, 5; 15, 12; 6, 4; 12, 13; 3, 2; 17, 15; 20, 17; 10, 10; 11, 10; 1, 1; 13, 11; 8, 8; 10, 8; 12, 10; 10, 9.

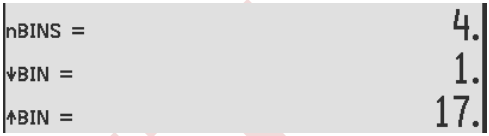
Solution:

STAT **CLΣ**

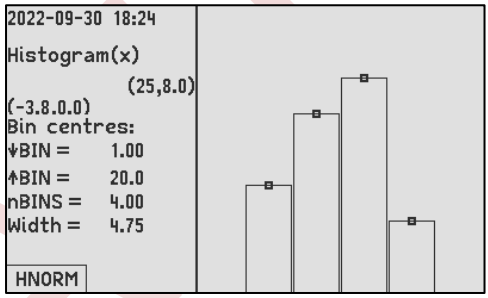
13 ENTER↑ **Σ+** **5** ENTER↑ **4** **Σ+** … until **9** ENTER↑ **10** **Σ+**

HISTOG

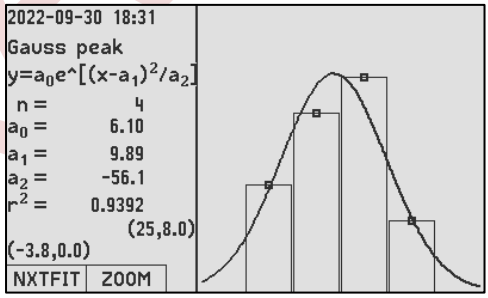
Choose **HISTOX** and you will get:



I.e. 4 bins between 1 and 17, each being 4 wide. To continue with these automatic parameters, open **HPLOT** and you will see:

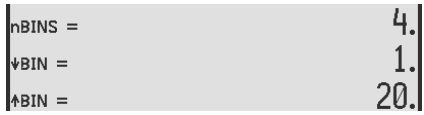


Let's check how this histogram compares with a *Gauss* curve. Just press HNORM and you will get:

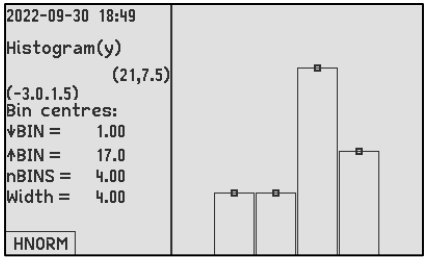


… looking quite well.

To do the same for the y column of your STATS data, EXIT this diagram and select **HISTOY**:



To continue with these automatic parameters, open **HPLOT** again:



Obviously, a *Gauss* curve will not be a good fit model for these rather asymmetric data.

2D Statistics – Curve Fitting, Interpolating, and Forecasting

FIT contains functions for curve fitting, featuring 10 different regression models (linear, exponential, logarithmic, power, root, hyperbolic, and more, → *ReM*), forecasts [x̂] and [ŷ], and the *coefficient of correlation* [r]. The fit model applied will be displayed heading numeric output after every related command (like **corr**, x̂, and ŷ). And in return of [L.R.], even its generic formula will be shown (→ examples below).

BestF tells your *WP43* to automatically select the regression model 'fitting your data best' (i.e. resulting in the biggest absolute *coefficient of correlation*). Then an elevated asterisk (*) will trail the name of the model chosen this way (please check the *IOI* for the 9 models available for this choice and the parameter of **BestF**). Like with all auto-functionality, you should know what you are doing here.

Example (from the *HP-27 OH*):

If *Galileo* had wished to investigate quantitatively the relationship between the time (***t***) for a falling object to hit the ground and the height (***h***) it has fallen, he might have released a rock[129] from various levels of the *Tower of Pisa* (which was leaning even then) and timed its descent by counting his pulse. The following data are measurements *Galileo* might have made:

t (*pulses*)	2	2.5	3.5	4	4.5 [130]
h (*Pisan feet*)	30	50	90	130	150

Unlike *Galileo*, you are equipped with a *WP43* – so what can you learn from this reported experiment? Let's look with **FIX 3**:

[FIT] **CLΣ**

CLΣ	ASSESS				PLOT
Σ-		s_{xy}	cov		HISTOG
Σ+	L.R.	s(a)	r	x̂	ŷ

[129] *TN:* Heaven beware! A big pebble had done as well. Just its impact on the ground must be heard – and it was more quiet in Pisa back then (far less tourists!).

[130] *TN:* These raw data showing half pulses don't look very plausible (*Galileo's* pulse had certainly not staid constant after pushing a rock up there), and actually it is dubious whether he made such experiments using the *Tower of Pisa* at all. But at least *HP* believed their calculator customers would love that story.

30 ENTER↑ 2 Σ+ returns

001 data point 30.000
2.000

Your next input after Σ+ will overwrite x and shift previous y up:[131]

50 ENTER↑ 2.5 Σ+ 90 ENTER↑ 3.5 Σ+ 130 ENTER↑ 4 Σ+
150 ENTER↑ 4.5 Σ+

005 data points 150.000
4.500

▼

GaussF	CauchF		BestF		
ParabF	HypF	RootF			
LinF	ExpF	LogF	PowerF		OrthoF

BestF 448 instructs your *WP43* to choose – out of all 2-parameter fit models provided – the one suiting these recorded data best.[132]

▼

CLΣ	ASSESS				PLOT
Σ-		s_{xy}	cov		HISTOG
Σ+	L.R.	s(a)	r	$\hat{x}$	$\hat{y}$

L.R.

Power* a_1 = 1.994
$y = a_0\ x^{a_1}$ a_0 = 7.723

Your *WP43* chose power regression as the mathematical model fitting these experimental data best. Let's check the *correlation coefficient*:

r returns Power* r = 0.998

This is an almost perfect correlation. To be precise, what are the estimated errors of the curve parameters (we are doing physics after all)?

---

[131] Thus, accumulating *2D* data will slowly overwrite the *stack* – if you want to save it, use STOSTK before accumulation and recall it by RCLSTK thereafter (→ example at the end of *OMS 4*).

TN: The mechanics of Σ+ were designed for *1D* data on the *HP-80, HP-45, etc.* They stayed this way also for *2D* data, supported from the *HP-32* on. Compare the first problem in the chapter about *1D* statistics.

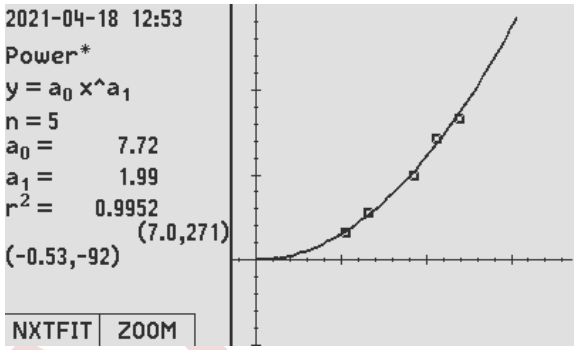
[132] → *IOI* for more about BestF.

s(a)

$s(a_1) =$ 0.080
$s(a_0) =$ 0.095

Hence, the equation expressing the overall results of these experiments best is $h = (7.72 \pm 0.09) \times t^{1.99 \pm 0.08} = 7.72(9) \times t^{1.99(8)}$ with ***t*** measured in *pulses* and ***h*** in *Pisan feet*.[133] See the plot produced by ASSESS and ZOOM showing the data points and the fit curve. *Galileo* could not know around 1600 yet, but we know today that

$$h = \frac{1}{2}\, g\, t^2.$$



Determining the size of a *Pisan foot* and *Galileo's* heartbeat frequency during his reported arduous experimental work with rocks (hope he had an assistant) is left as an exercise for the reader.

In addition, we found the following **example** in various *HP* calculator manuals of 1976 - 78. It reads typical for the thinking at that time:

Big *Lyle Hephaestus,*[134] owner-operator of the *Hephaestus Oil Company*, wishes to know the slope and y-intercept of a least squares line for the consumption of motor fuel in the *United States of America*[135] against time since 1945. He knows

[133] This equation shows 2 alternative ways for specifying and printing uncertainties.

[134] *TN*: Perhaps his ancestors emigrated from Greece: *Hephaistos* was the ancient Greek god of fire, forging, and technology in general. Sources tell he even designed a rotating space station orbiting the Earth, looking like *Kubrick's* in '*2001*' (another evidence that ancient Greek knew the Earth being a sphere – that knowledge got lost for many centuries).

[135] *TN*: Compare *los Estados Unidos Mexicanos*. But I bet you have guessed the location by the weird unit. Note a *barrel* of oil deviates from a barrel of whiskey and a barrel of whisky, of course. And don't forget the barrels of vine and beer. Also the chemical industry uses barrels. So many different volumes! Now you know why some islanders use *barrels* for measuring. Though there's help in *OMS 6*.

the data given in the table:

Motor fuel demand (millions of *barrels*)	696	994	1330	1512	1750	2162	2243	2382	2484
Year	1945	1950	1955	1960	1965	1970	1971	1972	1973

Solution (continuing with **BestF 448** but **FIX 1**):

Hephaestus could draw a plot of motor fuel demand against time. However, with his *WP43*, *Hephaestus* has only to key the data into the calculator using … [Σ+] …, then press [L.R.].

[FIT] [CLΣ]

696 [ENTER↑] **1945** [Σ+] [136] **…** **…** **2484** [ENTER↑] **1973** [Σ+]

L.R.

Linear* a_1 = 61.2

$y = a_0+a_1x$ a_0 = −118 290.6

Your *WP43* selected linear regression as the mathematical model fitting these data best. Let's check the *correlation coefficient*:

r returns Linear* r = 1.0

Perfect! Based on this good correlation result, *Hephaestus* confirms the automatic choice and is even tempted to extrapolate the computed trend of motor fuel demand to (then) future years.

Example (continued):

If *Hephaestus* wishes to predict (in 1976) the demand for motor fuel for the years 1980 and 2000, he keys in the new ***x*** values and presses [ŷ].

1980 [ŷ] returns Linear* ŷ = 2 808.6

2000 [ŷ] returns Linear* ŷ = 4 031.9

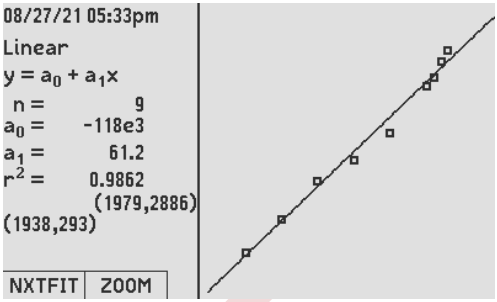
Similarly, to determine the year that the demand for motor fuel is expected to pass 3 500 million *barrels*, *Hephaestus* keys in **3500** (the new value for ***y***) and presses [x̂].

3500 [x̂] returns Linear* x̂ = 1 991.3

– so he expected the demand to pass 3.5 million *barrels* in 1992.

[136] *TN*: The boss computes himself! And he even knows how to do it properly! Seems that was a time before general managers, *CEOs*, and big staffs became fashionable.

Though if he had plotted his data (which was impossible on his calculator at that time – but a plain sheet of quadrille paper had done), *Hephaestus* could have been warned looking at his last 3 data points.[137]



Another **example** (from the *HP-15C OH*):

Agronomist *Silas Farmer* has developed a new variety of high-yield rice, and has measured the plant's yield as a function of fertilization. Use the [Σ+] function to accumulate the data below to find the values for Σx, Σx^2, Σy, Σy^2, and Σxy for nitrogen fertilizer application (x) versus grain yield (y).

x	Nitr. applied (kg/ha)	0.0	20.0	40.0	60.0	80.0
y	Grain yield (t/ha)	4.63	5.78	6.61	7,21	7,78

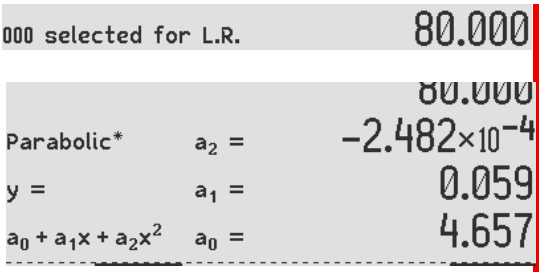
Solution (with FIX 3):

We are confident you can accumulate the data yourself using [Σ+] and recall the sums from [Σ] now. Let's proceed to the regression *Silas* is really interested in:[138]

[FIT] [▼] **BestF** **000** returns

[▼] **L.R.** returns

Out of all fit models available, your *WP43* chose the parabolic model.[139] Plotting the data with the fit curve by

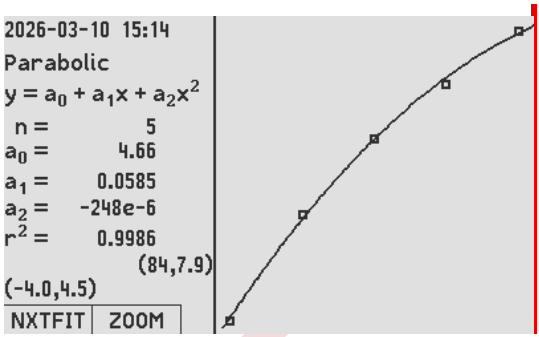


[137] *TN*: Information may be lost easily when dealing with numbers only. This problem turns out being a suboptimal choice for extrapolating without plotting – actually *HP* printed an inadvertent warning of thoughtless number crunching.

[138] *TN*: Hardly any real-world agronomist would be interested in the sums *HP* asked for here, and also mean and *SD* were computed thereafter for demonstration of the respective *HP-15C* functions only.

ASSESS shows an excellent correlation. So you can use it for forecasting how much grain is expected with 70 kg/ha fertilizer:

70 $\hat{y}$ returns 7.537 t/ha.



Chi-Square Statistic – Checking Expectations

The *chi-square statistic* measures the goodness of fit between two sets of frequencies.[140] It's used to test whether a set of observed frequencies differs from a set of expected ones sufficiently to reject the hypothesis under which the expected frequencies were obtained.

In other words, you are testing whether discrepancies between the observed frequencies (O_i) and the expected frequencies (E_i) are *significant*, or whether they may reasonably be attributed to chance. The formula generally used is

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

If there is a close agreement between the observed and expected frequencies, χ^2 will be small. If the agreement is poor, χ^2 will be large.[141]

---

[139] *TN*: *Silas'* problem was originally solved with linear regression since the *HP-15C* can't offer anything better here. That returns $r^2 = 0.976$ for the correlation coefficient.

Above curve looks more realistic since showing a tendency towards saturation.

[140] *Goodness of fit* tells how good both sets match, e.g., observed experimental data and a theoretical model. This text and the following example are quoted from the *HP-27 OH*.

[141] *TN*: Do not confuse this χ^2 defined here with the χ^2 distribution mentioned above. They are different! Unfortunately, however, both chi-squares are called and spelled identically. Seems the mathematical naming commission was inattentive here at the crucial time, perhaps distracted by discussing the outcome of a recent casino visit (note probability theory and statistics were invented for comprehending gambling).

Example:

A suspect dice from a Las Vegas casino is brought to an independent testing firm to determine its bias, if any. The dice is tossed 120 times and the following results obtained:

Number	1	2	3	4	5	6
Frequency	25	17	15	23	24	16

Solution (with **FIX 0**):

The *expected* frequency is $^{120}/_{6} = 20$ for each number here. For calculating χ^2, just enter:

25 ENTER↑ **20** − EXP x^2	25
17 ENTER↑ **20** − x^2 +	34
15 ENTER↑ **20** − x^2 +	59
23 ENTER↑ **20** − x^2 +	68
24 ENTER↑ **20** − x^2 +	84
16 ENTER↑ **20** − x^2 +	100
20 /	5

Now, is this χ^2 large or small? Statisticians have found it is to be considered 'small' if χ^2 is less than the value of the inverse χ^2 *CDF* for the *degrees of freedom* (here $n - 1 = 5$) and *confidence level* applicable (here 95%). As observed above already, also this χ^2 function is provided in your *WP43*. Simply key in:

5 STO I	5 for the *degrees of freedom*;
.95 PROB χ^2: $(\chi^2)^{-1}$	11.

Since 5 is less than 11, χ^2 is small enough to conclude that this dice is fair (with 95% confidence). [142]

[142] *TN*: A *confidence level* of 95% corresponds to 5% *error probability*. These values were typical settings for the *USA*, as in the vintage problem quoted here. For comparison, in Germany 99% / 1% are standard. Choose yourself. See further examples below.

Such *chi-square tests* may be also applied to entire curves measured: then every data point of the curve shall be compared with the value predicted by some theoretical model at the same position.

TNG: *Confidence limit = Vertrauensbereichsgrenze, confidence level = Vertrauensniveau,.*

Some Industrial Problems Solved

To get an idea of real-life opportunities covered by your *WP43* and of some constraints inherent to statistics, see the 5 applications shown below. All of them are demonstrated here using the *4LS* but work with the *8LS* as well, of course.

Application 1 (quick and easy *MSA*):

Whenever you want to know something about specific properties of an object, you shall measure it. Therefore, first knowing the properties of the measuring instrument (or system) you are going to use is essential.

Example:
Your colleagues in *R&D* have happily specified that particle accelerator beam pipes made of a special stainless steel shall have a magnetic susceptibility ≤ 0.01 also at their welding seams.[143] How can you verify whether the susceptibility meter available in your lab is sufficiently precise (a.k.a. '*accurate enough*') to control series production of such pipes? Or is it inevitable to invest in a better instrument?

Solution (measure the real-life precision of your measuring system):

1. Collect ≥ 30 metal samples covering the susceptibility range you are interested in (this range could extend, e.g., from 0. to about 0.02 here).[144] Mark each sample unambiguously (e.g., by numbering it). Prepare a table with as many numbered rows as you have samples.
2. Use the measuring system under investigation to measure each sample carefully under controlled conditions. Write each measured value next to the respective sample number; record as many decimals as possible.
3. Measure all samples a 2nd time under the same conditions, but in another sequence (just shuffle the samples). Do not allow for looking at the 1st data

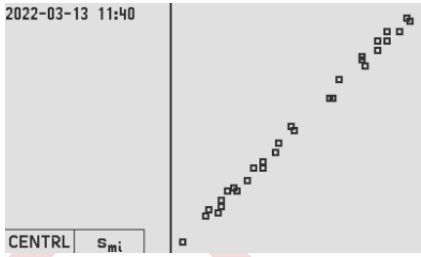
[143] Specifications and demands are written easily – actually verifying them in a production environment may be a real challenge, however. Check if your means suffice!

[144] Nature allows for positive susceptibilities only.

➡ It is not required to know the exact susceptibilities of your samples beforehand: they shall just fall in said range, cover it fairly homogenously (→ the plot overleaf), and must stay constant in your measurements. No expensive gauges needed here – real life has proven various pieces of stainless steel scrap sufficed. But 30 samples are the minimum required for this measurement of instrument precision.

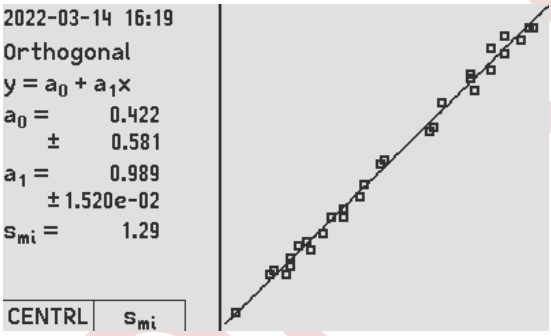
measured (hiding them is essential if you acquire the values manually)! Open a 2nd column and record each 2nd measured value in the row carrying the respective sample number again.

4. Get your *WP43*. Press [FIT] [CLΣ] to clear all statistical data. Then enter all pairs of values using [Σ+]. The 1st measured value shall be y, the 2nd x – thus, input will be ***mv1*** [ENTER↑] ***mv2*** [Σ+] for each sample (instead, you can provide your data in a *matrix*, → 180).



5. Call [ ] [PLOT] to plot all these measured points in a new window. The plot should look like an ant trail along the diagonal – see a real-world plot here →

6. Press [CENTRL] to fit an orthogonal regression line to your data points. Check if this deviates *significantly* from the diagonal $y = x$. Coarsely, this applies (with a confidence of some 95%) if $a_0 \pm 2\,s(a_0)$ excludes **0** or $a_1 \pm 2\,s(a_1)$ excludes **1**.



7. If true then your system may be running up still (wait some more time for warmup) or show an unstable zero (check and fix); then return to step 2.[145]

Else (i.e. if $a_0 \pm 2\,s(a_0)$ includes **0** and $a_1 \pm 2\,s(a_1)$ includes **1**) press [s_{mi}] to compute the precision of your measuring instrument investigated. This measured precision applies under the boundary conditions prevalent during steps 2 and 3 of this test and is definitely more realistic than catalog values stated by manufacturers.[146]

Then enter **30** [×] [1/x] and multiply with the tolerance you have to control (**0.01** in this case). A result **≥ 1** will tell you that this measuring system may be used for controlling production within said tolerance (i.e. it is a *capable instrument* for this job) under said conditions. Else you shall strive for a more precise device, better measuring conditions, or a wider tolerance (→ *ReMA I*).

[145] If the gross shape of your scatter plot deviates fundamentally from the one shown here, however (like the ants don't know where to walk to), this may point to surprises hidden in your setup or data acquisition system (→ *ReMA I*).

[146] Although displaying all measured points clearly separated might exceed the resolution of a small pocket calculator screen by far, the calculations in steps 6 and 7 carry full precision, and the resulting s_{mi} will be correct.

Application 2 (scrap rate, confidence limits):

Assume you own a little tool shop producing axle pins automatically in series, and want to know the quality of the parts produced after setup of the machine. You drew a *representative sample* of pins (all being nominally equal parts!) and measured their actual sizes precisely using a *capable* micrometer (→ Application 1).

Example:

A batch of turned pins is produced on a precision lathe. A representative sample of 10 pins is drawn and their diameters measured with a *capable* gauge: 12.356, 12.362, 12.360, 12.364, 12.340, 12.345, 12.342, 12.344, 12.355, and 12.353 mm. From earlier large scale investigations, you know that diameters from this production process follow a *normal* distribution.

How do you know your batch will be ok? What pin diameters will you get in larger scale batch production? Statistics can't tell you about all of them but it will tell you where to expect almost all (e.g., 99%) of them.

Solution:

DISP FIX 3

STAT CLΣ

0 ENTER↑ 56 Σ+ [147]

001 data point 0.000
56.000

Continue accumulating the remaining measured sample data:

62 Σ+ ...

53 Σ+

010 data points 0.000
53.000

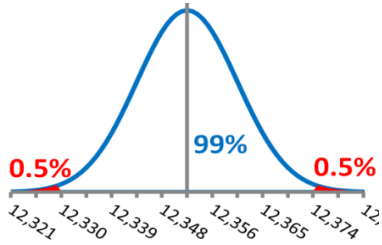
Knowing these pins are drawn from a *Gaussian* process, you get the best estimates for mean and standard deviation of your batch by

[147] Experimental data sets may consist of a series of ***x***-values (or ***y***-values) differing from each other by a relatively small amount (like here). Then you can save considerable input time by typing only the differences between each value and a chosen convenient offset ***q***. This ***q*** must then be added to computed results of $\bar{x}$, $\hat{y}$, $\hat{x}$, and to a_0 (of **L.R.**) later on.

Thus, you can simply enter **56 Σ+**, **62 Σ+**, etc. above. Computing $\bar{x}$ afterwards, divide the answer by **1000**, and add **12.3**. The computed *SD* shall be divided by **1000** as well.

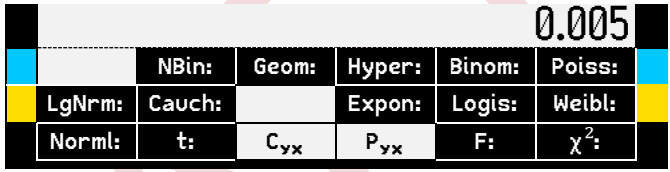
$\bar{x}$ 1000 / 12.3 +	$\bar{x}$ = 12.352
STO I	
s 1000 /	s_x = 0.009
STO J	We store $\bar{x}$ and s_x here for the next steps already.

If 99% of such a batch is found inside some symmetric limits of a *Gaussian* process then 0.5% will be out on either side.

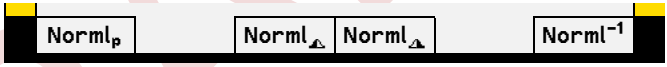


Based on the 10 pins analyzed, you may expect 0.5% of all pins with diameters less than

.005 PROB



Norml:



$Norml^{-1}$	12.330

and another 0.5% with diameters greater than

.995 $Norml^{-1}$	12.375

If you should observe significantly more than 0.5% of your pins beyond either limit, this indicates your process may be running out of control.

Assume these pins shall have a nominal diameter of 12.35. Then – based on this sample analysis – you can safely commit holding a tolerance of ± 0.05 (you will hardly produce any scrap as long as your process continues running the way you found it). If your customer would try, however, forcing you to deliver a tolerance of ± 0.02, you must expect some losses:

12.35 ENTER↑ ENTER↑	12.350	
.02 −	12.330	= lower limit.
$Norml_{\triangle}$	0.006	= lower scrap = 0.6%.
x⇄y	12.350	
.02 +	12.370	= upper limit.

Norml↗	0.021	= upper scrap = 2.1%.
+	0.026	= total scrap.

But what will hurt you even more than these 2.6% scrap you must expect now (i.e. more than 1 out of 40 pins) will be the inevitable necessity to set up, establish, and maintain a very precise and reliable sorting device to warrant only pins within 12.350 ± 0.020 will pass to your customer. Thus, stay firm (if you can afford it) and refuse that customer request to constrict your tolerance limits – it may well be you can't afford becoming weak here: the best and most cost effective parts filter in production is the one you don't need.

Are you interested in the mean pin diameter of your batch? So you know how much space you must provide to store a stack of, e.g., 50 pins? Determine the applicable mean and its variation; then use them to find both upper and lower limit confining the mean with a probability of, e.g., 95%.

Example (continued):

Since we drew the sample out of a *Gaussian* process, the *arithmetic mean* is applicable, the *standard error* indicates its variation, and *Student's* ***t*** is required for confidence limits. For the latter, we need its *degrees of freedom*:

Σ n	10.000	recall the number of data points.
1 − STO I	9.000	store the *degrees of freedom* (for computing t^{-1} below)
STAT s_m	0.003	is the standard error.

Having 95% inside means having 2.5% outside at either end (→ previous diagram).[148] Thus, one must generally take 0.025 and 0.975 as arguments in two subsequent calculations using the *quantile function* to get both 95% limits below and above the sample average:

.025 PROB t: $t^{-1}(p)$	−2.262
×	−0.006
STAT $\bar{x}$ x⇄y R↓ [149]	12.352

[148] The value of 95% is called the *confidence level* of this calculation. In this case, you calculate the 95% *confidence limits for the mean value*.

Instead of 95%, also 99% are frequently applied. We recommend checking the applicable valid standards before blindly copying any calculations from here. And feel free to apply other *confidence levels* wherever they suit your needs and you can justify them.

x≷y	-0.006	
+	12.346	= lower limit.
RCL L	-0.006	= last x.
2 x −	12.358	= upper limit.[150]

Now you know what to expect for the future average diameter of such batches. Hence a rail being

50 x 617.900 long inside will hold 50 pins in 97.5% of all cases.

12.346 and 12.358 are the *95% confidence limits* of the mean calculated above. So there is a chance of 2.5% that the mean will be < 12.346 (don't bother here) and an equal chance that it will be > 12.358. These chances[151] are inevitable consequences of the fact you know something about a small *sample* only (drawn out of a large *population*), but want or have to tell something about said entire *population*.

If you can't live with these uncertainties or the widths of the confidence limits, don't blame statistics but draw a larger sample or collect more precise data instead. At the bottom line, improving precision and stability of production processes is the key.

Application 3 (significant change):

Assume you have drawn a representative sample out of an arbitrary industrial process at *day 1*. Then you have changed the process parameters, waited for stabilization, and have drawn another representative sample of same size at *day 2*.[152] Being serious, you have meticulously measured and recorded a critical quantity (e.g., a characteristic dimension) for each specimen investigated at both days. Now, is there any

[149] $\bar{x}$ returns both $\bar{x}$ and $\bar{y}$. Only $\bar{x}$ is interesting here, so pressing x≷y R↓ moves $\bar{y}$ quickly out of the way. In a program, STK DROPy will be a better alternative – 1 step only (→ *OMS 4*).

[150] The upper *confidence limit* can be calculated so easily since $t^{-1}(p)$ is symmetric around the mean value. Else you had to repeat above calculation for an input of 0.975.

[151] Statisticians call these chances '*probabilities of a type I error*' or '*probabilities of an error of the 1st kind*'.

TNG: *Type I error = Fehler 1. Art.*

[152] There may well have been a longer time interval between both sampling days.

significant difference?

The following 3-step test may save you unwanted embarrassment in your next presentation or after your next publication:

1. Accumulate your sample data. Then let your *WP43* compute the mean values and *standard errors* for both samples,[153] and their *normalized distance*

   $d = |\overline{x} - \overline{y}| \,/\, \sqrt{s_{mx}^2 + s_{my}^2}$. Then the calculation could look like this:

   STAT s_m returns both standard errors in **X** and **Y**.

   EXP x^2 x≷y x^2 + $\sqrt{x}$ so this is the entire denominator.

   STAT $\overline{x}$ returns both $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$.

   − |x| thus, this is the numerator and …

   x≷y / STO D this is ***d*** (choose another destination if SSIZE8 is set).

2. Let your *WP43* compute the critical limit t_{cr} of *Student's t* for f *degrees of freedom* and a probability of 97.5% now:

   Σ n recall the number of samples measured.

   1 − STO I store the *degrees of freedom* f for *Student's* ***t***.

   .975 PROB t: $t^{-1}(p)$ as mentioned above, the requested *quantile function* lives in PROB. It takes the *degrees of freedom* stored in **I** to get t_{cr}.

   If $d < t_{cr}$ then this test indicates the difference between both samples is due to random deviations only. Congratulations – you have got a resilient process regarding the parameters you changed!

   Else continue.

3. Let your *WP43* compute a new critical limit t_{cs} for f and 99.5%:

   .995 $t^{-1}(p)$ get t_{cs}.

   If $d \geq t_{cs}$ then this test indicates a *significant difference* between both samples. Congratulations – your parameter change caused a significant effect!

   Else (i.e. for $t_{cr} \leq d < t_{cs}$) you simply cannot decide seriously based on the information you have – your samples may contain too little data, your measurements were not precise enough, the

[153] This test assumes your samples were both drawn from a *Gaussian* process which holds frequently in real life but shall be verified.

process observed is scattering too far, or whatever else. Though don't let your excited audience lead you in temptation! Keep silent or mumble something like "*investigation in progress*" at the utmost – management might want a quick answer but there is no solid basis for it (yet).

Application 4 (operating characteristics):

Assume you draw a representative sample of 20 parts out of a production batch of 100 parts and check this sample thoroughly. What is the probability ***P*** to find at least 1 random defect in such a sample if the overall probability for a defect in such a batch is known to be 5% (or 1%), for instance?

This is almost a textbook **example** for applying the *hypergeometric* distribution. P(***n*** ≥ 1) = 100% – p(***n*** = 0). Thus, the solution is:

DSP 3		
100 STO K	batch size	
20 STO J	sample size	
0.05 STO I	5% overall defect probability	
0 PROB Hyper: Hyper	returns	0.319 for p(***n*** = 0)
1 − +/−	returns	0.681 for P(***n*** ≥ 1)
0.01 STO I	1% overall defect probability	
0 Hyper	returns	0.800 for p(***n*** = 0)
1 − +/−	returns	0.200 for P(***n*** ≥ 1)

Even with 5% defects in the batch, the odds are about 1 out of 3 that no defect at all is detected in such a relatively large sample. Such sample tests are simple, hence popular, but definitely inadequate for controlling industrial processes with overall (target) defect probabilities less than some 1%.[154]

Application 5 (lifetime of punching dies):

Assume you are responsible for a workshop punching lots of small precision parts out of sheet metal. The dies you use, manufactured in batches by

[154] Traditionally, such kind of problems were solved *approximately* using *Poisson* or *binomial* distributions since these mathematical models are significantly easier to calculate. But you are equipped with your *WP43* which does the drudgery for you – be happy!

your supplier to the state of the art, will only withstand a limited number of some 100 000 punching cycles. For each batch supplied, your technicians meticulously record the number of cycles performed until a die breaks.

Example:

The following cycle numbers were recorded for a batch of 20 dies (sorted according to their lifetimes): 37 974, 51 341, 62 590, 77 832, 79 637, 106 144, 110 631, 112 436, 130 502, 156 003, 166 979, 167 335, 183 419, 205 518, 213 993, 252 171, 254 617, 301 190, 322 296, and 386 315. Use these data to derive a characteristic lifetime for this batch of dies.

Solution:

Such lifetime data follow a *Weibull distribution*. For evaluating above record, we shall compute summed transformed 'frequencies' y_i with $n = 20$ and i counting the dies:

$$y_i = ln\left\{ ln\left[\frac{1}{1-\left(\frac{i-0.3}{n+0.4}\right)} \right] \right\}$$

Solving this formula cries for a program. Programming will be covered in *OMS 4* – here just press [GTO] [.] [.] [P/R] and key in:

[LBL] [α] [√x] [×] [e^x] [9] [ENTER↑] which will be echoed **LBL 'FREQ'**.

.3 [−] **20.4** [/] **1** [x≷y] [−] [1/x] [ln] [ln] this is the entire expression.

[RTN]

Press [P/R] again to return to *RUM*.

Set [DISP] **FIX 3**.

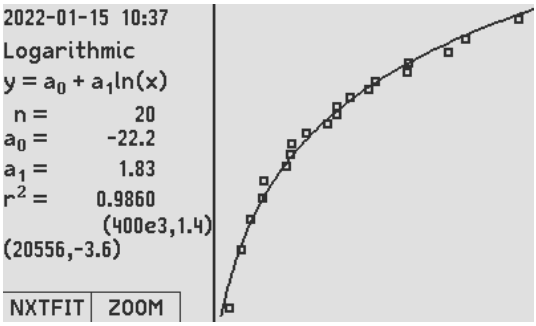
Now enter **1** [XEQ] **PROG** [FREQ] – you will get y_1. Repeat for 2 to 20. Results will be:

i	1	2	3	4	5	6	7
y_i	−3.355	−2.442	−1.952	−1.609	−1.340	−1.116	−0.921

i	8	9	10	11	12	13	14
y_i	−0.747	−0.587	−0.438	−0.297	−0.160	−0.026	0.107

i	15	16	17	18	19	20
y_i	0.243	0.384	0.535	0.704	0.910	1.216

Call [FIT] **CLΣ**. Then enter these y_i and the corresponding numbers of cycles c_i via [Σ+] into STATS as usual. Plotting these twenty data points (c_i, y_i) over



a logarithmic cycle scale should return a straight line.

Instead, let your *WP43* perform a logarithmic curve fit through these data points. Select **LogF** and call **ASSESS** – it will return a plot as shown here.

The resulting *characteristic life-time* of this batch of twenty dies is

0 $\hat{x}$

Logarithmic $\hat{x}$ = 191 185.

After tool composition or manufacturing processes were refined at your supplier, you receive a new batch of dies. Based on a record of their lifetimes, you can then assess the benefit of these changes if there is any (→ *Weibull distribution* in the *IOI*).

➡ You must not use neither $\bar{x}$ nor **s** here.

STAT and FIT contain many more statistical functions (like *covariances*, means and standard deviations for weighted data, *geometric means,* etc.) – just look them up there and check their entries in the *IOI*.

More examples of statistical applications can be found in the manuals of various vintage *HP* calculators, especially of *HP-27* and *HP-21S*.

We sincerely recommend consulting a good statistics textbook for more information about statistical methods in general, the terminology used, the mathematical models and their constraints, before actually applying them.

➡ Most methods make only sense from a certain minimum sample size on (e.g., percentiles and histograms).

Summary of Functions Operating on Reals

Your *WP43* provides many more than the numeric functions shown on pp. *22ff*, *34ff*, and *81ff* in various applications and examples – all the *real number* functions are listed below:

- **General mathematics:**
  - *Monadic* functions:

    [1/x], [|x|], [+/-], [√x] and [x²], [∛x] and [x³], [√(1+x²)], [2^x] and [lb x], [e^x] and [ln], [10^x] and [lg], [sin], [cos], [tan], and their inverses work as shown above and you learned in school (→ *128ff* for more information about angular I/O),

    for [sinh], [cosh], [tanh], and their inverses compare pp. *95f*,

    [e^x-1] and [ln(1+x)] return more accurate results for $x \approx 0$,

    [ceil] returns the smallest integer $\geq x$, while [floor] returns the greatest integer $\leq x$,

    [SDL] ***n*** ([SDR] ***n***) shifts digits left (right) by ***n*** decimal positions, equivalent to multiplying x times (dividing x by) 10^n (feel free to use SDL and SDR in analogy to the *short integer* commands ASR, SL, and SR, → *191ff*),

    [(-1)^x] returns $cos(\pi x) + i\, sin(\pi x)$ for non-integer x (→ *152ff*);

    [#] converts a closed *real* to a *short integer* of base 2…16 (→ *OMS 3*); [→BIN], [→OCT], [→DEC], and [→HEX] cover the most popular bases.

  - *Dyadic* functions:

    [+], [−], [×], [/], [y^x], and [∜y] work as was shown above; use

    [IDIV] for integer division (and [IDIVR] if you want also the remainder returned in Y),

    (**Example:** **7.8** [ENTER↑] **3.2** [INTS] **IDIV** returns 2)

    [log_x y] for the logarithm of y for the base x

    (**Example:** **655.5** [ENTER↑] **5** [EXP] **log_x y** returns 4.029 6…),

    [RMD] for the remainder of y/x and [MOD] for $y \bmod x$ (→ *194* for examples),

    [max] (or [min]) for the maximum (or minimum) of x and y; and

〔||〕 returns $\left(\frac{1}{x}+\frac{1}{y}\right)^{-1}$ for $x \times y \neq 0$, else **0**, being handy in electrical engineering in particular.

- *Triadic* functions:

  〔×MOD〕 returns $(z \times y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$;

  〔^MOD〕 returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$.

- **Isolating and separating specific parts of numbers** (all operations but 〔x|〕 contained in PART): Use the monadic functions …

  〔EXPT〕 for the exponent of x and 〔MANT〕 for its mantissa,

  〔x|〕 for the absolute value of x,

  〔FP〕 (or 〔IP〕) for the fractional (integer) part of x, preserving its sign,

  and 〔sign〕 for the *signum* of x; **sign** will return **1** for $x > 0$, **–1** for $x < 0$, and **0** for $x = 0$ or non-numeric data.

- **Rounding** (all operations but SHOW contained in DISP):

  〔RDP〕 ***n*** rounds x to n decimal places in FIX format (e.g., **1.234 567 89×10^{-95}** RDP **99** returns 1.2346×10^{-95}),

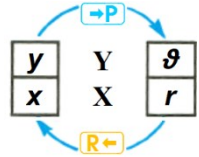
  〔ROUND〕 rounds x using the current display format,[155]

  〔ROUNDI〕 rounds x to next integer (½ rounds to 1),

  〔RSD〕 ***n*** rounds x to n significant digits, and

  SHOW displays the full precision of x in top row until next keystroke.

- **Conversions:**

  →P converts rectangular to polar *coordinates* (→ 22f), while R← converts vice versa.

  For *angular*, *date, and time conversions* → 142ff and 202ff, for *unit conversions* → 304ff.

[155] Works like RND did on *HP-42S*.

- ***Boole's* algebra** (all operations contained in BIT):

  [AND], [NAND], [OR], [NOR], [XOR], [XNOR], and [NOT] operate on *reals* like they did in *HP-28S*, i.e. x and y are interpreted before executing the operation. Zero is **false** (= **0**); any other number is **true** (= **1**).

  Example: **13.5** [ENTER↑] **–7.2** [BIT] [AND] returns **1** .

- **Probability & statistics** (unless introduced on pp. *96* - *121* already):

  [Γ(x)] computes the *gamma function* of x,

  [lnΓ] returns the natural logarithm of the *gamma function*, allowing also for computing great factorials.

  Example: What is *5432!* ?
  Remember $\Gamma(x+1) = x!$ So, entering **5433** [X.FN] [lnΓ] **10** [ln] [/] returns 17 931.480 374 010 87 as decadic logarithm of the result. Then calling [PART] [FP] [10^x] will return 3.022 55… for its mantissa.
  Thus, $5\,432! \approx 3.022\,55 \times 10^{17\,931}$.

  [RAN#] returns a uniformly distributed (pseudo) random *real* $x \in [0, 1)$,

  [SEED] stores a seed (i.e. a start value) for **RAN#**,

  Beyond all this, PROB also comprises **COMB**, **PERM**, and the 14 model distributions introduced on pp. *97ff*.

  [Σ] contains all the accumulated sums of your STATS data. Summon each of them by calling its *name* (no need to memorize any register numbers in this matter anymore).

  [STAT] comprises [Σ+], [Σ-], and [CLΣ], various mean values ([$\bar{x}$], [$\bar{x}_w$], [$\bar{x}_G$], [$\bar{x}_H$], [$\bar{x}_{RMS}$]), sample *SDs* ([s], [s_w]) and standard errors ([s_m], [s_{mw}]), population *SDs* ([σ], [σ_w]), various scattering factors ([ε], [ε_m], [ε_p]), furthermore [MEDIAN] and related values ([MAD], [Q_3-Q_1], [%ile]), **HISTOG** for histograms, [x_{min}], [x_{max}], [Δx], and a link to FIT. Look up the *IOI* for details.

  [FIT] contains [Σ+], [Σ-], and [CLΣ] once more, then correlations and covariances ([r], [cov]), all the commands concerning curve fitting ([L.R.], the 10 fit models, **ASSESS** for plotting and assess-

ing fits, as well as forecasting for ***x*** and ***y***), and **PLOT** for determining the real-life performance (or quality) of your measuring instruments and systems (→ examples above).

Look up the *ReM* (in particular the *IOI* and *ReMA I*) for comprehensive information about each such function provided on your *WP43* and some caveats.

- **Percentages** (all operations but [Δ%] contained in FIN):

  [%] returns $^{xy}/_{100}$, leaving y as is (so you can easily calculate another percentage of the same base after **CLX**).

  Example (from the *HP27 OH*):
  If you buy a new car, you have to figure the sales tax percentage, then add that to the purchase price to find the total cost of the car. ... For example, if the sales tax on a $6200 car is 5%, what is the amount of the tax and total cost of the car?

  6200 [ENTER↑] **5** [FIN] **%** returns 310. US$ for the sales tax;
  [+] returns 6 510. US$ for the total cost.

  If the dealer[156] gives you a 10% discount on the car, what will be your total cost?

  6200 [ENTER↑]
  10 **%** [−] returns 5 580. US$ for the discounted price;
  5 **%** [+] returns 5 859. US$ for the total cost.

  [Δ%] computes the percentage of change from y to x, returning $100\ ^{x-y}/_{y}$, leaving y as is (for the same reason as with [%]).
  Feel free to use [Δ%] also for calculating *markup* or *margin*:

  Example:
  You purchase ink cartridges for 21.99 US$ wholesale and retail them for 26.50 US$. What percent is your *markup*[157] and what percent is your *margin*?[158]

[156] *TNG:* Dieser Dealer ist ein ehrenwerter Mann.

[157] *Markup* is the price difference as a percentage of cost (or wholesale) price.
TNG: *Markup* = *Aufschlag* (z.B. auf die Produktionskosten).

[158] *Margin* is the price difference as a percentage of selling (or retail) price.
TNG: *Margin* = *Marge* (bezogen auf den Endpreis).

21.99 [ENTER↑] **26.5** [Δ%] returns 20.5 % *markup*.
[x≷y] **26.5** [x≷y] [Δ%] returns −17.0, i.e. 17 % *margin*.

[%MRR] computes the mean rate of return in % per period with y = present value, x = future value after z periods (→ *ReM*),

[%T] calculates $100\ x/y$ (called "% of total"), leaving y as is,

[%Σ] returns $100\ x/\sum x$, and

[%+MG] calculates a sales price by adding a *margin*[158] of x % to the cost y (→ *ReM*); you may use [%+MG] for calculating net amounts as well – just enter a negative percentage in x.

Example:
Total billed = 221,82 €, VAT = 19%. What is the net?

221.82 [ENTER↑] **19** [+/−] [FIN] [%+MG] returns 186,40 (€).[159]

- **Advanced mathematics** (→ the *IOI* and *ReMA I* for comprehensive information about the functions following):
  - *Monadic* functions:

    [B_n] and [B_n^*] return the *Bernoulli numbers*,

    [erf] and [erfc] the *error function* and its complement,

    [FIB] the extended *Fibonacci number*,

    [g_d] and [g_d^{-1}] the *Gudermann function* and its inverse, and

    [NEXTP] the next *prime number* greater than x (note [PRIME?] tests if $|\mathrm{IP}(x)|$ is *prime*, → *OMS 3*);

[159] Every engineer or scientist should be able to produce the very same result distinctly faster via **221.82** [ENTER↑] **1.19** [/].

TN: Finding [%+MG] and [%T] on financial calculators (the latter even as a primary key on the *HP-12C,* the gold standard of such devices), you might get the impression that average financial people might be mathematically slightly challenged and need some extra support. On the other hand, there was a saying in technical quarters (even before 2008):

"*Wenn man sieht, was Kaufleute allein mit Plus und Minus alles anstellen, sollte man sie an höhere Rechenarten erst gar nicht ranlassen.*" (≈ '*Looking at the results financial / business people produce using just plus and minus alone, their access to more advanced operations should be strictly denied.*')

〔sinc〕 (〔sincπ〕) returns $sin(x) / x$ (or $sin(\pi x) / \pi x$) for $x \neq 0$ and **1** for $x = 0$,

〔W_p〕 returns the principal branch of *Lambert's W* for given $x \geq -1/e$,

〔W_m〕 its negative branch, and

〔W^{-1}〕 returns x for given W_p (≥ -1);

〔K(m)〕 and 〔E(m)〕 return the *complete elliptic integrals of the 1st and 2nd kind*, and

〔ζ(x)〕 ***Riemann's*** *Zeta function* at location x .

Furthermore, call

〔H_n〕 (〔H_{nP}〕) for the *Hermite polynomials* for probability (or physics),

〔L_n〕 for *Laguerre's polynomials* and 〔$L_{n\alpha}$〕 for his *generalized* ones,

〔P_n〕 for the *Legendre polynomials*,

〔T_n〕 (〔U_n〕) for the *Čebyshev polynomials of 1st (or 2nd) kind*.

- *Dyadic* functions:

  〔AGM〕 returns the *arithmetic-geometric mean*,

  〔$J_y(x)$〕 (〔$Y_y(x)$〕) the *Bessel function of 1st (or 2nd) kind and order* y,

  〔sn(υ,m)〕, 〔cn(υ,m)〕, and 〔dn(υ,m)〕 compute the *Jacobi elliptic sine*, co*sine*, and *amplitude functions*, respectively,

  〔β(x,y)〕 returns *Euler's Beta function* and 〔lnβ〕 its natural logarithm,

  〔Γ_p〕 and 〔Γ_q〕 compute the *regularized gamma functions*,

  〔γ_{xy}〕 the *lower* and 〔Γ_{xy}〕 the *upper incomplete gamma function,*

  〔Z(φ,m)〕 returns ***Jacobi's*** *Zeta function*, and

  〔Π(n,m)〕 the *complete elliptic integrals of the 3rd kind*,

- *Triadic* function*:*

  〔I_{xyz}〕 returns the *regularized Beta function*.

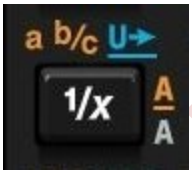
Rational Numbers (Fractions)

In addition to *reals*, you can also work with fractions on your *WP43*, with their absolute values between 1 000 000 and 0.000 1. Maximum denominator is 9 999.[160]

Enter a fraction directly by keying in a 2nd [.] in numeric input. Herein, the 1st [.] is interpreted as a blank space, the 2nd as a fraction mark:

Examples:

Key in consecutively:	… and get in *SDC* (~~FRCSRN~~)	
1 2 . 3 . 4 EXIT	$12\ ^3/_4$	
. 1 . 2 EXIT	$0\ ^1/_2$	
3 . 1 . EXIT	$3\ ^1/_2$	takes the last denominator entered [161]
1 . . 4 EXIT	$1\ ^0/_1$ [162]	since input was **1 $^0/_4$**
. . 1 5 EXIT	$0\ ^0/_1$	input was **0 $^0/_{15}$**



Every closed *real* with $|x| \in [10^{-4}; 10^6]$ on the *stack* will be displayed as a fraction after [a b/c] was pressed, after a fraction is entered as demonstrated, or after x is combined with a fraction by an arithmetic operation. If the fraction displayed is slightly greater or slightly less than the underlying *real*, < or > will precede this fraction, respectively (→ examples following). Get a more verbose notation with FRCSRN (like $x = ^1/_2$ or $x < ^3/_7$, applying to the other *stack registers* by analogy).[163]

---

[160] **DENMAX** accepts greater values but will reduce them as soon as input is closed. Fractions work almost like on *HP-32SII* and its successors but with higher precision.

[161] … following an idea of *Craig Finseth*. *Startup default* denominator is **4**.

[162] This display of a pure integer tells you unambiguously your *WP43* is in *proper fraction display mode*. In *improper fraction display mode*, $^1/_1$ will be displayed instead.

TN: The *HP-32SII* reads 1 . . 4 as $^1/_4$ – although this is neither coherent with its other input interpretations nor does it save any keystrokes. In our opinion, the calculator cannot and shall not know better than the user what (s)he meant.

[163] Your *WP43* continues calculating with *reals*, it will just display fractions as close to those as possible within the constraints set by you. FRCSRN stands for *show register names*.

Vice versa, any closed number displayed as a fraction will show its *real* value again after [.d] or after [DISP] **ALL**, **FIX**, **SCI**, or **ENG**. And a closed fraction will be decomposed to its *long integer* numerator in **Y** and denominator in **X** by [PART] **DECOMP**; this can be simply reverted by division.

Your *WP43* features 2 fraction display modes: *proper* and *improper fractions*.[164] [a b/c] toggles them, starting with *proper fractions* as illustrated below. The following **example** comprises most aspects of fraction display (assuming *SDC*):

Enter:	and you will see:	
3 [1/x]	$3.333\,333\,333\,333\,333\times10^{-1}$	
[a b/c]	$< 0\ {}^{1}/_{3}$	*proper fraction*
[a b/c] [165]	$< {}^{1}/_{3}$	*improper fraction*
	since $x = 0.333\,333\,333\,333\,333\,3 < {}^{1}/_{3}$.	
[1/x]	${}^{3}/_{1}$	since this is exact
78.40625 [−]	$-2\ {}^{413}/_{32}$	and stays exact
[ENTER↑] [×]	${}^{5\,822\,569}/_{1\,024}$	
2 [×]	${}^{5\,822\,569}/_{512}$	

Now, press [a b/c] for displaying this *improper fraction* as a *proper* one:

	$11\,372\ {}^{105}/_{512}$	
11 [/]	$> 1\,033\ {}^{4\,713}/_{5\,632}$	

This fraction is less than the underlying *real* x, deviating $< {}^{0.5}/_{5\,632}$ from it.

Now, let's reduce the maximum denominator to

64 [MODE] **DENMAX**	64	
[DROP↓]	$> 1\,033\ {}^{41}/_{49}$	
CF **SYS.FL** **DENANY**	$< 1\,033\ {}^{27}/_{32}$	

since ~~DENANY~~ allows for denominators 2, 4, 8, 16, 32, and 64 here only (DENFIX is clear since startup). The last fraction shown is greater than the underlying *real* x; the difference is $< {}^{0.5}/_{32}$ (and $> {}^{0.5}/_{64}$ – else the display would read $1033\ {}^{53}/_{64}$ instead).

---

[164] *TNG*: *Proper fractions* decken sowohl *echte Brüche* (wie ¾) als auch *gemischte Brüche* (wie 2 ½) ab. Bei *improper fractions* wird der ganzzahlige Anteil nicht herausgezogen, so dass hier der Zähler größer als der Nenner sein kann.

[165] This conversion was introduced on *RPN* calculators with the *WP 34S* in 2011.

➡ **DECOMP** will decompose a fraction preserving full precision always; thus, it will return here $y = 5\,822\,569$ for the numerator and $x = 5\,632$ for the denominator. So you can revert **DECOMP** simply by dividing these integers but note some *real* precision may be lost (compare results).

Another **example**, now with a maximum denominator 60:

DENANY allows for any denominator ≤60 (i.e. 60, 59, 58, ..., 6, 5, 4, 3, and 2).

~~DENANY~~ & DENFIX allows for a fix denominator 60 only.

~~DENANY~~ & ~~DENFIX~~ allows factors of 60 (i.e. 60, 30, 20, 15, 12, 10, 6, 4, 3, and 2).[166]

Before closing this chapter about rational numbers, let's not forget those isolated irrational islands in the vast ocean of *SI* where you may come across dimensions like in the following **example**:

A calculator stand is specified to measure 9" × 3 $^1/_2$" × $^5/_8$". It goes without saying that your *WP43* will support you also in such harsh environments. Only absolute greenhorns, however, will expect that a tight thin-walled box around this stand will displace

9 ENTER 3 . 1 . 2 ×	31 $^1/_2$
. 5 . 8 ×	19 $^{11}/_{16}$ *cubic inches* of water there.

Instead, an absolutely magic conversion from *cubic inches* to so-called *fluid ounces* is required now, even depending on the country you are in! [167] Though do not despair: *OMS 6* will tell you how to do this magic – it will take just a little more time and effort than calculating with rational units.

---

[166] 60 was a sacred number in ancient Babylon, hence our sexagesimal angles and hours. Actually, educated Babylonians calculated with base 60. See *OMS 3* for factorization.

[167] No such differentiation in *SI* – here a volume stays unchanged regardless of the matter filling it, be it air, steam, liquid water, mercury, ice, or solid lead, for examples.

TN: Honestly, unless you were raised on such an isolated island, we bet you assumed *fluid ounces* being a unit of mass, didn't you? Since you have heard of *ounces* once before and just thought … terribly wrong! Don't think there – you may run into deep troubles easily (thinking less, however, may pave your road to top positions in administration – see evidence between 2016 and 20 and after 2025-01).

Those islanders seem to have got plenty of spare time to devote to solving such problems with units. Sometimes, however, their high flying endeavors might fail (→, e.g., *https://www.latimes.com/archives/la-xpm-1999-oct-01-mn-17288-story.html*). In consequence, *NASA* turned to *SI*.

Proposal for Configuration Setting

The following settings (deviating from *SDC*) may be beneficial for your scientific or engineering work and provide for a clean screen as well:

1. Choose your regional preferences (→ *76ff*): just call DISP ▲ CHINA, EUROPE, INDIA, JAPAN, UK, or USA,
2. DSTACK **2** may do (→ *52*, f. 64),
3. **99** RANGE should suffice for real-world problem solving (→ *ReMS 1*),
4. ▼ SCI **5** or ENG **5** (whatever you prefer) should be precise enough for real life and covers the full number range.
5. SF SYS.FL 〔SPCRES〕 allows for infinities (**±∞**) and **NaN** (else such results would throw errors), → *300*.
6. If you want or have to work with *fractions*, set a suiting maximum denominator via ***n*** MODE DENMAX. Even ***n*** = **16** (for *Imperial* tape-lines), **32**, **64**, **100**, or **120** may suffice, **4800** catches almost all.

Having made your choices, call STOCFG to save your personal calculator *configuration* in a *user variable* (→ *62* and *78*). Use RCLCFG to recall it.

Alternatively, store it in a named *state file* in *FM* (→ *253f*).

SECTION 3: MORE OBJECTS AND DATA TYPES

Your *WP43* can do more for you than just handling *reals* and fractions: it can also deal with *angles*, *complex numbers*, *real* and *complex vectors* and *matrices,* as well as *integers*, *dates*, *times*, and *text strings*. All these *DTs* are covered in dedicated chapters of this *Section* (compare *ReMA B* for all *DTs* available).

But how shall your *WP43* learn what you want, just getting your input? Some **examples** will demonstrate:

Input in *SDC*	Display in *SDC*	*DT*, meaning
12345.678901 EXIT	12 345.678 901	*Reals* (∈ ℝ), → 78ff above
12.3 E **45** ENTER↑	1.23×10^{46}	
901.23.4567 ENTER↑	> 901 23/4 567	*Fraction* (∈ ℚ), → 128ff above
270 (→∠) →MULπ	1.5π	*Angle* shown in *multiples of π*
123.45678901 d.ms	123°46' 7.89"	*Angle* entered in *sexagesimal degrees*, → 138ff
12.34 CC **5.67** +/- EXIT	$12.34-i\times5.67$ $12.34\angle-5.67°$	*Complex numbers* (∈ ℂ) in rectangular or polar notation, → 152ff
12345678901 ENTER↑	12 345 678 901	*Integers* (∈ ℤ) of various bases (2, 3, …, 16) and lengths, → 185ff
1234567890 # H	$12\ 34\ 56\ 78\ 90_{16}$	
10100110111 # 2	$101\ 0011\ 0111_{2}$	
9.1121012345678 .d	0009-11-21 1:23:45	*Extended date*, → 202ff
19.345678901 h.ms	19:34:56.789 01	*Sexagesimal time*, → 205ff

Depending on your settings of particular *system flags*, modes, and parameters, *WP43* output might differ slightly from above (displays in *SDC* are in light blue fields below, further widespread formats in light red):

	DECIM.	~~DECIM.~~
12345.678901 EXIT		
GAP 4	1 2345.6789 01	1 2345,6789 01
GAP 3	12 345.678 901	12 345,678 901
GAP 2, **1**, or **0**	12345.678901	12345,678901
12.3 E **45** ENTER↑		
MULT×	1.23×10^{46}	$1{,}23\times10^{46}$
~~MULT×~~	$1.23\cdot10^{46}$	$1{,}23\cdot10^{46}$
12.34 CC **5.67** +/- EXIT		
MULT×	$12.34-i\times5.67$	$12{,}34-i\times5{,}67$
~~MULT×~~	$12.34-i\cdot5.67$	$12{,}34-i\cdot5{,}67$
19.345678901 h.ms		
TDM24	19:34:56.789 01	19:34:56,789 01
~~TDM24~~	7:34:56.789 01pm	7:34:56,789 01pm

	Y.MD	D.MY	M.DY
9.1121012345567 .d	0009-0…	09.11.2101 23:45:56	09/11/2101 11:45:56pm
		for the 9th of November	for September 11th

Obviously, your *WP43* reacts to your input very flexibly, and you will recognize the various *DTs* and formats easily.

Knowing these now, how can you combine data of various types in calculations? Ten *DTs* (not only numeric ones) of your *WP43* are listed in column 1 of the matrix overleaf. An object of the *DT* as indicated in one of the narrow columns at right (y) plus or minus an object of the *DT* in

column 1 (***x***) results in an object ***y*** +/− ***x*** of the *DT* printed at the intersection; thus, wherever a *DT* is shown there, the corresponding combination is legal for addition or subtraction – else **`Illegal input data type for this operation`** will be thrown. Grey fields indicate restrictions, light yellow fields highlight asymmetries.

DT and meaning ***x***	***y***									
	1	2	3	4	5	6	7	8	9	10
1 ℤ *Long integer* [168]	**1**	2	3	4	5	6 [169]	7	8	9	10
2 ℝ *Real number*	2	**2**	3	4	5	6 [169]	7	8	9	2
3 ℂ *Complex number*	3	3	**3**	—	—	—	7	9	9	3
4 *Angle* (in various *ADMs*) [170]	4	4	—	**4**	—	—	7	—	—	4
5 *Time interval* (in H.MS)	5	5	—	—	**5**	6 [171]	7	—	—	5
6 *Date* (in various formats)	6 [169]	6 [169]	—	—	6 [171]	**2** [172]	7	—	—	6 [169]
7 *Text string* [173]	7	7	7	7	7	7	**7**	7	7	7
8 *Real matrix* or *vector* [174]	8	8	9	—	—	—	7	**8**	9	—
9 *Complex matrix* or *vector*	9	9	9	—	—	—	7	9	**9**	—
10 *Short integer* [168]	1	2	3	4	5	6 [169]	7	—	—	**10**

Examples:

A *complex number* (*DT* 3) plus or minus a *real* (*DT* 2) will result in a *complex number*. And a *text string* (*DT* 7) plus a *time* (*DT* 5) will append the time string as displayed to said *text string* in small font. See respective chapters below.

---

[168] If *integers* of different types (*DTs* 1 and 10) or different bases are combined, output will be an *integer* of the type and base given in **Y**.

[169] A *date* ***y*** plus (minus) a *number* ***x*** adds (subtracts) the respective number of decimal *days* to (from) said *date*. The result will be an *extended date* – containing a time of day. A number ***y*** plus a *date* ***x*** works alike. A number ***y*** minus a *date* ***x*** is not supported. → *Dates* and *Times* further below. → *ReMA B* for *date* limits.

[170] → *Angles* further below.

[171] A *time* plus a *date* (or vice versa) returns an *extended date* (→ respective chapter further below). Same for a *date* minus a *time* (interval). A *time* minus a *date* is not supported.

[172] A *date* minus a *date* returns the number of decimal *days* between both *dates*. Plus is not supported.

The following matrix shows the resulting *DTs* of products and quotients in the same way (note that neither *dates* nor *text strings* can be multiplied or divided; actually some further combinations are physically forbidden – but feel free to use .d to convert *angles* or *time* intervals into untagged *reals*, → *138*):

… **times** an object x of the *DT* below returns a **product** of the *DT* printed at the intersection.[175]	An object y of *DT* … 1	2	3	4	5	8	9	10
1 ℤ *Long integer* [168]	**1**	2	3	4	5	8	9	10
2 ℝ *Real number*	2	**2**	3	4	5	8	9	2
3 ℂ *Complex number*	3	3	**3**	3	—	9	9	3
4 *Angle*	4	4	3	**2**	—	8	9	4
5 *Time interval*	5	5	—	—	—	—	—	5
8 *Real matrix* or *vector*	8	8	9	8	—	**8**	9	8
9 *Complex matrix* or *vector*	9	9	9	9	—	9	**9**	9
10 *Short integer* [168]	1	2	3	4	5	8	9	**10**
… **divided** by an object x of the *DT* below returns a **ratio** of the *DT* printed at the intersection.								
1 ℤ *Long integer* [168]	**1/2**[176]	2	3	4	5	8	9	10
2 ℝ *Real number*	2	**2**	3	4	5	8	9	2
3 ℂ *Complex number*	3	3	**3**	3	—	9	9	3
4 *Angle*	2	2	3	**2**	—	8	9	2
5 *Time interval* [177]	2	2	—	4	**2**	—	—	2
8 *Real matrix*	8	8	9	8	—	**8**	9	8
9 *Complex matrix*	9	9	9	9	—	9	**9**	9
10 *Short integer* [168]	1	2	3	4	5	8	9	**10**

[173] Only additions are supported for *DT* 7: Any numeric x or y will be converted to a *text string* according to the display format at execution time; for a *matrix*, its *descriptor* will be taken (e.g., [3×4 ℂ Matrix], see further below). String x will be appended to string y then.

[174] Where a blue number is printed on this or next page, matrix dimensions must match (→ *Vectors & Matrices: Calculating* further below).

[175] Where a red number is printed on this page, the *angle* tag will be discarded and the underlying *real number* will be processed instead of the *angle*. Note any angular input

Following tables are for powers and roots:

	A number $y > 0$ of *DT* …		
	1	2	10
… raised to a power of $x \geq 0$ of the *DT* below returns a result of the *DT* printed at the intersection.			
1 or 10 (i.e. *long* or *short integer*) [168]	1	2	10
2 ℝ *Real number*	2		
… raised to the power of $x < 0$ (with x of *DT* 1, 2, or 10 [168]) returns a number of *DT*:	2 (1 / 10 for 1^{-1} only)		
The x^{th} root ($x > 0$ of the *DT* below) of y returns a number of the *DT* printed at the intersection.			
1 or 10 (i.e. *long* or *short integer*) [168]	1 [178] / 2	2	10 [178]
2 ℝ *Real number*	2		

	A number $y < 0$ of *DT* …		
	1	2	10
… raised to a power of $x \geq 0$ of the *DT* below returns a result of the *DT* printed at the intersection.			
1 or 10 (i.e. *long* or *short integer*) [168]	1	2	10
2 ℝ *Real number*	2 / 3 [179]		
… raised to the power of $x < 0$ (with x of *DT* 1, 2, or 10 [168]) returns a number of *DT*:	2 / 3 [179] [1 / 10 for $(-1)^{-1}$ only]		
The x^{th} root ($x > 0$ of the *DT* below) of y returns a number of the *DT* printed at the intersection.			
1 or 10 (i.e. *long* or *short integer*) [168] x odd	1 [178] / 2	2	10 [178]
2 (or 1 or 10 with even x)	3 [179]		

is treated the same way also in the operations on the 2 pages following.

[176] E.g., **15** ENTER↑ **3** / returns *integer* 5 while **14** ENTER↑ **5** / will return *real* 2.8.

[177] The *time interval* will be converted to a *real* number of decimal *hours* before division.

[178] Square roots of squares, cube roots of cubes, logarithms of powers, etc. stay in the *DT* they were. For other *integers* $y > 0$, results will become *real* (for *DT* 1) or truncated *short integers* (for *DT* 10); for $y < 0$, results will become *complex* instead of *real*.

Any number of *DT* 1 or 2 raised to *complex* power will return a *complex number*, as well as any *complex number* raised to arbitrary power. Raising a *short integer* to *complex* power is not supported, nor are powers involving *DTs* 5 … 9.

This table is for general logarithms:

… combined in $\log_x y$ with a number $x > 0$ of the *DT* below returns a number of the *DT* printed at the intersection.	A number $y > 0$ of *DT* … 1	2	10
1 or 10 (i.e. *long* or *short integer*) [168]	1[178] / 2	2	10[178]
2 ℝ *Real number*	2	2	10[178]

And for $y < 0$ of *DT* 1 or 2, logarithms will return *DT* 3 if this is allowed (→ chapters about *complex numbers* below); logarithms of negative *short integers* are not supported.

Here a table for some *monadic* functions:

… processed by an operation printed below returns a number of the *DT* printed at the intersection.	A number $x > 0$ of *DT* 1	2	10
lg , lb x , √x , ∛x [180]	1[178] / 2	2	10[178]
ln , e^x	2	2	10[178]
10^x	1	2	10

The following table is for integer divisions:

[179] Complex results require CPXRES set (i.e. ℂ lit in the status bar). Else an error message will be thrown.

[180] For roots of negative operands, return to previous page

… IDIVR-divided by a number x of the *DT* below returns an **integer ratio** in $\mathbf{X}$ and a remainder in $\mathbf{Y}$ of the *DTs* printed at the intersection.	A number y of *DT* … 1	2	10
1 or 10 (i.e. *long* or *short integer*) [168]	**1**; 1	**1**; 2	**10**; 10
2 ℝ *Real number*	**1**; 2	**1**; 2	**10**; 2

Additionally, explicit *DT* conversions are available:

Any closed object x of *DT* … 1 *long integer*	2 *real*	4 *angle*	5 *time*	6 *date*	10 *short integer*	… will be converted into the *DT* below by the keystrokes printed at the intersection.
#I, ROUNDI		—	—	—	#I	1 ℤ *long integer*
.d						2 ℝ *real number*
∡→ …			—	—	—	4 *angle*
h.ms		—	—	—	—	5 *time*
# ***n***		—	—	—	# ***n***	10 *short integer* of base ***n***

The Status Bar Indicating Calculator Configuration and Status

Radix marks and digit group gaps are obvious in the numeric display; so are date and time display modes (YMD / DMY / MDY and TDM24) in the *status bar*. The following indicators may trail the date-and-time string, depending on *system flag* settings. They are listed here in order of appearance – those shown in *SDC* are printed in light blue fields again: [181]

[181] The symbol "&" denotes a logical "and" and a comma a logical "or" in this table.

Indicator	Set by	Deleted by	Explanation, remarks
ℂ	CPXRES	~~CPXRES~~	With CPXRES, *complex* results of operations on *reals* are allowed, like $\sqrt{-1}$. Else a domain error will be thrown in such a case.
ℝ	~~CPXRES~~	CPXRES	

⊾	~~POLAR~~	POLAR	Rectangular or polar notation chosen for displaying *complex numbers* (→ 152ff).
⊙	POLAR	~~POLAR~~	

∠°	DEG	setting any other *angular display mode* (*ADM*)	Display settings for *angles*: *decimal degrees*, *grads* or *gon*, *radians*, *multiples of π*, *mils*, or *sexagesimal degrees* (→ 142ff).
∠ᵍ	GRAD		
∠ʳ	RAD		
∠π	MULπ		
∠‾	MIL		
∠''	d.ms		

/max or /2345	DENANY	Can only be modified by DENMAX	Fraction display settings. The current value of the maximum displayable denominator D.MAX is shown behind the fraction bar (/max is *SDC* and equivalent to 9999). With ~~DENANY~~, DENFIX toggles a specific symbol trailing D.MAX in the *status bar* (as shown on pp. 128ff).
/2345f	~~DENANY~~ & DENFIX	DENANY	
/2345× or /2345·	~~DENANY~~ & ~~DENFIX~~	DENANY	

64:1	1COMPL	setting any other *integer sign mode* (*ISM*)	*Short integer* settings. First 2 digits tell the *word size*, the symbol after the colon tells the *ISM*. *Startup default* size is 64 *bits* (the max.) and 2's complement. CARRY and OVERFLOW may follow the *ISM* but are only lit if set (→ 185ff).
64:2	2COMPL		
64:U	UNSIGN		
64:S	SIGNMT		

wrap	M.WRAP	M.GROW	*Matrix grow mode*, displayed instead of ***ws***:***ISM*** as long as a *matrix* is open for editing (→ 162ff).
grow	M.GROW	M.WRAP	

Indicator	Set by	Deleted by	Explanation, remarks
END	ENDP	BEGINP	Payment mode in the *Time Value of Money* application, displayed instead of ***ws*:*ISM*** as long as TVM is open (→ *294ff*).
BEG	BEGINP	ENDP	
A	▲ if α is set	ENTER↑ in *AIM*, EXIT in Myα, ~~ALPHA~~	α or SF ALPHA will set *AIM*. For A (α), upper (lower) case letters can be entered then. *AIM* will start with A except for editing equations or creating *variables*. → *208ff*.
α	▼ if A is set		
8	SSIZE8	~~SSIZE8~~	*Stack* size.
4	~~SSIZE8~~	SSIZE8	
	→ remarks	*WP43* idling	Operation executing in *RUM*. Not lit with a program running.
	→ remarks	*WP43* idling	Program running (→ *OMS 4*).
	program waiting for user input	program running (→ *OMS 4*)	Will also be lit if a program is stopped by EXIT or R/S – then will be cleared by next keystroke.
	timer running in background	idle timer	→ TIMER (a.k.a. stopwatch) application in *OMS 6*.
	serial I/O in progress	idle communication line	
U	USER	USER	Toggles *user mode* (→ *OMS 7*).
	→ remarks		Lit if power is supplied through the *USB* cable, else clear.
	low battery	supply voltage > 2.5 V	A low battery will reduce processor speed. Your *WP43* may shut off when voltage drops below 2.0 V.

In *SDC*, the *status bar* looks like on p. 74. With MDY and ~~TDM24~~, CPXRES, POLAR, MULπ, DENFIX and a 4-digit D.MAX, unsigned *short integers*, CARRY and OVERFLOW, being in *alpha input mode*, with ~~SSIZE8~~, a program waiting for input and the timer running in background, in *user mode* and with a low battery, however, the bar may look like this instead:

Note also or , , and might appear at its right end. See below and *ReMS 1* for the *system flags* and calculator modes mentioned above.

Display formats for

- *angles* (→ 142ff),
- *complex numbers* (→ 152ff), and
- *matrices* (→ 162ff) follow real number formatting.
- *Integers* (→ 185ff),
- *dates* (→ 202ff), and
- *times* (→ 205ff) obey their own display formats.

Also all these are included in your calculator *configuration* when you save or recall it (→ 131).

Angles and Trigonometric Functions

For dealing with *angles*, choose 1 out of 6 *angular display modes* (*ADM*) on your *WP43*: **DEG**, **RAD**, **GRAD**, **MULπ**, **MIL** (all these live in [MODE]), and **D.MS**.[182] *Angles* are entered as *reals* and interpreted according to the current *ADM* as indicated in the *status bar* by ∠°, ∠ʳ, ∠ᵍ, ∠π, ∠⁻ , or ∠" (→ *139*).

Exception: *Sexagesimal degrees* shall be entered in the format `ddddd.mmsshh` [d.ms] – with `ddddd` standing for integer *degrees*, `mm` for *angular minutes*, `ss` for *seconds*, and `hh` for hundredths of *seconds*.

Examples:

12.3456543 [d.ms] returns 12°34'56.54",

45.6789018 [d.ms] returns 46° 8'29.01".[183]

arcsin, **arccos**, and **arctan** operate on *reals* and return *angles* – such output will be automatically tagged according to the current *ADM*. Assume **ALL 1** set for the following **examples**:

In *ADM* ...	... **0.5** [√x] [TRI] **arcsin** returns ...
sexagesimal degrees ∠"	45° 0' 0.00"
decimal DEG*rees* ∠°	45.°
RAD*ians* ∠ʳ	0.523 598 775 598 299ʳ
MUL*tiples of* π ∠π	0.25π
GRAD*ians* or *gon* ∠ᵍ	50.ᵍ
MIL ∠⁻	800.⁻

[182] *TN*: GRAD is used in surveying and robotics/automation, MIL in military (artillery).

TNG: Vorsicht, *HP* verwenden GRAD für *Gon* – dagegen steht DEG für die üblichen *Winkelgrade*. MIL entspricht *Strich*.

[183] There are no leading zeros for angular *minutes* and *seconds*. And this *ADM* truncates everything smaller than 0.01" – though it will display down to 0.01" always and can't be shortened (→ examples following).

Whenever you see a number formatted alike you know it is an *angle*.

Other functions (e.g., **sin**) expect *angular* input; there, a plain (untagged) number is taken as an *angle* of current *ADM*. You can add and subtract *angles* with various tags (and without): results will always come in current *ADM*. Additionally, 22 direct angular conversions are provided in [∡→] for your convenience:

From … / to …	*sexag. degrees*	*decimal degrees*	*radians*	*multipl. of π*	*grads / gon*	*mil*	current *ADM* or tagging
sexag. degrees	—	D→D.MS	—	—	—	—	→D.MS
dec. degrees	D.MS→D	—	R→D	—	—	MIL→D	→DEG
radians	—	D→R	—	Mπ→R	—	MIL→R	→RAD
multiples of π	—	—	R→Mπ	—	—	—	→MULπ
grads / gon	—	—	—	—	—	—	→GRAD
mil	—	D→MIL	R→MIL	—	—	—	→MIL
current *ADM*	D.MS→	DEG→	RAD→	MULπ→	GRAD→	MIL→	—

Example (with FIX 5):

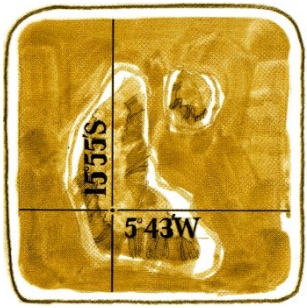
[MODE] MULπ Choose *multiples of π* as *ADM* and ∡π will appear in the *status bar* and stay there for the time being.

150 [1/x]	0.006 67	So π/150 correspond to …
[∡→] →RAD	0.020 94ʳ	some 0.020 94 *radians*
→DEG	1.200 00°	or precisely 1.2°
→D.MS	1°12' 0.00"	or 72 *angular minutes*
→MIL	21.333 33ᵐ	or some 21.33 *mil*
9 [1/x] [+]	0.117 78π	after adding π/9 in current *ADM*
→D.MS	21°12' 0.00"	is the result. By the way,
50 GRAD→ returns	0.250 00π	in current *ADM* immediately.

Note →RAD 'knew' it had to convert from *multiples of π* since →RAD expects *angular* input and takes the current *ADM* into account. *Angular* output is tagged always – thus, →DEG above converted from *radians,* →D.MS from

decimal and →MIL from *sexagesimal degrees*, since their inputs were tagged accordingly. Untagged **1/9** was taken as input in current *ADM* again, and **50** were introduced as *gradians* to your WP43.

You have learned about trigonometric functions at school. Thus, we demonstrate their operation on *angles* with 1 vintage **example** only:



Lovesick sailor *Oscar Odysseus* dwells on the island of Tristan da Cunha (37°03'S, 12°l8'W), and his sweetheart, *Penelope*, lives on the nearest island. Unfortunately for the course of true love, however, Tristan da Cunha is the most isolated inhabited spot in the world. If *Penelope* lives on the island of St. Helena (15°55'S, 5°43'W), use the following formula to calculate the great circle distance that *Odysseus* must sail in order to court her.[184]

Solution:

The formula for the great circle distance *d* in *nautical miles* is:

$$d = 60 \times arccos[sin(B_s)\ sin(B_d) + cos(B_s)\ cos(B_d)\ cos(L_d - L_s)]$$

with B_s being the latitude and L_s the longitude of the start (*Tristan da Cunha*) and B_d being the latitude and L_d the longitude of the destination (*St. Helena*).[185] Hence, with the numbers inserted, this formula reads:

$$d = 60 \times arccos[sin(37°03'S)\ sin(15°55'S) + cos(37°03'S)\ cos(15°55'S) \times cos(5°43'W - 12°18'W)]$$

Set the display formats – remember the trigonometric functions assume their input being in current *ADM* – and calculate from inside out,

d.ms	set the *ADM* to *sexagesimal degrees*.
DISP FIX 2	no more decimals needed in display.
5.43 d.ms	5°43' 0.00"
12.18 d.ms	12°18' 0.00"

[184] *TN*: This problem is found first in the *HP-25 OH*. It was reprinted thereafter in each and every *HP* scientific pocket calculator manual until the *HP-41C/41CV OHPG*.

[185] *TN*: This formula means that 1 *nautical mile* (nmi) corresponds to 1 *angular minute* on a great circle on Earth (60 nmi correspond to 1° there, some 111.1 km). This doesn't hold exactly but precisely enough for practical sailing.

Latitude means *geographische Breite* in German. Hence it is represented by the letter ***B*** in the formula above.

[−]	-6°35' 0.00"
[TRI] cos	0.99
15.55 [STO] [J] cos	0.96
[×]	0.96
37.03 [STO] [K] cos	0.80
[×]	0.76
[RCL] [K] sin	0.60
[RCL] [J] sin	0.27
[×]	0.16
[+]	0.93
arccos	22°15' 36.15"
[.d]	22.26
60 [×] returns	1 335.60
[U→] x: m ← nmi	2 473 535.88

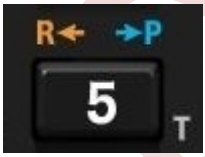
convert to an untagged *real*.

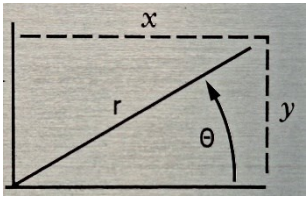
nmi or

, i.e. some 2474 km that *Odysseus* must sail to visit *Penelope*, presuming prosperous winds.

Mixed Calculations: Coordinate Transforms in 2D, Circuits, Flight Directions, Courses over Ground, etc.

2 functions are provided for converting polar or rectangular coordinates in 2 dimensions. Input and output data are both in *stack registers* X and Y here. [→P] converts *2D Cartesian* co-ordinates ***x*** and ***y*** to polar magnitude or radius ***r*** in X and angle ***ϑ*** in Y.





Example (assuming *SDC*):
Convert (***x***, ***y***) = (**6**, **4.5**) to polar.

Solution (with **FIX 2** set):

4.5 [ENTER↑] **6** [→P] returns

ϑ =	36.870°
r =	7.500

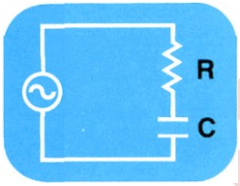
i.e. a *vector* of magnitude 7.5 pointing up at an *angle* of some 37° to the positive *x*-axis.

R← converts *2D* polar magnitude or radius ***r*** in X and angle ***ϑ*** in Y to *Cartesian* coordinates ***x*** and ***y***. Both functions honor the *ADM* settings and tags as described in previous chapter.

Example (continued):
Convert the returned ***ϑ*** of previous example to *radians*, and then convert the resulting coordinates (*r*, ***ϑ***) to rectangular.

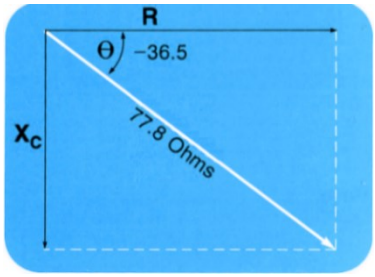
Solution:

x≷y		7.500 36.870°
∠→ →RAD		7.500 0.644ʳ
x≷y R← returns	y = x =	4.500 6.000 as expected.



Another **example** (from the *HP-29C OHPG*):
Engineer *Trigo Slothrop* has determined that in the RC circuit shown …, the total impedance is 77.8 Ω and voltage lags current by 36.5°. What are the values of resistance ***R*** and capacitive reactance X_C in the circuit?

Solution (with FIX 1):
See the picture: of the total impedance: ***R*** is its component parallel to the *x*-axis, and X_C is its perpendicular component parallel to the *y*-axis:



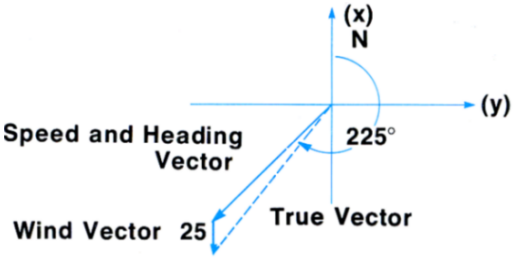
−36.5 ENTER↑ **77.8**

R← returns

y =	−46.3
x =	62.5

i.e. a resistance of 62.5 Ω and a reactance of 46.3 Ω.

Note you can use →P and R← also to convert *3D* cylinder coordinates to *Cartesian* and vice versa, since ***z*** is kept unchanged. – Knowing about R←, →P, and Σ+, we can profit from combining these functions now:



Example (from the *HP-25 OH*):

The instruments in fearless bush pilot *Apeneck Sweeney's* converted *P-41* indicate an air speed of 125 *knots* and a heading of 225°. However the aircraft is also being buffeted by a steady 25-*knot* wind that is blowing from north to south. What is the actual course and speed of the aircraft?

Solution (with FIX 2):

Combine the *vector* indicated on the aircraft instruments with the wind *vector* to yield the actual course and speed. Convert the *vectors* to rectangular, then combine the *x*- and *y*-coordinates in the statistical storage. Finally, recall the summed *x*- and *y*-coordinates and convert them to polar coordinates giving the actual *vector* of the aircraft. (North becomes the *x*-coordinate in order that the problem complies with navigational convention.)

CLR CLΣ	clears the statistical storage.	
225 ENTER↑ 125	indicated heading and air speed.	
R←	returns	y = −88.39 x = −88.39
STAT Σ+	puts x and y into the statistical storage.	
180 ENTER↑ 25	north wind.	
R←	returns	y = 0.00 x = −25.00
Σ+	adds x and y to the statistical storage.	
SUM	pushes the summation *registers* Σx and Σy on the stack.	
→P	returns	θ = −142.06° r = 143.77
x≷y 360 +	returns	217.94°

(we have to change the angle to become positive for being in line with navigational convention).

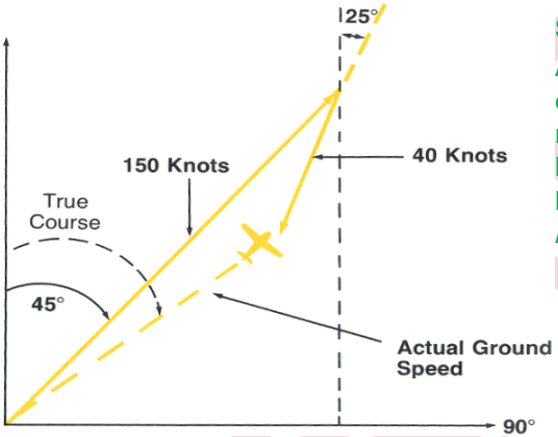
Mr. *Sweeney* is actually flying at 143.77 *knots* on a course of 217.94° over ground.[186]

[186] Another way of solving such *2D vector* problems will be shown below.

A similar problem appeared first in the *HP-55 OH* and was copied for some years. We quote it from the *HP-33 OH*:

Example:

On his way to search for an albino caribou, grizzled bush pilot *Apeneck Sweeney's* converted *Swordfish* aircraft has a true air speed of 150 *knots* and an estimated heading of 45°. The *Swordfish* is also being buffeted by a headwind of 40 *knots* from a bearing of 25°. What is the actual ground speed and course of the *Swordfish*? [187]



Start of solution:

A bearing of 25° equals a heading of 25° + 180° = 205°, so the corresponding headwind *vector* can be added. Solving this problem is left to you (for crosscheck: 113.24 *knots* at 51.94°).

Additionally, here is an advanced problem from a universe far, far away:

Example from the *HP-32 OH*:[188]

Federation starship *Felicity* has emerged victorious from a furious battle with the starship *Thanatos* from the renegade planet *Maldek*. However, its automatic pilot is kaput (sic!), and its main thrust engine is locked on at 37.2 MN directed along an angle of 25.2° from the star *Ultima*.[189] Consulting the ship's star map, the navigator reports a hyperspace entrance *vector* of 51 MN at an angle of 41.3° from *Ultima*. To what thrust and angle should the auxiliary engine be set, for *Felicity* to achieve alignment with the hyperspace entrance *vector*?

---

[187] This kind of plane is still in service after 70 years. Our grandparents could not imagine.

[188] *TN*: This problem is from 1978. The 1st episode of *Star Wars* was launched in 1977.

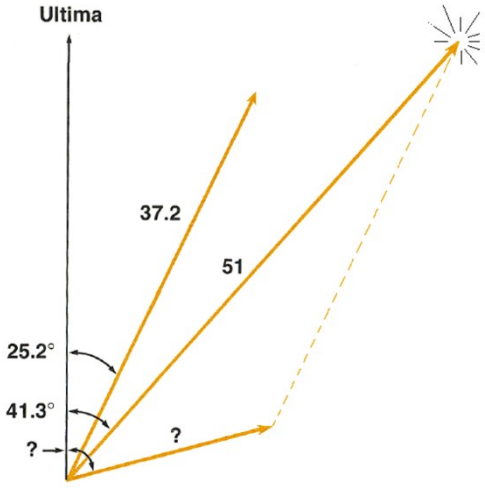
[189] *TN*: a) *Ultima* is Latin for '*the last*' (fem.).

b) *Thanatos* (Θάνατος) is ancient Greek for *death*, living (?) in *Hades*, and pronounced like *Tánatos* in Spanish, Italian, French, Dutch, German, Danish, Norwegian, Swedish, and Finnish, for examples, but like *Túnnatoss* in English.

c) Why is that planet called *Maldek* instead of, e.g., *Bendek*? Subliminal manipulation…

Solution (with FIX 1):

The required thrust *vector* of the auxiliary engine is equal to the hyperspace entrance *vector* minus the thrust *vector* of the main engine. The *vectors* are converted to rectangular coordinates using [R←], and their difference is calculated using [Σ+] … This difference is recalled to … **X** and **Y** … using [SUM]. Then, these rectangular coordinates of the auxiliary engine thrust *vector* are converted to polar coordinates using [→P].



[CLR] [CLΣ] clears all statistical data

41.3 [ENTER↑] **51** this is the hyperspace entrance *vector*

[R←] returns

y =	33.7
x =	38.3

[STAT] [Σ+] puts the ***x*** and ***y*** components of the hyperspace entrance *vector* into the statistical storage.

25.2 [ENTER↑] **37.2** main engine thrust *vector*

[R←] returns

y =	15.8
x =	33.7

[+/-] [x≷y] [+/-] [x≷y] inverts both its *Cartesian* components.[190]

[Σ+] adds the ***x*** and ***y*** components of the negative main engine thrust *vector* to the statistical storage.

[SUM] recalls the summation *registers*:

Σy =	17.8
Σx =	4.7

[→P] returns

ϑ =	75.4°
r =	18.4

meaning the auxiliary engine shall be set at 18.4 MN and an angle of 75.4° from *Ultima*.[191]

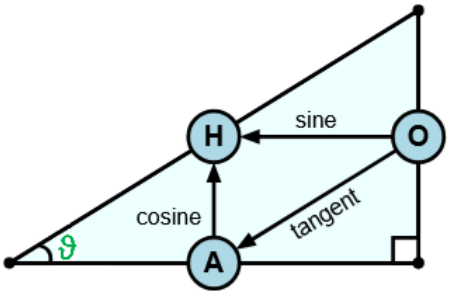
[190] [CC] [+/-] [CC] will do the same 1 keystroke faster (→ *Complex Numbers* below).

See the operating manuals of other vintage *HP* calculators (especially *HP-27*) for further applications in the area of angular mathematics (e.g., triangle solutions, navigation, and surveying).

Angles: Summary of Functions

The number of functions operating on and with *angles* is quite limited. They are important nevertheless. All are listed below. Generally, angular input is assumed being in current *ADM* unless tagged, angular output is always tagged. If you want to get rid of a tag, use [.d].

- General mathematics:
  - *Monadic* functions:



[sin], [cos], and [tan] operate on *angles* and return *reals*,[192]

[arcsin], [arccos], and [arctan] operate on *reals* and return tagged *angles* in current *ADM*,

[+/−] returns $x \times (-1)$ for closed input (a.k.a. 'unary minus').

  - *Dyadic functions:*

[+], [−], [×], and [/] work as specified in the matrices on pp. *134ff*, also for mixed tags (you can, e.g., add *radians* to *degrees*, subtract *mils* and get a correct result in current *ADM*),

[191] *TN*: Those looking for an extra challenge may compute next how flat the starship crew will become within *seconds* after the auxiliary engine is ignited.

You can't claim knowing physics was advantageous when solving such problems or watching *Star Wars* – on the other hand, we had a couple of good laughs in the cinema. Being at it, there is generally far too much noise in space movies – *Stanley Kubrick* knew better already.

Said universal problem was reprinted in the *HP-34C OHPG* in 1979. Thereafter, it vanished in hyperspace without any trace.

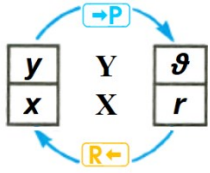
[192] See the picture: **H** denotes the hypothenuse, **A** the adjacent, and **O** the opposite side of ϑ. **Example:** $cos(\vartheta) = A/H$.

〔max〕 (or 〔min〕) returns the maximum (or minimum) of x and y.

- Rounding commands (RDP, ROUND, ROUNDI, and RSD) as well as SHOW operate on angles as they do on *reals* (→ 123f).

- Conversions:

  →P converts rectangular to polar *coordinates* (→ 22f), while R← converts vice versa (→ 145ff for examples).

  Angular conversions are covered comprehensively on p. 143.

- Advanced mathematics (→ the *IOI* and *ReMA I* for comprehensive information about the functions following):

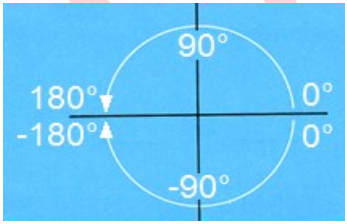
  - *Monadic* functions:

    〔sinc〕 returns $sin(x) / x$ and

    〔sincπ〕 returns $sin(\pi x) / \pi x$ for $x \neq 0$ and **1** for $x = 0$.

Further commands like 〔√x〕, 〔ln〕, 〔e^x〕, 〔y^x〕, or 〔lg〕 operate on the underlying *reals* only (→ 135, in particular f. 175).

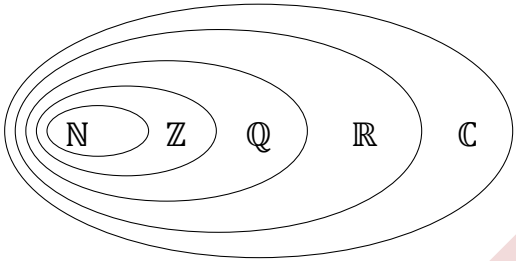
Periodic trigonometric functions (e.g. **sin**) actually operate on the interval (−180°, 180°] only (or the equivalent in *mils*, *gradians*, *multiples of* π, or *radians*); thus, any greater angular input will be reduced modulo 360° to (−180°, 180°] (or equivalents) before letting the respective function operate on it. → *ReM* for details.



Hence, also angular output is confined to (−180°, 180°] – or to (−1, 1] *multiples of* π, (−π, π] *radians*, etc. – unless in angular conversions.

Complex Numbers: Introduction

So far, we dealt with *real numbers* only – *fractions* {ℚ} and *angles* are mere subsets of *reals* {ℝ}. Your *WP43* can do more for you. Mathematicians can deal with more complex numbers than *reals*; these are called *complex numbers* (sic!) {ℂ}. If you don't know of them, feel free to skip the next 3 chapters, you can profit from your *WP43* perfectly without them – else note your *WP43* supports most functions operating on *reals* also for *complex numbers (CNs)*.



***Complex* results in calculations**: As long as you put in just *reals*, you will get only *real* results in *SDC*. To allow for *complex* results, set CPXRES (or *flag* I for *imaginary*). Try 1 +/- √x and see the different returns.

Use CC to **enter a *CN* directly** (→ *132*). With *SDC*, CC separates and concatenates the *real* and *imaginary* part in numeric input.

Examples (assuming *SDC* but FIX 1):

3 + *i* × 4 is keyed in 3 CC 4 ENTER↑ while the display shows in X row:

3	3_
CC	3.+*i*×_
4	3.+*i*×4_
ENTER↑	3.0+*i*×4.0

You enter the *real* part 1st – CC closes it – the *imaginary* part 2nd, as you write the number.[193] Input of negative *CNs* works by analogy with *real* input (→ 26). Following our case above,

3 – *i* × 4 is keyed in 3 CC 4 +/- ENTER↑,

[193] We follow *HP-42S* input tradition here. – Note that pushing 4 *CNs* on a *4-level stack* is easy on your *WP43*. It's a challenge on a *HP-42S* and *DM42* though.

–3 + *i* × 4 is keyed in 3 +/- CC 4 ENTER↑,

–31 – *i* × 42 is keyed in 31 +/- CC 42 +/- ENTER↑.

Alternatively, use 31 CC 42 ENTER↑ +/- here.

In scientific notation, e.g., SCI 5, this last number will be displayed like

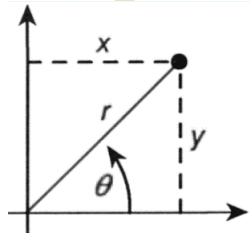
$-3.100\,00\times10^{1}-i\times4.200\,00\times10^{1}$

Different display formatting allows for showing SCI 6:

$-3{,}100\,000\cdot10^{1}-i\cdot4{,}200\,000\cdot10^{1}$

With a *CN* entered (or at least 1 *complex* input parameter in arithmetic operations or function calls), your *WP43* will set CPXRES automatically (indicated by ℂ in the *status bar*).

Instead of rectangular notation, a *CN* may be written in **polar notation** as well. Polar notation is set by SF SYS.FL (POLAR) (or SF X), causing ⊙ lit in the *status bar*. Then, for a new *CN*, its *magnitude* (or *radius*) ***r*** shall be entered 1st and its *phase* (or *argument*) ***ϑ*** second.



➡ ***ϑ*** may be entered in any *ADM* (→ 132); often, however, *multiples of* π or *radiants* make most sense here.

Example:

Set polar notation and MODE MULπ. Then a *CN* (**5**; **1.5π**) is keyed in **5** CC **1.5** ENTER↑ while the display (set, e.g., to FIX 2) shows in X successively:

5 CC **1.5** → $5.\angle1.5$

ENTER↑ → $5.00\angle-0.50\pi$

➡ ***ϑ*** is normalized to stay within the interval (**−π**, **π**], → 151. Also, if a negative magnitude is entered, it is made positive, ***ϑ*** is incremented by **π**, and then normalized. If **0** is entered for ***r*** then ***ϑ*** will be set to **0**, too.

Composing and decomposing a *CN*: Instead of entering a *CN* directly, you can generate it from 2 closed *reals* ***x*** and ***y***. If ⌊ is lit, CC will take

y as *real* part and ***x*** as *imaginary* part composing the *CN*. If ⊙ is lit, CC will take ***y*** as magnitude and ***x*** as phase of the new *CN* (compare numeric input above).

Example:

DISP ALL 00 — These 4 entries return to *startup default* settings.

MODE DEG

CF X

CF I

4 ENTER↑ −3 EXIT

```
                                  4
                                 -3
```

Note EXIT closes input without disturbing the stack.

CC composes a *CN* out of ***x*** and ***y*** now and returns[194]

```
                          4.-i×3.
```

SF X turns ⌞ to ⊙ and displays

```
                  5.∠-36.869 897 645 844 02°
```

Vice versa, CC may also cut a closed *CN* ***x*** into 2 *reals* in **X** and **Y** following the same rules.

Example (continued):

CC returns

```
r =                               5.
ϑ =     -36.869 897 645 844 02°
```

ℂ⊙ remain lit in the *status bar.*

RAD CC returns

```
                  5.∠-6.435 011 087 933×10^-1r
```

CF X turns ⊙ to ⌞ and shows

```
                          4.-i×3.
```

CC returns

```
Re =                              4.
Im =                             -3.
```

ℂ⌞∠r remain lit in the *status bar.*

Generally, *CN* outputs follow *real* formatting (→ *78ff*). The number of displayable decimals, however, may be limited by screen space.[195] If you

[194] Whenever a *CN* is returned, your *WP43* will set CPXRES and light ℂ in the *status bar.*

want to see more digits of a *CN* x, call SHOW; it will show full precision of both parts of x until next keystroke.

In calculations, press ENTER↑ for separating *CN* input as you do in *real* domain.

Example (assuming *SDC*):
(1 + 2 *i*) × (3 + 4 *i*) is entered and solved like this:

1 CC 2 ENTER↑	1. +*i*×2.
3 CC 4	3. +*i*×4_
× returning	-5. +*i*×10.

With a *CN* x closed and ~~POLAR~~, …

- +/- will change the signs of both the *real* and the *imaginary* part (as shown above),
- CPX conj will change the sign of the *imaginary* part only, and
- CPX Re⇄Im will swap *real* and *imaginary* parts.
- CC cuts it so you can continue with either part of it. This also allows returning from a *CN* with zero imaginary part to a *real*.

Also many transcendental functions will operate on *CNs* (sin, cos, tan, ln, e^x, y^x, $\sqrt{x}$, etc.). Please check pp. 159ff.

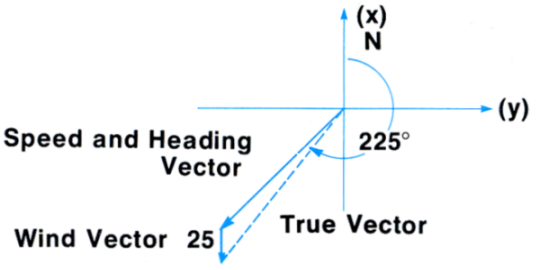
Complex Numbers Used for 2D Vector Algebra

You can use *complex* domain for *2D vector* algebra as shown below. The functions |x|, +, -, cross, dot, and UNITV wait for you; → CPX and the *IOI*.

[195] Choosing *rectangular* notation and multiplication dots allows for displaying both components using the large font in SCI 4 within $[10^{-999}, 10^{1000})$ together. It will work in SCI 5 for both components within $[10^{-99}, 10^{100})$. Staying with the *SDC* (MULT×) instead will cost you 1 displayed decimal for each component in *complex* domain.

In *polar* notation, angles are normalized to $(-\pi, \pi]$, so no space is needed for powers of 10 here. Hence, this allows for SCI 6 within $[10^{-999}, 10^{1000})$ regardless of the multiplication symbol chosen, and for SCI 7 within $[10^{-99}, 10^{100})$.

→ HIDE and RANGE in the *IOI*.



We can, for **example**, compute Mr. *Sweeney's* ground course (as explained on p. *147*) according to the following alternative method:

DISP FIX 2

MODE DEG

SF SYS.FL POLAR (or SF X)

125 CC **225** 125. ∠ 225_ indicated air speed & heading.

ENTER↑ 125.00 ∠-135.00° normalized angle (→ 151)

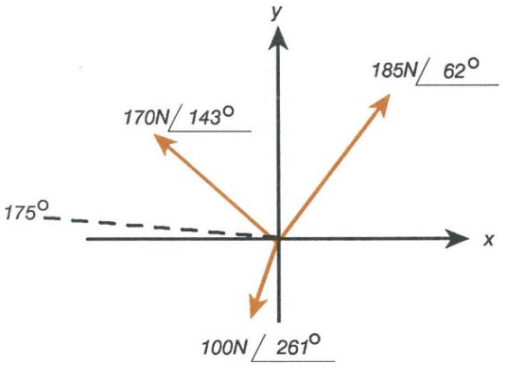
25 CC **180** 25. ∠ 180_ for north wind

\+ 143.77 ∠-142.06° the resulting *vector*.

CC r = 143.77
ϑ = -142.06°

360 + 143.77
217.94° (according to navigational convention, the angle must be positive. – Compare pp. *147f*).

2 complex **examples** more (from the *HP-42S OM*):



Dot Product of *CNs*:

The figure … represents three *2D* force *vectors*. Use *complex numbers* and add the 3 *vectors*. Then use the **dot** (*dot product*) function to find the component of the resulting *vector* along the 175° line.

Solution:

DISP FIX 2

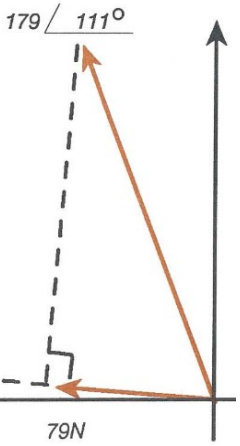
MODE DEG

SF X ensure proper *ADM* and coordinate system.

170 CC **143** ENTER↑ 170.00 ∠ 143.00°

185 CC **62** 185. ∠ 62_

⊞	270.12 ∠ 100.43°
100 CC 261	100. ∠ 261_
⊞	178.94 ∠ 111.15°



Now, take the *unit vector* at 175°:

1 CC 175	1. ∠ 175_

and multiply

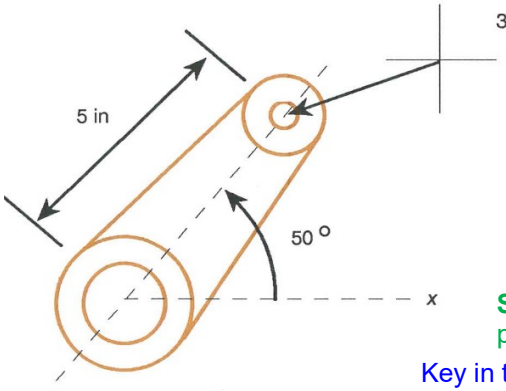
CPX dot	78.86

Thus, the resulting *vector* sum has a component of approx. 79 N in the direction of 175°.

Computing Moments:

To compute the moment of two *vectors*, use the **cross** (*cross product*) function. The cross product of two *vectors* is a third orthogonal *vector*. However, when two *complex numbers* are crossed, the *WP43* simply returns a *real number* that is equal to the signed magnitude of the resulting moment *vector*.

Find the moment generated by the force acting through the lever in the illustration …, where $\vec{M} = \vec{r} \times \vec{F}$.



➡ Lever and force are both acting in the drawing plane here.

Solution (continue with the settings of previous problem):

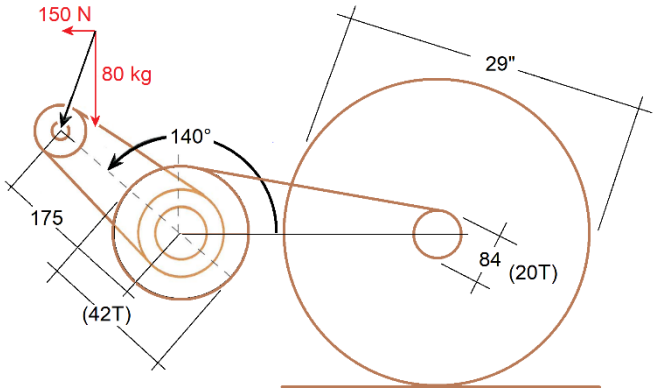
Key in the radius *vector* and the force *vector*:

5 CC 50 ENTER↑	5.00 ∠ 50.00°
300 CC 205	300. ∠ 205_
CPX cross returns $y \times x$	633.93

The moment *vector* has a magnitude of 634 *pounds times inches* and, since the result is positive, the *vector* points up, perpendicular to the plane of this page.[196]

Another **example** using a bicycle:

Assume *Willy Wheelie* starting his 29" bike with his full weight (80 kg) on the right pedal pointing 40° up. By drawing at the handlebars, he generates an additional horizontal force of 150 N. The pedal crank is 175 mm long. The front sprocket has 42 teeth, the rear sprocket of 20 teeth measures 84 mm in diameter. What moment is exerted by him initially? What force is acting along the chain then? How big is the force at the contact area between back wheel and ground? (Extra points: What about sliding?) Assume full pressure in the tires.



Solution (continue with the settings of previous problem):

See the sketch (not to scale at all!). Finding the forces and moments is complicated by the different units used, but such is real life. So let's begin with bringing everything to coherent units:

80 [+/-] [CONST] [$g_\oplus$] [×] returns -784.53 N.

Set rectangular coordinates by

[FLAG] CF **SYS.FL** [POLAR] and enter

150 [+/-] [CC] -784.53 -*i*×150.00

Set polar coordinates by

SF SYS.FL [POLAR] and enter the *vector* of the pedal crank

.175 [CC] **140.** Since $\vec{M} = \vec{r} \times \vec{F}$, we shall swap x and y before crossing:

[x≷y] [CPX] **cross** returns 108.36 Nm for the moment (→ previous page).

Now, what about the forces? The effective radius of the front sprocket is

84 [ENTER↑] **42** [×] **20** [/] **2** [/] i.e. 88.20 mm or

[E] **3** [/] 0.09 m.

---

[196] If the problem you're working requires a true (*3D*) *vector* as a result, use a 1×3 or 3×1 *matrix* to represent each *vector* in three dimensions (→ chapters after next).

Hence the force acting in the chain is

[/] 1 228.56 N.

This force acts also on the rear gear. The force at the circumference of the rear tire, i.e. at

29 [→U] [x:] [m←] inch 0.74 m,

results in

.084 [x≷y] [/] [×] 140.10 N.

Actually, it will be slightly greater since the effective radius of the tire will become about 2 cm smaller under load.[197]

Complex Numbers: Summary of Functions

Many of the functions operating on *reals* also work for *CNs*:

- General mathematics:
  - *Monadic* functions:

    [√x] and [x²], [∛x] and [x³], [√(1+x²)], [1/x], [eˣ] and [ln], [10ˣ] and [lg], [2ˣ] and [lb x], as well as [sin], [cos], [tan], [sinh], [cosh], [tanh], and their inverses work as usual (note that [arcsin] etc. returning *CNs* in polar notation will display the phases in *radians* but not tagged),

    [eˣ-1] and [ln(1+x)] return more accurate results with $x \approx 0$,

    [+/-] returns $x \times (-1)$ (a.k.a. 'unary minus') for closed input and ~~POLAR~~, while it turns x by 180° for POLAR, and

    [(-1)ˣ] returns $cos(\pi x) + i\, sin(\pi x)$ for non-integer x.

[197] "*Bei keiner anderen Erfindung ist das Nützliche mit dem Angenehmen so innig verbunden wie beim Fahrrad*." (quote by *Adam Opel*; he started as producer of sewing machines, turned to making bicycles, then automobiles; translation: '*No other invention links utility/benefit and convenience/pleasure as closely as a bicycle.*')

- *Dyadic* functions*:*

  [+], [−], [×], [/], [y^x] and [ˣ√y] work as usual,[198]

  [log_x y] computes the logarithm of y for base x,

  [dot] and [cross] allow for using *CNs* for *2D vector* computations, and

  [1/x] returns $\left({}^{1}/_{x} + {}^{1}/_{y}\right)^{-1}$ for $x \times y \neq 0$ and 0 else.

- Isolating and manipulating parts of *CNs*: use …

  [CC] for composing and cutting,

  [Re] for returning the *real* part of x and [Im] for its *imaginary* part,

  [Re⇄Im] for swapping its *real* and *imaginary* part,

  [|x|] for the *magnitude* of x and [∠] for its *phase* (a.k.a. *argument*),

  [FP] ([IP]) for the fractional (integer) parts of x (preserving their signs);

  [UNITV] returns the *unit vector* of x, and

  [conj] returns its *complex conjugate*.

- Rounding:

  [RDP] n rounds x to n decimal places in FIX format
  (**Example: 0.023456789 + *i* × 98.7654321** RDP **5** will return 0.023 46+*i*×98.765 43 , → 123),

  [ROUND] rounds x using the current display format,[199]

  [RSD] n rounds x to n significant digits, and

  [SHOW] displays the full precision of x until next keystroke.

---

[198] For odd integer x and $y < 0$, $y^{1/x} \neq \sqrt[x]{y}$. The latter function returns the correct solution here.

[199] Works like RND did on *HP-42S*.

- Advanced mathematics (→ the *IOI* and *ReMA I* for comprehensive information about the functions following):
  - *Monadic* functions:

    〔**FIB**〕 returns the extended *Fibonacci number*,

    〔$\mathbf{g_d}$〕 and 〔$\mathbf{g_d^{-1}}$〕 the *Gudermann function* and its inverse,

    〔**sinc**〕 returns $sin(x) / x$ and 〔**sincπ**〕 returns $sin(\pi x) / \pi x$ for $x \neq 0$ and **1** for $x = 0$,

    〔$\mathbf{W_p}$〕 returns the principal branch of *Lambert's W*,

    〔$\mathbf{W^{-1}}$〕 returns x for given $\mathbf{W_p}$,

    x! (= $\Gamma(x + 1)$) and 〔**Γ(x)**〕 calculate the *complex gamma function*, and 〔**lnΓ**〕 returns its natural logarithm.

  - *Dyadic* functions:

    〔**AGM**〕 returns the *arithmetic-geometric mean*,

    〔$\mathbf{C_{yx}}$〕 and 〔$\mathbf{P_{yx}}$〕 calculate with *complex gamma*,

    〔**β(x,y)**〕 returns *Euler's Beta function*, and 〔**lnβ**〕 its natural logarithm.

Vectors and Matrices: Introduction and Input

So far, we dealt with just 1 or 2 or (seldom) 3 numbers at once. Your *WP43* can do more for you – e.g., manipulate a set of numbers in a column or a row (of 2, 3, 4, or more numbers) or even in an array (of 4, 6, 8, 9, 10, 12, 14, 15, or more numbers) simultaneously. Such number columns or rows are called *vectors* and such arrays are called *matrices* by mathematicians. If you don't know of *vectors* and *matrices* yet, feel free to set them aside; your *WP43* will serve you perfectly without them.

If you know of them, however, note the function set of your *WP43* covers, e.g., *vector* additions, subtractions, and multiplications. It also allows for adding, subtracting, multiplying, inverting, and transposing *matrices*, as well as for editing and manipulating parts of such *matrices*. Furthermore, it provides functions for computing *determinants*, *eigenvalues* and *eigen-vectors*, and for solving *systems of linear equations*. Most of these operations are collected in MATX.

Generally, we talk of an ***n*×*m*** *matrix* if it features ***n*** rows and ***m*** columns of numbers. A *vector* may be regarded as a *matrix* featuring 1 column or row only.

Example:

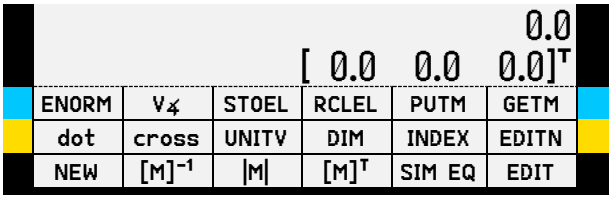
Enter a *vector* $\begin{bmatrix} 4 \\ -5 \\ 6.7 \end{bmatrix}$ and a *matrix* $\begin{bmatrix} -1 & 12 & 7 \\ 25 & 0 & 3 \end{bmatrix}$ subsequently. The *stack* shall be clear at beginning.

Solution:

DISP FIX 0 1

3 ENTER↑ 1 MATX NEW to initialize the *3D* column *vector* (i.e. a 3×1 *matrix*). See the new *matrix* in **X**:

The top *view* of MATX is displayed in the *menu section*:

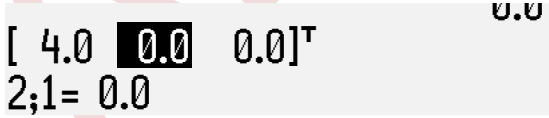


For saving screen space, your *WP43* displays each column *vector transposed* (thus the superscript **T** trailing it), i.e. in 1 row instead of 1 column on the screen. The *vector* is initialized with all its components being zero. To enter the *vector* components, press **EDIT** and the *Matrix Editor* will appear in the *menu section*:

The first *element* of the *vector* is displayed inverted now, indicating the position of the edit cursor. This particular *element* is shown below in the format set (i.e. **FIX 1** here). So we need 2 display rows for **X**.

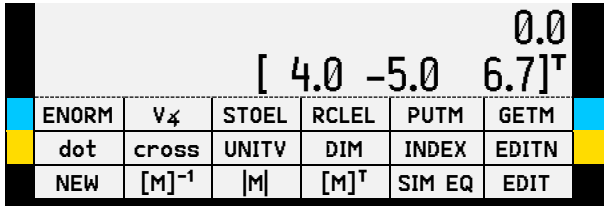
Now press **4**

Move the cursor to the next *element* (2nd row, 1st column): **→**



Continue editing: **5** **+/-** **→** **6.7**

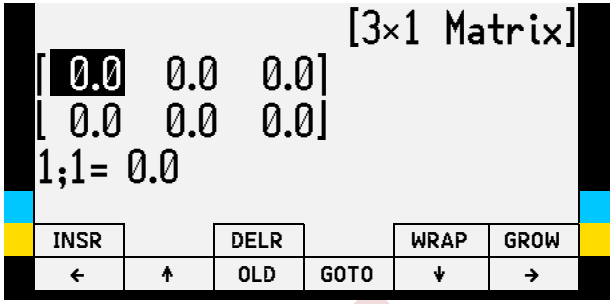
EXIT



This **EXIT** closed the last *matrix element*, left the *Matrix Editor* returning to the top *view* of MATX, closed input for the *matrix*, and shifted ***x*** to the right.

Now, let's initialize the 2×3 *matrix* via
2 **ENTER↑** **3** **NEW**

and once again call EDIT:

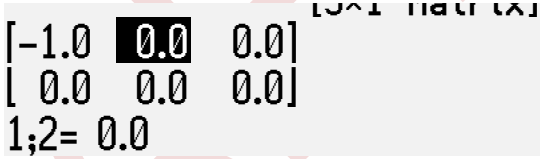


3 numeric rows are required for editing x now.

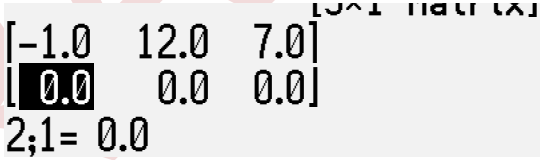
The 3×1 *matrix* in Y is the *vector* we entered just before. Every *matrix* is displayed in this short form (a.k.a. *matrix descriptor*, using a multiplication cross even for ~~MULT×~~) in any *stack register* but X.

Again, all *elements* of the new *matrix* contain zero at the beginning. Its first *element* is displayed inverted as the first *element* of the *vector* was above. *Matrix* editing will continue in the same way:

1 **+/−** **→**



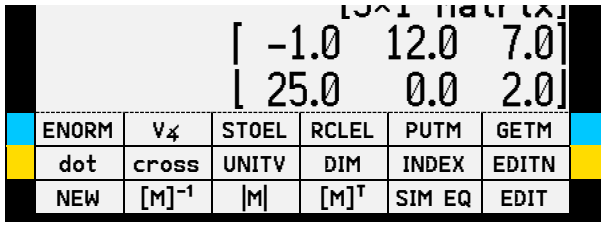
12 **→** **7** **→**



Entering the last → moved the cursor from the last (3rd) *element* of row 1 to the 1st *element* of row 2. So you can simply continue editing row-wise:

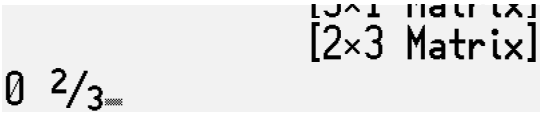
25 **→** **→** **2**

EXIT

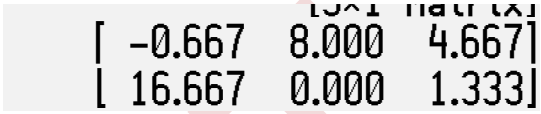


Now, also this *matrix* is closed and ready for calculating. Assume you want to multiply it by $^2/_3$ and see 3 decimals in the result:

DSP 3
. 2 . 3



Press × and you will get all *matrix elements* multiplied by $^2/_3$ at once:



You may store such *matrices* in any *RoV* . So let's store our result in **R00** – just press STO 0 0 for this.

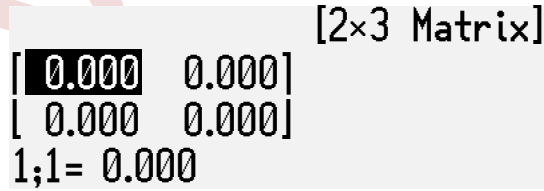
You can also create and fill a matrix directly in a *variable* (you don't have to create the matrix on the *stack* and store it afterwards).

Example:

Create a square *matrix* $[MA] = \begin{bmatrix} 4 & -3 \\ -2 & 1 \end{bmatrix}$ and fill it directly.

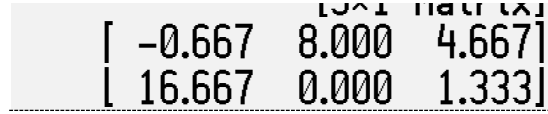
2 ENTER↑ DIM α M A ENTER↑ creates **MA** as a 2×2 *matrix*.

EDITN VAR MA



4 → 3 +/- → 2 +/- → 1

Now, press EXIT and you are done with **MA** as well – and the screen looks just as before again:



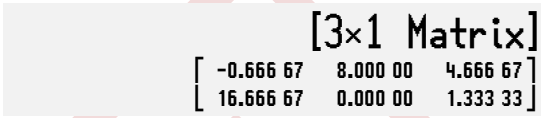
Vectors & Matrices: Displaying and Editing Larger Objects

Whenever **X** contains a *matrix*, your *WP43* will try to show it completely (i.e. display all its *elements* in the format you chose for *reals*). Objects in higher *stack registers* will be indicated in a single row each (abbreviated if necessary) or shifted out of the display window.

If space doesn't suffice for showing the entire current *matrix* in the format chosen, your *WP43* will switch to the small font automatically.

Example (continued):

DSP 5

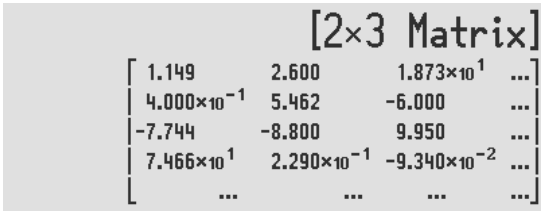


If font switching should not suffice for gaining enough screen space, your *WP43* will automatically turn to abbreviated SCI 3 format for the *elements* of the respective *matrix*. This allows for showing arbitrary *real matrices* up to 5×4 entirely.[200] If a *real matrix* exceeds 5 rows, its 5th row is displayed filled with ellipses (…); if it exceeds 4 columns, its 4th column is shown filled with ellipses.

Example:

Assume a 6×5 *matrix* $x = \begin{bmatrix} 1.1493 & 2.6 & 18.725 & 3 & 9.2 \\ 0.4 & 5.462 & -6 & 95.1 & 51.6 \\ -7.744 & -8.8 & 9.95 & 54.5 & 0.17 \\ 74.66 & 0.229 & -0.0934 & 2 & -3.829 \\ 33.9 & -79.4 & 3.436 & 9.08 & 4.256 \\ 0.0488 & 7 & 5.98 & -0.68 & -22.492 \end{bmatrix}$

was entered on the *stack* and is in **X** now. Then the screen will look like this:



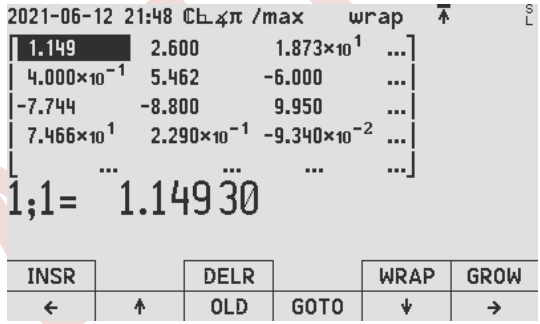
[200] ALL may be an even more favorable format sometimes. It will not be set automatically, however.

Editing such a large *matrix* will push also y from the screen as long as the *Matrix Editor* stays open. You can browse the entire *matrix* regardless of its size always.

For *matrices* larger than 5 rows and/or 4 columns, the display may vary depending on the cursor position: ellipses may appear on top and bottom, left and right side. A view of 3×3 *real matrix elements* including the one selected by the cursor can be seen always at least – this selected *element* is also displayed below of the *matrix* in the format you have chosen for *reals*. Since the indices of this *element* are shown there as well you always know where you are.

Example (continued):

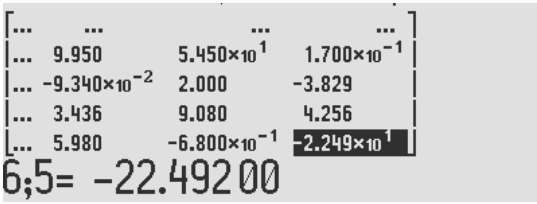
Press [MATX] EDIT and you will see:



The first *matrix element* is selected; and the **X** row shows it in FIX **5** as we had chosen.

Now, go to the bottom row of this *matrix* by pressing ↓ 5 times and you will get:

Go to the very last (bottom right) *element* of this *matrix* by pressing → 4 times:



Wherever you are within a *matrix*, you can replace or modify the currently selected *element* in 3 different ways:

1. Recall a number from any *RoV* via [RCL].
2. Simply key in a new number replacing the old one.

Example (continued):
Replace this last *matrix element* by **17.435**.

Type 1 7 . 4 3 5

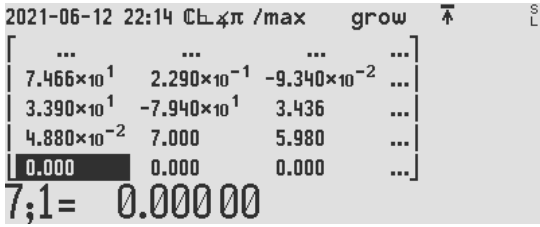
3. You can as well calculate a new value either operating on the present value of the current element or entirely from scratch, using the full function set of your *WP43*. If you need any *menus* to reach a function, they will temporarily replace the *Matrix Editor menu*; exiting those *menus* will return to it.

   Example (continued):
   If you now decide you want to keep the old *element* (**−22.492**), however, call OLD. This old content will stay available until you leave this *matrix element* by pressing ←, ↑, ↓, →, or EXIT after entering the new number.

You can also store (copy) the current *matrix element* into an arbitrary *RoV* using STO as long as the *matrix* is open for editing.

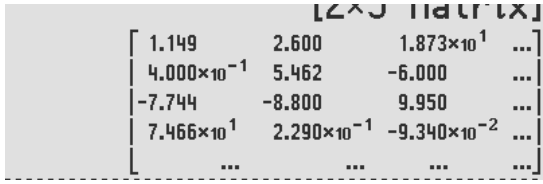
Repeatedly pushing the cursor in one direction (e.g., by →) will jump through the *matrix* rows to the last one and then return from the bottom right *element* to the top left in default ***wrap*** mode. If ***grow*** is set instead, then → from the bottom right *element* will add a new row to the *matrix*.

Example (continued):
GROW
→

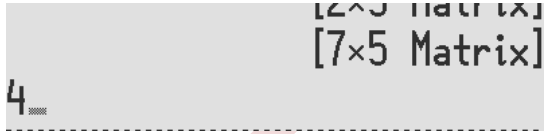


Here, we are done with this *matrix* for now. So press EXIT and you will see again:

The 1st *matrix element* is not highlighted anymore since you left the *Matrix Editor*.



Thus, just entering 4 will display, due to *ASL*:



So *matrix* editing is straightforward. The *IOI* contains additional information, also about the commands **DELR**, **INSR**, **C⇄C**, and **R⇄R** showing up in the *Matrix Editor menu*.

Vectors & Matrices: Complex Stuff

Your *WP43* supports also *complex vectors* and *matrices*, i.e. *matrices* comprising *complex elements*. They are created and initialized like *real* objects by NEW or DIM as explained above (luckily, *matrix* dimensions stay integer). Or you can recall a *real matrix* and edit it; if you enter 1 or more *complex numbers* for its *elements* it becomes a *complex matrix* – you can store it at the same or another place after editing.

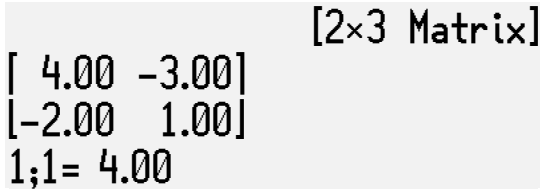
Example (continuation of p. 165):
Create and store a *complex matrix* $\begin{bmatrix} 5+8i & \pi i \\ -2 & 4-3i \end{bmatrix}$.

Solution:
Remember we have created a 2×2 *matrix* just a few pages ago. So it is most easy to recall it for using it as a template:

RCL VAR MA
DSP 2 since this will suffice for the process following.

EDIT

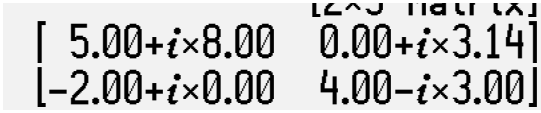
We can now enter the new *elements* here as we have done before:



5 CC 8 → 0 CC π → → 4 CC 3 +/-

EXIT

STO 0 1



Since we edited on the stack and stored the resulting new *complex matrix* in a new location, the old *real matrix* **MA** is not affected at all.

Compare pp. *152f* for the input and formatting of *complex numbers*. Everything else works as it does for *real* matrices. *Complex matrices* are no complex topic at all for you with your *WP43*.

Vectors & Matrices: Calculating

As we have seen on p. *165*, your *WP43* can multiply a *matrix* times a plain number (a.k.a. a *scalar*); doing this, each *element* of said *matrix* is multiplied by said number. Additions, subtractions, and divisions work alike for a *matrix* ***y*** combined with a *scalar* ***x***. Vice versa, with a *scalar* ***y*** and a *matrix* ***x***, additions, subtractions and multiplications will work the same way (divisions are special, see below). Also *monadic* functions will operate on each *matrix element* in your *WP43*, if applicable (except 1/x).

Examples:
With an arbitrary *matrix* in X, pressing ...

- √x will extract the square root of each *matrix element* individually. If CPXRES is set, a *real matrix* ***x*** comprising at least 1 negative *element* will become *complex* this way.
- x² will square each *matrix element* individually (instead, use ENTER↑ × for squaring the *matrix*).
- |x| will calculate the absolute value of each *matrix element* (instead, use MATX ENORM for calculating the *Euclidean norm* of the *matrix* or take |M| for getting its *determinant*).
- +/- will change the sign of each *matrix element*.

You can also let the *dyadic* functions [+], [−], [×], or [/] operate on 2 *matrices* or *vectors* (i.e. *DT* 8 and 9), as long as the rules of *linear algebra* are obeyed:

	y	x	Operation	Resulting x
[+] / [−]	[m×n Matrix]	[m×n Matrix]	$[y]+/-[x]$	[m×n Matrix]
[×]	"	[n×p Matrix]	$[y]\cdot[x]$	[m×p Matrix]
[/]	"	[n×n Matrix]	$[y]\cdot[x]^{-1}$	[m×n Matrix]

The first row reads as follows: For adding or subtracting 2 arbitrary *matrices*, both must be of identical size and shape, and the result will be of the same size and shape. Subsequent rows read by analogy.[201] If either *matrix* is *complex*, the result will often be *complex* as well.

Example (continuation of p. 166):
Multiply the *matrices* in **R00** and **MA**. Output format shall be FIX **3**.

Solution (we omit the *menu section* in the following pictures):

[DSP] [3]

[RCL] [VAR] MA (or [RCL] [α] [M] [A] [ENTER↑], → 61)

```
[2×2 ℂ Matrix]
[ 4.000 -3.000]
[-2.000  1.000]
```

The 2×2 ℂ Matrix in Y is the *complex matrix* we entered in previous chapter – the *stack* handles *matrices* as it handles other objects. Now let's recall *r00*:

[RCL] [0] [0]

```
[2×2 Matrix]
[ -0.667  8.000  4.667]
[ 16.667  0.000  1.333]
```

[201] Remember *matrix* multiplication behaves different than multiplication of *scalars*. For two arbitrary *matrices* [A] and [B] of matching size and shape, [A]·[B] ≠ [B]·[A] holds generally. Only square *matrices* can be squared.

And for *matrix* division, x must be an invertible (i.e. non-singular) *matrix* – else you can't divide by that *matrix*. Only square *matrices* can be invertible.

The '2×2 Matrix' in **Y** now is the one we have recalled from **MA** into **X** before recalling ***r00***. We multiply ***y*** times ***x*** as usual by [×] resulting in

```
[2×2 Matrix]
[-79.000  48.000  19.000]
[ 27.000 -24.000 -11.000]
```

You see that arithmetic operations on *matrices* are almost as easy as on *scalars* using your *WP43*.

And it features further *matrix* operations: |M| for computing its *determinant*, $[M]^{-1}$ for inverting, $[M]^{T}$ for transposing, M.LU and M.QR for computing the *LU* and *QR* decompositions, and 2 *norms* (*Euclid's* ENORM and the *row norm* RNORM) – please look them up in the *IOI*.

Example:

We want to invert a 2×2 *matrix* $[M] = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

Solution:

Just enter the *matrix* as usual

2 [ENTER↑] [MATX] [DIM] [α] [M] [ENTER↑] creates ***M*** as a 2×2 *matrix*.

[RCL] [VAR] M

EDIT etc. until [EXIT]

```
[2×3 Matrix]
[ 1.000  2.000]
[ 3.000  4.000]
```

[M]$^{-1}$ (or [1/x])

```
[2×3 Matrix]
[-2.000  1.000]
[ 1.500 -0.500]
```

Thus, the inverted *matrix* reads $[M]^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0{,}5 \end{bmatrix}$.

For 2 *vectors* of identical size, there are 2 special multiplications provided: **dot** and **cross**. Here, **dot** will return the dot product, a *scalar* – exactly what the table above says for ***m*** = ***p*** = **1**. And **cross** works exclusively for two *2D* or *3D vectors* and will return their cross product.

Example from the *HP-27 OH*:

The force $\vec{F}$ on a particle with charge ***q*** which is moving with a velocity $\vec{v}$ through a magnetic field $\vec{B}$ is given by $\vec{F} = q\,\vec{v} \times \vec{B}$. Suppose a proton

($q = -e = 1.6 \cdot 10^{-19}\ coulomb$) is moving with $\vec{v} = (0.4 \quad 2.8 \quad -1.2) \cdot 10^7\ {}^{m}/_{s}$. A uniform magnetic field surrounding the proton is of a strength $\vec{B} = (1.3 \quad -0.3 \quad 0.7)\ tesla$. Calculate the force on the proton. This can be written as $\vec{F} = q\ \vec{v} \times \vec{B} = 1.6 \cdot 10^{-19} \cdot 10^7 \cdot (0.4 \quad 2.8 \quad -1.2) \times (1.3 \quad -0.3 \quad 0.7)$.

Solution (FIX **1** will do):

In cross products, *vectors* must be entered in proper sequence as written from left to right:

3 [ENTER↑] **1** [MATX] **DIM** [α] [V] [ENTER↑] creates ***v*** as a 3×1 *matrix*.

[RCL] **VAR** **v**

EDIT etc. until [EXIT]

[ENTER↑]

```
                    1
[ 0.4   2.8  -1.2]ᵀ
```

[STO] [α] [▲] [B] [ENTER↑] creates also ***B*** as a 3×1 *matrix*.

[RCL] **VAR** **B**

EDIT etc. until [EXIT]

```
[3×1 Matrix]
[ 1.3  -0.3   0.7]ᵀ
```

cross returns

```
[2×2 Matrix]
[ 1.6  -1.8  -3.8]ᵀ
```

[E] **7** [×]

[CONST] [E] [F1] [×] returns

$$\begin{bmatrix} 2.6\times10^{-12} & -2.9\times10^{-12} & -6.0\times10^{-12} \end{bmatrix}^T$$

... *newtons*, of course. The total 'size' or absolute value of this force is

ENORM

$$7.2\times10^{-12}$$

Compare the weight of a proton:

[CONST] [M] [P] [F1] recall the proton mass m_p.

[CONST] [G] [F6] [×] recall the gravity acceleration $g_\oplus$ and multiply:

$$1.6\times10^{-26}$$

So this is a force ratio of

[/]

$$4.4\times10^{14}$$

Thus, physicists deliberately neglect gravitational effects in such microscopic calculations as long as no higher precision is required.

If you just want to perform elementary *vector* operations in *2D*, however, there are 2 easy alternatives (known from earlier calculators):

1. Enter the *Cartesian* components of each *vector* in $\mathbf{X}$ and $\mathbf{Y}$ (if necessary, convert polar into *Cartesian* components by [R←] before) and use [Σ+] for additions (→ *145ff* for examples – never use [Σ-] here!). Recall the result via [SUM]; it may look like this, for instance:

   ```
   Σy =      1 464.21
   Σx =        123.58
   ```

2. Calculate with *complex numbers*. In *complex* domain, [+], [−], and also *2D vector* multiplications are easily accessible since **dot** and **cross** are contained in CPX as well (→ *152ff*).

Vectors & Matrices: Solving Systems of Linear Equations

Your *WP43* can also solve simultaneous linear equations (of the kind $[A] \cdot \vec{X} = \vec{B}$) for you. To deal with such problems, proceed as follows:[202]

1. Specify the number of unknowns (e.g., 3) by entering

   [MATX] **SIM EQ** [0] [3]

   Your *WP43* automatically creates and dimensions 3 *matrices* (if necessary): Mat.A, Mat.B, and Mat.X. You will see a new *menu* showing up:

2. Press **Mat A**. The *Matrix Editor* will open and you can enter the 9 *elements* of the 3×3 *coefficient matrix* **Mat.A** (you learned on pp. *162ff* how to do this). When done, leave the *Matrix Editor* by [EXIT] to return to the *menu* shown in step 1.

3. Press **Mat B** and enter the *elements* of the 3×1 *constant matrix* **Mat.B** the same way (this is a *vector* actually).

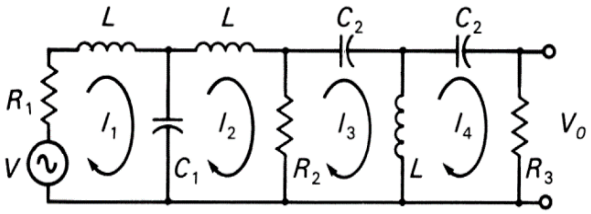
[202] This procedure works as it did on *HP-42S*. The number of unknowns is only limited by the free memory available in your *WP43* at execution time.

4. Press **Mat X** to let your *WP43* compute the 3×1 *solution matrix* **Mat.X** (a *vector* again). You are done!

To work another problem with the same number of unknowns, return to step 2 or 3. For a problem with a different number of unknowns, press **EXIT** for leaving the *menu* shown above, and start over with step 1. It goes without saying this works for *real* and *complex matrices*.

Example taken from the *AFH*:

Find the output voltage V_0 for the filter network shown. Input voltage shall be $V = 10\ \mathrm{V}$ at frequency $\omega = 15 \times 10^3\ \mathrm{rad/s}$, and $R_1 = 100\ \Omega$, $R_2 = 1\ \mathrm{M\Omega}$, $R_3 = 100\ \mathrm{k\Omega}$, $C_1 = 250\ \mathrm{nF}$, $C_2 = 25\ \mu\mathrm{F}$, and $L = 10\ \mathrm{mH}$.



Solution:

Describe the network using loop currents (via *Kirchhoff* equations):

$$\begin{bmatrix} (R_1+i\omega L-i/\omega C_1) & (i/\omega C_1) & 0 & 0 \\ (i/\omega C_1) & (R_2+i\omega L-i/\omega C_1) & (-R_2) & 0 \\ 0 & (-R_2) & (R_2-i/\omega C_2+i\omega L) & (-i\omega L) \\ 0 & 0 & (-i\omega L) & (R_3+i\omega L-i/\omega C_2) \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} V \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This system shall be solved for the 4 unknown loop currents I_1 through I_4. Then, $V_0 = R_3\, I_4$. ENG **4** will suffice for the output.

With the values given above, this system of linear equations will look like:

$$\begin{bmatrix} 100 - i\,116.67 & i\,266.67 & 0 & 0 \\ i\,266.67 & 10^6 - i\,116.67 & -10^6 & 0 \\ 0 & -10^6 & 10^6 + i\,147.33 & -i\,150 \\ 0 & 0 & -i\,150 & 10^5 + i\,147.33 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Start the solver: **MATX** **SIM EQ** **0** **4**.

Press **Mat A** and enter the elements of the big *complex matrix* using the *Matrix Editor* as explained above.

Press **Mat B** and key in the voltage *vector*.

Finally, call **Mat X** and you will get the *vector* of the loop currents:

$[\ 199.50\times10^{-6}+i\times4.096\ 4\times10^{-3}\ \ldots]^T$

Its last (4th) *element* is the current we need. Separate it with:

INDEX **VAR** **Mat X**

4 **STO** **I** set the index pointers to 4th row …

1 **STO** **J** … and 1st column and recall this *element*:

RCLEL $53.446\times10^{-6}-i\times2.259\ 9\times10^{-6}$

100 **E** **3** **×** multiply this current with R_3 and set POLAR

SF **X** $5.349\ 4\angle-2.421\ 2°$

So the output voltage is V_0 = 5.35 V with a little phase shift.

Vectors & Matrices: Eigenvalues and Eigenvectors

An *eigenvalue*[203] is a *real* or *complex number* λ solving the matrix equation $[A]\cdot\vec{X}=\lambda\cdot\vec{X}$. Then, the *vector* $\vec{X}$ is called an *eigenvector* of $[A]$. Usually, there will be more than one λ and a multitude of *vectors* $\vec{X}$ solving this problem. Thus, the simplest set of linearly independent *vectors* $\vec{X}$ is chosen to build the base of the *eigenspace* belonging to a particular *eigenvalue* found. And the simplest set of *eigenvectors* building a base of a space of the same dimension as $\vec{X}$ are called the *eigenvectors* of $[A]$. Your *WP43* can solve such problems for you as well:

Example 1:

What are the *eigenvalues* of the *matrix* $[M]=\begin{bmatrix}2 & 1\\ 6 & 1\end{bmatrix}$?

Solution:

We have got a 2×2 *matrix* named **M** already. We don't need its old contents anymore so we simply recall and edit it:

RCL **VAR** **M**

DISP **FIX** **0** **1**

[203] *TN*: The German term *eigen* may translate here to English *inherent*. *Eigenwert* can be traced to an era when German was spoken in physics and mathematics. Gone due to WWII.

MATX EDIT etc.

```
[ 2.0  1.0]
[ 6.0  1.0]
```

STO VAR M

The *eigenvalues* are the solutions of the *characteristic polynomial* of this problem $(2-\lambda)(1-\lambda)-6=0$

▲ EIGVAL returns

```
M = [2×2 Matrix]
[ 4.0  0.0]
[ 0.0 -1.0]
```

being the *matrix* with the *eigenvalues* as its diagonal *elements*. This resulting diagonal *matrix* is pushed on the *stack*.

Example 1 (continued):

Now, what are the *eigenvalues* of $[N]=\begin{bmatrix}3 & 4\\ -4 & 3\end{bmatrix}$?

Solution:

▲ EDIT etc.

```
M = [2×2 Matrix]
[ 3.0  4.0]
[-4.0  3.0]
```

STO α N ENTER↑

▲ EIGVAL returns

```
M = [2×2 Matrix]
N = [2×2 Matrix]
[ 3.0+𝑖×4.0  0.0+𝑖×0.0]
[ 0.0+𝑖×0.0  3.0−𝑖×4.0]
```

Note we get *complex eigenvalues* here although **N** comprises only *reals*.

Example 2:

What are the *eigenvalues* of $[Q]=\begin{bmatrix}0 & 0 & -2\\ 1 & 2 & 1\\ 1 & 0 & 3\end{bmatrix}$?

Solution:

3 ENTER↑ MATX DIM α Q ENTER↑ creates **Q** as a 3×3 *matrix*.

RCL VAR Q

EDIT etc.

```
[2×2 ℂ Matrix]
[ 0.0  0.0 -2.0]
[ 1.0  2.0  1.0]
[ 1.0  0.0  3.0]
```

▲ EIGVAL returns

```
Q = [3×3 Matrix]
[ 1.0   0.0   0.0]
| 0.0   2.0   0.0|
[ 0.0   0.0   2.0]
```

Note one *eigenvalue* comes twice here.

Let's get the *eigenvectors* of **Q** now – they will be put out as row *vectors* of a *matrix*:

x≷y returns **Q** into **X**

```
[3×3 Matrix]
[ 0.0   0.0  -2.0]
| 1.0   2.0   1.0|
[ 1.0   0.0   3.0]
```

EIGVEC pushes this *matrix* on the *stack*

```
Q = [3×3 Matrix]
[ 1.0   0.0  -2.0]
| 0.0   1.0   1.0|
[-1.0   0.0   1.0]
```

STO α V ENTER↑

x returns

```
[3×3 Matrix]
[ 2.0   0.0  -2.0]
| 0.0   2.0   1.0|
[-2.0   0.0   1.0]
```

RCL L recalls **V**.

1/x x≷y x returns

```
[3×3 Matrix]
[ 1.0   0.0   0.0]
| 0.0   2.0   0.0|
[ 0.0   0.0   2.0]
```

This looks very much like what was returned for the *eigenvalues* of **Q** above. Let's check:

DSP 4

− returns

```
[2×2 Matrix]
[ 0.000 0   0.000 0   0.000 0]
| 0.000 0   0.000 0   0.000 0|
[ 0.000 0   0.000 0   0.000 0]
```

So the result of $[V]^{-1} \cdot [Q] \cdot [V]$ with **V** being the *matrix* of the *eigenvectors* of **Q** is exactly the diagonal *matrix* of the *eigenvalues* of **Q**. This holds generally for any square *matrix*.

Your *WP43* can compute *eigenvalues* and *eigenvectors* for *matrices* featuring rational *elements* as well:

Example 3:

What are the *eigenvalues* of $\begin{bmatrix} -38 & 43/7 & 63/2 & 1149/14 \\ -14 & 19/7 & 7 & 181/7 \\ -8/7 & -122/49 & 24/7 & 177/49 \\ -16 & 26/7 & 13 & 244/7 \end{bmatrix}$?

Solution:

4 [ENTER↑] [MATX] **NEW** creates a 4×4 *matrix*.
EDIT **38** [+/−] → **.43.7** → **.63.2** → **.1149.14** → etc.

Each *matrix element* can be entered as an integer or fraction but is converted to a *real* following the current display format as soon as said *element* is closed.

[3×3 Matrix]
[-38.000 0 6.142 9 31.500 0 82.071 4]
[-14.000 0 2.714 3 7.000 0 25.857 1]
[-1.142 9 -2.489 8 3.428 6 3.612 2]
[-16.000 0 3.714 3 13.000 0 34.857 1]

[DSP] [0][1] shall suffice here

[-38.0 6.1 31.5 82.1]
[-14.0 2.7 7.0 25.9]
[-1.1 -2.5 3.4 3.6]
[-16.0 3.7 13.0 34.9]

[MATX] [▲] **EIGVAL** returns

[-5.0 0.0 0.0 0.0]
[0.0 -2.4×10^{-32} 0.0 0.0]
[0.0 0.0 5.0 0.0]
[0.0 0.0 0.0 3.0]

➡ The 2nd *eigenvalue* is zero here – you could hide such very small elements using **HIDE** (→ *ReMS 1*).

[RCL] [L]
[DSP] [2]
EIGVEC displays

[4.00 1.25 5.00 1.33]
[3.00 -0.50 -4.00 0.67]
[1.00 -1.00 5.00 -1.00]
[1.00 1.00 1.00 1.00]

Generally, your *WP43* solves *characteristic polynomials* numerically.

Vectors & Matrices: Dealing with Statistical Data

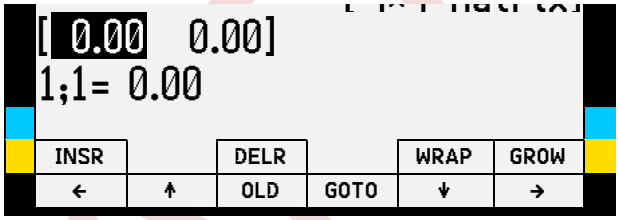
You can enter *2D* data using an input *matrix* as well as keying them in point after point. How is this done?

Let's return to the *MSA* introduced on p. *112*. Resume it with its step 4 – remember there were ≥ 30 constant samples measured twice in a special way using the measuring system under investigation, resulting in a corresponding number of pairs of measured values recorded:

4. Create a 1×2 named *matrix* and open it for editing:

   1 ENTER↑ 2 MATX DIM (α) M S A ENTER↑ creates **MSA** accordingly.

   EDITN VAR MSA



   GROW allows the *matrix* to grow with data entered.

   Now key in all recorded pairs of measured values: The 1st value shall be y, the 2nd x – so the keystroke sequence will be ***mv1*** → ***mv2***. Then, another → lets the *matrix* grow by 1 row for the next data point.

   With all points entered in the *matrix*, eventually key in

   EXIT

   (FIT) CLΣ

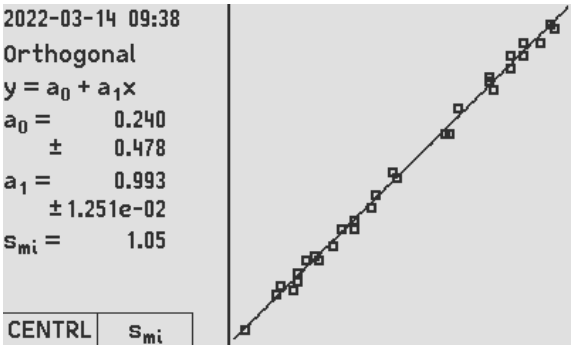
   RCL VAR MSA

   Σ+ Calling Σ+ with such a *matrix* in **X** will accumulate all your data at once and display the very last point in **X** and **Y** (and save a copy of the input *matrix* in **L**).[204]

5. Call ▲ ▭ PLOT for plotting these points in a new window.

6. Press CENTRL to fit a straight line through all these data points (a real-life dataset is plotted overleaf). Check if this line deviates significantly from the diagonal $y = x$; coarsely (with some 95% confidence), this will apply if

[204] The steps 5 through 7 here are the same as shown above in *Some Industrial Problems Solved*. They are repeated here just for your convenience. – This input method may be applied as well to other data sets, → *App. 4*.



$a_0 \pm 2\,s(a_0)$ excludes **0** or $a_1 \pm 2\,s(a_1)$ excludes **1**.

If it does then your system may be running up still (wait for equilibrium) and/or show an unstable zero (check and fix). Then restart the procedure with a new set of measurements (step 2 on p. 112).

7. Else (= if $a_0 \pm 2\,s(a_0)$ includes **0** and $a_1 \pm 2\,s(a_1)$ includes **1**) press [s_{mi}] to push the precision of your measuring instrument investigated on the *stack*. This measured precision applies under the boundary conditions prevalent during steps 2 and 3 of this test, and is definitely more realistic than most catalog values stated by manufacturers.

   Then enter **30** [×] [1/x] and multiply with the tolerance you have to control (**0.01** in this case). A result **≥ 1** will tell you this measuring instrument, device, machine, or system may be used for controlling series production with said tolerance zone (i.e. it is a *capable* instrument for this job) under said conditions; else you shall strive for a more precise device, better measuring conditions, or a wider tolerance. → *ReMA I* for more.

Vectors & Matrices: Summary of Functions

There are functions operating on an entire *matrix* (in $\mathbf{X}$, for instance) and others operating on its *elements* (x_{ij}) individually. Let's list the first set first:

- General mathematics:
  - Mostly *monadic* functions operating on the entire *matrix* $\boldsymbol{x}$:

    [ENORM] returns the *Euclidean norm* of $\boldsymbol{x}$ [205] (a *real number*),

    [RNORM] computes the *row norm* of $\boldsymbol{x}$ [205] (a *real number*),

    [RSUM] computes the row sums of $\boldsymbol{x}$ (a column *vector*),

    [|M|] returns the *determinant* of $\boldsymbol{x}$ (a *real* or *complex number*),

[205] This may be a *vector* as well – a *matrix* with just 1 row or 1 column.

〔[M]ᵀ〕 returns the transpose *matrix* of x,[205]

〔[M]⁻¹〕 and 〔1/x〕 return the inverse *matrix* of x, if applicable.

〔EIGVAL〕 returns the *eigenvalues* of x, and

〔EIGVEC〕 its *eigenvectors* (→ 176ff), while

〔UNITV〕 returns the *unit vector* of x (→ *ReM*); [205]

〔INDEX〕 indexes the *matrix* specified. [205]

- Functions operating on the *indexed matrix* (*IM*) [205] or its pointers:

  〔STOIJ〕 sets the index pointers i and j according to x and y,

  〔RCLIJ〕 recalls i and j into $\mathbf{X}$ and $\mathbf{Y}$,

  〔I+〕, 〔I-〕, 〔J+〕, and 〔J-〕 increment or decrement i and j,

  〔STOEL〕 stores x in the current *element* a_{ij} of the *IM*,

  〔RCLEL〕 recalls the current *element* a_{ij} into $\mathbf{X}$.

  〔GETM〕 copies an IP(y) × IP(x) sub-*matrix* out of the *IM* into $\mathbf{X}$, starting with its *element* a_{ij} (→ *IOI*), and

  〔PUTM〕 inserts a (smaller) *matrix* x into the *IM*, starting with target *element* a_{ij} (→ *IOI*).

  〔R⇄R〕 (or 〔C⇄C〕) swaps row (column) IP(x) and row (column) IP(y),

  〔FIND〕 searches for the value x in the *IM*,

  〔CMAX〕 (or 〔CMIN〕) finds the greatest (smallest) *element* in column j.

- *Monadic* functions operating on each *element* x_{ij} of x [205] individually:

  〔+/−〕, 〔|x|〕, 〔∠〕, 〔√x〕 and 〔x²〕, 〔∛x〕 and 〔x³〕, 〔√(1+x²)〕, 〔2ˣ〕 and 〔lb x〕, 〔eˣ〕 and 〔ln〕, 〔10ˣ〕 and 〔lg〕, 〔sin〕, 〔cos〕, 〔tan〕, 〔sinh〕, 〔cosh〕, and 〔tanh〕 as well as their inverses work as explained above,

  〔eˣ−1〕 and 〔ln(1+x)〕 return more accurate results with $x_{ij} \approx 0$;

  〔sinc〕 (or 〔sincπ〕) returns a *matrix* containing $\frac{sin(x_{ij})}{x_{ij}}$ $\left(\frac{sin(\pi\, x_{ij})}{\pi\, x_{ij}}\right)$ for $x_{ij} \neq 0$ and **1** for $x_{ij} = 0$,

〔(-1)^x〕 returns $cos(\pi\, x_{ij}) + i\, sin(\pi\, x_{ij})$ for non-integer x_{ij}.

〔RDP〕 ***n*** rounds each x_{ij} to ***n*** decimal places in FIX format,

〔ROUND〕 rounds each x_{ij} using the current display format, and

〔RSD〕 ***n*** rounds each x_{ij} to ***n*** significant digits;

〔erf〕 (or 〔erfc〕) returns the *error function* (or its complement) for each x_{ij},

〔FIB〕 the extended *Fibonacci number* for each x_{ij},

〔x!〕 calculates the *factorial* of each x_{ij} or $\Gamma(x_{ij}+1)$, respectively,

〔Γ(x)〕 the *gamma function* for each x_{ij}, while

〔lnΓ〕 computes its logarithms, and

〔ζ(x)〕 returns *Riemann's Zeta function* for each x_{ij}.

For a *real matrix*, [205]

〔ceil〕 returns a *matrix* with the smallest integers ≥ x_{ij} and

〔floor〕 with the greatest integers ≤ x_{ij} ,

〔FP〕 returns a *matrix* with the fractional parts of x_{ij} and

〔IP〕 with their integer parts, preserving their signs,

〔EXPT〕 returns a *matrix* with the exponents of each x_{ij} and

〔MANT〕 with their mantissas, while

〔sign〕 replaces each x_{ij} by its *signum* returning **1** for $x_{ij}>0$,

–1 for $x_{ij} < 0$, and **0** for $x_{ij} = 0$ or non-numeric data.

For a *complex matrix*,[205] 〔conj〕 returns a *matrix* comprising the *complex conjugates* of x_{ij}.

- *Dyadic* functions operating on entire matrices x and y:

  〔+〕, 〔−〕, 〔×〕, and 〔/〕 work as explained on pp. 170ff,

  〔dot〕 represents the dot product operating on 2 arbitrary *vectors* or *matrices* of matching size (→ 172f),

〔cross〕 computes the cross product operating on 2 *real 2D* or *3D vectors* of identical size (→ 172f), and

〔∠〕 returns the angle between 2 *real 2D* or *3D vectors* of identical size.

- *Dyadic* functions operating on each element y_{ij} of $\boldsymbol{y}$ [205] individually:

  〔y^x〕 raises y_{ij} to the power of $\boldsymbol{x}$,

  while 〔$\sqrt[x]{y}$〕 extracts the $\boldsymbol{x}^{th}$ roots of y_{ij}.

  〔$\log_x y$〕 calculates the logarithms of y_{ij} for base number $\boldsymbol{x}$.

- Isolating and manipulating bulk parts of a *complex matrix* $\boldsymbol{x}$: [205]
  Use …

  〔CX→RE〕 for cutting $\boldsymbol{x}$ in its 2 parts and storing them in $\mathbf{X}$ and $\mathbf{Y}$,

  〔RE→CX〕 for composing a complex matrix $\boldsymbol{x}$ from its 2 parts $\boldsymbol{x}$ and $\boldsymbol{y}$,

  〔Re〕 for returning its *real* part and 〔Im〕 for getting its *imaginary* part,

  〔|x|〕 for isolating its *magnitude* and 〔∠〕 for returning its *phase*, and

  〔Re⇄Im〕 for swapping its *real* and *imaginary* part.

The *Matrix Editor* was demonstrated comprehensively on pp. 162ff. Turn to the *IOI* for additional information about all *matrix* operations provided, in particular about those in MATX not mentioned in this summary.

Integers: Input and Displaying

Let's go back to the roots for a bonus *DT*: Any number you enter without using [.], [E], or [CC] is taken as an *integer* by your *WP43* (→ *22* and *132f*). It allows *integer* computing in 15 bases from binary to hexadecimal.

Any number your *WP43* displays without any punctuation mark is an *integer* (e.g., a counted value, → *96f*). And it will stay integer as long as it is exclusively combined with *integers* in additions, subtractions, and multiplications, and other integer functions operate on it; else it will be converted to another *DT* (→ matrices on pp. *134f* and *ReMS 3*). Generally, [.d] converts any closed *integer* x into *DT* 2; and any angular conversion transforms even an open *integer* into an *angle* (→ *138*).

Your *WP43* features 2 kinds of *integers*: *integers* of finite length (called *short, DT* 10) and of almost infinite length (called *long, DT* 1).

Long integers are useful for precise numeric tasks. If you enter a number of arbitrary length just without using [.], [E], or [CC], it will be taken as a *long integer* of base 10. For **example**,

1 111 111 111 [EXP] x^2 returns 1 234 567 900 987 654 321

Note this 19-digit result is right adjusted again, though no radix mark is displayed. Such a number is shown with ease. Greater *long integers* ($\geq 10^{21}$) will be displayed using a smaller font. Very big ones ($\geq 10^{43}$) will be shown with an exponent of 10 instead of their least significant digits; nevertheless, all their digits are kept internally, thus *long integers* can be of very high precision (up to 1000 digits[206]).

Example:[207]

Integers n from 1 up to 100 could all[208] be expressed as sum of 3 cubes $n = i^3 + j^3 + k^3$ – except 42 (sic!). In 2019-09, two mathematicians of Bristol and Boston published that the triple 12 602 123 297 335 631, 80 435 758 145 817 515, and −80 538 738 812 075 974 should solve this

[206] → *ReMA B* for the exact limits.

[207] → *https://phys.org/news/2019-09-sum-cubes-solvedusing-real-life.html*

[208] Unless $\mathrm{mod}(n; 9) = 4$ or 5. See 2 chapters below for explanation of mod.

problem. Verify!

12 602 123 297 335 631 EXP x^3 returns

2 001 387 454 481 788 542 313 426 390 100 466 780×10^12

SHOW shows all digits of this number:

2 001 387 454 481 788 542 313 426 390 100 466 780 457 779 044 591

80 435 758 145 817 515 x^3 + SHOW returns

522 413 599 036 979 150 280 966 144 853 653 247 149 764 362 110 466

−80 538 738 812 075 974 x^3 + returns 42 !

Another **example**:

Presumably you know that **69*!*** is the maximum factorial for all common scientific calculators – with your WP43 you can compute this factorial exactly:

69 x! SHOW

171 122 452 428 141 311 372 468 338 881 272 839 092 270 544 893 520 369 393 648 040 923 257 279 754 140 647 424 000 000 000 000 000

.d returns

1.711 224 524 281 413×10^98

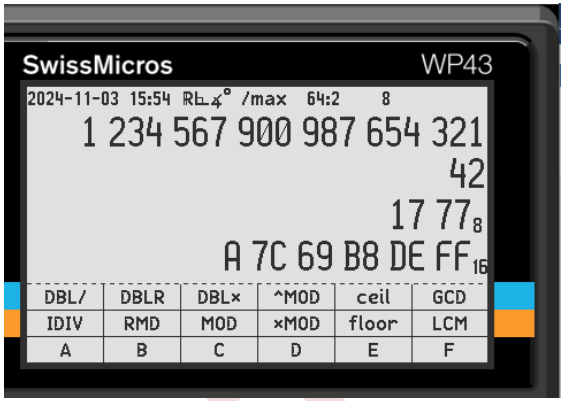
the 16-digit approximation to that *long integer* – note you can go far beyond (→ *ReM*).

This is all you need to know about entering and displaying *long integers* – feel free to jump to p. 193 for more about calculating with them.

Short integers, on the other hand, feature a finite *word* size of up to 64$bits$ ($\approx 18 \times 10^{18}$); they are particularly useful for assembler programming and close-to-hardware system design incl. debugging.[209] A *short integer* of base 2, …, 16 is entered specifying its digits

[209] Your *WP43* includes all the integer and bit manipulations of the dedicated *Computer Scientist's HP-16C*, plus all bases and the entire extended function set of the *WP 34S*, rendering a 2nd calculator obsolete. It can perform some clever calculations with hexadecimal, octal, and binary numbers that most calculator apps on a mobile phone can't do, and neither *Excel* nor whatever else one uses on the *PC* in the office.

trailed by [#] ***base*** (find [#] above [EXP]). You may abbreviate [#] [1][0] by [#] [D], and [#] [1][6] by [#] [H]. Open INTS for direct access to the digits **A** … **F** required for numeric input in bases >10 →



From the 2nd input on, you can save keystrokes: When you enter a new *integer*, your *WP43* will take it as a *short integer* of the base you specified last – as long as you did not enter any other *DT* in between.[210]

Word size and *integer sign mode* (*ISM*) set are indicated in the *status bar* using a format ww:x (e.g., **64:2** here). There, ww denotes the *word* size in *bits*, and x is **1** or **2** for *1's or 2's complement*, **u** for *unsigned*, or **s** for *sign-and-mantissa mode* (→ 139); these *ISM's* control the handling of negative *integers* (→ the example following).

CARRY and OVERFLow – if set – will be shown as **c** or **o** or **oc**, respectively, trailing *ISM* display in the *status bar*. Both are *system flags*[211] (→ 59) – if you want to set, clear, or check them, apply the commands provided in FLAG.

Example:

[SF] **SYS.FL** [LEAD.0] (or [SF] [L])	
[INTS] [▲] **WSIZE** [1][2]	This allows seeing all bits easily at a glance.
147 [EXIT]	Enter 147 as a (decimal) *long integer*.
[▲] **→BIN**	Convert it to a binary *short integer*

[210] This shortcut will close when you key in [.], [E], [CC], or [#], even if you delete this keystroke thereafter.

Illegal input (e.g., **2** in base 2 or **B** in base 10) can be detected no earlier than said input is completed. And you may key in more than the current *word* size can take; also this will be checked when input is closed. An error message can be thrown after closure the earliest.

[211] They behave like they did on *HP-16C* and *WP 34S*.

▼ 1COMPL set 1's complement, and you will see [212]

$0000\ 1001\ 0011_2$ and – after +/− – $1111\ 0110\ 1100_2$.

Obviously +/− in 1COMPL flips every bit, equivalent to NOT here.

Return to the original number via +/−, press 2COMPL, and you will get

$0000\ 1001\ 0011_2$ and – after +/− – $1111\ 0110\ 1101_2$.

Note the negative number equals the inverse + 1 in 2COMPL.

Return via +/− again, press SIGNMT and you will see

$0000\ 1001\ 0011_2$ and – after +/− – $1000\ 1001\ 0011_2$.

Negating a number will just flip the top bit in SIGNMT (hence its *name*).

Return via +/− once more, press UNSIGN and you will get

$0000\ 1001\ 0011_2$ and – after +/− – $1111\ 0110\ 1101_2$.

The 2nd number looks like in 2COMPL but an *overflow* is indicated here – see ᴼ in the *status bar* trailing the *ISM*.[213] Thus, pressing +/− will not suffice for returning to the original number here anymore; you must clear OVERFLOW explicitly by CF SYS.FL (OVERFL) or CF B.

So positive numbers stay unchanged in all these 4 modes. Taking a negative *short integer*, however, in one *ISM* and switching to another will lead to different interpretations and, thus, different displays.

Example (continued):
The fixed bit pattern representing the *short integer* …

-147_{10} in 12:2 will be displayed as…

-146_{10} in 12:1 , as…

$-1\ 901_{10}$ in 12:S , and as…

$3\ 949_{10}$ in 12:U . You can verify this easily.

[212] Note the gap automatically inserted every 4 *bits* here for easier reading.

[213] Actually, changing signs should have no meaning in unsigned mode per definition. Thus, +/− should be either illegal here or result in no operation. "In unsigned mode, the most significant bit adds magnitude, not sign, so the largest value represented by a 12-*bit* word is 4095 instead of 2047" (quoted from the *HP-16C Computer Scientist Owner's Handbook* of April 1982, p. 30).

Keeping the *ISM* and changing bases will produce different views on the constant bit pattern as well.

Example (continued):

WSIZE 16, 2COMPL

−16123 EXIT	−16 123	to base 10 corresponds to
# 2	$1100\ 0001\ 0000\ 0101_2$	to base 2,
# 3	$-211\ 010\ 011_3$	to base 3,
# 4	$30\ 01\ 00\ 11_4$	to base 4,
# 5	$-1\ 003\ 443_5$	to base 5,
# 6	$-202\ 351_6$	to base 6,
# 7	$-65\ 002_7$	to base 7,
# 8	$14\ 04\ 05_8$	to base 8,
# 9	$-24\ 104_9$	to base 9,
# D	$-16\ 123_{10}$	to base 10 as expected,
# 11	$-11\ 128_{11}$	to base 11,
# 12	$-9\ 3B7_{12}$	to base 12,
# 13	$-7\ 453_{13}$	to base 13,
# 14	$-5\ C39_{14}$	to base 14,
# 15	$-4\ B9D_{15}$	to base 15, and
# H	$C1\ 05_{16}$	to base 16.

You may have noticed that the displays for bases 2, 4, 8, and 16 look similar, presenting all 16 *bits* to you, while a signed mantissa is displayed for the other bases instead. There are also different separator intervals

---

+/− in unsigned mode was allowed, however, by the designers of *HP-16C* and implemented as shown above. So we followed that implementation for sake of backward compatibility (as we did with *WP 34S*), though frowning.

Luckily, however, this inherited error carries a benefit: You can easily (by a single key press) determine that a very big positive unsigned number in display may correspond to a small negative number for the *word* size chosen. A quite useful feature.

(4 bits for binary, 2 digits for bases 4, 8, and 16, and 3 for all other bases); they are fixed unless **GAP 0** is set by you.

These different formats consider that bases 2, 4, 8, and 16 are most convenient for bit and byte manipulations and further close-to-hardware applications. The bases in between will gain most interest in dealing with different number representations and calculations therein, where base 10 is the common reference standard. [214]

Let's turn to bigger *words* now:[215]

Example (continued):

CF L

WSIZE 6 4

UNSIGN

▼ 9 3 A 1 4 C 6 # H (→ 187).

Then your *WP43* will display

$9\ 3A\ 14\ C6_{16}$

In binary representation, this number will need 28 digits, looking like

$1001\ 0011\ 1010\ 0001\ 0100\ 1100\ 0110_2$

Obviously, such a big binary number can't be displayed in 1 row this way. Look what your *WP43* does instead – enter ▼ →BIN and you will get:

$1001\ 0011\ 1010\ 0001\ 0100\ 1100\ 0110_2$

It takes the small font here; then **44 digits can be shown in 1 row**, sufficient for a 64-*bit* number to base 3. If also leading zeros were turned on by SF L, all **64 *bits* can be shown in 1 row** using a minimal font for binary display:

.... 1..1 ..11 1.1. ...1 .1.. 11.. .11.₂ [216]

---

214 During numeric input, gaps are inserted suiting the base chosen (as above). If there is no such base yet, **GAP 3** applies since your *WP43* cannot know your intentions in advance.

215 Up to 16 hexadecimal digits are possible for unsigned *short integers*.

216 If you want to convert hexadecimal numbers to *split octal*, you can do so inserting a **0** after every 2 hex digits. Vice versa you shall insert a **0** after every 3 octal digits.

Example: AB**0**CD**0**EF**0**FF$_{16}$ = 253**0**315**0**357**0**377$_8$.

Integers: Bitwise Operations on Short Integers

As mentioned, your *WP43* carries all the bitwise operations you may know from the vintage *HP-16C Computer Scientist*, plus some more you might have learned with *WP 34S*. You find them all in BIT.

SB	BS?	#B	FB	BC?	CB
NAND	NOR	XNOR	MIRROR		
AND	OR	XOR	NOT	MASKL	MASKR

Generally, the bits in a *word* are counted from right to left, starting with number **0** for the least significant bit (this is important for specifying correct bit numbers in the operations **BC?**, **BS?**, **CB**, **FB**, and **SB**).

The **examples** below all deal with 8-*bit words* with leading zeros for easy reading:

SF L
BIT ▼ WSIZE 8

Common input	
Y	0110 1011₂
X	1011 1001₂

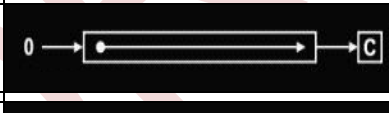
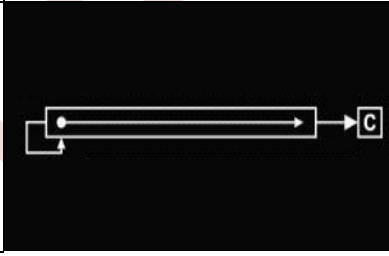
Let's look at the 6 bitwise *dyadic* (*Boolean*) operations (all found in the 1st view of BIT shown above) →

Operation	Symbol	Output
AND	∧	0010 1001₂
NAND	⊼	1101 0110₂
OR	∨	1111 1011₂
NOR	⊽	0000 0100₂
XOR	⊻	1101 0010₂
XNOR		0010 1101₂

See 12 bitwise *monadic* operations (first 5 from the 1st view of BIT, last 7 from its 2nd view) in the table below with schematic pictures as they were printed on the back of the *HP-16C* (the boxed C represents the *carry* bit, indicated in the *status bar* if set):

Common input → 1011 0011₂

Operation	Schematic picture if applicable	E. g.	Output
Clear Bit		CB 4	1010 0011₂
Flip Bit		FB 5	1001 0011₂
Set Bit		SB 6	1111 0011₂

Operation	Schematic picture if applicable	E. g.	Output
		Common input →	1011 0011$_2$
Negate[217]		NOT (¬)	0100 1100$_2$
Mirror		MIRROR	1100 1101$_2$
Rotate Left		RL 1 RL 2	0110 0111$_2$ C 1100 1110$_2$
Rotate Left through Carry		RLC 1 RLC 2	0110 0110$_2$ C 1100 1101$_2$
Rotate Right		RR 1 RR 2 RR 3	1101 1001$_2$ C 1110 1100$_2$ C 0111 0110$_2$
Rotate Right through Carry		RRC 1 RRC 2 RRC 3	0101 1001$_2$ C 1010 1100$_2$ C 1101 0110$_2$
Shift Left		SL 1 SL 2	0110 0110$_2$ C 1100 1100$_2$
Shift Right		SR 1 SR 2	0101 1001$_2$ C 0010 1100$_2$ C
Arithmetic Shift Right		ASR 3	in 1/ 2COMPL: 1111 0110$_2$ in UNSIGN: [218] 0001 0110$_2$ in SIGNMT: [218] 1000 0110$_2$

[217] *TN*: Remember: $(🍏\wedge\neg🍐)\vee(\neg🍏\wedge🍐)=(🍏\vee🍐)\wedge(\neg🍏\vee\neg🍐)$.
What the English do with apples and oranges, Germans do with apples and pears.

[218] *TN*: The schematic picture correctly describes **ASR** for *1's* and *2's complement* only. Executing **ASR** three times on the *HP-16C*, however, equals a signed division by 2^3 in all its modes, hence the different results for the latter two *ISMs* above.

Furthermore, you can create bit masks using **MASKL** and **MASKR**.

Example:
For **WSIZE 8**, **MASKL 3** will return a mask *word* $1110\ 0000_2$. Use it, e.g., for extracting the top 3 bits of an arbitrary *byte* via **AND**.

See the *IOI* for these and more commands operating on bit level on *short integers* (**LJ**, **RJ**, **#B**, and the tests **BS?** and **BC?**). Finally, note that no such operation will set OVERFL. CARRY is only settable by shift or rotate functions as demonstrated above. And **ASR** is the only bitwise operation being sensitive to *ISM* – **ASR** is the link to integer arithmetic operations.

Integers: Arithmetic Operations

Of the 4 basic arithmetic operations (+, −, ×, and /), the first 3 work with *integers* as they do with *reals*; the only difference lies in precision: up to 64 digits for *short integers* in binary representation or even up to 1000 digits for *long integers*. Take [+/−] as a multiplication times −1, and [y^x] as repeated multiplication. Depending on input parameters and mode settings, OVERFL or CARRY may be set in such an operation (→ *196ff*).

Divisions, however, must be handled differently in integer domain since the quotient can't feature a fractional part here. Generally, the formula

$$\frac{a}{b} = (a \operatorname{idiv} b) + \frac{1}{b} \times \operatorname{rmd}(a : b)$$

applies where the horizontal bar denotes a *real* division, idiv represents integer division, and rmd stands for the remainder of the latter. So

$$\operatorname{rmd}(a : b) := a - b \times (a \operatorname{idiv} b)$$

Examples:

$\frac{25}{7} = 3 + \frac{1}{7} \times 4$ (and for a *real* case: $\frac{25}{7.5} = 3 + \frac{1}{7.5} \times 2.5$)

$\frac{-25}{7} = -3 + \frac{1}{7} \times (-4) \quad \Rightarrow \quad \operatorname{rmd}(-25 : 7) = -4$

$\frac{25}{-7} = -3 + \frac{1}{-7} \times 4 \quad \Rightarrow \quad \operatorname{rmd}(25 : -7) = 4$

$$\frac{-25}{-7} = 3 + \frac{1}{-7} \times (-4) \quad \Rightarrow \quad \mathrm{rmd}(-25 : -7) = -4$$

There is another function doing almost the same: it is called mod (read *modulo*). With the same input as above, it returns:

$$\mathrm{mod}(25 : 7) = 4\ ,$$
$$\mathrm{mod}(-25 : 7) = 3\ ,$$
$$\mathrm{mod}(25 : -7) = -3\ ,$$
$$\mathrm{mod}(-25 : -7) = -4\ .$$

So mod returns the same as rmd only if both arguments have equal signs. The general formula for mod reads:

$$\mathrm{mod}(a : b) := a - b \times \mathrm{floor}\left({}^{a}/_{b}\right) \quad \text{with, e.g.,} \quad \mathrm{floor}\left(\frac{25}{7}\right) = 3$$
$$\text{and} \quad \mathrm{floor}\left(-\frac{25}{7}\right) = -4\ .$$

Also this formula applies to *reals* as well. Use it straightforwardly for calculating, e.g.,

$$\mathrm{mod}(25.3 : -7.5) = 25.3 - (-7.5) \cdot (-4) = -4.7$$

These 4 functions are called **IDIV**, **RMD**, **MOD**, and **floor**[219] on your *WP43* for obvious reasons. They are all found in the first view of INTS (→ *187*), together with more integer operations like **ceil**, **×MOD**, and **^MOD** (→ 200 for an instance). MOD is also printed on the keyboard as 🟦-shifted function of /.

Furthermore, many exponential and logarithmic operations, x^2 and √x, x^3 and $\sqrt[3]{x}$, x!, **COMB**, **PERM**, as well as **sin**, **cos**, and **tan** can operate on *integers*, too. Some of them will stay in integer domain while others may return *real* or even *complex numbers*. See the summary on pp. *199f* for further information.

[219] Both **floor** and **ceil** can operate also on *reals* and return *long integers*.

Integers: Overflow and Carry with Short Integers

In arithmetic operations on *short integers* on your *WP43*, OVERFLOW and/or CARRY may be touched under certain conditions.

Note for each *word* size and *ISM* there is a maximum and a minimum integer displayable: let's call them I_{max} and I_{min}.

Example:

For 4-*bit words* (i.e. WSIZE **4**), we get

- I_{max} = 15 and I_{min} = 0 for 4:U, while
- I_{max} = 7 and I_{min} = −8 for 4:2,
- I_{max} = 7 and I_{min} = −7 for 4:1 and 4:S.

Let's start from **1** incrementing by **1** and see what will happen in these modes. And whenever OVERFLOW and/or CARRY will be lit in the *status bar* in this course, we will clear them (using CF SYS.FL [OVERFL] (or CF B) and/or CF SYS.FL [CARRY] (or CF C)) before next increment:

4:U		4:2		4:1		4:S	
0001_2	1	0001_2	1	0001_2	1	0001_2	1
0010_2	2	0010_2	2	0010_2	2	0010_2	2
…	…	…	…	…	…	…	…
0111_2	7	0111_2	7	0111_2	7	0111_2	7
1000_2	8	1000_2	−8	1000_2	−7	1000_2	−0
1001_2	9	1001_2	−7	1001_2	−6	0001_2	1
…	…	…	…	…	…	0010_2	2
1110_2	14	1110_2	−2	1110_2	−1		
1111_2	15	1111_2	−1	1111_2	−0		
0000_2	0	0000_2	0	0001_2	1		
0001_2	1	0001_2	1	0010_2	2		

For comparison, we start another turn from **1** following the same rules but decrementing by **1** instead:

4:U		4:2		4:1		4:S	
0001_2	1	0001_2	1	0001_2	1	0001_2	1
0000_2	0	0000_2	0	0000_2	0	0000_2	0
1111_2 O C	15	1111_2 C	−1	1110_2 C	−1	1001_2 C	−1
1110_2	14	1110_2	−2	1101_2	−2	1010_2	−2
…	…	…	…	…	…	…	…
1001_2	9	1001_2	−7	1000_2	−7	1111_2	−7
1000_2	8	1000_2	−8	0111_2 O	7	1000_2 O	−0
0111_2	7	0111_2 O	7	0110_2	6	1001_2	−1
0110_2	6	0110_2	6	…	…	1010_2	−2
…	…	…	…	…	…		
0010_2	2	0010_2	2	0001_2	1		
0001_2	1	0001_2	1	0000_2	0		

The most significant bit is #2 in 4:S and #3 in all other modes here.

With these results, I_{max}, and I_{min}, the general rules for setting and clearing CARRY and OVERFL are as printed below:

Operation	Effect on CARRY	Effect on OVERFL
Shift and rotate	As demonstrated on pp. *191f*.	None.
Boole's, MIRROR	None (→ *191f*).	None.
|x|, ABS	None.	Clears OVERFL (but sets it for $x = I_{min}$ in 2COMPL).
+, RCL+, STO+, INC, etc.	Sets CARRY if there is a carry out of the most significant bit, else clears CARRY.	Sets OVERFL if the result exceeds the interval $[I_{min}, I_{max}]$, else clears OVERFL.

Operation	Effect on CARRY	Effect on OVERFL
-, **RCL-**, **STO-**, **DEC**, etc.	Sets CARRY in a subtraction $m - s$ • in 1COMPL or 2COMPL if the binary subtraction causes a *borrow* [220] into the most significant bit, • in UNSIGN if $m < s$, • in SIGNMT if $m < s$ & $m \cdot s > 0$ Else clears CARRY.	Sets OVERFL if the result exceeds $[I_{min}, I_{max}]$, else clears OVERFL.
×, **RCL×**, **STO×**, **+/-**, **(-1)^x**, x^2, x^3, **LCM**, **x!**, etc.	None.	Thus, in **UNSIGN**, +/- always sets OVERFL and **(-1)^x** does so for odd x.
2^x	Clears CARRY. Sets CARRY only if $x = -1$, or in UNSIGN if $x = wsize$ or in the other modes if $x = wsize - 1$	
y^x, 10^x	Sets CARRY for $x < 0$ (as well as for 0^0), else clears CARRY.	Sets OVERFL if the result exceeds $[I_{min}, I_{max}]$, else clears OVERFL.
e^x	Sets CARRY for $x \neq 0$, else clears CARRY.	
DBL×	None.	Clears OVERFL.
/, **RCL/**, **STO/**, **DBL/**, **ln**, **lg**, **lb**, $\log_x y$,	Sets CARRY if the remainder is ≠ 0, else clears CARRY.	Clears OVERFL but sets it in **2COMPL** for the division $I_{min} / (-1)$

[220] Compare the examples on previous page.

TN: The so-called *borrow* in subtraction seems to be a specialty of the *USA*. Compare the subtle methodic differences in manual subtracting explained in *http://de.wikipedia.org/wiki/Subtraktion#Schriftliche_Subtraktion* (the corresponding article in the English *Wikipedia* is distinctly less instructive).

TNG: Sowohl *carry* als auch *borrow* entsprechen einem *Übertrag*.

Short Integers Applied: A Pseudo Random Number Generator

A straight *PRNG* **example** program for *HP-16C* was posted (search for '*63-bit LFSR*'). Here it is translated for your *WP43* (starting in *SDC*):

567890 1234567890 123 # D STO K	store seed
P/R	turn to *program entry mode*
GTO . .	go to free space
LBL α A ENTER↑	begin of program
RCL K	
BIT ▲ RR **62**	rotate bit 62 into bit 0
RCL K	
RR **61**	rotate bit 61 into bit 0
▼ XOR	bit 0 = bit 61 XOR bit 62
▲ SR **01**	shift bit 0 into CARRY
RCL K	
RLC **01**	multiply ***k*** times 2 and add CARRY
▼ MASKR **63**	set up a 63-*bit* mask right adjusted
AND	ensure a 63-*bit* output
STO K	store next seed
RTN	end of program

program A

P/R	return to *run mode*	
XEQ PROG A	returns	$2\,134\,430\,432\,281\,004\,439_{10}$
XEQ PROG A	returns	$4\,268\,860\,864\,562\,008\,878_{10}$
XEQ PROG A	returns	$8\,537\,721\,729\,124\,017\,757_{10}$
XEQ PROG A	returns	$7\,852\,071\,421\,393\,259\,706_{10}$
XEQ PROG A	returns	$6\,480\,770\,805\,931\,743\,604_{10}$

This *PRNG* has a period of $2^{63} - 1 \approx 9 \times 10^{18}$, so it is 'pretty random'. The sequence of its returns, however, lacks randomness – hence it is called *pseudo* (→ *ReMA J*). After all, this is for demonstration purposes only.

Being at it, an astounding good while simple *PRNG* is given by the sequence ***r*** := FP(**997** × ***r***).[221]

[221] Thanks to *Namir Shammas* for pointing me to this.

Integers: Summary of Functions

Many of the numeric functions operating on *reals* also work for *integers*. Functions operating on *short integers* (*DT* 10) will return *DT* 10 except for the cases given on pp. 134ff. Functions operating on *long integers* (*DT* 1), on the other hand, may return results of *DT* 1, 2, 3, or 4:

- General mathematics:
  - *Monadic* functions:

    [√x], [∛x], [lb x], and [lg] operating on *DT* 1 return *DT* 1 if possible (else *DT* 2 or 3) [222]
    or just the integer part of the solution for an input of *DT* 10;

    [e^x] and [ln] operating on *DT* 1 return *DT* 2 or 3 always;
    for *DT* 10, they work in analogy to [√x];

    [#] converts a closed *integer* into an arbitrary other base; [→BIN], [→OCT], [→DEC], and [→HEX] cover the most popular bases.

    [|x|], [x²], [x³], [2^x], and [10^x] return *integers* as you expect, and

    [+/-] works for *short integers* as demonstrated on pp. 188f.
    For *long integers*, it works as for *reals*.

    [SHOW] displays full precision of x in top numeric row(s) until next keystroke. For very *long integers* >10^{296}, see the *IOI*.

  - *Dyadic functions:*

    [+], [−], and [×] return *integers* as expected (→ 134f),

    [/] returns a *short integer* for *short integer* input while it may return a *real* for *long integer* input as specified in f. 176 on p. 136,

    [IDIV] returns just the integer part of the division,

    [RMD] returns the remainder of y/x (→ 193 for examples),

[222] For instance, **64** [√x] returns *integer* 8 while **65** [√x] will return 8.062… By analogy, **8** [∛x] returns 2, **1024** [lb x] returns 10, **1000** [lg] returns 3, **1296** [ENTER↑] **4** [ˣ√y] returns 6, and **625** [ENTER↑] **5** [log_xy] returns 4 (all these results being *integers*). For **−64**, [√x] may return 0.+*i*×8. (→ *Complex Numbers* above).

[IDIVR] combines IDIV and RMD, returning the remainder in Y,

[MOD] returns $y \bmod x$ (→ *194f* for examples),

[y^x], [$\sqrt[x]{y}$], and [$\log_x y$] work as specified on pp. *136f*;[222]

[max] (or [min]) return the maximum (or minimum) of x and y,

[GCD] returns the *Greatest Common Divisor* of x and y and

[LCM] their *Least Common Multiple*.

- *Triadic* functions:

  [×MOD] returns $(z \times y) \bmod x$ and

  [^MOD] returns $(z^y) \bmod x$, both for $x > 1$, $y > 0$, $z > 0$

  (**Example:** **73** [ENTER↑] **55** [ENTER↑] **31** [INTS] [^MOD] returns **26**).

- *Boole's* algebra:

  [AND], [NAND], [OR], [NOR], [XOR], [XNOR], and [NOT] operate bitwise on *short integers* as shown on p. *191*.
  They operate on *long integers* like they do on *reals*, → *124*.

- Bitwise operations are for *short integers* exclusively:

  [CB], [FB], [SB], [ASR], [SL], [SR], [RL], [RLC], [RR], [RRC], and [MIRROR] work as demonstrated on pp. *191f*.

  [LJ] ([RJ]) justifies the bit pattern to the left (right) within its *word* size,

  [MASKL] and [MASKR] create mask *words* (→ *193*),

  [BC?] and [BS?] test if the specified bit is clear or set,

  [#B] counts the number of bits set in x.

- Probability (→ *96f*):

  [x!] returns the factorial of $x \le 450$,

  [C_{yx}] computes the number of *combinations* of x and y,

  [P_{yx}] the number of *permutations*; and

〔RANI#〕 returns a (pseudo) random *integer number* $\in [x, y]$. Use it for throwing dice, for example.

- Advanced mathematics (→ the *IOI* and *ReMA I* for comprehensive information about the functions following):

〔B_n〕 and 〔B_n*〕 return the *Bernoulli numbers* and

〔FIB〕 the *Fibonacci number*;

〔PRIME?〕 tests if $|x|$ is a *prime number* and

〔NEXTP〕 returns the next *prime number* > $|x|$. For large long integers, NEXTP may take many seconds. Feel free to abort it by 〔EXIT〕.

〔FACTOR〕 returns the *prime factors* of a positive x in a *real matrix*. Factorizing an integer containing a large prime factor may take many seconds – we recommend checking x with PRIME? first. Feel free to abort FACTOR by 〔EXIT〕.

Example:

123 456 789 012 FACTOR returns

```
[ 2.   3.   10288065751.]
[ 2.   1.            1.]
```

since 123 456 789 012 = $2^2 \times 3 \times 10\,288\,065\,751$

Many more functions accept *integer* input but will return different, mostly *real* output. → *IOI* and *ReMS 3* for more about them.

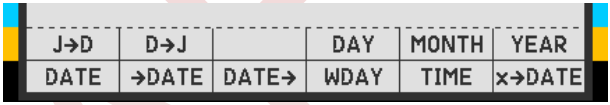
Dates

For *date* calculations, choose 1 out of 3 *date display modes* (*DDM*) on your *WP43*: YMD (*startup default*), DMY, and MDY – find the respective mode-setting commands in the 2nd *view* of CLK.

Date input is decimal according to the *DDM* chosen and terminated by .d (as shown on pp. *132f*).

Example:
The 26th of November in 2024 is entered

2024.1126 .d after Y.MD,
26.112024 .d after D.MY, and
11.262024 .d after M.DY.



Any *real* may be converted into a *date* by CLK x→DATE, [223] and any triple of *reals* or *integers* on the *stack* by →DATE (→ *44*).

Vice versa, DATE→ splits a *date* in 3 *integers* and pushes them on the stack as demonstrated on p. *44*. Both DATE→ and →DATE (and also x→DATE) observe the current *DDM*. If you want to extract particular information from a *date* independent of current *DDM*, use one of the operations DAY, MONTH, or YEAR.

Like in the *status bar*, a closed *date* input or a *date* output returned by a function (e.g., by DATE) is displayed as in the following **example**:

Thursday 2023-03-09 18:29:39 for YMD,
Thursday 09.03.2023 18:29:39 for DMY,
Thursday 03/09/2023 18:29:39 for MDY (with TDM24 set)..

So you know the present *DDM* immediately from looking at the date string top left in the *status bar*.

[223] Digits (actually decimals) exceeding those needed for a *date* in the *DDM* chosen will be interpreted as daytime, → next chapter.

CLK **WDAY** takes a *date* from the *stack* – or a decimal input of, for **example**, **2013.0504** in YMD (equivalent to inputs of **4.052013** in DMY or **5.042013** in MDY) – and returns an *integer* indicating the position of this day in the corresponding week, headed by the name of this weekday as *temporary information*:

```
Saturday                                6 .
```
[224]

Expect similar returns of **DATE** .

Use + to add an *integer* (or *real* [225]) number of days to a *date* – and − to subtract such a number from a *date*. The result will be a *date* again. And a *date* may be subtracted from another *date*, resulting in a difference of days. Compare the matrix on p. *134*.

Example (assuming *SDC*):
The interval between 1931-11-09 and 2022-03-26 amounts to 90 years and almost 5 months. How many days were these? And how many hours?

---

[224] Numeric output of **WDAY** corresponds directly to Chinese weekdays 1 to 6. For Portuguese weekdays (*segunda-feira* etc.), add 1 to days 1 to 5.

Calculation of weekdays for the past depends on the calendars used at that time – there may be different true results for different countries depending on the date the particular country introduced the *Gregorian calendar*. Officially, it became effective at 1582-10-15 for the catholic world. Large regions took their time and switched later. Store the proper switch date for your geographic area of interest using **J/G** (→ *Localizing Calculator Output*) – check *Wikipedia* for the precise date applicable.

Dates before the year 8 AD may be indicated differently than they were experienced at that time due to inconsistent application of the leap year rule. We count on your understanding and hope this shortcoming will not affect too many calculations.

TN: 8 AD should better read AD VIII instead. But nobody counted years this way at that time – around the Mediterranean Sea, it was the year DCCLXI A.V.C. (AB VRBE CONDITA – since the founding of Rome) in best case (actually, this notation was broadly implemented some XL – or even CD – years later). Also note the *Julian calendar* became valid not earlier than DCCVIII A.V.C. – before, months were organized differently. *Julius Caesar* was daggered in DCCIX A.V.C. so he did not live long with his calendar. (Note the Roman way of writing numbers was hampering calculations severely – one reason why Europe fell back in mathematics until Arabic numerals were taken over, including the zero invented in ancient India or Babylon.)

Note there are also other calendars widespread on this planet nowadays, e.g., in the Muslim or Buddhist world.

[225] → next chapter.

Solution:

2022.0326 [.d]	2022-03-26 0:00:00	
1931.1109 [.d]	1931-11-09 0:00:00	
[−]	33 010	days.
24 [×]	792 240	hours.

Using [CLK] **J→D** and **D→J**, you can deal with *Julian day numbers* useful in astronomy (→ *IOI*). And there is **SETDAT**, serving obvious purposes on your *WP43* (note it may take up to 1 minute until you see the date string updated in the *status bar*). But that's it – these are all the legal operations on *dates*. (If you want to convert a *date* to a plain *real*, use [.d], → 138.)

GAP, **ALL**, **ENG**, **FIX**, or **SCI** have no effect on *dates*.

Extended Dates

You may perform more precise calculations with *dates* if and when you add a time-of-day information:

Example (assuming *SDC*):

Enter **2023.1219142634567** [.d] and your *WP43* will interpret this input as

2023-12-19 14:26:34

Note the *seconds* are truncated. Now, if you want to know the date and time, e.g., 300 000 *seconds* later (for whatever reason) then key in

300000 [U→] **Time: s → day** returning 3.472 222… days.

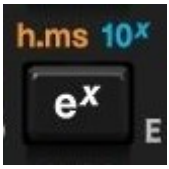
Press [+] and you will get

2023-12-23 01:46:34

Subtracting two such *dates* results in a difference in decimal *days*. Multiply its fractional part times 24 and press [h.ms] to get this part converted into *hours*, *minutes*, and *seconds* formatted `hh:mm:ss.fff` (→ next chapter).

Times

There is also a dedicated *DT* for *time* calculations on your *WP43*: *Sexagesimal* *times* are entered most easily using the format `hhh.mmssfff` terminated by [h.ms] – with `hhh` standing for *hours*, `mm` for *minutes*, `ss` for *seconds*, and `fff` for decimal fractions of *seconds* (these fractions as well as the *hours* may take more or less than 3 digits).



Example (assuming *SDC*):
Enter 45 *hours*, 39 *minutes*, and 7.8642 *seconds*:

45.39078642 [h.ms]

This is displayed with *startup default* **TDISP 0**:	45:39:07.864 2
Choosing [CLK] [▲] **TDISP** [3] will return	45:39:07
… instead, while **TDISP** [2] will return	45:39

The latter 2 formats also allow for compact returns when calling **TIME**.

➡ There is no display rounding for *times*.

The **:** (or a trailing **s**, → bottom of next page) is the unambiguous indicator for a *time* displayed on your *WP43*. There may be leading zeros in the *minutes* and *seconds* sections and a settable number of digits after the 2nd **:**. You can choose 12- or 24-hour display for *time of day* output.

Example (continued):

Call **TIME** in the evening and you might get	21:47
[CF] **SYS.FL** **TDM24** will return[226]	9:47p.m.

Add and subtract *sexagesimal* *times* simply by [+] and [−]. Multiply or divide such *times* by any *integer*, rational, or *real* – the result will stay a *time*. If you add any *integer*, rational or *real* to a *time*, that number will be taken as decimal hours and automatically converted to a *time* before adding. This applies to subtractions by analogy. If you divide by a *time*

[226] When a *time of day* is returned by a function, it will be displayed according to your choice – it is stored, however, as standard 24-hour *time* for further calculations. Hence, we recommend keeping TDM24 set in *time* calculations, else interpretation is needed.

interval, it will be converted to decimal hours before. Compare the matrices on pp. *134ff*.

Example (assuming *SDC*):
To meet your date at 5:25 p.m. at Stanford, you need 15' from your office to get your car out of the parking garage, 1.5 *hours* for the ride, and 12' for walking from the parking lot to the lecture hall. Being careful, you count in another quarter of an hour for a possible traffic jam on the expressway. When do you have to leave your office? If Stanford is 57 *miles* away, what time do you need for a *mile* on average?

Solution:

CLK ▲ TDISP 2		
.15 h.ms 1.5 +	returns	1:45
.12 h.ms +		1:57
.1.4 +		2:12
5.25 h.ms x⇄y −		3:13.

You shall leave at 3:13 p.m. the latest.

This result looks like a 12h-time here even in *SDC* – your *WP43* can't know any better based on the input given.

And TDISP 4 1.5 ENTER↑ 57 / h.ms returns 0:01:34.7 per *mile*.

If a result of your *time* calculations will become less than the display limit you set by TDISP (so no non-zero digit of the resulting *time* can be shown within the limit set) the result will be displayed in a fixed format like SCI 4 or ENG 4, depending on ALLENG. With ALLENG set, the following three *times* will be displayed like this, for **example**:

Time:	5 min 8.432 1 s	6.140 32 s	0.093 421 5 s
TDISP 1 or 2	0:05	6.140 3s	93.421×10⁻³s
TDISP 3	0:05:08	0:00:06	
TDISP 4	0:05:08.4	0:00:06.1	
TDISP 5	0:05:08.43	0:00:06.14	0:00:00.09
TDISP 6	0:05:08.432	0:00:06.140	0:00:00.093
TDISP 0	0:05:08.432 1	0:00:06.140 32	0:00:00.093 421 5

To convert such a closed *sexagesimal time* to decimal hours, e.g., for further calculations, use [.d]. Press [h.ms] to reconvert closed decimal hours to a *sexagesimal time*. For converting a bigger number of *seconds* into HMS format do as follows, for **example**:

123456789 [ENTER↑] **3600** [/] [h.ms] returning 34 293:33:09

Feel free to add this to a *date* like

2022-11-11 [.d] [+] returning 2026-10-09 21:33:09

There is only 1 more dedicated *time* command – **SETTIM** – on the calculator: enter a 24h time here and note it may take up to 1 *minute* until you see the time string updated in the *status bar*.[227]

The commands **GAP**, **ALL**, **ENG**, **FIX**, or **SCI** have no effect on *times*. Look up *OMS 6* for the stopwatch a.k.a. TIMER.

If you want to convert a *time* to a plain *real*, use [.d] (→ *138*).

[227] The real time clock of your *WP43* may deviate by up to 1 *minute* per month or 2 *seconds* per day from true time (i.e. ± 25 ppm approximately, caused by parts tolerances). This doesn't affect neither real-world time calculations nor the TIMER application described in *OMS 6*.

Note you live with these small inaccuracies wearing a quartz watch as well. Mechanical watches are less accurate. Radio controlled time pieces (incl. smartphones) need a suiting external reference signal to synchronize, once a day at least.

Alpha Input Mode: Introduction and Virtual Keyboard

This mode is designed for text entry, e.g., for typing messages, prompts, comments, and answers (for applications in programs → *OMS 4*). It is entered via [α] typically. Within *AIM*, …

1. primary function of most keys is appending or inserting the letter printed in grey bottom right of the respective key to ***x***, → *virtual keyboard* overleaf;
2. the *menu* Myα pops up immediately, containing your favorite special characters;[228]
3. [▮] leads to homophonic Greek letters (→ overleaf).[229]

Upper and lower case are set by [▲] and [▼], respectively, also for the letters in Myα and CATALOG'CHARS'αINTL (→ *140* and *210f*).

Wherever a *default primary function* is not primary anymore in *AIM* but continues being meaningful, it is reached by [▮]. For instance, [▮] is required for appending a digit to ***x***. And [▮] is also a shortcut to some special symbols, like [▮] [+/−] calling **±** or [▮] [XEQ] appending **qu**.

---

[228] For people writing German, Myα might look like pictured overleaf, for instance. Feel free to put other letters in, → *OMS 7* for how to populate Myα.

[229] This works wherever applicable, with 'homophonic' following classical Greek pronunciation. We assigned **Gamma** also to [C] following the alphabet (sic!), and **Chi** to [H] since this letter comes next in pronunciation.

Three Greek letters lack single-letter equivalents in English: **Psi** is accessed via [▮] [W] (since **ψ** just looks like **w** in a way), **Theta** via [▮] [U] (following **T**), and **Eta** via [▮] [J]; these three letters are printed blue on the physical keyboard as reminders.

Omicron is not needed at all since looking exactly like the Latin letter **O** in either case.

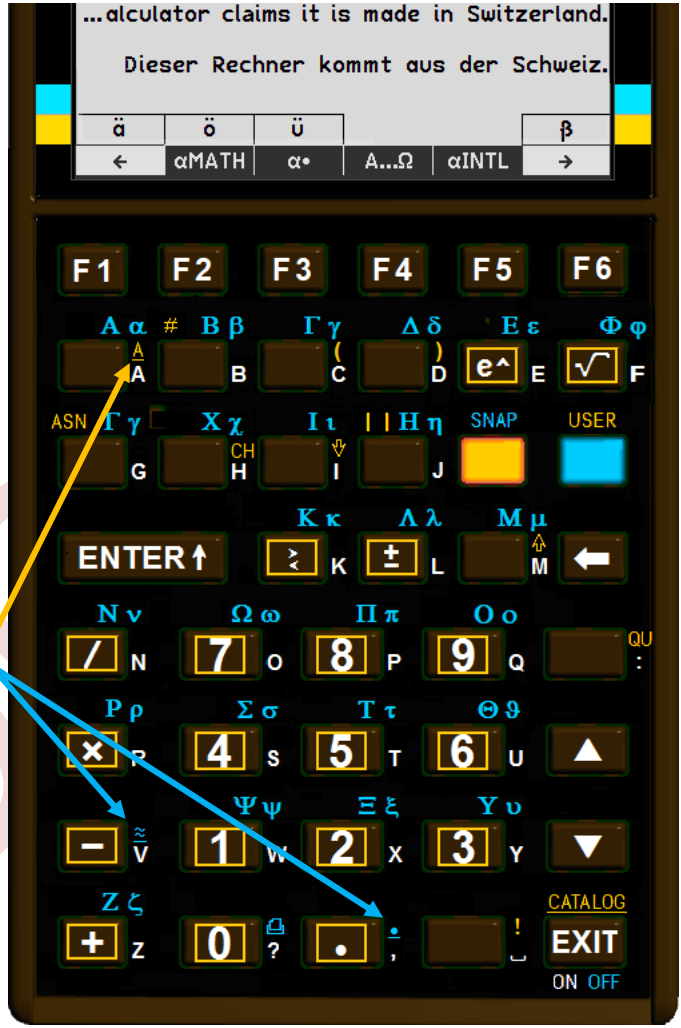
There is an '*alpha helper*' on the calculator backside for your orientation.

Being at it, kudos to *Thales*, *Pythagoras*, *Heraclitus*, *Leucippus*, *Democritus*, *Aristotle*, *Archimedes*, *Euclid* (i.e. ὁ Θαλῆς ὁ Μιλήσιος, ὁ Πυθαγόρας ὁ Σάμιος, ὁ Ἡράκλειτος ὁ Ἐφέσιος, ὁ Λεύκιππος, ὁ Δημόκριτος, ὁ Ἀριστοτέλης, ὁ Ἀρχιμήδης ὁ Συρακούσιος και ὁ Εὐκλείδης), and their colleagues for laying the foundations of logics, mathematics, and physics (τῆς λογικὴς και μαθηματικής τέχνης και τῆς φυσικής ἐπιστήμης) as we know them today – starting 2600 years ago. Note that the first two disciplines were called *practical arts* and just the latter *theoretical natural science*.

And kudos to those unnamed Babylonian mathematicians who laid the foundations for the Greek, actually recording what we now call "*Pythagoras' theorem*" as early as 1200 years (!) before him, for instance. We all stand on the shoulders of giants.

2 additional *prefixes* operate in *AIM*: [f] + [R↓] makes the next direct keyboard input a subscript if provided, while [f] + [E] makes it a superscript. See the golden arrows printed to the right of these keys, above the letters I and M.

Virtual keyboard in AIM.



➡ You can move the edit cursor at an arbitrary position in the string ***x*** and correct or improve ***x*** or add something to it as long as ***x*** is open for editing and shorter than 197 characters. 3 alpha *menus* become accessible for accented letters, punctuation marks, mathematical and further symbols (find the abbreviated labels for αINTL (A), α• (•), and αMATH (≈) on the keyboard). Look up their contents in *ReMS 2*.

Outside of *AIM*, you can call [α.FN] FBR to browse the entire character sets provided.

Alpha Input Mode: Entering Simple Text and More

Your *WP43* features a large font mainly for numeric output and a small font for *text strings*. See here all characters of the small font directly accessible through the *virtual keyboard* shown on previous page:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e *e* f g h i j k l m n o p q r s t u v w x y z

Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω

α β γ δ ε ζ η ϑ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω

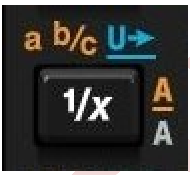
0 1 2 3 4 5 6 7 8 9 + - × or · / . , : ? ! ≷ ± ( ) | √‾ # 🖶 ,

the subscripts $_{A B C D E F G H I J K L M N O P Q R S T U V W X Y Z}$

$_{a b c d e f g h i j k l m n o p q r s t u v w x y z}$ $_{α δ μ}$ $_{0 1 2 3 4 5 6 7 8 9 + -}$

and the corresponding superscripts ^{A}…^{Z}, ^{a}…^{z}, and ^{0}…$^{9 + -}$.

The 26 plain Latin letters can be found in CATALOG'CHARS'αINTL, too, (together with 73 more, supporting international communication). Short-cut to αINTL is [f] [A] in *AIM*. This *catalog* will stay open until closed explicitly by [EXIT].



The 24 plain Greek letters are found as well in CATALOG'CHARS'A...Ω (plus 11 accented letters).

→ *ReMS 2* for both these catalogs.

So you may, for **example**, easily store and display an actual modern Greek message like

Οι μελλοθάνατοι σε χάιρετουν.[230]

Actually, we could have written the major part of this *OM* just using said font. It covers at least 47 languages from Afrikaans to Zhōngwén (→ *ReM*), providing the means that your display messages or prompts can be read and understood most easily by some 50% of all mankind.

---

[230] … corresponding to Latin MORITVRI TE SALVTANT. I created this example in 2015 looking at a banner shown in Athens during the Greek economic crisis.

Example:

You can even spell Dèng Xiǎo-píng de famous and evidently successful slogan in Pīnyīn straight ahead correctly: [231]

Bùguǎn bái māo, hēi māo, dàizhù lǎoshǔ jiù shì hǎo māo!

Taking advantage of this character set, it is also absolutely easy spelling, e.g., French, Spanish, or German prompts correctly « en français », "en español", or „auf Deutsch", as well as message strings in many more languages using extended Latin alphabets.

Your WP43 supports you in climbing the very first step of politeness and due respect by allowing you to adapt the software you write to the language your customers speak - instead of superciliously hacking in everything in English, using merely the very meager plain Latin letter set.

2 more *menus* (αMATH and α•, called by [f] + [−] and [f] + [.] in *AIM*) contain further symbols and punctuation marks (→ *209* and *ReMS 2*):

¡ ¿ ; " ' § $ £ ¥ € % & * < ≤ = := ≡ ≙ ≈ ≠ ≥ > @
[ ] { } \ ^ _ | ~ ° ¢ « » ÷ ← ↑ → ↓ ↕ ∅ ∞ ⊾ ⊥ ∡ ‖
¬ ∧ ∨ ∫ ⊙ ◷ ℤ ℚ ℝ ℂ $^{-1}$ ^{T} * E ∞ ⊙ ⊕ etc.

Pressing [USER] in *AIM* toggles USERα, allowing you to assign your favorite symbols to particular locations on the keyboard or in *menus* (→ *322ff* for how to do this). Such assignments will then become accessible when USERα is set (indicated by U and A or α being both lit in the status bar).

Open alphanumeric input can be deleted character by character using [←]. If there is no character left anymore, input will be canceled (→ *26*).

AIM is typically terminated by [ENTER↑] (duplicating string x in Y) or [EXIT] (just closing string x) pressed when only Myα is on screen. Then you will return to the menu displayed before entering *AIM*.

[231] Alas, he deviated from his own pragmatic motto later. And his successors follow that deviation so far, for whatever reasons or fears.

Example (continued):

Pressing [EXIT] with *Dèng Xiǎo-píng's* slogan entered will display it in **X** as shown above.

Pressing [ENTER↑] instead will display the text twice - entirely in **X** and truncated to 1 line in **Y**, showing only its first 7 words trailed by an ellipsis here:

```
                    Bùguǎn bái māo, hēi māo, dàizhù lǎoshǔ ...
Bùguǎn bái māo, hēi māo, dàizhù lǎoshǔ jiù
                                        shì hǎo māo!
```

Text strings exceeding 2 display lines will unveil their full contents after [SHOW] only.

Combining Text Strings and Numeric Data

Due to the *data type* concept of your *WP43* (a.k.a. object orientation), adding numeric data to a *text string* is as simple as pressing [+].

Example:

Assume the 2 lowest *stack registers* look like this:

```
                    The train will arrive at
                                       23:55
```

So here is a *text string* in **Y** and a *time* in **X**. Pressing [+] will merge ***x*** and ***y***, returning

```
                    The train will arrive at 23:55
```

Here, ***x*** was converted to a *text string*, taking its present display format into account, and was appended to ***y*** (→ the matrix on p. 134).

Let's enter a 2nd string now by pressing [α] and typing the necessary letters, starting with a blank:

```
                    The train will arrive at 23:55
sharp at Victoria station.
```

Press [EXIT] to close ***x*** and leave *AIM*:

```
The train will arrive at 23:55
sharp at Victoria station.
```

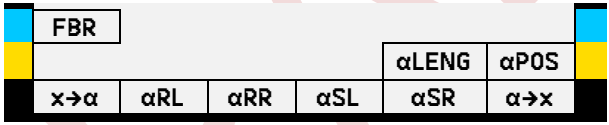
Now we have got 2 *strings* in **X** and **Y**, so pressing [+] will append string ***x*** to string ***y***, returning:

```
The train will arrive at 23:55 sharp at
Victoria station.
```

Any *string* may contain up to 196 characters in total (about 5 lines of text on display). Once numeric data (like the time above) became part of such a string, they are fixed and will not vary even if format will be changed.[232]

Working with Text Strings

Your *WP43* provides some more commands for dealing with strings. You find them all in α.FN:



x→α ***destination*** converts a character code ***x*** to the corresponding symbol and appends it to the destination; the character code is saved in **L**. For destination **X**, **x→α** returns the symbol only.

If **X** contains a *string* already, the entire *string* is appended to the destination (in this case **x→α** executes like **STO+**).

If **X** contains a *matrix* instead, **x→α** uses each *element* in the *matrix* as a character code. It starts with the 1st (top left) *element* and continues following usual reading habits until reaching the (bottom right) end of the *matrix*.

[232] Some *text strings* displayed may be hard to tell apart from numbers. *Text strings*, however, are always shown in small font, and there are no automatic gaps in them.

α→x ***source*** converts the leftmost symbol in the source string to the corresponding code, pushes this on the *stack*, and removes said symbol from said string. If the source is empty, **α→x** returns **0**.

αRL ***source*** rotates the source string by x symbols to the left.

αRR ***source*** rotates the source string by x symbols to the right.

αSL ***source*** shifts the source string by x symbols to the left, deleting the first x symbols from the string.

αSR ***source*** shifts the source string by x symbols to the right, deleting the last x symbols from the string.

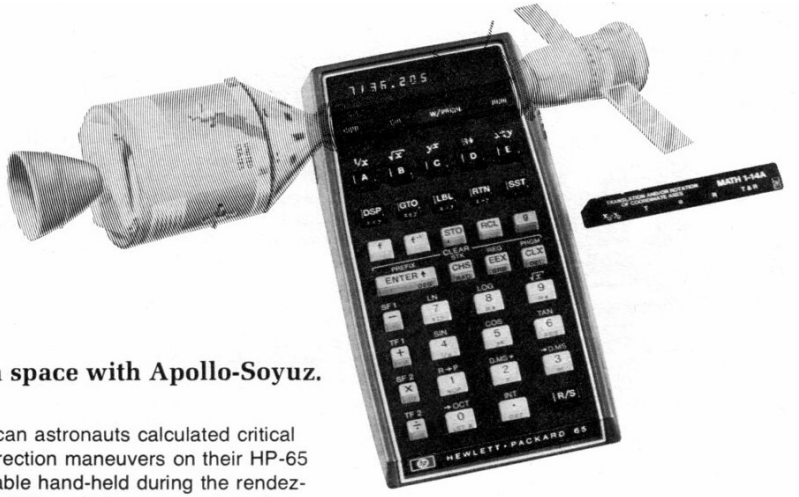
αLENG ***source*** pushes the length of the source string on the *stack*.

αPOS ***source*** searches the source string for the target symbol or string given in **X**. If a match is found, **αPOS** returns in **X** the position where the target was found starting in the source (counting the leftmost symbol as position 0); else it returns **−1**. The target is saved in **L**.

FBR shows all characters defined in both fonts provided.

You can also compare *text strings* using commands found in TEST, e.g., to create a sorted list. String ***A*** is called "smaller" than string ***B*** if ***A*** precedes ***B*** in sorting (→ *ReM* for the sorting order).

Nevertheless, don't forget that your *WP43* is designed to serve you mainly as a programmable calculator. Please turn overleaf to see what could be performed with such devices.



HP-65 in space with Apollo-Soyuz.

The American astronauts calculated critical course-correction maneuvers on their HP-65 programmable hand-held during the rendezvous of the U.S. and Russian spacecraft.

Twenty-four minutes before the rendezvous in space, when the Apollo and Soyuz were 12 miles apart, the American astronauts corrected their course to place their spacecraft into the same orbit as the Russian craft. Twelve minutes later, they made a second positioning maneuver just prior to braking, and coasted in to linkup.

In both cases, the Apollo astronauts made the course-correction calculations on their HP-65. Had the on-board computer failed, the spacecraft not being in communication with ground stations at the time, the HP-65 would have been the only way to make all the critical calculations. Using complex programs of nearly 1000 steps written by NASA scientists and pre-recorded on magnetic program cards, the astronauts made the calculations automatically, quickly, and with ten-digit accuracy.

The HP-65 also served as a backup for Apollo's on-board computer for two earlier maneuvers. Its answers provided a confidence-boosting double-check on the coelliptic (85 mile) maneuver, and the terminal phase initiation (22 mile) maneuver, which placed Apollo on an intercept trajectory with the Russian craft.

Periodically throughout their joint mission, the Apollo astronauts also used the HP-65 to calculate how to point a high-gain antenna precisely at an orbiting satellite to assure the best possible ground communications.

The first fully programmable hand-held calculator, the HP-65 automatically steps through lengthy or repetitive calculations. This advanced instrument relieves the user of the need to remember and execute the correct sequence of keystrokes, using programs recorded 100 steps at a time on tiny magnetic cards. Each program consists of any combination of the calculator's 51 key-stroke functions with branching, logical comparison, and conditional skip instructions.

The HP-65 is priced at $795*. See it, and the rest of the HP family of professional hand-helds at quality department stores or campus bookstores. Call 800-538-7922 (in California, 800-662-9862) for the name of the retailer nearest you.

HEWLETT hp PACKARD

Sales and service from 172 offices in 65 countries

An *HP* advertisement of 1975 (above), two of 1982 (overleaf), a picture of

'astronaut' *Sally Ride* floating with 3 *HP-41Cs* in a space shuttle in 1983, followed by an advertisement for the *HP-41C* on next page. Compare the capabilities of your *WP43* and imagine the opportunities you have.

THE HEWLETT-PACKARD 65 IN SPACE. Aboard the U.S. *Apollo* in its historic 1975 space linkup with the Russian *Soyuz*, an HP-65 calculated critical linkup maneuvers to place both spacecraft into the same orbit and to aid *Apollo* when it was 22 miles from *Soyus*. Commander Thomas P. Stafford and crew also relied on the HP-65 to precisely pinpoint *Apollo's* high-gain antenna at an orbiting satellite to assure communications with earth.

TWO HP-41C'S, CARRIED BY ASTRONAUT ROBERT CRIPPEN, ABOARD THE SPACE SHUTTLE COLUMBIA, contained critical programs. One to tell *Columbia* the next ground station to contact, when contact would be made and for how long. The other to calculate both the pre-entry center of gravity (balancing point) and the amount of fuel that must be burned to maintain the precise balancing point during descent.

Quando sono i risultati che contano

HEWLETT PACKARD

Gli astronauti della "Columbia" hanno utilizzato due HP-41C per effettuare alcuni calcoli, giudicati "critici" dalla NASA, al momento del rientro nell'atmosfera: posizione del centro di gravità della navetta spaziale e continua visualizzazione dei parametri e dei dati inviati dalle stazioni a terra in costante contatto con la "Columbia".

SECTION 4: PROGRAMMING AND AUTOMATIC PROBLEM SOLVING

Your *WP43* is a powerful *keystroke-programmable* calculator. If already this statement makes you smile with delight, this *Section* is for you. Else we will bring a smile on your face by mentioning the following facts:

Your *WP43* allows you to store a sequence of keystrokes like you would use them to solve a problem manually; this will save you time on repetitive calculations (→ *23ff*). Once you have written the keystroke sequence (a.k.a. *procedure* or *routine*) for solving a particular problem and recorded it in your *WP43*, you need no longer devote attention to the individual keystrokes that make up the procedure. Instead, you can let your *WP43* solve each similar problem for you. And because you can easily check the routine stored, you have more confidence in your final answer since you don't have to worry each time about whether or not you have pressed an incorrect key. Your *WP43* performs the drudgery, leaving your mind free for more creative work.

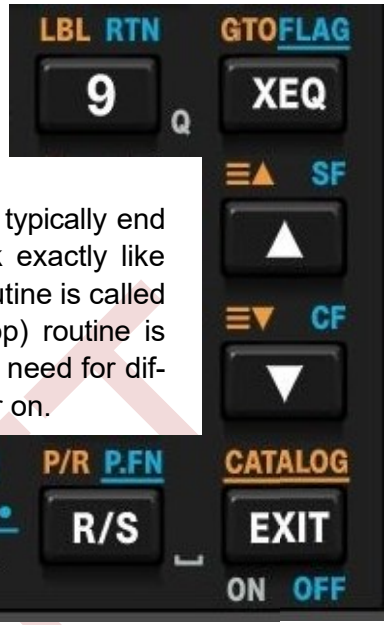
And even better: you may use *program memory* (*PM*) for storing many routines containing thousands of keystrokes (→ *59*). For telling your *WP43* where a particular routine begins and ends, each one is confined by 2 steps: It starts with **LBL** (for LaBeL) and typically ends with **RTN** (for ReTurN); → *25*. These 2 steps separate it from other routines you may add for other tasks. And **LBL** puts a *label* (a.k.a. *tag*) on your routine so you can go to and call it when you want it to be executed.

You may structure *PM* even more: Collect a set of routines and separate them by **END** from other routines or sets. What is found between two **END** steps is called a *program* – they are the basic building blocks within *PM*. Think of the beginning and end of the entire used *PM* section containing implicit **END** steps;[233] there will be at least 1 program in *PM* at any time.

Within routines, you may store any sequence of keystrokes representing commands, operations, labels, or constants. Most executable operations are programmable. The commands in your routine may also access each

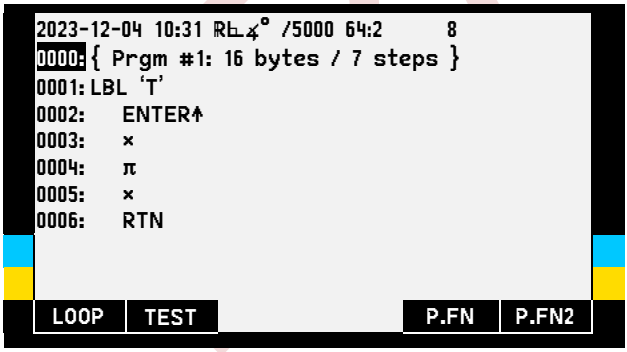
[233] You can't see that first END but the last one is visible – pictured, e.g., on p. 220.

and every *register*, *variable*, or *flag* provided – there are almost no limits. You are the sole and undisputed master of *WP43* memory.[234]



Each such routine itself may contain *subroutines* – also these start with **LBL** and typically end with **RTN**. Actually, *sub*routines may look exactly like routines: the only difference is that a *sub*routine is called from another routine, while a main (or top) routine is called from the keyboard. Thus, there is no need for differentiating these 2 kinds of routines further on.

Enough of theory – press [RTN] and switch to *program entry mode* (*PEM*) by [P/R]. The display of your *WP43* will change to something like this:



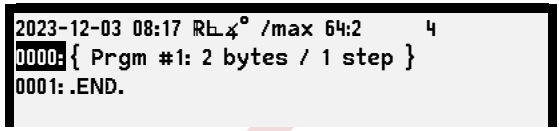
This screen shows the routine you entered on p. 25 (the *status bar* on top will differ according to your actual time and settings). In the *LCD* section used for numeric output so far, the first 6 steps in *PM* are shown. Label steps are 'outdented' for visual

[234] This freedom has a price: Take care that your routines don't interfere with each other in their quest for storage space. Recording the *global labels* and *register*s, *variables*, and *flags* a particular routine uses, and documenting their purposes and contents for later reference is good practice.

An alternative – using *local registers* and *flags* instead of *global* ones – will be covered in a chapter further below.

structuring. The position of the program pointer (*PP*) – the *current step* – is highlighted by inversion. The routine the *PP* is in is called the *current routine*; the corresponding program is the *current program*. MyPFN is shown in the *menu section*.

Turning to *PEM* the very first time after unpacking your *WP43* (or after CLPALL or RESET), the display may look like this, instead.



Writing and Recording a New Routine

Whenever you want to create a new routine, ensure you are in *PEM*. Then enter GTO . . – this will bring you to the very end of the used section of *PM*, so you can start keying in your new routine right there without interfering with anything coded previously.

Start the routine with LBL giving it a descriptive *label*. Then press the keys as you would do in manual problem solving (→ *23f*). Each new step will be inserted right after the *current step*. You find the following keys and labels related to programming and programs on your keyboard as shown on previous page; clockwise, these are:

- LBL for LaBeLing a routine or just the program section following,
- RTN for ReTurNing to the caller of the *current routine*,
- GTO for unconditionally Going TO a specified label (i.e. moving the *PP* onto the respective LBL step),
- XEQ for eXECUting or calling a specific (sub-) routine,
- ▲ and ▼ (or ≡▲ and ≡▼ if there is an open *multi-view menu*) for browsing program steps,
- EXIT for exiting *PEM* if just MyPFN (meant to be your personal *menu* of programming commands) is open, returning to *RUM*,
- R/S for Running or Stopping the *current routine*, and
- P/R for toggling *PEM* and *RUM*.

These are continued to the left by the *menus* for Programming FuNctions (P.FN, containing END for closing the *current program*), system INFOrmation, I/O, LOOPs, and TESTs.

Example (from the *HP-15C OH*):

Mother's Kitchen, a canning company, wants to package a ready-to-eat spaghetti mix containing three different cylindrical cans: one of spaghetti sauce, one of grated cheese, and one of meatballs. *Mother's* needs to calculate the base areas, total surface areas, and volumes of the three different cans. It would also like to know, per package, the total base area, surface area, and volume.

Solution:

The program to calculate this information uses these formulas and data:

$$base\ area = \pi\, r^2$$
$$volume = base\ area \times height = \pi\, r^2\, h$$
$$surface\ area = 2 \times base\ area + side\ area = 2\,\pi\, r^2 + 2\,\pi\, r\, h$$

r	*h*	Base Area	Volume	Surface Area
2.5	8.0	?	?	?
4.0	10.5	?	?	?
4.5	4.0	?	?	?
	TOTALS	?	?	?

Method:

1. Enter an ***r*** value into the calculator and save it for other calculations. Calculate the base area ($\boldsymbol{\pi\, r^2}$), store it for later use, and add the base area to a *variable* which will hold the sum of all base areas.
2. Enter ***h*** and calculate the volume ($\boldsymbol{\pi\, r^2\, h}$). Add it to a *variable* to hold the sum of all volumes.
3. Recall ***r***. Divide the volume by ***r*** and multiply by **2** to yield the side area.
4. Recall the base area, multiply by **2**, and add to the side area to yield the surface area. Sum the surface areas in a *variable*.

Don't enter the actual data while writing the program – just provide for their entry. These values will vary and so they will be entered before and / or during

each program run. – Now, turn to *PEM* using P/R and go to free *PM* by GTO . .
Key in the following program to solve above problem:

LBL K ENTER↑	LBL 'K'	For kitchen
INPUT α R ENTER↑	INPUT 'r'	Ask for radius input
EXP x^2	x^2	
π	π	
×	×	Compute base area
STO α▲ BASE ENTER↑	STO 'BASE'	
STO+ α▲ ■S B ENTER↑	STO+ 'ΣB'	Sum of bases
VIEW VAR BASE	VIEW 'BASE'	Show base
P.FN PAUSE 10	PAUSE 10	for 1 *second*
INPUT α H ENTER↑	INPUT 'h'	Ask for height input
RCL VAR BASE	RCL 'BASE'	
×	×	Compute volume
STO α▲ VOLUME ENTER↑	STO 'VOLUME'	
STO+ α▲ ■S V ENTER↑	STO+ 'ΣV'	Sum of volumes
VIEW VAR VOLUME	VIEW 'VOLUME'	Show volume
PAUSE 10	PAUSE 10	for 1 *second*
RCL VAR r	RCL 'r'	
/	/	
2	2	
×	×	Compute side area
RCL VAR BASE	RCL 'BASE'	
2	2	
×	×	
+	+	Compute surface
STO+ α▲ ■S S ENTER↑	STO+ 'ΣS'	Sum of surfaces
RTN	RTN	End of routine
P/R		Leave *PEM*

Now, let's run the program (…):

GTO K makes **K** the *current program*.

CLR CLCVAR clears all *variables* of the *current program*. By doing so, it creates the *variables* not defined yet, avoiding errors caused by undefined *variables* later.

DISP FIX 0 1

XEQ K	r? 0	1st can: ask for **r** [235]
2.5	2.5	
R/S	BASE = 19.6	show base for 1 s
	h? 0	ask for height input
8 R/S	VOLUME = 157.1	show volume for 1 s
	164.9	surface
XEQ K	r? 2.5	2nd can: radius?
4	4	
R/S	BASE = 50.3	show base for 1 s
	h? 8	ask for height input
10.5 R/S	VOLUME = 527.8	show volume for 1 s
	364.4	surface
XEQ K	r? 4	3rd can: radius?
4.5	4.5	
R/S	BASE = 63.6	show base for 1 s
	h? 10.5	ask for height input
4 R/S	VOLUME = 254.5	show volume for 1 s
	240.3	surface
RCL VAR ΣB	133.5	sum of bases
RCL VAR ΣV	939.3	sum of volumes
RCL VAR ΣS	769.7	sum of surfaces

[235] *TN*: No units are mentioned in this problem. Assume they talked about *inches*, the output is in *square* and *cubic inches*. Do you need conversion into *fluid ounces*? But which ones? (→ *OMS 6*)

If you want to run this routine again for another set of cans, remember to clear the *variables* **ΣB**, **ΣV**, and **ΣS** before. `CLCVAR` will do this for you most easily.

The preceding program illustrates the basic techniques of programming. It also shows how data can be manipulated in *PEM* and *RUM* by entering, storing, and recalling data (input and output) using [ENTER↑], [STO], [RCL], store arithmetic, and programmed I/O.

See next chapters (and the *IOI*) for comprehensive information about all the commands used in this case.[236]

Labels

Each routine or subroutine shall begin with a `LBL` step. Though you may tag *labels* not only to the 1st but to any step in your routine. Structuring *PM* and jumping within is eased by these *labels*. Your *WP43* allows for specifying a wide variety of alphanumeric *labels* as elaborated overleaf.

Whenever a step like, e.g., `GTO` ***labl*** is encountered (with ***labl*** representing an arbitrary *label*) with either a program running or a single program step executed in *RUM*, your *WP43* will **search this *label*** using 1 of the 2 following methods:

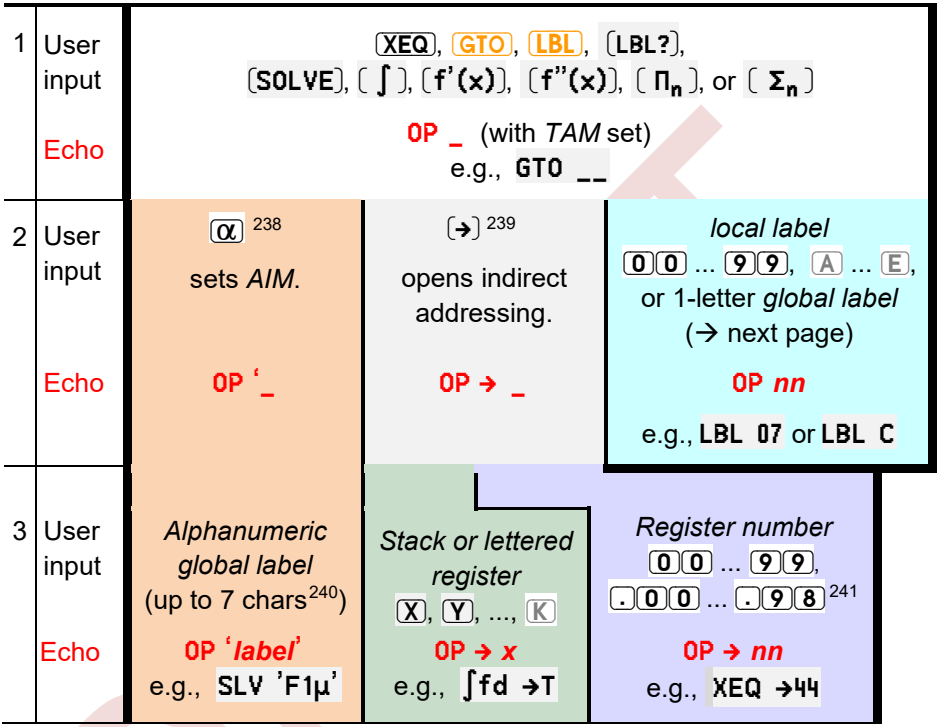
1. If ***labl*** is just numeric (**00 ... 99**) or **A ... E**, it will be searched forward from the current *PP* position. When an `END` step is reached without finding ***labl*** so far, the quest will continue right after previous `END` (so the search will stay in the *current program*). This is the procedure for ***local labels*** – they may be hence reused in other programs.
2. If, however, ***labl*** is alphanumeric (up to 7 characters of arbitrary case, starting with a letter and automatically enclosed in ‘ like ‘`M`’ or ‘`Ab1`’), searching will start at the end of *PM*, go upwards, and cover the entire *PM* independent of the current *PP* position. This is the procedure for ***global labels***.[237]

[236] *TN: Mother's Kitchen* might tend to use a spreadsheet application on its tablet to compute such outputs nowadays. Your *WP43* boots faster and is smaller.

[237] These procedures for *local* and *global labels* were established with the *HP-41C*. Though the range of *local labels* varies from calculator model to model.

Thus, *global labels* can be accessed from anywhere in *PM* while *local labels* can only be reached from within their own program.

Addressing *labels*, on the other hand, follows the rules given below:



`SLV 'F1μ'` solves the function given in the routine **F1μ** (→ 273ff).

`∫fd →T` integrates the function given by `PGMINT` over the *variable* whose label is on *stack register* **T** (→ 280ff).

`XEQ →44` calls the routine whose label is found in **R44**.

238 You can skip pressing [f] here – see overleaf. See also an alternative there.

239 Indirect addressing works with all these operations except LBL.

240 Said *label* must contain at least 1 letter. *Labels* are case sensitive. The 7th character will terminate entry and close *AIM* – shorter *labels* need a closing ENTER↑.

241 ... if the respective *local register* is allocated.

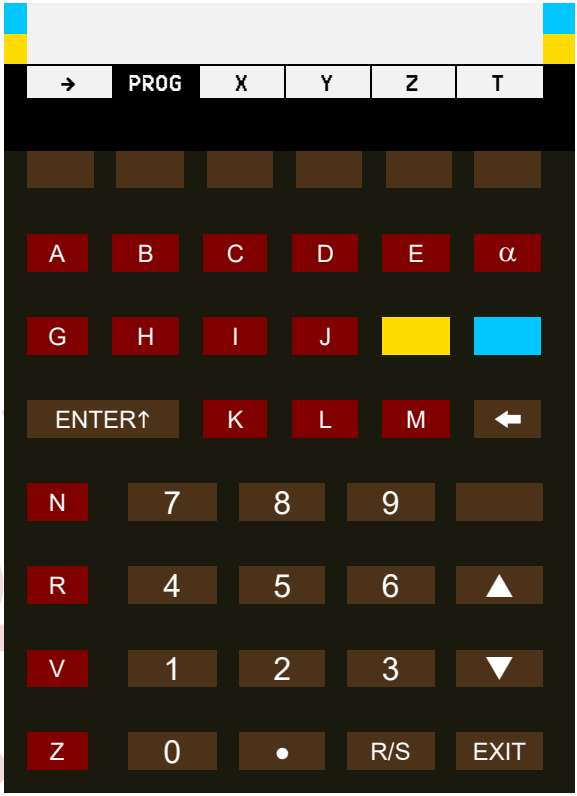
➡ `XEQ.` is not allowed neither in *RUM* nor *PEM* – it turns to `GTO.` automatically instead. Some lettered *registers* may be dedicated to special applications; → *Addressing and Manipulating Objects in RAM*.

Furthermore, **GTO** provides 4 special cases: see **GTO↑**, **GTO↓**, **GTO.**, and **GTO..** in next chapter.

And remember *TAM* is set also during *label* addressing, so the *virtual keyboard* of your *WP43* will look like this:

You can access the *local labels* **A** - **E** directly as well as the *global labels* **G** - **N**, **R**, **T**, **V**, **X**, **Y**, and **Z**. This *virtual keyboard* allows for calling 14 programs and 5 different subroutines per program by just 2 keystrokes each.

Note also the changed assignment of [F2] compared to p. 61. Thus, instead of typing a longer *global label* via [α] trailed by the necessary characters in *AIM*, just press **PROG** and pick the label from this *submenu* comprising all *global labels* defined at execution time – this may well be easier and faster.



Editing a Routine

Whenever you want to edit (modify, correct, expand, or shorten) an existing routine, start in *RUM* entering [GTO] ***labl***. This will move the *PP* onto the corresponding step **LBL** ***labl*** (as explained on p. 224). Then turn to *PEM* using [P/R] and start browsing from this LBL step on.

Example:

Let's browse the program steps in our first routine. Press

GTO T

P/R

▼ ▼ ▼

```
0001: LBL 'T'
0002:    ENTER↑
0003:    ×
0004:    π
0005:    ×
0006:    RTN
0007: END
```

Unless you are next to the begin or end of a program, the *PP* will always be placed in the middle of the display with 3 steps shown above and 3 steps below of it.

Navigating in *PM*, you may execute various actions. If you want to ...

- continue browsing forward, press ▼ (or ≡▽ if a *multi-view menu* is displayed); when reaching the END, the *PP* will return to the first step of the current program and start browsing there again.
- browse backwards, press ▲ (or ≡△ if a *multi-view menu* is displayed); when reaching program top, browsing will stop.
- delete a program step, go to said step (make it the *current step* by moving the *PP* onto it), then press ⬅. It will vanish (this can't be undone!) and the *PP* will move on the step before.
- insert something, go to the program step before, and then press the keys to be inserted after it.

Example (continued):

Replace ENTER↑ × in our routine above by x^2 doing the same.

Solution:

Press ▲ –	now, the *PP* is on step	0003: ×
⬅	This step vanished; the *PP* is on step	0002: ENTER↑ now.
⬅	This step vanished; the *PP* is on step	0001: LBL 'T' now.
EXP x^2	This command is now stored in step while π moved to step 3	0002: x^2 0003: π

- jump to a particular *global label* (without inserting a GTO in the *current routine*), press GTO . α *label* ENTER↑ or GTO . PROG (*label*).

For a *local label*, press GTO . ***nn*** instead. 1-letter *labels* can be accessed, e.g., by GTO . D (→ 226).

- jump to the top of current program, press GTO ▲.
  If the *PP* is there already, GTO ▲ jumps to the top of previous program.
- jump to the top (i.e. the first step) of next program, press GTO ▼.
- start writing a new routine, press GTO . ., then LBL ... (as experienced in *Mother's Kitchen* on pp. 221f).

That's almost all. When you are done, press P/R or EXIT to leave *PEM*, returning to *RUM*.

Please note your *WP43* can't know in advance if all the necessary pre-conditions for a certain program step will be fulfilled at the time of its execution. Thus, respective error messages can appear no earlier. Debugging may be required.

Running a Routine from the Keyboard (also for Debugging)

Whenever you want to execute an existing routine, ensure you are in *run mode.* Then there are 3 alternatives:

1. **Automatic execution of the *current routine*:** Press RTN to return the *PP* to the first step of the *current routine*. Then press R/S. This will run this routine, i.e. start executing the following steps automatically until an error happens, a **STOP**, a final **RTN**, or an **END** will be encountered[242] where it will halt and display x.
2. **Automatic execution of a selected routine:** Press XEQ and specify the *global label* of the program you want to execute (or press PROG and pick it from this *menu*).[243] This will move the *PP* to the corresponding **LBL** (→ 224) and start executing the following steps auto-

[242] ... or you interrupt it by pressing R/S or EXIT – then it will stop after the *current step* is executed completely. For resuming program execution, press R/S again.

[243] This is the standard way to run routines. Furthermore, you can define shortcuts to your favorite routines by customizing your *WP43* as described in *OMS 7*.

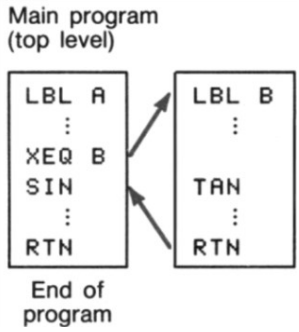
matically until an error happens, a **STOP**, a final **RTN**, or an **END** will be encountered[242] where it will halt and display x (→ 23f).

3. **Stepwise execution of a selected routine:** Press [GTO] and specify the *global label* of the program you want to execute (or press **PROG** and pick it from this *menu*). This will move the *PP* to the corresponding **LBL** (→ 224) and wait. Each following program step will then be executed 1 at a time as you press ▼ (or [≡▽] if a *multi-view menu* is displayed): pressing the key will show the *current step* at left end of the top numeric row, releasing will execute it.[244]

   Where your routine asks for input (→ 238ff), close this input by ▼ (or [≡▽]). Reaching the end of the *current routine*, ▼ (or [≡▽]) will return to its first step. Following this procedure, you will go through the routine as in normal execution but slower – and you may call **RBR** or **STATUS** or perform additional checks after each program step.[245] This procedure is especially useful for *debugging*.

If an error occurs while a routine is running, it stops immediately at the step generating said error and throws the corresponding error message (→ *ReMA C* for a compilation of all error messages provided). To view the program step causing said error, press [P/R].

Subroutines: Running a Routine From Another Routine



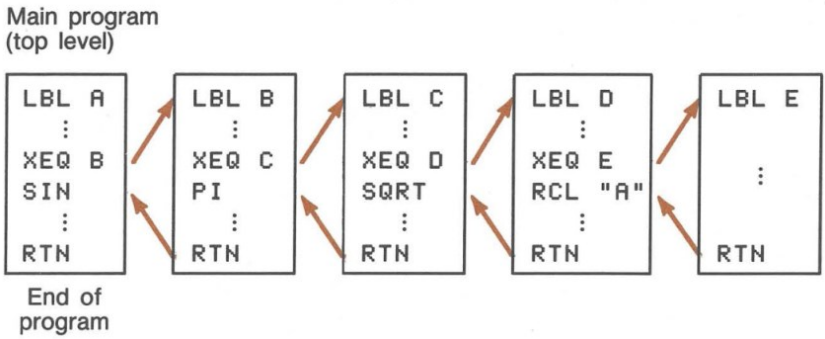
[XEQ] is programmable as well. Whenever a running routine encounters an **XEQ**, it will search for the associated label as described on p. 224, move the *PP* to it, and continue program execution with the step after this **LBL** until it encounters a **RTN**; this will return the *PP* to the step right

[244] Pressing ▲ or [≡△], on the other hand, moves the *PP* backwards in the *current routine* without executing anything.

[245] Watch that your additional checks don't alter the status of your *WP43* in a way deviating from its status in automatic execution; else you shall compensate. Also take care when browsing backwards.

after said XEQ, where execution will continue. Compare the picture above where routine **A** calls routine **B**.

You can also nest subroutines – your *WP43* can remember as many pending subroutine return addresses as *RAM* allows. Every routine may allocate its own *local registers*, up to 99 each (→ 255).



> Be aware that all return addresses will be lost for the *current program* if you alter the *PP* (e.g., by RTN) while this program is stopped – pressing R/S, ≡∇, or ▼, however, will not cause such a loss.

If you need any of your subroutines elsewhere in another routine then you can call it again and again at no expense of memory. If you want to call a particular subroutine from another program than the one it is defined in, the label of the subroutine you call must be global.

Automatic Binary Testing and Conditional Branching

So far, we were talking about linear programs, running straight from their beginning (LBL) to end (final RTN or END). Your *WP43* can do more for you: like other programmable calculators before, it features a set of binary tests checking various calculator states. Most of them are collected in TEST. There are also 2 tests in BIT, and 8 *flag* tests are stored in FLAG. *Names* of binary test commands contain a '**?**', mostly as their last character.

Generally, binary tests will return `true` or `false` as *temporary information* at left end of the **Z** row if called from the keyboard. If called in a running routine automatically instead, they will execute the next program step if the test is true at execution time, else skip that step. So the general rule reads "**do if true**". Think of the step immediately following the test containing a `GTO` and you see how conditional branching comes into play.

Example:

```
...
0020:    x≤ ? Y
0021:    GTO 'Join'
0022:    x⇄y
...
0032: LBL 'Join'
0033:    ln x
...
```

If this test is true (i.e. x is not greater than y)
then go to the label **Join** (at step 32);
else swap x and y
and continue working here.

Most binary tests operate on x. 6 tests check its *DT*:

- `REAL?` **tests if x is a *real* object** (*DT* 2 or 8)
  and executes the next program step if true, else skips it.
- `CPX?` tests if x is a *complex* object (*DT* 3 or 9) with an *imaginary* part $\neq$ **0** and …
- `MATR?` tests if x is a *matrix* (*DT* 8 or 9) ...
- `STRI?` tests if x is a *text string* (*DT* 7) ...

5 tests check its numeric content:

- `INT?` **tests if x is integer** (note `INT?` will return `true` if `FP?` returns `false`, i.e. also for x being of *DT* 2 with FP(x) = 0)
  and executes the next program step if true, else skips it.
- `EVEN?` tests if x is integer (→ `INT?`) and even ...
- `ODD?` tests if x is integer (→ `INT?`) and odd ...
- `FP?` tests if x has a nonzero fractional part ...
- `PRIME?` tests if |IP(x)| is a *prime number* ...
- `SPEC?` tests if x is special (i.e. ∞, −∞, or 'Not a Number') ...
- `NaN?` and `±∞?` subtilize `SPEC?`

7 tests compare x with **0**, **1**, or the content of a source specified (let's call it s, → 62 and 64 as well):

- **x<? s tests if x is less than s and executes the next program step if true, else skips it.**
- **x≈? s** tests if the rounded values of x and s are equal ...
- **x≤?**, **x=?**, **x≠?**, **x≥?**, and **x>?** work in analogy to **x<?**.

4 tests check its internal structure:

- **BC?** (or **BS?**) tests if X contains a *short integer*, then checks its bit specified and executes the next program step if said bit is clear (or set), else skips it.
- **LEAP?** tests if X contains a *date*, then extracts the year and tests for a leap year ...
- **M.SQR?** tests if X contains a *matrix*, then checks if it is square ...

8 *flag* tests operate on the *flag* specified:

- **FC?** tests this *flag* and executes the next program step if said *flag* is clear, else skips it.
- **FS?** works like **FC?** but executes the next program step if said *flag* is set.
- **FC?F** (**FS?F**) works like **FC?** (**FS?**) but flips this *flag* after testing (clears it if it was set or sets it if it was clear).
- **FC?C** (**FS?C**) works like **FC?** (**FS?**) but clears this *flag* after testing.
- **FC?S** (**FS?S**) works like **FC?** (**FS?**) but sets this *flag* after testing.

Finally, there are 4 special tests:

- **LBL?** tests for the existence of the label specified, anywhere in *PM*.
- **TOP?** will return true if the *PP* is in the top level routine (→ sketches on pp. 229f).
- **ENTRY?** checks the (internal) entry *flag*. This is set if:
  - a character is entered in *AIM* or

- a command is accepted for entry (be it by ENTER↑, a function key, or R/S with a partial command line).

ENTRY? is useful, e.g., after **PAUSE**.

- **KEY?** 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 (exception!); else it will be skipped and the code of said key will be stored in the address specified. Key codes reflect the rows and columns on the keyboard (→ *241* for an application).

11	12	13	14	15	16
1/x 21	EXP 22	TRI 23	ln 24	e^x 25	$\sqrt{x}$ 26
STO 31	RCL 32	R↓ 33	CC 34	35	36
ENTER↑ 41		x⇄y 42	+/- 43	E 44	⬅ 45
/ 51	7 52	8 53	9 54	XEQ 55	
x 61	4 62	5 63	6 64	▲ 65	
− 71	1 72	2 73	3 74	▼ 75	
+ 81	0 82	. 83	R/S 84	EXIT 85	

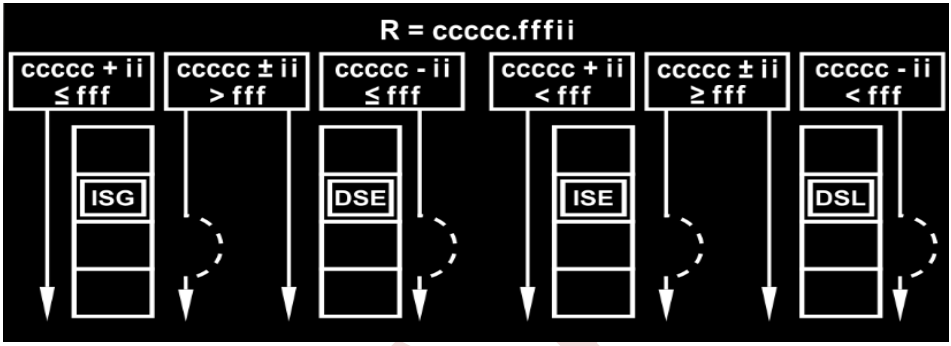
See the *IOI* for comprehensive information about all binary tests provided, also beyond the ones mentioned above.

As mentioned, routines end with **RTN** (typically) and programs with **END**. Executing a program, both **RTN** and **END** work in a very similar way and show only subtle differences: an **RTN** immediately after a binary test returning **false** will be skipped – an **END** will not.

Loops and Counters

The commands **DSE**, **DSL**, **DSZ**, **ISE**, **ISG**, and **ISZ** (all contained in LOOP) are for controlling loops in routines. Each of them Decrements or Increments a counter in a *RoV* as specified, tests the result, and executes or skips the following program step. See the picture below illustrating **ISG** (Increment and Skip if Greater), **DSE** (Decrement and Skip if

Equal), **ISE** (Increment and Skip if Equal),, and **DSL** (Decrement and Skip if Less). [246] The control *variable* contains a number formatted *ccccc.fffii*, i.e. the start value of a counter ***ccccc***, its 3-digit final value ***fff***, and a 2-digit increment/decrement ***ii***. If ***ii*** is entered as zero, it defaults to **1**. (The commands **ISZ** and **DSZ** simply skip if Zero – anything between **−1** and **+1** is assessed as **0** here.)



With a **GTO** placed in the skipped step pointing to a label upstream in the same routine you can create loops running until the specified condition is met, → examples below. E.g., if the loop counter should go from **5** to **1**, use **5** [STO] [K] **...** [DSE] [K]; on the other hand, if it should go from **1** to **5**, use **1.005** [STO] [K] **...** [ISG] [K].

Without such an exit condition you can deliberately create an infinite loop. Such a loop will run until you interrupt it manually by [EXIT] or [R/S], or until battery voltage falls below the limit (note your WP43 will not turn off automatically with a program running!).

Created inadvertently, infinite loops in a running program may be difficult to recognize – will stay lit in the *status bar* and your *WP43* will accept no keystrokes except [EXIT] or [R/S] – it may look like locked or caught in a hung state. Thus, implementing some outputs in regular intervals is recommended especially in programs not completely checked yet.

Let's demonstrate loops, use of *user flags*, tests, and also indirect addressing – outputs will be covered in next chapter:

[246] Such a picture is also printed on the back of your *WP43*.

Example 1:

The '*Cullen numbers*' are $C_n = n\,2^n + 1$. It is proven that no C_n is prime for $n \leq 1000$ except very few cases. One is C_1. What else do you find?

Solution:

Reset the *PP* to the start of free *PM* by [GTO] [.] [.]. Switch to *PEM* via [P/R] and key in:

[LBL] [α] [C] [N] [ENTER↑]	LBL 'CN'	
[RCL] [I]	RCL I	
[PART] IP	IP	
[FILL]	FILL	n
[EXP] 2^x	2^x	2^n
[x]	×	$n\,2^n$
[LOOP] INC X	INC X	$n\,2^n + 1 = C_n$
[TEST] PRIME?	PRIME?	
[GTO] [A]	GTO A	… if prime
[x≷y]	x≷y	else return n
[GTO] [B]	GTO B	and go to *local label* **B**.
[LBL] [A]	LBL A	
[x≷y]	x≷y	return n
[R/S]	STOP	… to show n. Resume with [R/S]
[LBL] [B]	LBL B	
[LOOP] ISG I	ISG I	if $n + 1 \leq f$ …
[GTO] PROG [Cn]	GTO 'Cn'	then repeat with $n + 1$
[P.FN] END	END	else end (with $n + 1 > f$).
[P/R]		

Now, load the loop counter: **1.999** [STO] [I] (**999** is the maximum final count possible) and start the program by pressing [XEQ] PROG [Cn]. It will display 1 as expected. Press [R/S]. What will be found next? And thereafter?

Example 2 (following an idea of *Gene Wright*):

Write a little routine to store random numbers in **R25** through **R39**.

Solution:

Enter GTO . . P/R and key in:

LBL X	LBL 'X'	
PROB ▲ RAN#	RAN#	Get a new random number and…
STO → 24	STO →24	store it where *r24* is pointing to.
LOOP ISG 24	ISG 24	Increment the pointer.
GTO X	GTO 'X'	If *r24* ≤ **39** then return to label **X**
RTN	RTN	else finish this routine.
P/R		

Initialize the loop counter via **25.039** STO 24 and start the program by pressing XEQ X. It will stop with the last random number in display. Check the target *registers* using RBR.

Example 2 (continued):

Now, write a routine to sort these 15 stored numbers so the smallest number moves to the *register* with the smallest address.

Solution:

We will use the so-called '*bubble sort*' algorithm. Re-initialize the loop counter via **25.039** STO 24 (*r24* was modified by program **X** above). Enter

GTO . . P/R		
LBL Y	LBL 'Y'	
P.FN LocR 02	LocR 02	Allocate 2 local *registers*.
REM α B ▼ E…	REM 'Begin of main loop'	
LBL A	LBL A	
RCL 24	RCL 24	Put the start pointer *r24* …
STO .00	STO .00	into local *register* **00**;
LOOP INC X	INC X	increment this pointer and …
STO .01	STO .01	store it in local *register* **01**.
CF .00	CF .00	Clear local *flag* **00**.
LBL C	LBL C	
RCL → .00	RCL →.00	Recall the contents of the *registers* …
RCL → .01	RCL →.01	where *r.00* and *r.01* are pointing to.

TEST x< ? Y	x< ? Y	Is $x < y$?
GTO B	GTO B	Then go to local label **B** below …
LBL D	LBL D	else…
LOOP ISG .00	ISG .00	increment ***r.00*** and…
ISG .01	ISG .01	if ***r.00*** ≤ **39** increment ***r.01*** and…
GTO C	GTO C	if (***r.00*** > **39** or ***r.01*** ≤ **39**) then return to local label **C** above;
FLAG FS? .00	FS? .00	else check local *flag* **00**:
GTO A	GTO A	if set, return to local label **A** above,
RTN	RTN	else stop executing this routine.
LBL B	LBL B	
SF .00	SF .00	Set local *flag* **00**.
STO → .00	STO →.00	Store the smaller value …
x≷y	x≷y	where ***r.00*** and the greater …
STO → .01	STO →.01	where ***r.01*** is pointing to.
x≷y	x≷y	Restore the *stack* and …
GTO D	GTO D	return to local label **D** above.
P.FN END	END	
P/R		

Start the program by pressing XEQ Y . Then check the target *registers* using RBR . You will find the smallest value in **R25**, a greater one in **R26**, etc., up to the greatest in **R39**.

➡ This program allocates 2 *local registers* for its exclusive use (**R.00** and **R.01**). Furthermore, it uses 1 *local flag* and 4 *local labels*.

The following sorting program is even shorter (kudos to *Jean-Marc Baillard*). Start it by **25.039** XEQ Z . Finding out how it works is left as an exercise for the reader:

```
LBL 'Z'
   sign
LBL A
   RCL L
   RCL L
   RCL →L
```

```
LBL B
  RCL →Y
  x> ? Y
  GTO C
  x≷y
  RCL L
  +
LBL C
  R↓
  ISG Y
  GTO B
  x≷ →L
  STO →Z
  ISG L
  GTO A
END
```

→ *HP-42S OM*, pp. 152 – 154, for more.

Programmed User Interaction and Dialogues

A number of commands are provided for controlling the interaction of programs with you. A program shall output a result to you at least, and it may also ask for your input. In the *IOI*, the behavior of those I/O commands is described if they are entered from the keyboard. Executed by a program, however, they will work differently.

When you start a program by XEQ or R/S, the symbol will be lit in the *status bar*. While in *RUM* each command executed may change the display immediately, only **AVIEW**, **INPUT**, **PAUSE**, **PROMPT**, **STATUS**, **STOP**, and **VIEW** will update the screen with a program running. **AVIEW** and **VIEW** will display until the next of these 6 commands is encountered or the program stops. For programmed I/O, see the following **examples**:

- Feel free to use **STATUS** for monitoring, e.g., *flags* (→ *291ff*).
- Use **AVIEW** for displaying a text string (or message) stored in the source, without showing the *name* of the source.

- Take **VIEW** for displaying, e.g., intermediate results. Specify any *RoV* you want as source of information – also **X** is a valid parameter of **VIEW**. The *name* of the source will tag the output.

  ➡ Frequent display updates will slow down program execution.

- Use

  ```
  VIEW xyz
  PAUSE n
  ```

  for showing output for a fixed time interval of $\frac{n}{10}\,s$.

- Ask ('prompt') for numeric input employing

  Use **RCL**, **VIEW**, or **AVIEW** to display the *RoV* ***xyz***, then …

  ```
  STOP
  STO xyz
  ```

  … stop and wait for user reaction, finished by [R/S].
  stores what the user entered.

  A stop sign ❗ will be displayed in the *status bar* when the program hits a **STOP**. Whatever you will key in will be put into ***xyz*** when you continue program execution by [R/S].

  More elegant is using **PROMPT** or **INPUT** for this task:

  ```
  PROMPT yx
  STO xyz
  ```

  displays a message (like **AVIEW yx**) and stops, lighting ⍰.
  stores what the user entered.

  or:

  ```
  INPUT xyz
  ```

  displays and stops in just 1 step – also lighting ⍰.

- Prompt for text (string) input using the following steps:

  ```
  SF ALPHA
  RCL xyz
  STOP
  ```

  sets *AIM* for upcoming input.
  displays a stored message string and lights **A**.
  waits for your input. Whatever you key in now is appended to x when you continue by pressing [R/S].

  ```
  CF ALPHA
  STO yzx
  ```

  returns to the numeric mode previously set.
  stores x to wherever you like.

  Again, more elegant is using **INPUT** or **PROMPT**:

  ```
  SF ALPHA
  INPUT xyz
  CF ALPHA
  ```

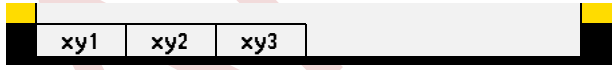
  sets *AIM* for upcoming input.

  returns to the numeric mode previously set.

- If you need to enter values for several *variables* then the following way is most efficient (although it may look lengthy here):

```
LBL 'Var.In'      we will need this label for VARMNU later.
  REM 'Following are the variable declarations:'
  MVAR 'xy1'
  MVAR 'xy2'
  MVAR 'xy3'
  REM 'And this creates and displays a menu for all these vars:'
  VarMNU 'Var.In'
  STOP            stops for user interaction.
  EXITall         exits the menu when program continues.
  RCL 'xy2'       recalls what you need first (it may have been entered in
                  the menu in arbitrary order).
```

The label called 'Var.In' here should be located close to the program top. It may be followed by up to 18 MVAR steps defining your *variables* required. When the program encounters the step VarMNU, it will set up a *menu* for all these *variables* and display it. In our case, this would look like



As long as such a menu is displayed and you want to…

- **write** a new value into 1 of these *variables*, key in the value or calculate it, then press the *softkey* corresponding to your target *variable*, and x will be stored in it.
- **recall** the present value of 1 of these *variables*, press [RCL] and then the corresponding *softkey*.
- **view** the present value of 1 of these *variables*, press [VIEW] and then the corresponding *softkey* (→ 63).[248]
- **exit** this *menu*, press [EXIT].
- **close** this menu **and continue** program execution, press [R/S].

If you continue by pressing a *softkey*, the *name* of the corresponding

[247] [P.FN] [REM] inserts a comment in your program. It will be skipped in execution.

[248] *HP-42S* allowed for viewing the present value of one of the variables displayed by just pressing [▼] and the corresponding *softkey*. We can't support this on *WP43* since this offers you 3 *menu* rows.

variable is stored into **K**. Your program can use this information to determine which key was pressed. Else, if you just continue by pressing [R/S], ***k*** is not altered.

- Directly react on particular keys pressed. The key codes returned by **KEY?** (→ 233) allow for 'real time' response to user input from the keyboard. **KEY?** takes a *register* argument (**X** is allowed but will not lift the stack) and stores the key most recently pressed during program execution or pause in the *register* specified.[249] Although the keyboard is active during program execution it is desirable to display a message and suspend the routine by **PAUSE** while waiting for user input. Since **PAUSE** will be terminated early by any key press, simply use **PAUSE 99** in a loop to wait 10 s for input. And since **KEY?** acts as a test as well, a user input loop may well look like the following **example**:

```
LBL 'User.in'
   RCL xyz
   PAUSE 99
   KEY? 00
   GTO 'User.in'
   LBL? →00
   XEQ →00
   GTO 'User.in'
```

displays the *register* with the message string.
waits 9.9 s for user input unless a key is pressed.
tests for user input and puts the key code in **R00**;
if there was no input then return to the beginning;
else if a label corresponding to the key code exists
then call it, …
else return to the beginning.

Instead of the dumb waiting step, the routine can do some computations in between and update the display before the next call to **PAUSE**.

To be even more versatile, you can use **?KTYP** to return the type of the key pressed if its row / column code is given (→ *IOI*).

If you decide not to handle the key in your program you may feed it back to the main processing loop of your *WP43* with the command **PUTK *nn*** . It will cause the program to halt, and the key will be handled as if pressed after the stop. This is especially useful if you want to allow numeric input while waiting for some special keys like, e.g., the arrows. After execution of the **PUTK** command you are responsible for letting the routine continue its work by pressing [R/S].

See the *IOI* for more about the commands mentioned in this chapter.

[249] Just [R/S] and [EXIT] can't be queried since they stop program execution immediately.

Solving Differential Equations

The following **examples** use the programmability of your *WP43* for solving differential equations, frequently appearing in various disciplines:

A simple ecological model[250] of interacting populations consists of rabbits with an infinite food supply and foxes that prey on them. The system can be approximated by a pair of nonlinear, first-order differential equations:

$$\frac{dr}{dt} = 2\,r - \alpha\,r\,f \quad \text{and} \quad \frac{df}{dt} = -f + \alpha\,r\,f$$

where r is the number of rabbits, f is the number of foxes, and α is a positive constant to show how frequently rabbits and foxes meet. When α = **0** there are no encounters; the rabbits keep breeding and the foxes starve. For a specific α, the probability of encounter is proportional to the product of the numbers of foxes and rabbits. A reasonable choice for α is **0.01**.

One way to solve this problem numerically is a simple *Euler* method, solving equations of the form:

$$x_{n+1} = x_n + \Delta t\; g(x_n)$$

using a … small increment of time Δt. The equations become:

$$r_{n+1} = r_n + \Delta t\;(2\,r_n - \alpha\,r_n\,f_n) \quad \text{and} \quad f_{n+1} = f_n + \Delta t\;(-f_n + \alpha\,r_n\,f_n)$$

The features of the *WP43* make it ideally suited for problems of this type.

```
LBL 'Foxes'
  REM 'Initialization taking input from the stack'
  STO 'n.fox'
  R↓
  STO 'n.rabb'
  FIX 5
LBL 01
  REM 'Begin of time loop'
  RCL 'n.fox'
  ENTER↑
  ENTER↑
  RCL 'n.rabb'
  RCL 'meet'
  ×
```

[250] *TN*: This is quoted from the *HP Journal* of 1975-11 for demonstration purposes only. I modified the original *HP-25* program using *named variables* instead of *registers* and adding comments for better readability and comprehending.

```
×                 number of meetings between rabbits and foxes
STO 'n.meet'        store it for later use in the other equation
x≷y
-                 change in number of foxes
RCL 'Δt'
×
+
x<0.?
0
STO 'n.fox'       new number of foxes
RCL 'n.rabb'
2
×
RCL 'n.meet'
-                 change in number of rabbits
RCL 'Δt'
×
RCL 'n.rabb'
+
x<0.?
0
STO 'n.rabb'      new number of rabbits
IP
RCL 'n.fox'
IP
E 5
/
+                 merge numbers of rabbits and foxes into 1 output number
PAUSE 20          pause for 2 seconds for data recording
GTO 01            end of (infinite) time loop, return to its begin.
```

Let's compute:

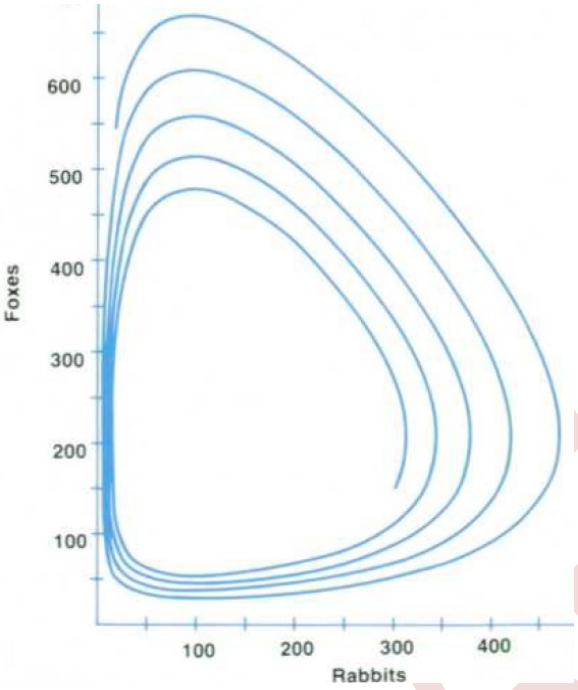
$r_0 = 300,\ f_0 = 150,\ \alpha = 0.01,\ \Delta t = 0.02$. Start with

0.01 [STO] **meet** **0.02** [STO] **Δt** **300** [ENTER↑] **150** [XEQ] **PROG** [Foxes]

The results can be plotted by hand on a graph of foxes versus rabbits. Note that the calculations give … floating-point numbers, but the display is always truncated to an integer.

A plot of foxes versus rabbits (→ overleaf) is a circular loop with a period of about five time units (250 iterations). The rabbit minimum is 14, the maximum is 342 on the first loop. The fox minimum is 54 and the maximum is 478 on the first loop.

$r_0 = 100,\ f_0 = 200$ is a constant solution.



The following method [251] solves ordinary 2nd order differential equations, a type frequently occurring in physics. In a 1st **example**, we will solve the equation of motion for the fall of a parachutist (or skydiver):

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

with Earth's gravity acceleration *g* and taking care of drag. Proceeding in small constant time steps ***Δt*** again, the following set of equations controls the vertical motion of this skydiver:

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + \left[g - b\left(\frac{df}{dt}\right)_0^2\right]\frac{\Delta t}{2} = \left(\frac{df}{dt}\right)_0 + A_0\,\frac{\Delta t}{2}$$

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

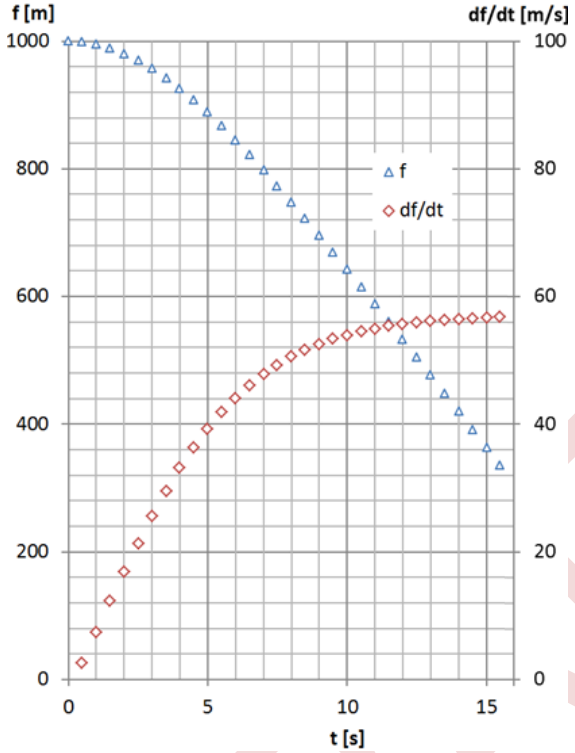
$$f_2 \approx f_1 + \left({}^{df}/_{dt}\right)_{3/2}\Delta t \quad \text{etc.}$$

Assume start height at time zero ($t = 0$) is 1000 m and vertical velocity is zero (i.e. $f(t_0) = f_0 = 1000$ and $\left(\frac{df}{dt}\right)_0 = 0$). Using *named variables* ***Δt***, ***b***, ***t***, ***f***, and ***dfbydt*** (a ⁄ must not appear in a *variable name*), the following routine will compute height and velocity of the skydiver as functions of time: [252]

[251] See *ReMA I* for more about this method.

[252] ***A*** must be ≥ 0 always. Thus, this routine will work for velocities $< \sqrt{g/b}$ only. Furthermore, it will not work for abruptly decelerating fast prior movements (e.g., by opening a parachute in free fall).

```
LBL 'PFall'
   .5                initialize all variables used
   STO 'Δt'
   .003              assumed realistic drag value for a body (with closed
   STO 'b'                                          parachute) falling in air
   1000              start height
   STO 'f'
   0                 start time and velocity
   STO 't'
   STO 'dfbydt'      end of initialization
LBL 01
   REM 'Begin of time loop'
   # g⊕              recall g⊕ from CONST
   RCL 'b'
   RCL 'dfbydt'
   x²
   ×                 b × (df/dt)²
   -                 g – b × (df/dt)² = A
   RCL 't'
   x>0.?             check time – it will be zero in 1st run
   GTO 02            from 2nd run on go to local label 02
   REM 'for the first run only'
   DROP↓             forget t
   2
   /                 A / 2
   GTO 03            go to common part
LBL 02
   REM 'from second run on'
   DROP↓             forget t
LBL 03
   REM 'common part resumes here for plotting next data points'
   STO+ 'dfbydt'     calculate the new df/dt
   RCL 'Δt'
   STO+ 't'          calculate the new time
   RCL× 'dfbydt'     df/dt × Δt
   STO- 'f'          calculate the new f
   VIEW 't'          new time
   STOP                for reading
   VIEW 'f'          new height
   STOP
   VIEW 'dfbydt'     new velocity
   STOP
   GTO 01            ad infinitum
END
```



Now, leave *PEM* and start the program: XEQ PROG (PFall).

Plotting the points calculated for ***f*** and ***df/dt*** will result in a diagram like this. Height decreases following a parabola over time at beginning but becomes linear later since vertical velocity does not increase much after some 12 s here, approaching 57 m/s (205 km/h) while skydiving with closed parachute.

For comparison: The velocity limit with an open parachute ($b \approx 0.3$) will be < 6 m/s, so touchdown will be like falling from a wall 1.65 m high. Don't forget to open it in time!

In a 2nd **example**, we will look at a horizontal harmonic oscillator with spring rate ***r*** and mass ***m***. Oscillating freely, its equation of motion is simply

$$\boldsymbol{m}\,\frac{d^2f}{dt^2} = F = -\boldsymbol{r}\,f$$

Adding an external stimulating force and velocity-dependent damping, the equation will change to

$$\boldsymbol{m}\frac{d^2f}{dt^2} = -\boldsymbol{r}\,f - \boldsymbol{b}\frac{df}{dt} + \boldsymbol{s}\sin\omega t \quad\Leftrightarrow\quad \frac{d^2f}{dt^2} = -\alpha\,f - \beta\,\frac{df}{dt} + \gamma\sin\omega t = A$$

with $\alpha = {}^{r}/_{m}$, $\beta = {}^{b}/_{m}$, and $\gamma = {}^{s}/_{m}$. Then the following set of equations controls the oscillator motion for small constant time steps $\boldsymbol{\Delta t}$:

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + \left[-\alpha f_0 - \beta\left(\frac{df}{dt}\right)_0 + \gamma\sin\omega t_0\right]\frac{\Delta t}{2} = \left(\frac{df}{dt}\right)_0 + A_0\,\frac{\Delta t}{2}$$

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

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

Now, all you have to do is rewriting the 1st part of previous program:

```
LBL 'Osci'
   .2                    initialize all variables used
   STO 'Δt'
   1                     assumed relative spring rate
   STO 'α'
   .5                    assumed relative damping
   STO 'β'
   1                     size factor for stimulation
   STO 'γ'
   RAD
   .3                    frequency of stimulation
   STO 'ω'
   1.5                   start position
   STO 'f'
   0                     start time and velocity
   STO 't'
   STO 'dfbydt'          end of initialization
LBL 01
   REM 'Begin of time loop'
   RCL 't'
   RCL 'ω'
   ×
   sin                   sin(ω t)
   RCL 'γ'
   ×                     γ sin(ω t)
   RCL 'β'
   RCL 'dfbydt'
   ×
   -                     γ sin(ω t) – β df/dt
   RCL 'α'
   RCL 'f'
   ×
   -                     γ sin(ω t) – β df/dt – α f  = A
   RCL 't'
   x>0.?                 check time – it will be zero in 1st run
```

The remaining code can be taken over from the 1st example above as is (feel free to delete the output of *df/dt* if you are not interested in it). Play with the parameters and observe the results.

Very similar equations apply to vertical oscillators, actually to almost all vibrating, swinging, or bouncing objects and parts. Enjoy!

Consult *ReMA I* for more about solving differential equations, also in *2D* and using another method.

The Programmable Menu (MENU)

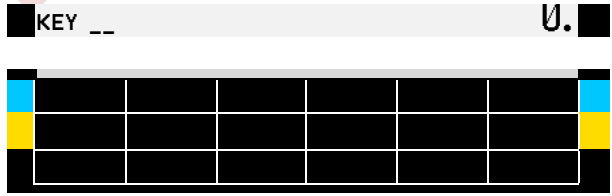
Your *WP43* features a *programmable menu* you can use to cause program branching. By this, you can create *menu*-driven programs. The command **MENU** selects this *programmable menu*; it will be displayed when the program stops. You can define each *item* in this *menu* so that when the respective key is pressed, a particular **GTO** or **XEQ** instruction will be executed. You can even redefine ▲, ▼, and EXIT.[253]

To define a softkey in the programmable menu:

1. Store a *text string* of up to 7 characters in *register* **X**. This text will appear in the *menu space* for the *softkey* specified (if free space doesn't suffice, only the first characters of ***x*** will be displayed). **X** is neither used nor dropped when defining ▲, ▼, or EXIT.

2. Call **KEYG** (i.e. on key, go to)
   or **KEYX** (i.e. on key, execute). You find them in P.FN.

3. The *menu view* changes – the black fields indicate you shall specify which *softkey* you want to define:

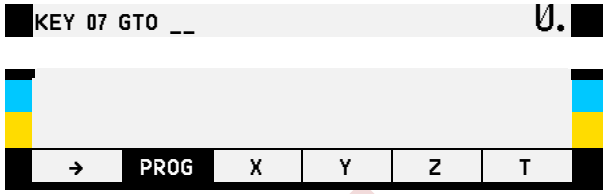
   

   Call 1 of the 18 *softkeys* available (including 🟧 or 🟦, if applicable), ▲, ▼, or EXIT. Alternatively, enter the respective key number, 1 through 21 (the unshifted leftmost *softkey* carries #1, the 🟦-shifted leftmost *softkey* #13, ▲ #19, etc.).

[253] See the *HP-42 OM*, pp. 145 - 148, and its *Programming Examples and Techniques*, pp. 29 - 39, 92 - 99, 158 - 160, and 184 - 192, for examples using the *programmable menu*.

Let's assume you called KEYG in step 2 and pressed the 🟨-shifted leftmost *softkey* now.

4. The command echo changes:



5. Specify a program label using 1 out of the following 4 methods (the *menu view* changes, → 226):
   a. Key in a single-letter local label out of **A** – **E**.
   b. Key in a 2-digit local label,
   c. Select a global label by pressing the corresponding *softkey* in PROG.
   d. Type a global label character by character using [α] and *AIM*.

   Unless you have just defined ▲, ▼, or [EXIT], *x* will be dropped.

   Repeat this procedure (steps 1 – 5) for each *softkey* in the *programmable menu* you want to define. A new definition replaces any previous definition that may exist for this *softkey*.

To display the programmable menu, execute the MENU function, e.g., by [P.FN] P.FN2 MENU.

To clear all softkey definitions in the programmable menu, press [CLR] CLMENU.

Basic Kinds of Program Steps

You have seen various program steps so far. Each step takes a single place in *PM*, and each step is numbered automatically. Basically, these steps fall into 5 categories – each may contain either ...

1. a global or local label (like LBL 'Osci' or LBL 02 above) or
2. an entire command (like - or y^x or $\log_x y$ or STO× →'Prod2' or REM 'Note ξ varies acc. to Fēi Cháng-hǎo et al., 2023') or

3. a *numeric constant*, either keyed in like `-1.902×10^-16`, `1 23 45₁₆`, `1.7+i×0.4`, `-23 4/5`, or recalled from CONST like `# λc`;[254] or
4. a fix *date* (like `2023-09-10`) [254] or
5. a fix *time* (like `21:45:17.654`) [254] or
6. an *alphanumeric constant* (like `This is the 2nd iteration output:`). Such a string may be up to 196 characters or 256 bytes or 4 lines long, whatever will be reached first. It will be displayed enclosed in ' (else some *strings* might be misinterpreted as numbers or commands in program listings), though automatically stored in X without these delimiters.[255]

Since each constant takes 1 step in a routine, there is no necessity for separating them by `ENTER↑` here.

Example:
Think of calculating 12.3 + 45.678 in a routine. Then keying in
12.3 `ENTER↑` ← **45.678** `+` will result in a program snippet

```
12.3
45.678
+
```

which will do for returning 57.978. The missing `ENTER↑` saves 1 *byte* of program space and makes the routine a tiny bit faster. This may be not really crucial but you should know.

Constant *vectors* and *matrices* can't be entered directly in a program. Instead, however, you can create such objects in advance, store them in *registers* or *variables*, and manipulate these stored *items* (as described in *OMS 3*) in routines as well, of course.

Each program step requires at least 1 *byte* of memory (→ *ReMA B*). Use `?MEM` to learn about the free space remaining in the pool. We think you will hardly ever run out of program space – but it is possible. If you do while trying to enter a new program step, you will read the message `RAM`

[254] The respective value will be automatically stored in X.

[255] The character ' is no member of any *menu*.

does not suffice ; → *ReMA C* and *B* for ways to escape from such a situation.

You can use the vast majority of operations provided on your *WP43* to solve repetitive and iterative problems of almost any kind you can imagine, with a speed exceeding the capabilities of earlier *HP* pocket calculators by far.

Especially when you begin with developing and creating keystroke programs, such routines will be more difficult to read than programs written in a high level language – in particular since inline comments are not supported (but feel free to use **REM** frequently wherever beneficial) and indenting is limited; also obvious, easily recognizable control structures are missing. Loops in particular may need some habituation.

Let's look at an exemplary *PASCAL* structure and the equivalent keystroke sequence doing the same on your *WP43*:

Pascal	Keystrokes	Comments for the keystroke program steps
.....	.....	
.....	.....	
.....	19	
.....	STO J	Load the comparative value.
.....	1.03403	Initialize loop control (begin with **1**,
.....	STO I	end at **34**, increment by **3**).
for I = 1 to 34 by 3 do;	LBL 01	Begin of the big loop.
begin;	REM 'xxx'	Write something meaningful here.
.....	.....	
.....	.....	
.....	.....	
.....	.....	
.....	RCL I	Recall the loop counter.
if I < 19 then do;	x≥ ? J	If ***i*** ≥ **19**
begin;	GTO 02	then jump to local label 02
.....	REM 'yyy'	else (for ***i*** < **19**) proceed here.
.....	.....	
.....	.....	
.....	.....	
.....	.....	
end;	GTO 03	When ready, skip next section.
else do;	LBL 02	Do what shall be done for ***i*** ≥ **19**.
begin;	REM 'zzz'	Write something meaningful here.
.....	.....	
.....	.....	
.....	.....	
.....	.....	
end;	LBL 03	End of if-condition.
.....	.....	
.....	.....	
.....	.....	
.....	.....	
.....	.....	
.....	ISG I	Eventually increment ***i*** and check.
end;	GTO 01	If ***i*** ≤ **34** then run the big loop again
.....	.....	else it is done now.
.....	.....	Continue.

Once you have learned the basics (like "**do if true**, skip if false", sometimes reverting the logic of a test, and checking the loop counter at the end of the loop), you will enjoy the freedom you have.

Deleting Programs

To delete some steps of a program, do as explained on pp. *226f*. Repeat as often as necessary.

To delete an entire program quickly, press CLR CLP, then either

- PROG to choose the target *program*, or
- α to enter the target *program name*, or
- ENTER↑ to delete the *current program*.

! CLP will not ask for confirmation! It will remove the entire *program* specified without further ado, not just the *current routine*. And CLP can't be undone! The space freed by CLP will be returned to the pool your *WP43* offers for your data and programs.

To delete all programs stored in *RAM*, press CLR CLPall, stop and **think**, and deny your command or confirm. Then CLPall will wipe out the entire *PM* at once, so you may get up to some 45 000 *bytes* for new programs and data (depending on the space your data in *RoV* are occupying, for example).

! Also CLPall, once confirmed, cannot be undone.

Flash Memory (*FM*)

In addition to the *RAM* provided, your *WP43* also allows you accessing its large *FM*. The major part of it is allocated to a so-called *FAT disk* (*FD*) of 6 MB (→ sketch in next chapter). You can use it for voltage-fail-safe storage of your programs and data, e.g., for backup. These will also survive RESETs, even hard ones.

Use SAVE for creating a safe on-board backup of your entire work (your calculator *configuration*, programs, *variables*, *register* contents, *flag* status, etc.).

➡ There will be only 1 backup file in *FM* – pressing SAVE again will overwrite previous backup.

Alternatively, I/O **SAVEST** saves the *calculator state* in a named file on the *FD*. Such a *state* encompasses the same data as a backup but you may store various *states* under different filenames, so you can return to various calculator *states* depending on your actual tasks and requirements.[256]

The different flavors of **Load...** (in I/O) are for recalling saved data from the backup or an arbitrary *state file*:

- **LoadΣ** recalls the saved statistical *matrices* STATS and HISTO as well as the *statistical summation registers*,
- **LoadR** recalls the contents of all *general purpose registers* saved (this includes the entire *stack* and **L**), and
- **LoadSS** recovers the saved system settings of your *WP43* (incl. its *configuration*),
- **LoadV** recalls all user-defined *variables* saved,
- **LoadEq** all equations,[257] **LoadPr** all programs,[258] and
- **LoadAll** recalls the entire backup or state file at once, overwriting your present *calculator state*.

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

While *FM* is the (even battery-fail) safe place, hence ideal for long-living data, it shall not be used for frequently changing temporary storage like in programmed loops.[259] Conversely, *registers* and standard user *PM* residing in *RAM* are designed for data changing with high frequency – but they will not hold any data when the battery dies or is removed for longer

[256] Instead, **STOCFG** stores the *WP43 configuration* in an *RoV*, i.e. in *RAM*.

[257] **LoadEq** will add the recalled equations not listed in *RAM* before. Identical equations will not be touched.

[258] **LoadPr** will add (append) the recalled programs to those present already. Since *program names* (i.e. their starting *global labels*) may be reused, programs carrying identical *names* may appear in *RAM* more than once, especially after **LoadPr**. This may fill the *RAM* faster than expected – your *WP43* will throw **RAM does not suffice** then; we recommend purging *PM* before if appropriate.

[259] *FM* may not survive more than 100 000 flashes (>27 years with 10 flashes per day, for example). Although this will be hardly reached in normal use, we made commands writing to *FM* (**SAVE**, **SAVEST**, and **WRITEP**) non-programmable. Better safe than sorry.

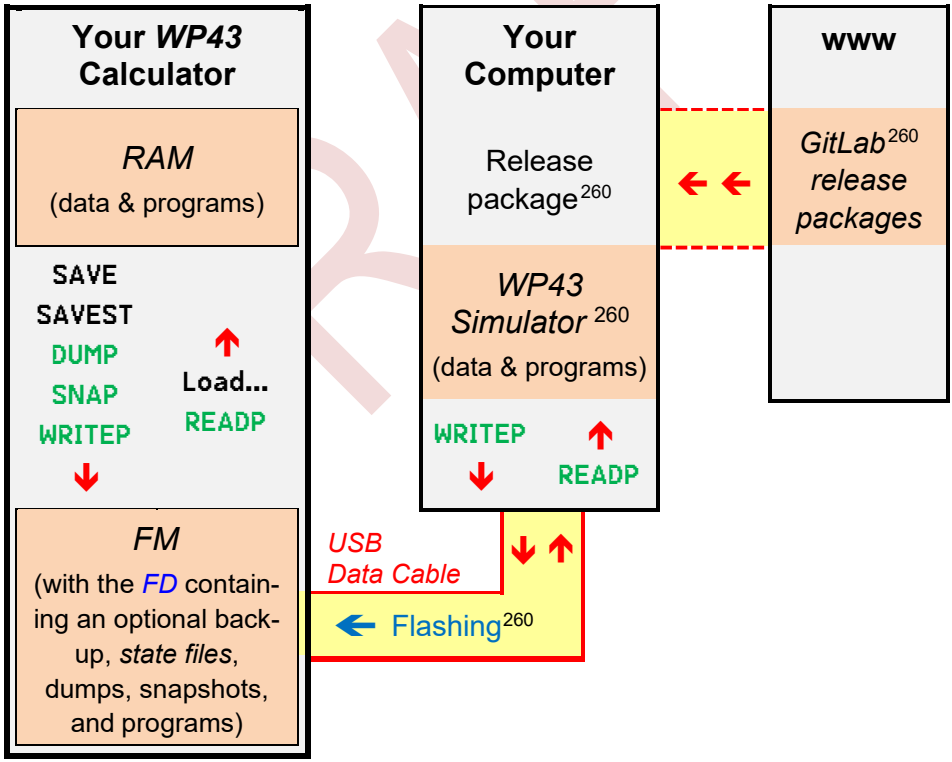
than some *seconds*. So both *RAM* and *FM* have their specific advantages and disadvantages you should take into account for optimum benefit and longevity of your *WP43*.

Communicating with Your Computer

Looking at I/O from your *WP43* calculator, this matter appears as pictured schematically below (not to scale).

For **SAVE**, **SAVEST**, and the different flavors of **Load** return to previous chapter. **DUMP**, **SNAP**, **READP**, and **WRITEP** are covered below.

➡ Neither system state nor `.p43` files (→ 256) nor *configurations* are easily user-readable – but there is an alternative further below.



[260] Please see *ReMA E* and *F*.

Data exchange with your *PC*:

There are 2 ways you can exchange data:

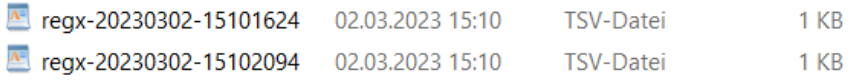
1. With a *USB* cable connecting your *WP43* with your *PC*, you can 'activate' the *FD* (a.k.a. *USB* disk) in *FM* by [USB]. Then said disk will be part of the address space of your *PC*, and you can use file management applications there (e.g., the *Explorer* on a *Windows PC*) to copy or move arbitrary files to and from this *FD*. Having done this, [EXIT] [ENTER↑] will return the *FD* to your *WP43* calculator exclusively, so you can continue calculating.

2. You can write programs from your *WP43* onto *FD*, dump specific results (e.g., *matrices* or very long integers), save snapshots, backups, or *system states* in files there. You may then employ method 1 and deal with these files on your *PC*, enjoying more editing comfort than our little calculator can offer. We will elaborate on this way in the following:

   [I/O] **WRITEP** writes your *current program* in a named `.p43` file on *FD* in the directory `PROGRAMS`.

   [I/O] **READP** reads a named `.p43` file from *FD* into *PM*, so you can execute or edit this program.

   [I/O] **DUMP** dumps ***x*** with its full precision onto the *FD* in a file `regx-`***`yyyymmdd-hhmmsscc`***`.tsv` (according to date and time of the dump; → two examples below) in directory `SAVFILES`. This is the way to permanently output all digits of very long integers ($>10^{44}$) – note you can SHOW such an integer up to 10^{296} easily but only its top 44 digits will stay on display after next keystroke. DUMP can write up to 1001 digits.

   Such files are user-readable on your *PC*: use a text editor (e.g., *Notepad*) for dealing with them, print them, etc.

SNAP stores a snapshot (a.k.a. screenshot, hardcopy) of the entire *WP43* calculator screen on the *FD* in a graphic file named ***yyyymmdd-hhmmsscc***.bmp (according to date and time the snapshot was taken) in directory SCREENS. The same can be achieved by pressing ■ and E simultaneously – this will also work in *PEM* and with *temporary information* (like messages) displayed.

➡ The *WP43* display is too large (it features too many pixels) to be printed on a vintage *HP* printer (→ *IOI*). Though you can copy such a file to your *PC* and deal with it there like you do with any .bmp file (edit, print, etc.).

SAVE and SAVEST were covered in previous chapter already.

What works on your physical *WP43* shall work on the *Simulator* as well – both platforms share the same set of functions. Just where your *WP43* will write *FD* files for **DUMP** and **SNAP**, the Simulator will write to the *clipboard*.

So you can write the current program you are working at from *PM* of your *WP43* calculator in a file on its *FD* (using **WRITEP**), connect your *WP43* to your *PC*, and read the program file from the *FD* into *PM* of the *Simulator* (using **READP**). Then test, debug, and edit it there profiting from the advantages of your *PC* keyboard (feel free to compute with your *WP43* calculator disconnected in this time). When finished and satisfied with program execution, **WRITEP** from your *PC* back on the connected *FD*, disconnect, **READP** into *PM* of your *WP43* calculator, and work with the refined program there then. Find more about this procedure in *ReMA E*.

Local Data

After some time with your *WP43* you will have a number of routines stored, and keeping track of their resource requirements may become challenging. Most modern programming languages take care of this by declaring *local variables*, i.e. memory space allocated from general data memory and accessible for the *current (sub-) routine* only; when the routine is finished, this memory is released. On your *WP43*, mainly *registers* are

used for data storage – so we offer *local registers* allocated to your current routine exclusively.

Example 1:

Let's assume you create a routine labeled **P1** and need 5 *registers* (in addition to the *stack*) for your computations therein. Then all you have to do is just enter *PEM*, go into the routine **P1**, and key in (close to its top)

P.FN **LocR** 5 ENTER↑

specifying that you want 5 *local registers*. After this step, you can access these new *registers* using local addresses **.00** – **.04** throughout routine **P1**.

Now, if you call another routine **P2** from **P1**, also **P2** may request *local registers* by **LocR**. Then these will carry *local register* addresses **.00** etc. again, but the *local register* **.00** of **P2** will be physically different from the *local register* **.00** of **P1**, so no interference will occur (neither **P2** can access *local registers* of **P1** nor vice versa). – As soon as **RTN** or **END** is executed, the *local registers* of the respective (sub-) routine will be released and the memory they took will be returned to the pool of free space.

Local data holding allows for recursive programs, since every time such a routine is called again it will allocate a new set of *local registers* and *user flags* being different from the ones it got before. → **LocR**, **?LocR**, and **POPLR** in the *IOI*.

You may allocate up to 99 *local registers* per routine (as far as memory allows). In addition, you get 32 local *user flags* as soon as you request at least 1 *local register*.

Example 2::

The following subroutine acts as a built-in test preserving the *stack*: it checks if **X** is tagged as *decimal* or *sexagesimal degrees*, returning to the next instruction if true and skipping 1 instruction if false:[261]

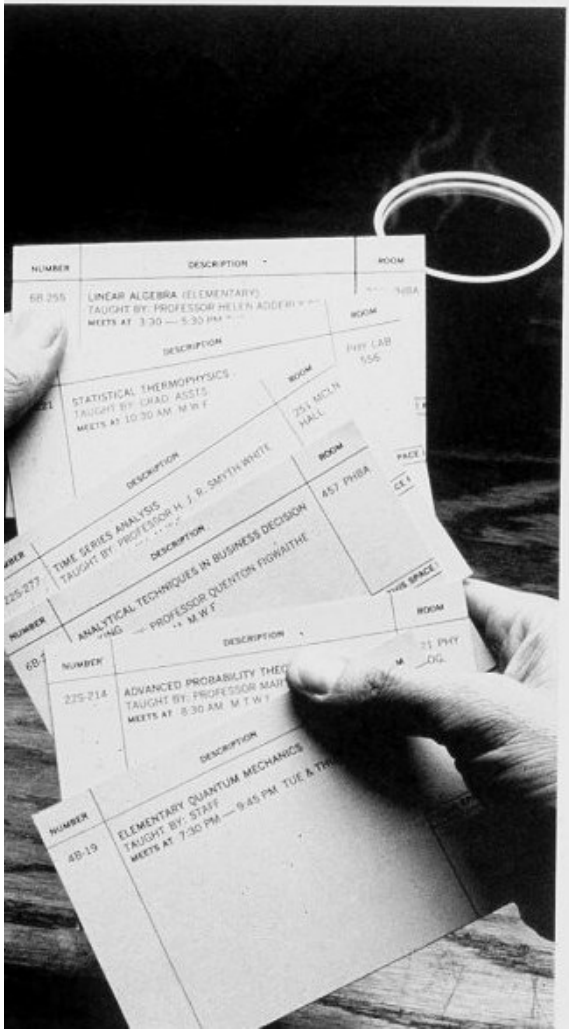
```
0001 LBL 'TAGGD?'
0002    LocR 08      allocate 8 local registers to the current routine
0003    STOSTK .00     for stack saving.
0004    ''           set up a minimum text string in X
```

[261] Kudos to *Didier*.

```
0005    +          and append y (= previous x).
0006    176        176 = B0₁₆ = °
0007    x→α
0008    αPOS Y     look for a ° in y.
0009    x< ? 0.     if there is none
0010    SKIP 002      then go to step 0013
0011    RCLSTK .00    else recall the entry stack
0012    RTN             and return to caller;
0013    RCLSTK .00    but if a ° was found then recall the entry stack
0014    RTN.SKP         and return to the caller but 1 step downstream.
```

Since you are free to nest subroutines (→ *229*), and each subroutine may allocate its own *local registers*, the total number of *registers* allocated may become quite big. The total number of accessible *registers* at one time, however, will never exceed 211 (→ *59*). Look up *ReMA B* for more information about the space required for various *DTs*.

If you're taking tough courses, you need all the help you can get.



If you've really done it to yourself this term, you need an advanced calculator you can count on through thick and thicker.

You need the most advanced functions and programming features. You need lots of pre-written programs to save you time. You need Continuous Memory and the utmost in dependability. You need an HP calculator.

The HP-34C.

All the help you can get.

Hewlett-Packard offers you eight different calculators priced from $55* to provide professional solutions in science, engineering and business.

So visit your nearest HP dealer for a hands-on demonstration. Then buy an HP calculator. It may be the last easy thing you do for a long time.

For details and the address of the dealer in your area, call toll free: (800) 547-3400, Dept. 265E, except Hawaii and Alaska. In Oregon, call 758-1010. (Or write Hewlett-Packard, Corvallis, OR 97330, Dept. 265E.)

*Prices are suggested retail excluding applicable state and local taxes — Continental U.S.A., Alaska and Hawaii. 611-12

HEWLETT PACKARD

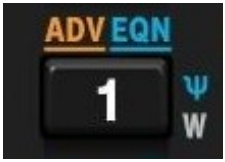
An *HP-34C* displaying ζ(3) – feel free to compare it with the result of your *WP43*.

The *HP-34C* was the first pocket calculator worldwide featuring a general *Solver* and *Integrator* on its keyboard. Read *OMS 5* below for more.

Previous page shows an advertisement for this calculator of 1980 or 81. Compare f. 55 on p. 46.

SECTION 5: ADVANCED PROBLEM SOLVING

Beyond what you have seen so far, your *WP43* provides some powerful commands for computing programmable sums and products, definite integrals, 1st and 2nd derivatives as well as for solving equations. All these are provided both in ADV and in EQN. Pressing [ADV] in *RUM* results in this *view*:



PGMSLV	SLVC	f"(x)			PGMINT
SOLVE	SLVQ	f'(x)	Π_n	Σ_n	$\int f dx$

The commands **Σn**, **Πn**, **SLVQ**, **SLVC**, **SOLVE**, **∫**, **f'(x)**, and **f"(x)** are explained below in this order. All of them are programmable. Also interactive integrating, deriving, and solving equations is possible – it can be reached through [EQN]. See below for details and examples.

Programable Sums

Do you need results of finite sums? **Σn** computes them for you. It is called with a loop control number (*LCN*) in **X** and a *label* trailing the command. Said *LCN* follows the format `ccccc.fffii` (→ **DSE** etc. in the *IOI*).

In its heart, **Σn** then works like this:

1. **Σn** sets the sum to **0** initially.
2. **Σn** fills all *stack registers* with ***c*** and calls the routine specified by ***label***. That routine returns a summand in **X**.
3. **Σn** adds this summand to said sum.
4. **Σn** decrements ***c*** by ***i*** (or by **1** if ***i*** = 0); if ***c*** ≥ ***f*** then **Σn** goes back to step 2, else it returns the final sum in **X**.

Example:

$$\sum_{k=0}^{100} \sqrt{k} = ?$$

Solution:

1. Write a little program for the internal calculation of the summands:

```
LBL 'ΣV'
  √x
  RTN
```
.

2. Enter

   100

   ADV Σ_n α S V ENTER↑ (or pick **ΣV** from PROG, → 226)

   and get 671.462 9 returned if **FIX 4** is set.

Σ_n deliberately sums from the last term to the first, on the assumption that summations will often be of convergent series so this summing sequence should generally increase accuracy of result.

Programmable Products

Finite products can be computed as well as sums. The command Π_n is called with a loop control number in **X** and a *label* trailing the command – its parameters are the same as those required for Σ_n above.

In its heart, Π_n works almost as Σ_n:

1. Π_n sets the product to **1** initially.
2. Π_n fills all *stack registers* with ***c*** and calls the routine specified by the label. That routine returns a factor in **X**.
3. Π_n multiplies this factor with said product.
4. Π_n decrements ***c*** by ***i*** (or by **1** if ***i*** = 0); if ***c*** ≥ ***f*** then Π_n goes back to step 2, else it returns the final product in **X**.

Example (with **SCI 4**):

$$\prod_{k=1}^{90} \sqrt{\frac{1}{\sqrt{1+k^2}}} = ?$$

Solution:

1. Write a little program for the internal calculation of the factors:

```
LBL 'PROD'
   √(1+x²)
   1/x
   √x
   RTN
```

2. Instead of calling PROD by [ADV] [Π_n] you can call it from another program and measure the time it takes. Thus write a 2nd little program:

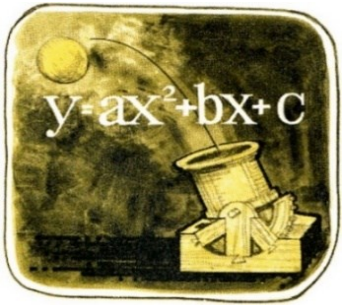
```
LBL 'P2'
   TICKS        Start time
   STO 00       Store it since Π_n is going to fill the entire stack.
   90           Start value. Default step size is 1, final value 0.
   Π_n 'PROD'
   TICKS        Stop time
   RCL- 00      Get the time difference in ticks.
   RTN
```

[XEQ] [PROG] [P2] and get 5.941 4×10⁻⁷⁰ and **9** *ticks* (0.9 s) returned when powered by battery, 0.3 s when powered by *USB*.

Solving Quadratic Equations

SLVQ finds the *real* and *complex* roots of an arbitrary quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input *stack* [***c***, ***b***, ***a***, …] :

- If $r := b^2 - 4ac \geq 0$ then SLVQ will return 2 *real* roots $(-b \pm \sqrt{r})/2a$ in **Y** and **X**. If called in a routine then the step after SLVQ will be executed.
- Else, SLVQ will return the 1st *complex* root in **X** and the 2nd (*complex conjugate* of the 1st) in **Y**. If called in a routine then the step after SLVQ will be skipped.



So actually SLVQ tests for *real* roots at its very end. In either case, it returns ***r*** in **Z**. Higher *stack registers* are kept unchanged. **L** will contain parameter ***c***.

Example:
Find the roots of $4x^2 - 3x - 2 = 0$.

Solution (with FIX 4 chosen):
4 ENTER↑ **3** +/− ENTER↑ **2** +/−
ADV **SLVQ** returns $x = 1.175\,4$, $y = -0.425\,4$, $z = 41.000\,0$. Since z is positive, x and y are the 2 *real* roots of this equation here.

Check: Store x in **J** and y in **K**. Then enter
RCL J FILL **4** × **3** − × **2** − returning $2.000\,0\times10^{-33}$ and
RCL K FILL **4** × **3** − × **2** − returning $0.000\,0$.

Remember your *WP43* calculates with 34 digits precision, so any result within $\pm\,3\cdot10^{-33}$ is equal to zero in this matter.

Solving Cubic Equations

SLVC finds the *real* and *complex* roots of an arbitrary cubic equation $ax^3 + bx^2 + cx + d = 0$ with its parameters on the input *stack* [***d***, ***c***, ***b***, ***a***, …] .[262] For real parameters and the *discriminant*

$$\Delta = b^2c^2 - 4ac^3 - 4b^3d - 27a^2d^2 + 18abcd\,,$$

there are 3 cases:

1. $\Delta > 0$ leads to 3 distinct *real* roots,
2. $\Delta = 0$ means a double or triple root,
3. $\Delta < 0$ leads to 1 *real* root and a pair of *complex conjugate* roots.

SLVC will return the 3 roots in **X**, **Y**, and **Z**, plus ***Δ*** in **T**. If called in a routine, **SLVC** will act as a test returning **true** for $\Delta \geq 0$, so the step after **SLVC** will be executed then – for $\Delta < 0$, this step will be skipped.

[262] For finding roots of higher grade polynomials, feel free to apply the *Lin-Bairstow* method (→ https://en.wikipedia.org/wiki/Bairstow's_method or in German language https://de.wikipedia.org/wiki/Bairstowverfahren). Or just apply the *Solver* explained further below.

Arbitrary Algebraic Equations

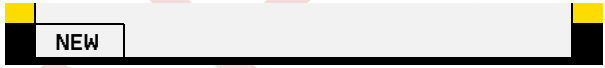
EQN lets you create, store, select, edit, and delete arbitrary *algebraic* equations. You may use each equation for …

- solving it interactively for any *variable* it comprises (→ next chapter),
- plotting it,
- integrating or differentiating it.

The total number of equations and *variables* storable is limited just by the amount of free memory available.

Example:

Press [EQN]. If there is no equation in memory yet, your *WP43* will return:



Press **NEW** to enter a new equation. You will get:

with *AIM* turned on (→ 208ff). Key in your equation, e.g.,

height = h [↓][0] [+] v [↓][0] [×] time [−] $g_{\oplus}$ [/] [2] [×] time [^] [2] [263]

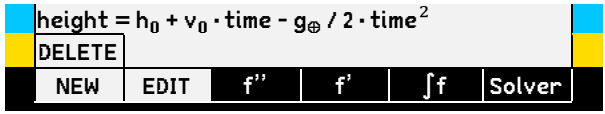
for the height of, e.g., a ball thrown vertically upwards with velocity $\boldsymbol{v_0}$ starting at height $\boldsymbol{h_0}$ (e.g., from the *Eiffel Tower* or a *Zeppelin* – well, rather drop it from the latter). Press [ENTER↑] for closing the *Equation Editor* and see:

---

[263] Type what you need – p. 270 shows what can be picked from the keyboard directly. The *Equation Editor* starts in lower case for good reasons.

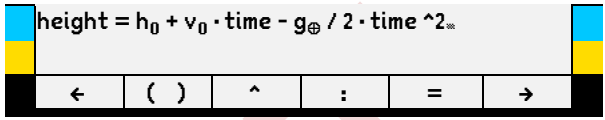
The index $_{\oplus}$ is stored in αMATH, and can be accessed via [▮] [−] [▼] … Picking a character from αMATH will insert this character into the equation and close αMATH, so you can continue editing. This applies also to the 2 other alpha *menus* when writing an equation.

Even easier: You can pick $g_{\oplus}$ directly from CATALOG'CONST.



You will get such a *view*[264] whenever 1 or more equations are stored. The equation displayed in the blue row is called the *current equation*.[265]

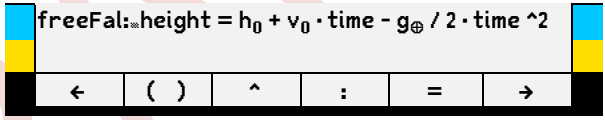
Pressing EDIT opens the *Equation Editor* for this equation:



You may modify this equation at any position by moving the edit cursor to the location behind the symbol(s) to be changed and pressing ⌫ as required followed by the new symbol(s) to be inserted here.

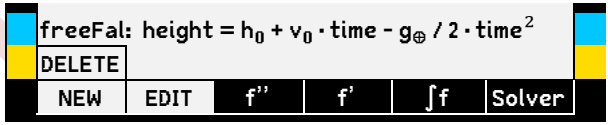
For labeling this equation, move the cursor left to its very begin using ←, and key in up to 7 characters (→ 60) plus a colon there:

F R E E ▲ F ▼ A L :



(If an equation becomes wider than the screen, ellipses will be displayed at its end(s); then use ← or → for scrolling.)

ENTER↑



ENTER↑ terminates the *Equation Editor* and stores the modifications you made.

➡ Editing an equation clears all its *variables*. You are free to create

---

[264] Here, your *WP43* uses multiplication dots for sake of better readability (you could hardly distinguish × and **x** in an equation displayed). For the same reason, spaces are inserted automatically. Note implicit multiplications are not supported in equations.

[265] A dashed line will appear over the current equation when there are more. To select another equation, press ▲ or ▼ then until the requested one appears.

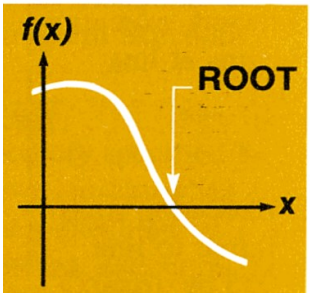
and baptize your *variables* to your liking, but remember *variable names* must be unique (→ *60*). Furthermore, you may employ any constant specified in your *WP43* by just calling its *name* (→ *298ff*). You must not, however, reuse any *name* of such a constant for any of your *variables* – else your *WP43* may misunderstand your intentions. Actually, do not reuse any *item name* already defined for your *variables*.[266]

And watch proper spelling of functions and *variables*. Your *WP43* will definitely not decode `TAN(α)` as you may have meant it, but as you wrote it. How shall it know any better? → *270*, the last chapter of *ReMS 3*, and *ReMA N*.

Evaluation of equations follows *PFEMDAS* (i.e. parentheses first, then functions, exponentiations, multiplications, divisions, additions, and eventually subtractions).[267] *Unary minus* is interpreted as a multiplication times −1. And the number of pending operations as well as the number of different parameters in each such equation must not exceed 9.

The Solver

Interactively Solving Arbitrary Algebraic Equations



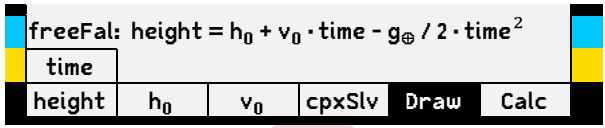
The built-in *Solver* application of your *WP43* is a special root finder that enables you to solve an equation for any of its *variables*. It allows solving for an arbitrary unknown as well as for finding the (real or complex) root(s) of an arbitrary equation.[268]

[266] Thus, the following hypothetical 1-letter *variables* must not appear in any equation: ***A***, ***B***, ***C***, ***D***, ***F***, ***G***, ***I***, ***J***, ***K***, ***L***, ***R***, ***T***, ***X***, ***Y***, ***Z***, ***a***, ***c***, ***e***, ***h***, ***i***, ***k***, and ***s*** – but, e.g., ***j***, ***x***, ***y***, or ***z*** may. See the *IOI* for all the *item names* specified.

[267] *TNG*: Klammer vor Funktion vor Hoch vor Punkt vor Strich.

[268] *TNG*: Der eingebaute Gleichungslöser („*Solver*“) Ihres WP43 erlaubt Ihnen das Auflösen nach einer beliebigen Variablen bzw. das Finden reeller oder komplexer Nullstellen in einer beliebigen Gleichung

Press EQN, make the equation you want to solve the *current equation* (→ previous chapter), and press **Solver**. Your *WP43* will check this equation for syntax errors (missing operators, misspelled functions, illegal *variable names*, etc.). It will then return a *menu* of all applicable *variables*, like the one in our **example**:



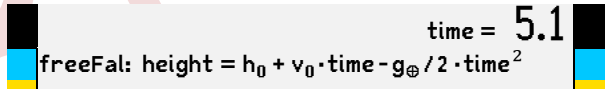
Your *WP43* recognized $g_\oplus$ being defined in CONST (→ 298ff). It will detect other constants as well, if applicable. Now you can enter values for each known *variable* by pressing the respective *softkey*, e.g.,

0 height **50** h_0 **15** v_0 corresponding to a target height 0, a start point on a 50 m platform, and a velocity of 15 m/s upwards at time zero. Only 1 unknown variable remains.

(Option: enter 1 or 2 initial guesses for the unknown like **5** time **10** time.)

Then press f F1 for the unknown …

time (now without any numeric input heading – the *Solver* takes what is in **X** and **Y** as initial guesses for *time*) and your *WP43* will solve the equation for this *variable* and return its value in **X** (for FIX 1 set):



i.e. 5.1 s until the ball hits the ground.[269]

Use RCL for the respective *variables* to recall your inputs or results later on.

Another **example** (from the *HP-27S OM*):

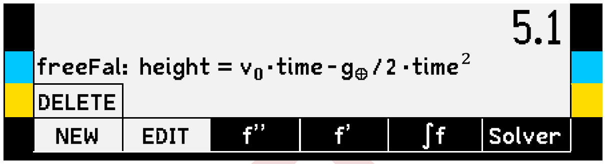
Carbon-14 Dating: Wood on the outer surface of a giant sequoia tree exchanges carbon with its environment. The radioactivity of this wood is 15.3 counts per minute per gram of carbon. A sample of wood from the center of

[269] The equation $h_0 + v_0\,t - \frac{1}{2}\,g\,t^2 = 0$ was solved here actually. Its exact physical solution is $t = \left(v_0 + \sqrt{v_0^2 + 2\,g\,h_0}\right)/g$. Take care that $(v_0^2 + 2\,g\,h_0)$ stays positive here – thrown from the bottom of a deep pit, the ball may never reach the ground outside. See the *IOI* for the contents of the other *stack registers*. Note you can as well use SLVQ for solving **freeFal**.

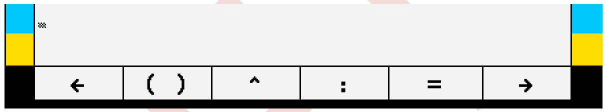
the tree yields 10.9 counts per minute per gram of carbon. The rate constant for the radioactive form of carbon, ^{14}C, is 1.20×10^{-4}. How old is the tree? What is the half-life of ^{14}C?

Solution (assuming you continue directly after solving previous problem): Exit previous equation and enter a new one for radioactive decay:

EXIT



NEW



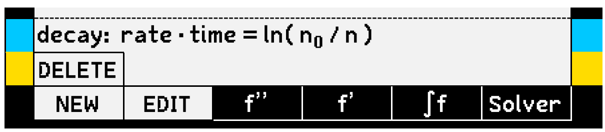
D E C A Y : R A T E × T I M E = L N ( ) N ↓ 0 / N

Press [f] e^x to enter e^( ), [f] √x to get √( ), and |x| to insert | |. All other functions like ln, lg, sin, arccos, tanh, etc. shall be either

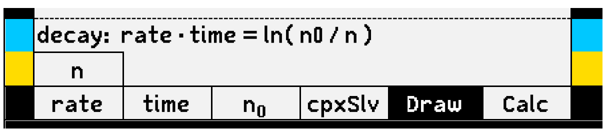
- picked from CATALOG'FCNS or
- called by typing their *names* as spelled in the *IOI*, trailed by ( ). Take care: e.g., SIN or LN will not be read as a function for obvious reasons.

Specify the argument within the parentheses (or bars) like above. This applies to dyadic operations by analogy – their arguments shall be separated by a colon (like COMB(x:y)). Leave the parentheses by →, if applicable.

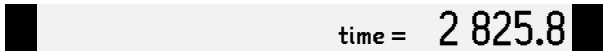
ENTER↑



Solver



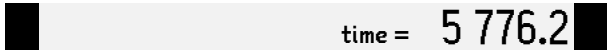
1.2 E +/- **4** rate **15.3** n_0 **10.9** n time



This is the computed age of this tree in years.

Now, calculate the half-life of ^{14}C; this is the time needed for half the material present to decay; thus, enter

2 n_0 **1** **n** **time**



Since the rate constant was given with 1% accuracy in the setting above, your calculatory result for the half-life can't be any better, i.e. (5776 ± 58) *years*; this is consistent with today's 'official' value of (5 730 ± 40) *years*.

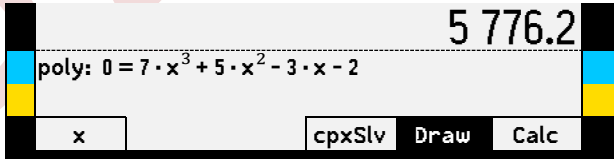
One more **example**:

Find the roots of $7x^3 + 5x^2 - 3x - 2 = 0$.[270]

Solution:

1. Enter the equation as demonstrated above.
2. Make this equation the *current equation* and press **Solver**.

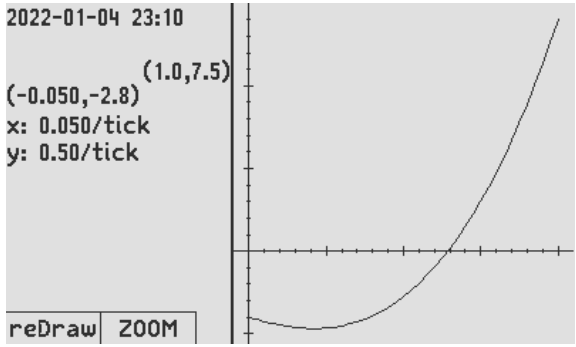
   You will see:



3. Optionally enter 1 or 2 initial guesses for the unknown like

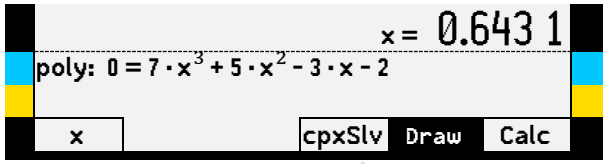
   0 **x** **1** **x** .

   Press **Draw** for drawing the equation between these 2 values. Counting the minor ticks you can estimate this function having a root at about 0.64. Let's see – press **EXIT** and you will get the previous screen again.

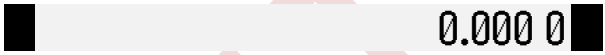


[270] This problem could be solved using **SLVC** as well. Feel free to compare the results.

4. Set the display format to **FIX 4**, then leave DISP by **EXIT**. Press **F1** for the unknown **x** (now without any numeric input heading), and your *WP43* will solve the equation for this *variable* and return its value in **X**: [271]



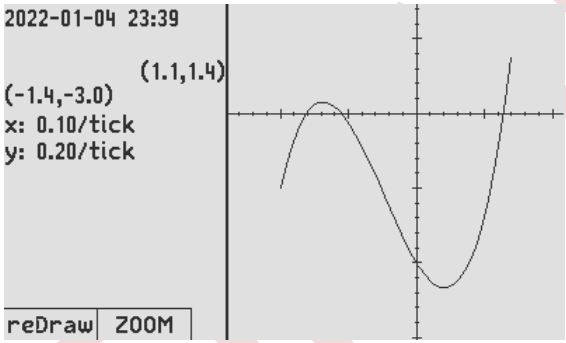
5. If you want to crosscheck you can press

Calc returning 

for the function value at this location, confirming the result of the *Solver*.[272]

6. There may be up to 2 more roots – you guess they are on the left side:

a. Enter 2 new initial guesses for the unknown like **–1** **x** **.7** **x** and check the function with **Draw**; use **ZOOM** to get a convenient tick size. You can read from the plot that there will be one more root close to −0.55 and another one at −0.8. Then press **EXIT** to close the diagram.



b. Overwrite the guesses with **–1** **x** **–.6** **x**, then press **x** once more, and you will get:

c. Enter 2 new initial guesses like **–.7** **x** **–.4** **x**. Press **x** once more and you will get:

---

[271] If you want the *Solver* looking for *complex* roots as well, press **cpxSlv** instead of the last **x** here. In any case, the *Solver* will also return values in **Y**, **Z**, and **T** – see **SOLVE** in the *IOI*.

[272] If you wish to compute the function at any other location – for whatever reason – you shall enter the location *x* using **x** before calling **Calc**.

Feel free to look into *Section 5* of the *HP-27S OM* for more about interactive solving of algebraic equations.

Interactive Solving Expressions Stored in Programs

Instead of operating on an equation as described in previous chapters, your *WP43* can also solve an expression ***f*** stored in a program. Then,

1. Write a program for ***f***.
2. Press ADV SOLVE and select the program you want solved.
3. Enter values for all known *variables* of ***f***.
4. Let your *WP43* compute the unknown *variable*.
5. Leave the *Solver*.

We will go through this step by step:

1. Write a program for ***f***:
   - It shall begin with a global label.
   - It must define all *variables* required for calculating ***f***.
   - It shall be as efficient as possible since it is going to be executed many times.

   For interactive solving, we recommend proceeding as follows for this program: From its 2nd step on, *menu variables* shall be declared using MVAR instructions (→ *240*) covering all *variables* of ***f***; no trailing VARMNU here! The subsequent body of the routine shall then evaluate ***f*** recalling these *variables*. For a *Solver* routine, the original expression shall be rewritten in a way that ***f*** = **0** is fulfilled.

   Example (with FIX 1 chosen):
   Let's return to the equation we dealt with in the last 2 chapters:

   $$height = h_0 + v_0 \cdot time - g_\oplus / 2 \cdot time^2$$

   This is easily rewritten: $h_0 + v_0 \cdot time - g_\oplus / 2 \cdot time^2 - height = 0$

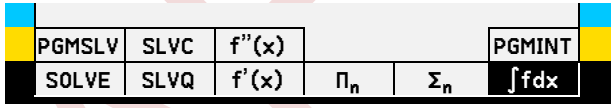
   So the required program might look like this:

```
LBL 'FreeF'
  MVAR 'height'
  MVAR 'h0'
  MVAR 'v0'
  MVAR 'time'
  # g⊕
  -2
  /
  RCL× 'time'
  RCL+ 'v0'
  RCL× 'time'
  RCL+ 'h0'
  RCL- 'height'
  RTN
```

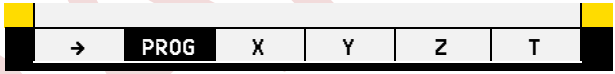
(no trailing VARMNU here!)
take this out of CONST.

now we have got ***f***.

2. Press [ADV] :



Choose SOLVE :



Press PROG and pick the proper program for ***f*** (here FreeF). You will get the corresponding *menu* of *variables*, i.e. here:

3. Enter values for all known *variables* of *f* and (optionally) 1 or 2 guesses for the unknown.

In our case, we may just take the values we know from above:

0 height **50** h_0 **15** v_0 **5** time **10** time

4. Let your *WP43* compute the unknown *variable*.

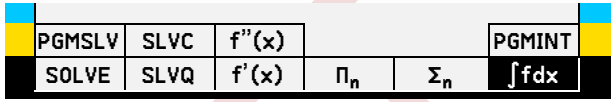
Press time once more but without a heading numeric entry now. Your *WP43* will return time = 5.1 as you have expected (→ 269).

5. Leave the *Solver*.

Pressing [EXIT] will return to the top *view* of ADV we know it from step 2.

Using the Solver in a Program

For using the *Solver* in a programs, it has to be told what you did tell it interactively so far. Calling ADV in *PEM*, you will need **PGMSLV** not used before:



PGMSLV is to specify the program calculating ***f***.
This step must be found in your program before **SOLVE** is called.

Furthermore, define the necessary *variables* in advance and load the known values, e.g., using **STO**. Eventually, the unknown *variable* must be specified calling **SOLVE**.

Example:

Let's return to the equation we dealt with in the last chapters. So the required program for *f* might look like this (like previous program but without the MVAR steps if the *variables* required were defined before):

```
LBL 'FreeFp'
  # g⊕
  -2
  /
  RCL× 'time'
  RCL+ 'v0'
  RCL× 'time'
  RCL- 'height'
  RTN
```

take this out of CONST.

now we have got ***f***.

The program 1 level above could contain a snippet like this:

```
...
PGMSLV 'FreeFp'
SOLVE 'time'
VIEW 'time'
...
```

specify the function to be solved.
solve for time.
display the solution.

Before starting this program (let's call it **L**), fill the *variables* of the equation to be solved, e.g., with the start values known from above:

0 STO VAR height 50 STO VAR h_0 15 STO VAR v_0

Optionally fill the unknown with a 1st guess, e.g., with **5** as we specified above (a 2nd guess will be taken from **X**):

5 STO VAR time

Alternatively, you can put these **STO** steps in **L** at any place before **SOLVE** is called. Or enter **MVAR** steps for all required *variables* as we did for **FreeF** above. Or mix both methods.

Call **L** via XEQ L and you will get time = 5.1 (→ 269).

Here is another example, taken from the *AFH*:

During the winter of '78, Arctic explorer *Jean-Claude Coulerre*, isolated at his frozen camp in the far north, began scanning the southern horizon in anticipation of the sun's reappearance. *Coulerre* knew that the sun would not be visible to him until early March, when it reached a declination of 5° 18' S. On what day and time in March was the chilly explorer's vigil rewarded?

The time in March when the sun reached 5° 18' S declination can be computed by solving the following equation for ***t***:

$$D = a_4\,t^4 + a_3\,t^3 + a_2\,t^2 + a_1\,t + a_0$$

where ***D*** is the declination in degrees, ***t*** is the time in days from the beginning of the month, and

$$a_4 = 4.2725 \times 10^{-8}$$
$$a_3 = -1.9931 \times 10^{-5}$$
$$a_2 = 1.0229 \times 10^{-3}$$
$$a_1 = 3.7680 \times 10^{-1}$$
$$a_0 = -8.1806.$$

This equation is valid for $1 \le t < 32$, representing March, 1978.

Solution:

Note that Southern latitudes are expressed as negative numbers for calculation purposes. Hence, *Coulerre* should solve

$$0 = a_4\,t^4 + a_3\,t^3 + a_2\,t^2 + a_1\,t + a_0 + 5°18'$$

5.18 [d.ms] [.d] **8.1806** [−] returns −2.8806

Using the *Horner scheme* (→ 55), we can write a little program for this equation most easily:

[GTO] [.] [.] [P/R]
[LBL] [α] SUNRISE
[RCL] [0] [4] [×]
[RCL] [0] [3] [+] [×]
[RCL] [0] [2] [+] [×]
[RCL] [0] [1] [+] [×]
[RCL] [0] [0] [+]
[RTN]

Store **−2.8806** in **R00** and the other constants in **R01 – R04**. Furthermore, enter two guesses

1 [ENTER↑] **32** and call
[XEQ] SUNRISE which will return 7.5137
[#] [F] separates the fractional days 0.5137
24 [×] [h.ms] returns 12:19:45

So *Coulerre* will have seen the Sun at 1978-03-07 12:19:45 *UTC*.

Eventually turn to *ReMA J* for more information about automatic root finding and some caveats.

Integration

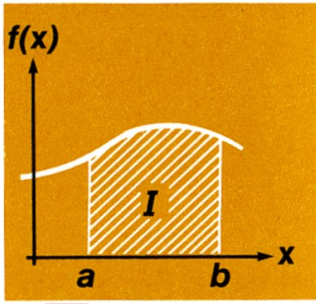
Numeric Integration of Arbitrary Algebraic Expressions

Many problems[273] in mathematics, science, and engineering require calculating the *definite integral* of a function. If the function is denoted by *f(x)* and the interval of integration is *a* to *b*, the integral can be expressed

[273] Quoted from the *HP-34C OHPG*, pp. 202f (this was the 1st pocket calculator worldwide featuring numeric integration, thanks to professor *William Kahan*, → *ReM*). – If you know the analytical solution of an integral, however, you should apply this knowledge.

mathematically as

$$I = \int_a^b f(x)\ dx$$



The quantity ***I*** can be interpreted geometrically as the area of a region bounded by the graph of *f(x)*, the *x*-axis, and the limits $x = a$ and $x = b$ (provided $f(x) \geq 0$ throughout this interval).

When an integral is difficult or impossible to evaluate by analytical methods, it can be calculated using numerical techniques frequently still. You can compute (almost) arbitrary definite integrals numerically using the command ∫ on your *WP43*.

Example (from the *HP-34C OHPG*):

Certain problems in physics and engineering require calculating *Bessel functions*. The *Bessel function of the 1st kind and order 0* can be expressed as

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} cos[x \sin(t)]\ dt$$

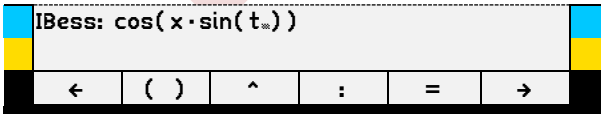
Solution:

This is calculated in *radians*, thus enter MODE RAD and press EQN.

Above function is not in the equation list yet. So, press NEW and start entering the integrand:

▲ I B ▼ E S S : 

Continue with C O S ( ) X × S I N ( ) T

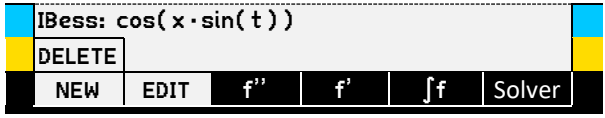


Note you did not have to step into any parentheses, your *WP43* put you at the right place automatically.[274]

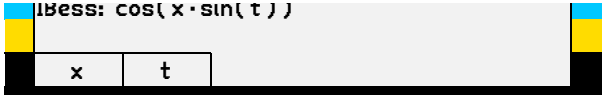
Close and store this function by pressing ENTER↑. The *menu* will return to the

[274] You will have noticed that **IBess** is not an equation but just one side of it. To keep the system lean, such 'functions' are listed under EQN nevertheless.

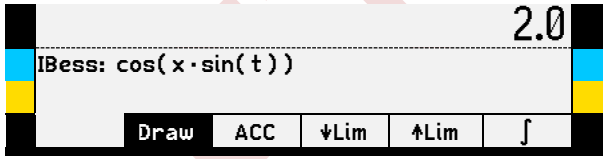
previous one:



Press **∫f** now. Your *WP43* will then check the current equation (→ 266ff) for syntax errors (missing operators, illegal *variable names*, misspelled functions, etc.). It will then return a *menu* of all applicable *variables*: [275]



Now, you can enter a value for each *variable* you know (i.e. for each *integration constant*) by pressing the respective *softkey* trailing the value, e.g., **2** **x**.[276] Since there is only 1 variable left here (***t***), this will be the *variable of integration*. The *menu* changed:



Even your *WP43* can't compute an integral exactly, it approximates its value numerically. The accuracy of this approximation depends on the accuracy of the integrand's function itself as calculated by your program. This is affected by round-off error in the calculator and also by the accuracies of the *integration constants* you specified.

ACC is a *real* that defines the relative error of the integration. With ***ACC*** = **0.001**, for instance, you can be sure that

$$\left|\frac{v_T - v_C}{v_C}\right| \leq 0.001$$

(with v_T being the true value and v_C the computed value of the integrand) at any point between **↓Lim** and **↑Lim**.

We want to see the integrand accurate to 3 decimals. Thus enter

.001 **ACC** for the accuracy of computation as specified above,

0 **↓Lim** for the lower and

π **↑Lim** for the upper integration limit, and start integrating by pressing **∫**.

After a short while, your *WP43* will return this:[277]

[275] Such a *menu* will not be displayed if just 1 *variable* is present in the integrand.

[276] For recalling such an *integration constant*, just press **RCL** **VAR** and choose the respecttive *softkey*.

Don't forget to divide this by π to get the correct value for $J_0(2)$:

DISP FIX 3 π /

Enter other values for ***x*** and integrate again to get $J_0(x)$ at other locations. [278]

Interactive Integrating Expressions Stored in Programs

Instead of operating on an 'equation' as described in previous chapter, your *WP43* can also integrate an expression ***f*** stored in a program. Therefore, you shall

1. write a program for ***f***.
2. press ADV ∫fdx.
3. enter values for all known *variables* (*integration constants*) of ***f***, for ***ACC***, and for the integration limits. Select the *variable of integration*.
4. Let your *WP43* compute the definite integral specified.

We will go through this step by step: [279]

1. Write a program for ***f***:
   - It shall begin with a global label.
   - It shall define all *variables* required for calculating ***f***.
   - It shall be as efficient as possible since it is going to be executed many times.

   We recommend proceeding as follows: From the 2nd step of this program on, *menu variables* shall be declared using MVAR instructions

[277] The value ***y*** above is the estimated accuracy of the integral.

[278] ∫≈ vanishes with ⬜ like every *TI* vanishes with the next keystroke.

We did not include the division by ***π*** in our function **IBess** since this is evaluated many times during the integration process; thus, the fewer steps **IBess** contains the faster the result can be returned.

You may have noticed a function **$J_y(x)$** is implemented in your *WP43* directly returning the value of the *Bessel function of 1st kind and order* ***y*** at location ***x***. Feel free to compare the results.

[279] This follows closely the procedure as described for the *Solver* above.

(→ 240) covering all *variables* of ***f***. The subsequent body of the routine shall then evaluate ***f*** recalling these *variables*.

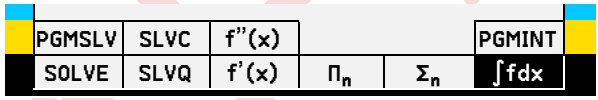
Example:

Let's return to the integrand we dealt with in the last chapter. Then the required program for ***f*** might look like this:

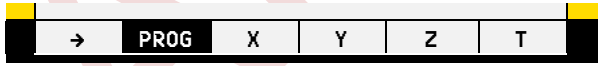
```
LBL 'IBessI'
   MVAR 'x'
   MVAR 't'
   RCL 't'
   sin
   RCL× 'x'
   cos
   RTN
```

now we have got ***f***.

2. Press ADV :



Choose ∫fdx :



Press PROG and pick the proper program for ***f*** (here IBessI). You will get the corresponding *menu* of *variables*:

3. Enter values for all known *variables* (*integration constants*) of ***f*** and select the *variable of integration*.

In our case, we may take the value we know from above: **2** **x** **t** . So ***t*** will be the *variable of integration*. The *menu* will change now:

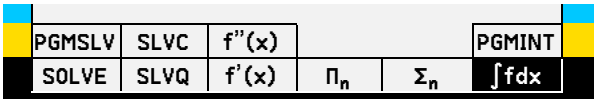
We enter (like in previous chapter) **.001** ACC **0** ↓Lim π ↑Lim

4. Let your *WP43* compute the definite integral specified.

Press ∫ to integrate with all the parameters as chosen, and your *WP43* will return ∫≈ 0.704 as you might have expected (→ previous chapter). Divide by **π** to get the value for $J_0(2)$ as above.

Using the Integrator in a Program

For using the *Integrator* in programs, it has to be told what you sent to it interactively so far. Calling ADV in *PEM*, you will need **PGMINT** not used so far:



It is for specifying the program calculating ***f*** before the integration is called. You shall define the necessary *variables* in advance. Use **STO** to load them with the known values before integrating. Then call the *menu* **∫fdx**:

The integration limits and the requested accuracy shall be stored before integrating. Eventually, the *variable of integration* must be specified calling ∫.

Example:

Let's return to the integrand we dealt with in the last 2 chapters. So the required program for *f* might look like this:

```
LBL 'IBessP'
  RCL 't'
  sin
  RCL× 'x'
  cos
  RTN
```

equals **IBessI** above but without the **MVAR** steps.

now we have got *f*.

The program one level above could contain a section looking like this:

```
LBL 'IntP'
  ...
  PGMINT 'IBessP'
  0
  STO '↓Lim'
  π
  STO '↑Lim'
  0.001
  STO 'ACC'
  ∫fd 't'
  VIEW X
  ...
  RTN
```

specifying the function to be integrated.

integrate over time.

display the solution.

Before starting this program, load the *variables* staying constant under integration, e.g., with the start values known from above:

2 [STO] [VAR] [x].

[XEQ] [PROG] [IntP] and you will see ∫≈ 0.704 as you may have expected (→ previous chapter). Divide by π to get $J_0(2)$ as above.

We recommend checking the integrand carefully for irregularities. Feel free to look up *ReMA J* for more information about automatic numeric integration, some caveats, and hints.

Differentiation

Differentiating Arbitrary Algebraic Equations

There are 2 commands provided returning the values of the first 2 derivatives of the function $f(x)$ at position x. This function $f(x)$ can be specified in an equation.

f'(x) returns the 1st derivative. For computing it, ...

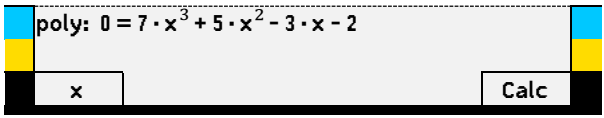
1. **f'(x)** will first look for a user routine labeled **'δx'** (or '**δX**', '**Δx**', or '**ΔX**', in this order[280]), returning a fixed step size ***dx*** in **X**. If that routine is not provided, ***dx*** = **0.1** is set for default.
2. **f'(x)** will fill all *stack registers* with x and call $f(x)$ then. It will evaluate $f(x)$ at 10 points equally spaced in the interval $x \pm 5\,dx$ (if you expect any irregularities therein, change ***dx*** to exclude them).
3. On return, the 1st derivative will be in *stack register* **X**, while **Y**, **Z**, and **T** will be clear and the position ***x*** will be in **L**.

Example (with **SCI 3** set):
Take the equation **poly:** again (→ 271f). Instead of checking 2 function values left and right of the root you could check the slope at the root just once.

Solution:
For each of the 3 roots of **poly:**, calculate the root first, then the 1st derivative of **poly:** at that point:

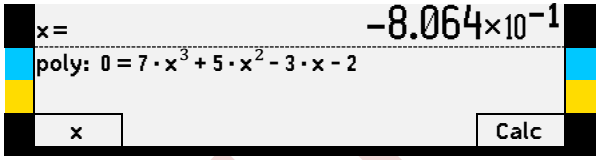
[280] There is a function **Δx** but that doesn't interfere with this global label.

1. Press EQN, make **poly:** the *current equation*, and press Solver. You will see then:

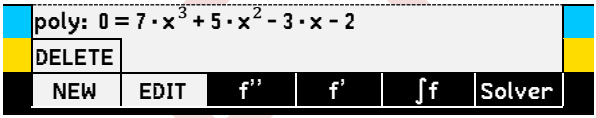


2. Find the 1st (leftmost) root as shown above:

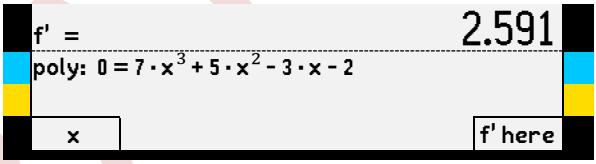
−2 x **−1** x x



Press EXIT :



Press f' :

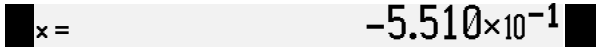


Note that **f'** returns the value of the 1st derivative at this location immediately since **poly:** features only 1 *variable*; else **f'** would have needed your input via the *softkeys* displayed and pressing **f' here** thereafter. So the slope of **poly:** at ***x*** = −0.8064 is 2.591. Get the slopes at the 2 other root positions the same way:

EXIT returns to the top *view* of EQN as above.

3. Solver

Find the 2nd root of **poly:** **−.7** x **0** x x



EXIT returns to the top *view* of EQN as above.

f' returns

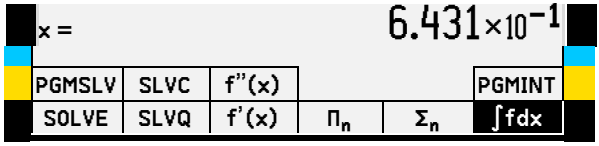
EXIT returns to the top *view* of EQN as above.

4. Solver

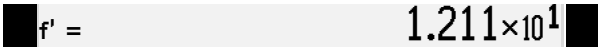
Find the 3rd (rightmost) root of **poly:**

0 x 1 x x

EXIT returns to the top *view* of EQN as above.



f' returns



So the slope of **poly:** at $x = -0.8064$ is 2.591, at $x = -0.5510$ it is −2.134, and at $x = 0.6431$ it is 12.11; so the sequence of slopes is positive, negative, and positive as expected.

f''(x) works in the same way, computing the 2nd derivative of the function.

Interactive Differentiating Expressions Stored in Programs

Instead of operating on an equation as described in previous chapter, your *WP43* can also derive an expression ***f(x)*** stored in a program. Then,

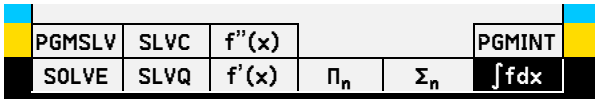
1. Write a program for ***f(x)***. It must begin with a global label. For interactive derivation, we recommend proceeding as follows: [281]

   From the 2nd step of this program on, *menu variables* shall be declared using **MVAR** instructions (→ *240*) covering all *variables* of ***f(x)***. The subsequent body of the routine shall then evaluate ***f(x)*** recalling these *variables*.

2. Optionally, write another program labeled '**δx**' (→ *283*).

3. Enter values for all known *variables* (*derivation constants*) of ***f(x)***. Put the location where you want to know the derivative into **X**.

4. Press ADV :



5. Press **f'(x)** or **f''(x)**. You will get:

[281] This follows loosely the procedures as described for the *Solver* and *Integrator*.

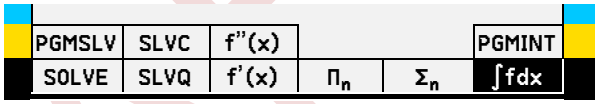
6. Choose **PROG** to pick the label of the program containing the function $f(x)$ (or enter its label directly as described on p. 225).

7. Let your *WP43* compute the 1st or 2nd derivative at location x.

Computing Derivatives in a Program

For computing derivatives in programs, proceed as demonstrated in previous chapter. Just remember you should omit the **MVAR** instructions in your program calculating $f(x)$; instead, define the necessary *variables* in advance and load them with the known values using **STO**.

Call ADV in *PEM*:



Press **f'(x)** (or **f''(x)**): You will be asked for the label of your program calculating $f(x)$ – type it or pick it from the list as explained in steps 5 and 6 in previous chapter. Your *WP43* will compute the requested derivative for you in this program step.

Nesting Advanced Operations

You can nest **SLV**, **∫**, **f'(x)**, **f''(x)**, **Σ**, and **Π** in routines to any depth as far as memory allows and your patience and power last. Solving nested advanced operations may require many, many calculations to be executed. When dealing with such problems, we recommend

a) checking the innermost routines for minimum execution time and

b) connecting your *WP43* to an *USB* outlet for supplying external power.

Example:

Light is observed to be diffracted when passing through very small holes, an effect most obvious with laser light. Its intensity behind a circular hole is

$$I(r) = I_0 \times \left(\frac{J_1(2\pi r)}{\pi r}\right)^2 \quad \text{with} \quad J_1(x) = \frac{1}{\pi}\int_0^{\pi} cos[\,t - x\,sin(t)]\,dt$$

at distance ***r*** from the center of the beam, with $J_1(x)$ being the *Bessel function of the 1st kind and order 1* (→ 277). Find the first 3 roots of the intensity (i.e. the radii where no light will be observed).

Solution:

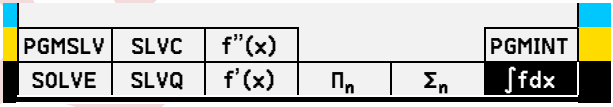
1. Write a little program for the integrand $f(t) = cos[\,t - x\,sin(t)]$:

```
LBL 'J1'
   RCL 't'
   ENTER↑
   sin
   RCL× 'x'
   -
   cos
END
```

$sin(t)$
$x\,sin(t)$
$t - x\,sin(t)$
$cos[\,t - x\,sin(t)]$

2. Write another little program for calculating the intensity $\boldsymbol{I(r)}$. Actually, to determine its roots you just need solving $\boldsymbol{J_1(2\pi r)}$. And remember ADV called in *PEM* displays:

```
LBL 'II'
   MVAR 'r'
   1.×10^-5
   STO 'ACC'
   CLX
   STO '↓LIM'
   π
   STO '↑LIM'
   RCL× 'r'
   STO+ X
   STO 'x'
   PGMINT 'J1'
   ∫fd 't'
END
```



→ 279.

πr
$2\pi r$

specifies the program of the integrand (see the *menu*).
computes $\pi \times J_1(2\pi r)$ and returns it in **X**.

3. Enter DISP SCI 3
   MODE RAD
   ADV SOLVE

4. Enter **PROG** (II).
   You will see

5. Enter **.1** **r** **1** **r** **r**.
   You will get 6.098×10^{-1} after some time.[282]

6. Enter **1** **r** **1.5** **r** **r** and you will get 1.117 .

7. Enter **1.5** **r** **2** **r** **r** and you will get 1.619 .

You will find further instructions and examples in *HP-42S RPN Scientific Programming Examples and Techniques*. Despite its title, this manual also comprises significant material about the *Solver*, *Integrator*, *matrices*, and statistics. Watch the differences between *HP-42S* and *WP43* though.

Just in case: If you find the *Solver* or *Integrator* consuming too much time for any computation whatsoever, feel free to abort it by pressing **EXIT**.

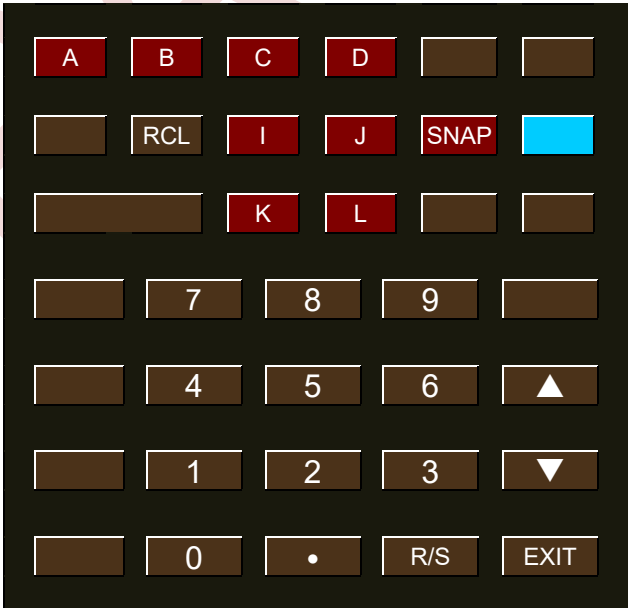
[282] The algorithm finds this root directly after some 15 s using the *WP43 Simulator* in *SDC* (→ *ReMA E*) on a *PC* running *Windows 10* or some 12 s with FASTFN.
This task will take 16 *minutes* on your *WP43* calculator in *SDC* powered by battery alone, 9.5 *minutes* with FASTFN, some 4 *minutes* with *USB* power supply but ~~FASTFN~~, and 3 *minutes* with *USB* and FASTFN. Look up FASTFN in *ReMS 1*.

SECTION 6: TWO BROWSERS, TWO APPLICATIONS, AND TWO SPECIAL MENUS

For monitoring free memory, used *registers* and *flags* easily, 2 *browsers* are provided (**RBR** and **STATUS**, see below). And there are 2 very useful applications: a **TIMER** (or stopwatch, → *292f*) and *The Time Value of Money (TVM)*, → *294ff*). Furthermore, 2 special *menus* providing constants and unit conversions will ease your path in science and engineering (→ *298ff*).

The Browsers RBR and STATUS

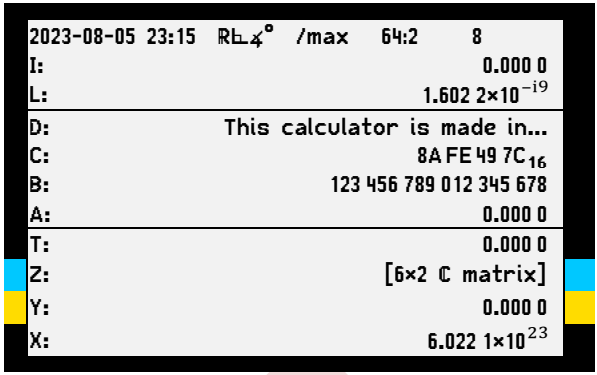
These 2 *browsers* may be called in all modes except *AIM*. They are not programmable. Some special keys and rules apply within these *browsers* as explained below.

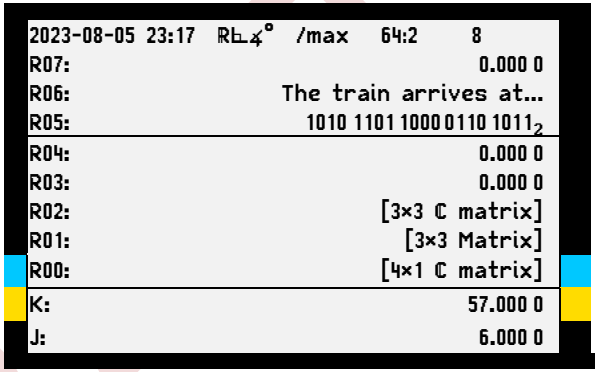


RBR browses all currently allocated *registers* and shows their contents or the space they are taking. See here its *virtual keyboard*; it looks similar to the one in *TAM* (→ *61*) but differs in detail.

The first display you will see after calling **RBR** covers *registers* **X** … **I** (contents will deviate on your screen – individual numbers will be shown

in the display format currently selected, while *text strings* may be abbreviated and *matrices* will be; fractions will be displayed with their decimal value):





▲ goes up the *stack*, continuing with the remaining lettered *registers*, then with **R00**, **R01**, etc. as shown below. Every fifth *register* is displayed overlined to guide the eye →

After **R99**, **X** will be shown again.

▼ works vice versa, going down. Both ▲ and ▼ cycle through the memory locations.

. turns to *local registers* if any are allocated to the current routine, starting with **R.00**; else (or another) . turns to *named variables* (incl. *system variables*), and one . more returns to the first screen of RBR as shown above. ▲ and ▼ work in each such view.

R/S toggles display to show the *Register* contents or the Space allocated to them (→ *ReMA B*).

A … D or I … L browse immediately to the corresponding *register*.

0 0 … 9 9 browse immediately to the corresponding (*global* or *local*) *register*. If no such *local register* is allocated, it returns to the first *global register*. If pressed while *named variables* are displayed, it jumps to the corresponding global *register*.

RCL in *RUM* recalls the content of the *register* displayed in the lowest row …
in *PEM* enters a corresponding step `RCL __`
} and leaves RBR.

SNAP writes a snapshot of the present **RBR** display onto the *FD* (→ *257*). Pressing ■ suffices for calling **SNAP** here.

EXIT leaves the register browser RBR.

FLAG **STATUS** displays the amount of free memory available and the *flags* set.[283] Local *user flags* will only be displayed if at least 1 *local register* is allocated Some global settings and all *system flags* set are displayed in the bottom rows.


```
2024-01-17 00:05 CL4r /max  64:2c   8

40556 bytes free in RAM.
Global user flags set:
0 2 11 A C D L I
No local flags and registers are allocated.

RM=½E  SDIGS=34  ULP of reg X = 10^-6176
ALLENG ASLIFT AUTOFF CARRY CPXRES DECIM.
DENANY ENDPMT LEAD.0 MULTx NUM.IN PROPFR
SPCRES SSIZE8 TDM24 YMD
```

▲ and ▼ will toggle between *views* if more *flags* are set (note only the top view is shown if **STATUS** is called in a program).

SNAP writes a snapshot of the present **STATUS** display onto the *FD* (→ *257*). Pressing ■ suffices for calling **SNAP** here.

EXIT leaves **STATUS**.

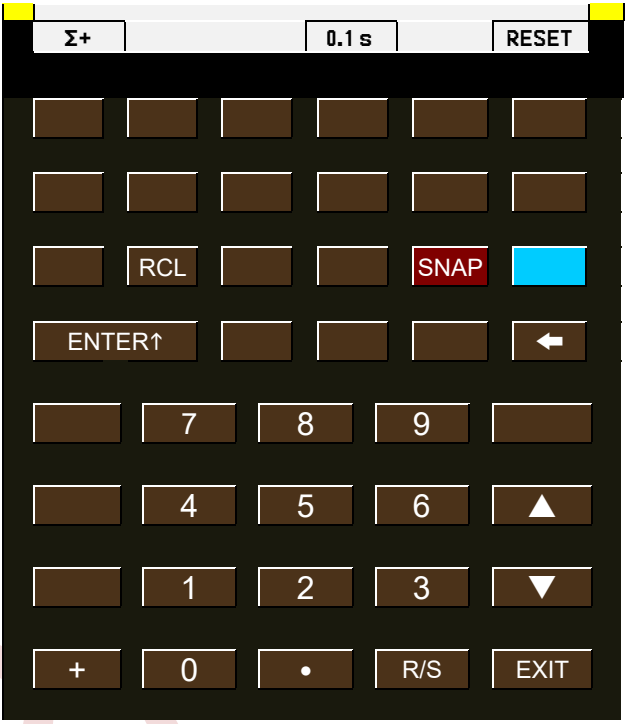
These are all the keys working in **STATUS**.[284]

[283] … inspired by STATUS on *WP 34S*.

[284] The font browser **FBR** shares the same virtual keyboard.

The Timer Application

Your *WP43* provides a timer, a.k.a. stopwatch. See its *virtual keyboard* here.



Press [⏱] to start it.

Then the top numeric row will be replaced by 0:00:00.0 [00], unless the timer was running before already (then the accumulated run time will replace zero here). Thereafter, pressing …

[R/S] starts or stops the timer without changing its value. [285]

[0.1 s] toggles displaying tenths of *seconds* (*startup default* = 'display').

[RESET] or [←] resets the timer to zero without changing its status (running or stopped). It deletes the *total time* if applicable (see [.]).
(Note this is not the global **RESET** command, it just shares its *label*.)

[Σ+] adds the present (total) time value (converted to decimal hours, i.e. a *real* ***y***) and the data point number (***x*** = 1, 2, 3, …) to STATS.

---

[285] The *WP43* timer works as in *WP 34S* – just the display differs. With respect to the timer of the vintage *HP-55*, there are 2 differences:

1. Start times (also decimal hours) are supported by **RCL** on your *WP43* (see the page after next). It will not take ***x*** (as it is when calling **TIMER**) as start time.
2. Your *WP43* will display tenths instead of hundredth of *seconds*. Reaction times of the hardware don't allow for more precision anyway.

[0] [0] … [9] [9] sets the *current register address* (*CRA*). Its *startup default* is 00. The *CRA* is displayed between rectangular brackets like

27:31:55.6 [08]

You shall enter both digits always.[286] Since the *CRA* may be incremented automatically, set it so there will be sufficient unused registers following for your records.

[ENTER↑] stores the present (lap) time value in the *CRA* without changing the timer status or value, then increments the *CRA* by 1.

[.] combines [ENTER↑] and ⬅ in 1 keystroke, resetting the (lap) time. Unless already displayed, the *Total time* counted since the last explicit press of ⬅ or [RESET] will be shown like

2:04:15ᵀ 0:02:29 [06]

or 26:02:31.7ᵀ 10:00:49.6 [11] and updated from then on.

[.] allows for automatic recording lap times in a series of *registers*.

➡ The *total time* is volatile – it will vanish and be reset when ⬅ or [RESET] is pressed (→ above).

[+] combines the functionalities of [Σ+], [ENTER↑], and ⬅ in 1 keystroke. This allows for recording lap time as well as *total time* for later offline analysis, e.g., for computing the *arithmetic mean* and *standard deviation* of lap times after leaving TIMER.

▲ (or ▼) increments (or decrements) the *current register address* by 1. ▼ allows for overwriting the last (lap) time stored.

[RCL] ***nn*** recalls ***r nn*** and adds it to the lap time recorded without changing the timer status. The *total time* is not affected by this addition.

[SNAP] writes a snapshot of the present display onto the *FD* (→ *257*). Pressing 🟧 suffices for calling **SNAP** here.

[EXIT] leaves the timer without changing its status. So the time string in top numeric row will vanish but the timer may continue incrementing

[286] On the *HP-55*, a single digit sufficed since only 10 *registers* were featured for this purpose. And there was no automatic *CRA* increment.

in background (indicated by ⏱ in the *status bar*) until ...

a) you will return to the timer and stop it explicitly by R/S or ...

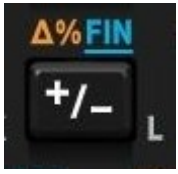
b) you will turn off your *WP43* manually (it will not turn off automatically with the timer running in foreground [287]) or ...

c) your *WP43* will shut down due to low battery (*USB* power supply is recommended for long timing).

No other keys will work in TIMER. The time counted can be indicated up to 99:59:59.9 (more than 4 days) – then the display will overflow. This applies to the *total time* as well.

For subtracting split times you have to leave the TIMER application. – TIMER is not programmable.

The Time Value of Money (TVM)



TVM is a financial application field proven for over 5 decades, computing, e.g., the future value (***FV***) of

1. a repeated investment or
2. a regular down-payment for a credit

based on its present value (***PV***), its interest rate per annum in percent (***i%/a***), the required payment per period (***PMT***), the number of periods per annum (***per/a***), and the total number of payment periods (n_{PER}).[288] Similar financial problems will often occur to technical people

[287] With the timer running in background, your *WP43* may turn 'off' but will continue incrementing nevertheless! Care for reliable power supply if you are up to record longer timings than 10 *minutes*, else you may easily deplete the battery.

[288] *TVM* was launched with *HP*'s very first financial '*problem solver*', the *HP-80* of 1972, and was implemented on each and every *HP business/financial calculator* thereafter. An early advertising sheet promised 'you can improve and simplify your time-and-money management' applying *TVM* quickly. 'Random-entry financial keys let you key in problems in any order. And you can change any number at any time.'

All this was true for certain – it was before any spreadsheet software became available. It may even hold nowadays to some extent still since hardly any modern financial tool solving such problems is more compact, handier, and booting faster than a pocket calculator featuring *TVM*. Note also the lasting success of the *HP-12C*, the 'gold

as well, so we included a *menu* FIN'TVM in your *WP43*.

For your information, the general formula for such problems reads

$$0 = PV + (1 + i\,p)\,\frac{PMT}{i}\,[1 - (1+i)^{-n_{PER}}] + FV\,(1+i)^{-n_{PER}}$$

with the deduced parameter $i = \frac{I\%/period}{100} = \frac{I\%/year}{100} \,/\, \frac{periods}{year}$ and the binary switch $\boldsymbol{p}$:[289] If payments occur at the…

- end of each period then $\boldsymbol{p = 0}$ (choose **End** in TVM).
- beginning of each period then $\boldsymbol{p = 1}$ (choose **Begin** in TVM).

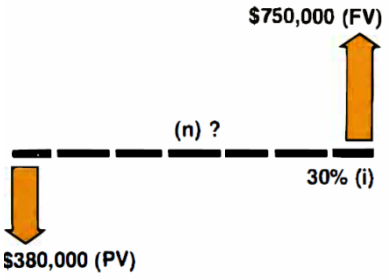
TVM uses the *cash flow convention* that cash outlays (payments) are input as negative, and cash incomes are entered as positive.[290]

The present value ***PV*** always occurs at the beginning of the 1st period. It can also be an initial cash flow or a discounted value of a series of future cash flows.

The future value ***FV*** is always meant to occur at the end of the $\boldsymbol{n_{PER}}^{th}$ period. It can also be a final cash flow or a compounded value of a series of cash flows.

Example for calculating the number of periods:[291]

A potential development site currently appraised at $380 000 appreciates at 30% per year. If this rate continues, how many years will it be before this land is worth $750 000?



standard' of financial calculators, continuously made and sold almost unchanged since 1981 (→ *ReMA J*).

TNG: TVM = Zeitwert des Geldes (offizielle Übersetzung von *HP*).

289 For the notation of the *TVM* formula, remember f. 61 on p. 50.

290 This convention was introduced with the *HP-92* in 1977. The *HP-42S OM* states:
The correct *sign* (positive or negative) for *TVM* numbers is essential. The calculations will make sense only if you consistently show payments out as negative and payments in (receipts) as positive. Perform a calculation from the point of view of either the lender (investor) or the borrower, but not both!

291 The examples in this chapter are quoted from the *HP-27 OH*.

Solution:

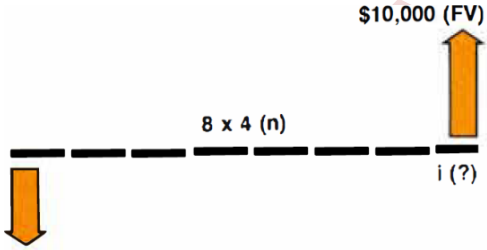
DISP FIX 2 will suffice for all financial problems here. Then all you have to do is key in the known parameters and boundary conditions:

Begin					End
RCL n_p	i%/a	per/a	PV	PMT	RCL FV
n_{PER}	i%/a	per/a	PV	PMT	FV

FIN TVM Begin

−380000 PV	−380 000.00	present value paid,
30 i%/a	30.00	% interest rate per year,
0 PMT	0.00	no payments,
1 per/a	1.00	default.
750000 FV	750 000.00	But how long does it take to reach this?
n_{PER}	2.59	*years*.[292]

Example finding the necessary interest rate for compounded amounts:



What annual interest rate must be obtained to amass $10 000 in 8 years on an investment of $6 000, with quarterly compounding? (Continue with as many settings of previous case as possible.)

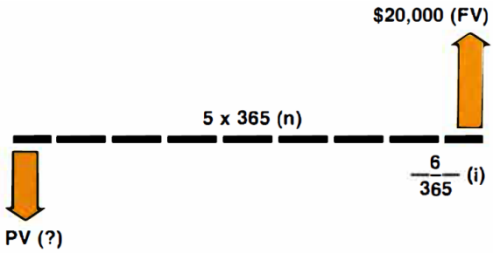
Solution (***p*** = **1** and ***PMT*** = **0** are taken over from above):

−6000 PV	−6 000.00	paid,
4 per/a	4.00	quarters,
10000 FV	10 000.00	target amount,
8 ENTER↑ 4 × n_{PER}	32.00	periods. Now, we need ...
i%/a	6.44	% interest rate per year to reach this.

TN: Enjoy the typical amounts of money and interest rates of a time long passed – and learn some financial (American) English. Furthermore, get used to an amount of 1234.56 US$ reading $1234.56 in the *USA* (even crying with dB95 ... emmh ... 95 dB will not help, alas). Apparently, not every unit is a unit there.

[292] *TN*: Generally, you shall enter values for the switch ***p*** and the 5 *variables* you know of above formula; then *TVM* will solve for the unknown 6th variable, pressed without a heading input value. *TVM* leaves ***p*** and above 5 *variables* unchanged, so you may continue solving the next problem with (some of) them.

Example for finding the present value of a compounded amount:

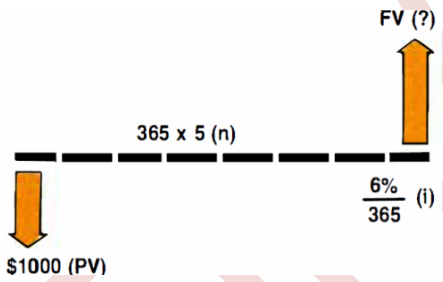


In 5 years when your son starts college, you will need $20 000. You deposit a lump sum in a certificate account that earns 6% compounded daily. How much do you need to deposit today to reach that goal? (Continue ...)

Solution (***p*** = **1** and ***PMT*** = 0 are taken over from above):

20000 FV	20 000.00	future value,
6 i%/a	6.00	% interest rate per year,
365 per/a	365.00	days per year,
5 ENTER↑ **365** × n_{PER}	1 825.00	periods. Now, we need...
PV	−14 816.73	to be deposited.

Example for finding the future value of a compounded amount:



The local trading post manager opened up a savings operation 5 years ago, offering 6% compounded daily. Gold miner *Yellowstone Sam* deposited $1000 at that time, and now wants to know his present balance and the total accrued interest after all this time. (Continue ...)

Solution (***p*** = **1** and ***PMT*** = 0 are taken over from above):

−1000 PV	−1 000.00	original deposit,
6 i%/a	6.00	% interest rate per year,
365 per/a	365.00	days per year,
5 × n_{PER}	1 825.00	periods. Now, Sam has...
PV	1 349.83	present balance meaning...
RCL PV +	349.83	accrued interest.[293]

Nominal interest rate converted to effective rate:

[293] *TN*: The adding of ***PV*** here is due to the *cash flow convention* of p. 295.

Example for finding the effective annual interest rate:
What is the effective annual rate of interest if the annual nominal rate of 12% is compounded quarterly? (Continue ...)

Solution (***p*** = **1** and ***PMT*** = 0 are taken over from above):

−100 PV	−100.00	base value,
12 i%/a	12.00	% nominal rate per year,
4 per/a	4.00	quarters per year,
4 n_{PER}	4.00	compound periods in total;
FV	112.55	
RCL PV +	12.55	% effective interest rate.

Turn to pp. *347ff* for more applications of *TVM* (annuities etc.).

The Catalog of Constants

Your *WP43* contains 81 important constants, sorted alphabetically. Pressing CONST the first time, you will get:

	G	G_0	G_C	g_e	$GM_\oplus$	$g_\oplus$	
	c_2	e	*e*	F	F_α	F_δ	
	a	a_0	a_{Moon}	$a_\oplus$	c	c_1	

Besides by browsing with ▲ and ▼, you can reach the constants most easily using the alphabetical search method demonstrated in *ReMS 2*.

Find the *names* of 15 astronomical and surveying and 11 mathematical constants printed on colored background in the table below, as well as the 6 fundamental physical constants and 4 constants defining standard conditions.

Name	Unit[294]	Remarks
a	*day*	*Gregorian* year
a_0	m	*Bohr* radius
a_{Moon}	m	Semi-major axis of the Moon's orbit around the Earth.
$a_\oplus$	m	Semi-major axis of the Earth's orbit around the Sun. Within its uncertainty, $a_\oplus$ equals 1 AU (*astronomic unit*).
c	m/s	Speed of light in vacuum
c_1	$m^2\,W$	1st radiation constant
c_2	$m \cdot K$	2nd radiation (*Wien*) constant
e	C	Elementary charge
e	1	*Euler's e*
F	C/mol	*Faraday* constant
F_α	1	*Feigenbaum's* α and δ
F_δ		
G	$m^3/(kg\,s^2)$	*Newtonian* constant of gravitation; also known as γ from other authors. See also $GM_\oplus$ below.
G_0	$1/\Omega$	Conductance quantum
G_C	1	*Catalan's* constant
g_e	1	*Landé's* electron g-factor
$GM_\oplus$	m^3/s^2	Geocentric gravitational constant (according to the Earth model *WGS84,* → *ReMS 2*)
$g_\oplus$	m/s^2	Standard Earth gravity acceleration
h	J s	*Planck* constant, a.k.a. elementary quantum of action
$\hbar$	J s	So-called '*Dirac* constant', actually just $h/2\pi$
k	J/K	*Boltzmann* constant

[294] Find the numeric values of all these constants and their uncertainties as well as all units used here explained in *ReMS 2*. – The photograph of *M31* (the *Andromeda* galaxy) above was taken by *Boris Štromar*; compare the vintage painting below.

Name	Unit[294]	Remarks
K_J	Hz/V	*Josephson* constant
l_p	m	*Planck* length
m_e	kg	Electron mass
M_{Moon}	kg	Mass of the Earth's Moon
m_n	kg	Neutron mass
m_p	kg	Proton mass
M_p	kg	*Planck* mass
m_p/m_e	1	Proton to electron mass ratio
m_u	kg	Atomic mass constant
$m_u c^2$	J	Energy equivalent of atomic mass constant
m_μ	kg	Myon mass
$M_\odot$	kg	Mass of the Sun
$M_\oplus$	kg	Mass of the Earth. See also $GM_\oplus$ above for more precision
N_A	$1/mol$	*Avogadro's* number
NaN		**NaN** means "Not a Number" (e.g., $0/0$ or $ln(x)$ for $x < 0$ or $tan(90°)$ unless in *complex* domain). **NaN** covers poles as well as regions where a function result is not defined at all. ➡ Infinities, on the other hand, are considered numeric in your *WP43* (see the end of this table). Any such *special* result of a calculation (±∞ or **NaN**) will throw an error message unless SPCRES (or *flag* D) is set; **NaN** allows that functions written by you can return **NaN** instead.
p_0	Pa	Standard atmospheric pressure
R	$J/mol\ K$	Molar gas constant
r_e	m	Classical electron radius
R_K	Ω	*Von Klitzing* constant
R_{Moon}	m	Mean radius of the Moon

Name	Unit[294]	Remarks
R_∞	$1/m$	*Rydberg* constant
$R_\odot$	m	Mean radius of the Sun
$R_\oplus$		Mean radius of the Earth
Sa	m	Semi-major axis
Sb		Semi-minor axis
Se^2	1	1st eccentricity squared
Se'^2		2nd eccentricity squared
Sf^{-1}		Flattening parameter
		… according to the Earth model *WGS84* (→ *ReMS 2*)
T_0	K	= 0°C, standard temperature
t_p	s	*Planck* time
T_p	K	*Planck* temperature
V_m	m^3/mol	Molar volume of an ideal gas at standard conditions
Z_0	Ω	Characteristic impedance of vacuum
α	1	Fine-structure constant
γ	$m^3/kg\ s^2$	*Newtonian* constant of gravitation; also known as G from other authors. See also $GM_\oplus$ above.
γ_{EM}	1	*Euler-Mascheroni* constant
γ_p	Hz/T	Proton gyromagnetic ratio
$\Delta\nu_{Cs}$	Hz	Hyperfine transition frequency of ^{133}Cs
ε_0	F/m	Electric constant or vacuum permittivity
λ_C	m	*Compton* wavelengths of the electron, neutron, and proton
λ_{Cn}		
λ_{Cp}		
μ_0	H/m	Magnetic constant or vacuum permeability
μ_B	J/T	*Bohr* magneton
μ_e		Electron magnetic moment

Name	Unit[294]	Remarks
μ_e/μ_B	1	Ratio of electron magnetic moment to *Bohr's* magneton
μ_n	J/T	Neutron and proton magnetic moment
μ_p		
μ_u		Nuclear magneton
μ_μ		Myon magnetic moment
σ_B	W/m^2K^4	*Stefan-Boltzmann* constant
Φ	1	Golden ratio
Φ_0	Wb	Magnetic flux quantum
ω	rad/s	Angular velocity of the rotating Earth according to the Earth model *WGS84* (→ *ReMS 2*)
$-\infty$	1	These infinities may result from *real* calculations. They are counted as special numeric values (compare **NaN**) and can be transferred if SPCRES (or *flag* D) is set; else an error message will be thrown.
∞		

Note CONST is also accessible via CATALOG, so you can summon these constants in equations as well (→ *266ff*). Employ them for further useful equivalences, e.g.:

- express *joules* in *electron-volts* $1\ \mathrm{eV} \approx 1.602 \times 10^{-19}\ \mathrm{A\,s\,V} \Rightarrow 1\ \mathrm{J} \approx 6.24 \times 10^{18}\ \mathrm{eV}$;
  $\Rightarrow\ 1\ {}^{\mathrm{eV}}/_{k} \approx {}^{1.602 \times 10^{-19}}/_{1.381 \times 10^{-23}}\ \mathrm{K} = 11\,604.5\ \mathrm{K},$ [295]
- calculate the wavelength from the frequency of electromagnetic radiation via $\lambda = {}^{c}/_{\nu}$ (so 100 THz correspond to 3.0 μm),
- determine the energy of electromagnetic radiation from its frequency via $E = h\nu$ (thus, $1\ \mathrm{THz} \times h = 6.63 \times 10^{-22}\ \mathrm{J} = 4.14\ \mathrm{meV}$). $1\ \mathrm{eV}$ corresponds to $242\ \mathrm{THz}$ (i.e. $\nu^{-1} = 4.1\ \mathrm{fs}$, $\lambda = 1.2\ \mu\mathrm{m}$).

[295] Another nice example for throwing *SI* in, getting *SI* out.

Another **example**:

Do you remember *Albert Einstein's* most popular formula $E = m\,c^2$? So if you want to see the energy equivalent (in *electron-volts*) of one of the small particle masses given in *kilograms* above, multiply its mass by

$$c^2/_e \approx 5.610 \times 10^{35}\ \mathrm{m^2/_{A\,s^3}}$$

and you are done: $m_e\ c^2/_e = 510\,999\ \mathrm{eV} \approx 511\ \mathrm{keV}$, for instance, m_p corresponds to 938 MeV, etc.

One final **example**:

Assume American advanced scientists will succeed in producing a tiny bit of anti-matter in one of their high-tech laboratories one day – let's say 0.1 µg of anti-hydrogen, carefully stored isolated in ultra-high vacuum (*UHV*).

Although in future, most probably *US* power transmission lines will still look like they do today since this is a well-tried American standard.[296] Thus, under slightly extreme weather conditions,[297] an accidental blackout may happen easily for some days – the electric vacuum pumps will run down, and a subsequent vacuum breakdown will let atmospheric gas leak into the shiny vacuum vessel where it will interact with the precious anti-matter and annihilate immediately. How much energy is going to be released then?

Solution:

You only need the same tiny amount of (common) matter, so 0.2 µg will annihilate in total within the vessel. 1 µg = 10^{-6} g = 10^{-9} kg. Thus enter:

DISP ENG 3	
.2 E +/- 9	0.2×10^{-9}
CONST c	299.8×10^{6}
ENTER↑ X	89.88×10^{15}

[296] This kind of open cable routing can be observed in India in the extreme as far as I am aware. Advantage: you always see where the cables run. Disadvantage: see above.

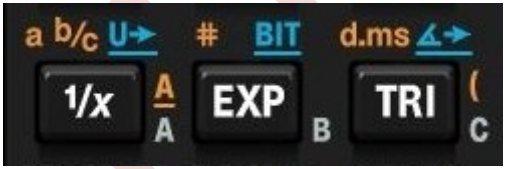
[297] Think of a thunderstorm, blizzard, or hurricane – expected happening more frequently due of *Anthropogenic Climatic Change*. But don't be afraid, this is all fake news created by insane minds according to the greatest president of all times of that blessed nation (feel free to swap adjectives – as *Albert Einstein* once said reportedly: "*Zwei Dinge sind unendlich, das Universum und die menschliche Dummheit, aber beim Universum bin ich mir noch nicht ganz sicher.*" Translated ~ "*There exist just two infinite entities: the universe and human stupidity; but I'm not yet certain about the universe.*")

☒ 17.98×10^6 J = 18 MJ set free. The odds are frightening high this lab will need no cleaning in next weeks.[298]

On the other hand, 0.1 µg of anti-hydrogen require $N_A/10^7$ atoms of it (with N_A being *Avogadro's* number), meaning 6×10^{16} atoms or 3×10^{16} molecules of this gas (i.e. 30 000 million millions of anti-hydrogen molecules). Luckily, this amount is far from being producible in any lab for the time being.[299]

Unit Conversions

Your *WP43* features 22 angular conversions stored in ∠→ (→ *143*). Furthermore, U→ contains 24 conversions of astronomical distances (*light years*, *parsec*, and *astronomic units*), time intervals (*years*, *days*, and *seconds*), levels (*decibel*), *carats*, and dedicated kitchen volumes (*spoons* and *cups*), being useful for the world. Also $\mathbf{T_0}$ (→ *301*) helps converting *centigrade* to *kelvin* and vice versa (this member of CONST is not repeated in U→ since it is only added or subtracted).



All conversions in U→ come as pairs: Select the pair you need, then press the softkey for the target unit and ***x*** will be converted (→ examples below). *SI* units are always on the left.[300]

Working within *SI*, you don't need any conversions – everything just fits in snuggly and returns proper *SI* units automatically (→ examples below).

---

[298] For comparison, 1 kg of *TNT* releases 4.6 MJ. So 18 MJ will cause a great blast.

[299] … and proper *UHV* vessels show very low leak rates, so the annihilation energy may be released in small quantums (you can determine their size) over a longer time interval – power supply may be re-established in time and vacuum pumps operating again.

For crucial applications, however, uninterruptible power sources based on batteries and/ or independent (diesel) generators are recommended to be installed locally wherever supplies are threatened by the actual state of public infrastructure being less than great. Personally, I've seen two big ship diesel generators dedicated exclusively to a medium size production plant in Punjab.

Furthermore, be aware that ordering *antipasti* and *pasta* together in an Italian *ristorante* is strictly at your own risk. You have been warned.

[300] In memory of *Massimo Gnerucci*: "*Left is right and right is wrong.*"

The remaining 118 entries of U→ are only needed if you should be confronted with old local units surviving on isolated islands.[301]

Overall, U→ is structured like a tree. Press U→ and you will see this first:

containing the labels of 8 branches for converting Energy, Power, Time, Force & pressure, mass, length (x), Area, and Volume units. See the entire structure of U→ below (with the *menu* rows printed top down following common reading habits). Some labels need more than 6 characters – then extra high *menu* rows will be displayed:

	F1	F2	F3	F4	F5	F6	Remarks
U→:	E:	P:	Time:	F&p:	m:	x:	branch headers, units of temperature, torque, and ratios – the latter in an extra-high row
	°C ↔ °F				V:	A:	
	dB ↔	power ratio	dB ↔	field ratio	Nm ↔	lbf·ft	
E:	J ↔	cal	J ↔	Btu	J ↔	Wh	units of energy
P:	W ↔	hp_E	W ↔	hp_{UK}	W ↔	hp_M	units of power
Time:	s ↔	day			s ↔	year	units of time
F&p:	N ↔	lbf	Pa ↔	bar	Pa ↔	psi	units of force and pressure
	Pa ↔	in.Hg	Pa ↔	torr	Pa ↔	atm	
			Pa ↔	mmHg			

[301] *TN*: These entries will help when you happen to be thrown back into such an obstinate environment (seems British heritage). U→ may also give you a minute idea of the mess we had in the world of measuring before going metric after the French Revolution over 220 years ago.

Just for balancing, we included also 9 (refined) traditional Chinese units – else those annoyingly loud and parochial 5% of the world's population (always expecting special treatment but lacking due respect for the laws and customs of the rest of the world) might get even more arrogant and boastful.

	F1	F2	F3	F4	F5	F6	Remarks
m:	kg ↔ lb.		kg ↔ cwt		kg ↔ oz.		units of mass
	kg ↔ stone		kg ↔ short cwt		kg ↔ troy oz.		
	kg ↔ ton		kg ↔ short ton		kg ↔ carat		
	kg ↔ liăng				kg ↔ jīn		
	g ↔ oz.		g ↔ troy oz.		g ↔ carat		kitchen units [302]
x:	m ↔ au		m ↔ l.y.		m ↔ pc		units of length
	m ↔ mile		m ↔ nmi.		m ↔ foot		
	m ↔ inch		m ↔ point		m ↔ yard		
	m ↔ lĭ		m ↔ yĭn		m ↔ zhàng		
	m ↔ chĭ		m ↔ cùn		m ↔ fēn		
	m ↔ fathm						
	mm ↔ inch		mm ↔ point		m ↔ surv. foot$_{US}$		'kitchen' units [302]
	km ↔ mile		km ↔ nmi.				
A:	m^2 ↔ acre		m^2 ↔ mŭ		m^2 ↔ acre$_{US}$		units of area
	ha ↔ acre		m^2 ↔ ha		ha ↔ acre$_{US}$		'kitchen' units [302]
V:	m^3 ↔ gal$_{UK}$		m^3 ↔ barrel		m^3 ↔ gal$_{US}$		units of volume
	m^3 ↔ floz$_{UK}$				m^3 ↔ floz$_{US}$		
	mℓ ↔ tsp$_{UK}$		mℓ ↔ tbsp$_{UK}$		mℓ ↔ cup$_{UK}$		*British Imperial* kitchen units
	mℓ ↔ floz$_{UK}$		mℓ ↔ pint$_{UK}$		ℓ ↔ gal$_{UK}$		
	mℓ ↔ tea spoon$_I$		mℓ ↔ table spoon$_I$		mℓ ↔ cup$_I$		international kitchen units

[302] Consecutive conversions and calculations in science and engineering are executed the easiest (and safest) way just using *SI* base or derived units without any prefixes.

TN: A project team member from down under, however, asked for 'kitchen units' (he meant *gram*, *liter* and *milliliter, millimeter* and *kilometer*, and *hectare* – large kitchens!) appearing explicitly on the *SI* side of some conversions. Hence, now you must watch out that you don't lose powers of 10 when using those conversions (→ *Example 4* below, for instance). That's the price to be paid for working with old cookbooks etc. We sincerely recommend avoiding those kitchen units in consecutive conversions.

Rest assured there exist lots of weird and incoherent units more. You are invited to compile your personal favorites in a *user menu* (→ *OMS 7*) so the vast majority of worldwide users doesn't have to suffer from a few requiring extensive space.

	F1	F2	F3	F4	F5	F6	Remarks
	mℓ ↔ tsp_{US}		mℓ ↔ $Tbsp_{US}$		mℓ ↔ cup_{US}		*US* customary kitchen units
	mℓ ↔ $floz_{US}$		mℓ ↔ $pint_{US}$				
			ℓ ↔ quart		ℓ ↔ gal_{US}		

Find comprehensive explanations of all the units used in these conversions in *ReMS 2*.

Combine conversions as you like. Units are matched carefully. **ENG 2** will do for all the examples following here:

Example 1:

For filling your bike tires with a maximum pressure of 30 psi the following will help you at gas stations in Europe and beyond:

30 U→ **F&p:**

Pa ← psi (press F5) returns 207.×10^3 Pa or

Pa → bar (press F4) returns 2.07 bar.

Now you can set the filler and will not blow your tires.

Example 2:

Your friend in Louisiana tells you she has got 20 *cubic feet* of debris on her veranda after flooding (the dams in the Mississippi delta turned out being of less use than once thought). What does this really mean?

1 U→ **x:** m ← foot returns 305.×10^-3

3 EXP y^x 28.3×10^-3

20 × 566.×10^-3 m^3. OK, some work – but manageable.

Note there is no need to provide a conversion from *cubic feet* to *cubic meters* and vice versa – you can easily create this yourself by applying the usual rules of calculation as shown here. Further applications go by analogy.[303]

Example 3:

A network switch is specified for 3 320 Btu/h. What?!?

[303] *TN* for engineers: That's no witchcraft – just remember that *square* means a power of 2 and *cubic* a power of 3 and you are done, on the *SI* side at least.

3320 U→ **E:** **J ← Btu** returns 3.50×10^{6} J/h.

Since $1J = c_c Wh \iff 1\,{}^{J}/_{h} = c_c W$ applies, you can use

J → Wh for converting and get 973. W, i.e. almost 1 kW.

Now you know what will be going on there.

Example 4:

In *Section 2*, there was a problem resulting in a box featuring a volume of 19 $^{11}/_{16}$ *cubic inches*. So, what does this volume mean in real units instead? And how much water can such a box contain in areas where people are condemned to deal with *British Imperial* units and alike nowadays still?

1 U→ **x:** ▼ **mm ← inch** returns 25.4 mm (kitchen unit!)

3 EXP **y^x** 303 16.4×10^{3}

19 . **11** . **16** **x** $323.\times10^{3}$ mm^3.

Since 1 mm^3 = 10^{-3} cm^3 and 1 cm^3 = 1 ml,

1000 / returns 323. ml ≈ $^{1}/_{3}$ *litre*. And

U→ ▼ **ml → floz$_{UK}$** returns 11.4 *fluid ounces* by another kitchen unit conversion.

But is this true? We actually don't know, maybe the result is

RCL L **ml → floz$_{US}$** 10.9 *fluid ounces* instead?

You have to ask for the target location first! Medieval times …!

Though you can solve this *British Imperial* problem straight ahead as follows (saving 6 keystrokes of 30 by using *SI* base units):

1 U→ **x:** **m ← inch** returns 25.4×10^{-3}

3 EXP **y^x** 16.4×10^{-6}

19 . **11** . **16** **x** $323.\times10^{-6}$

U→ **V:** **m^3 → floz$_{UK}$** 11.4 *fluid ounces*.

Example 5:

A celestial object moves with a velocity of 0.1 *parsec per year*. What does this mean in standard units? What does it mean in relation to the velocity of light? And how does this translate for air pilots?

.1 U→ **x:** **m ← pc** returns 3.09×10^{15} m.

EXIT returns to the top *view* of U→.

1 Time: s ← year /	returns	97.8×10^6	m/s.
CONST c /	returns	326.×10^−3	= 32.6% of *c*.
RCL L x		97.8×10^6	m/s.
3600 x	returns	352.×10^9	m/h (since 1 h = 3600 s).
… corresponding to U→ x: m → nmi		190.×10^6	nmi/h

or **190** *mega-knots*.[304]

Example 6:

What is the average velocity of the Earth orbiting the Sun?

1 full Earth orbit is approximately $2\pi r$ with $r = a_\oplus$:

2 π x CONST $a_\oplus$ x	returns	940.×10^9	m.

It takes 1 year for running this distance. This is:

1 U→ Time: s ← year		31.6×10^6	s.
Hence, /	returns	29.8×10^3	m/s
or 3.6 x		107.×10^3	km/h.

Example 7:

What on earth means '1 *furlong per fortnight*'?

1 *furlong* equals $^1/_8$ *mile*. And 1 *fortnight* is 14 *days*. Thus,

8 1/x U→ x: ▼ m ← mile	returns	201.	m and
14 EXIT Time: s ← day	returns	1.21×10^6	s. Hence,
/ 3600 x	returns	599.×10^−3	m/h ≈ **0.6** m/h.

Many snails are faster. Note 2688 *furl./fortn.* ≡ 1 *mile*/h. Easy to remember.

Example 8:

There are reports that some engineers on isolated islands (→ 130) measure loads in *slugs*.[305] 1 *slug* is defined as a mass accelerated by 1 ft/s² when a net force of 1 *pound* (lbf) is exerted on it. What does this mean for the rest of the world?

---

[304] Sounds like a unit created for the great *Alexander* visiting *Gordion* in Asia in 333 BC.

[305] *TN*: Remember those already have got *pounds*, *ounces* (also *troy* ones), *stones*, *short* and *long hundredweights*, *short* and *long tons* – linked by factors circumventing easy exponents of 10 like the plague. But all those don't suffice, thus *slugs* were created. What on earth does the world do wrong with *kilograms*?

$$1\ slug = 1\ \text{lbf} \cdot \frac{\text{s}^2}{\text{ft}} = 1\ \text{lbf} \cdot \frac{\text{N}}{\text{lbf}} \cdot \frac{\text{s}^2}{\text{ft}} \cdot \frac{\text{ft}}{\text{m}}$$

1 U→ F&p: N ← lbf	returns	4.45 N and
1 EXIT x: m ← foot	returns	$305.\times10^{-3}$ m. Thus,
/	returns	14.6 kg for 1 *slug*.[306]

Further research found 12 *slugs* = 1 *blob*.[307] You may even meet some islanders working with material densities given in *blobs per cubic inch.* What on earth does that mean?

$$1\ {}^{blob}/_{inch^3} = 12\ {}^{slugs}/_{inch^3} = 12 \times 14.6 \times \left(\frac{\text{m}}{inch}\right)^3$$

12 ×	returns	175. kg for one *blob* and
1 m ← inch 3 EXP y^x	returns	16.4×10^{-6} m^3. Thus,
×	returns	10.7×10^{6} kg/m^3 or **10.7** kg/cm^3

which is far denser than any engineering material available.[308]

Example 9:

There is more to find in those dark corners: 1 poundal = 1 ${}^{\text{lb ft}}/_{\text{s}^2}$ – what does this mean for the rest of the world?

$$1\ pdl = 1\ \frac{lb}{kg} \times \frac{ft}{m} \times \frac{kg\ m}{s^2}$$

1 U→ m: kg ← lb	returns	$454.\times10^{-3}$ kg and
1 EXIT x: m ← foot	returns	$305.\times10^{-3}$ m. Thus,
×	returns	$138.\times10^{-3}$ N.

Supported by your *WP43*, you will find further easy ways to produce whatever conversion you may need.

---

[306] Another nice example for throwing *SI* in, getting *SI* out.

[307] Sounds like a decent quantity of pudding hitting an empty plate.

[308] *TN*: Those islanders need that unit to keep their pre-*SI* software packages running, for sake of tradition ("that's the way we've always done that" – like cable routing on p. 303). Learned that seniority plays a major role there but didn't know that applies to software as well.

In emergencies of a particular kind, it may be helpful knowing that …

- *becquerel* (Bq) equals *hertz* in your *Geiger-Müller* counter,[309]
- *gray* (Gy) is the unit for deposited or absorbed energy (1 Gy = 1 $^{J}/_{kg}$; → *ReM*), and
- *sievert* (Sv) is *gray* times a radiation dependent dose conversion factor (≥1) for the damage caused in biological material including human bodies (remember the problem on pp. *91f*).

Also in this field, some outdated units may still be found in literature:

- For those admiring the very first *Nobel* laureate in physics (in 1901), *Wilhelm Conrad Röntgen*, for discovering the X-rays[310] (revolutionizing medicine), the charge generated by radiation in matter was measured by the unit *roentgen* with $1\,R = 2{,}58 \cdot 10^{-4}\ ^{A\,s}/_{kg}$.
- At the same time, *rem* (i.e. *roentgen equivalent in men*[311]) measured what *sievert* does today (1 rem = 10 mSv).
- Few decades ago, *rad* (*radiation absorbed dose*) was used, with 1 *rad* = 0.01 *gray* which is still a lot – values greater than *millirad* are

[309] Such emergencies or accidents would never ever happen, of course, the responsible parties affirmed. Those accidents, however, did not know those solemn affirmations – hence they simply happened against (almost) all official experts' claims. In consequence, mankind gathered (and still gathers) experience with radioactivity.

TN: Here, our warmest regards go to Algeria*, Argentina, Armenia, Australia*, Austria, Belarus*, Belgium, Brazil, Bulgaria, Canada, China, Czech Republic, Finland, France, Germany, Hungary, India, Iran, Israel, Japan, Kazakhstan*, Kiribati*, Lithuania*, the Marshall Islands* (incl. Bikini and Eniwetok), Mexico, Mururoa*, the Netherlands, North and South Korea, Pakistan, the Philippines, Romania, Russia, Slovakia, Slovenia, South Africa, Spain, Sweden, Switzerland, Taiwan, Tibet*, the Ukraine*, *United Arab Emirates*, *United Kingdom*, *USA*, and Xinjiang-Uighur* in alphabetical order.

The countries marked with a star suffer from actions of their so-called 'motherlands' between 1945 and today – spotting those colonialists is left as an exercise for the reader. The other countries in the list controlled their industry or military in a way that activated areas on their own territory could occur – see also the two maps below, though the 2nd not showing an emergency or accident but an experiment on humans.

[310] *Conrad Röntgen* (1845 – 1923) ruined his hands in those experiments – he couldn't know better yet.

[311] This unit must be outdated – it is not regarded gender equitable nowadays anymore.

seldom found in literature (but → *314*).

- Pour les fidèles amis et admirateurs de Madame *Marie Skłodowska Curie* (1903 *Nobel* laureate in physics and 1911 in chemistry), there was a unit *curie* with $1\ \mathrm{Ci} = 3{,}7 \cdot 10^{10}\ \mathrm{Bq} = 3{,}7 \cdot 10^{10}\ decays/s$.

  You can deduct from the size of this unit that lumps of radioactive material were 'absolutely no problem' for the pioneers in this field. How could they know any better at that time?[312]

  Luckily, mankind has advanced meanwhile as the following **example** proves:

Radiation Hotspots Resulting From the Chornobyl' Nuclear Power Plant Accident

[312] Pioneering is certainly exciting but may carry risks. *Marie Curie* (1867 - 1934) died from aplastic anemia, aged 66. Seems that *Conrad*, who died from carcinoma of the intestine, aged 78, was more cautious or just luckier than *Marie* with respect to collateral health damages from his experiments.

In the confiscated/closed zone around Chernobyl, ≥ 40 Ci/km² of ^{137}Cs were measured in 1996 (some 4300 km² – note the scale on the map[313]). How many *decays/s* (a.k.a. *becquerel*) did that mean per *square foot*? How many were these in 1986, how many are these in 2026, how many will these be in 2086 and 2186?

Solution (with **ENG 2**; the half-life of ^{137}Cs is 30.2 *years*, → 92):

40 ENTER↑ **3.7** E **10** ×	1.48×10¹² Bq/km²,
E **6** /	1.48×10⁶ Bq/m²,
1 U→ x: **m** ← **foot** EXP x²	92.9×10⁻³ m²/ft²,
×	137.×10³ Bq/ft² in 1996,
96 ENTER↑ **86** − **30.2** STO K / 2^x ×	173.×10³ Bq/ft² in 1986,
1986 ENTER↑ **2026** − RCL K / 2^x ×	69.1×10³ Bq/ft² in 2026,
26 ENTER↑ **86** − RCL K / 2^x ×	17.4×10³ Bq/ft² in 2086,
100 +/- RCL K / 2^x ×	1.76×10³ Bq/ft² in 2186,

i.e. 200 *years* after that event (remember all these activities are minima).

Feel free to compare the three maps printed in this *OM*.[314]

---

[313] The map above is a clip (cut, enlarged, and reassembled by me) of a map found at https://maps.lib.utexas.edu/maps/commonwealth/chornobyl_radiation96.jpg, copied from the *CIA Handbook of International Economic Statistics* (1996).

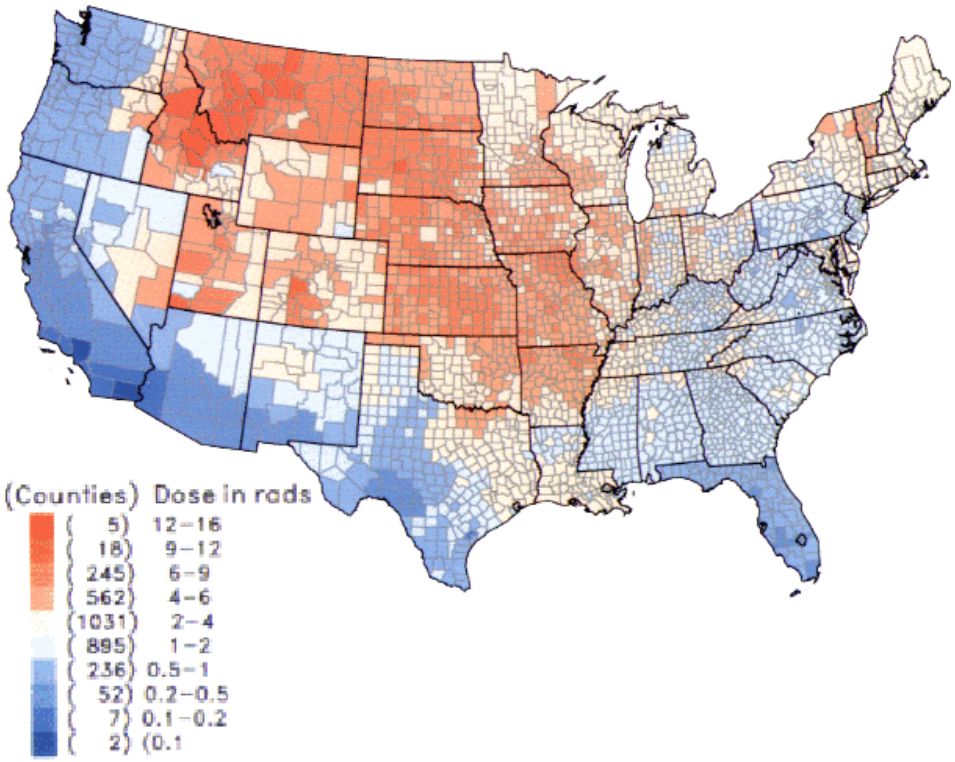
Note the Chernobyl disaster started with a planned experiment in good faith but immediately went out of control in the night shift of 1986-04-26 due to an inherent reactor design flaw unknown to the acting operators. Some of them were convicted by courts of the *USSR* while the reactor designers went unpunished. About 300 000 people were evacuated.

For the record: 40 years after, in 2026-04, 70 µSv/h were measured by a journalist visiting the ruins of said reactor. This corresponds to 1.7 mSv/d and 614 mSv/a.

Look forward to detecting some 2 µSv/h in 2186 at the same spot, corresponding to 1.3 mSv/*month* after 200 *years*. Feel free to extrapolate further and compare with p. 93.

[314] The map following overleaf is quoted from *Study Estimating Thyroid Doses of I-131 Received by Americans From Nevada Atmospheric Nuclear Bomb Test*, *National Cancer Institute* (1997), → https://commons.wikimedia.org/w/index.php?curid=524198. I enlarged its legend for better readability.

TN: Not everything on this map looks plausible – data are inconsistent from state to state. And north of Montana seem to have lived no Americans. Who held the overall responsibility for that nuclear bomb test and its consequences?



The remarkably huge delay between the test and the publication of this study can't be attributed to the half-life of the iodine isotope investigated (→ *100*).

Being at it and for the record: According to treaties between the *USA* and *Native Americans* around 1800 all the area west of the Mississippi should belong to these peoples. Those treaties, however, were broken shortly by people flocking east of the that river – their subsequent actions would be rated nowadays as a continuous cultural and physical genocide. Activities alike were observed, e.g., in Canada and Australia – seems the majority of immigrants from Europe carried a particular mindset. Apologizing recently doesn't make those misdeeds undone at all.

Some alleged 'great powers' (at their time) misbehaved alike in 20th century (like the *USSR* 1931 - 50, Japan 1932 - 45, and Germany 1938 - 45) – Ukrainians, East and South East Asians, and many Europeans suffered the same way (but shorter). Germany and Japan were punished eventually. Alas, Chinese (since 1950) and Russians (since 1994) misbehave until today – Tibetans, Chechens, Georgians, Crimean Tatars, Uighurs, and Ukrainians again have to cope with it. It's all peoples' right be free and living self-determined! Imperialism is so outdated – decolonialize!

At the bottom line, international law – based on centuries of mankind's painful experiences – carries many advantages for common people, smaller countries and nations. It shall be observed like civil law, by everybody on this planet. Period.

SECTION 7: CREATING YOUR VERY PERSONAL *WP43*

Beyond storing your personal programs and data, you can even modify the *user interface (UI)* of your *WP43* according to your needs. As far as we are aware of, *WP43* is the first calculator worldwide allowing for fully customizing its *UI*: You may assign an arbitrary functionality to almost any location, unshifted or shifted, on the keyboard or in a *menu*. *User mode* will then bring your assignments to the front, so you can interact with your *WP43* via a *UI* you designed yourself.

Use **ASSIGN** to store your personal assignments: [ASN] allows for reassigning almost the entire keyboard (but [f], [g], [α], and [F1] - [F6]; we recommend keeping also other basic functions accessible, → *324f* for some caveats).

In the explanations starting overleaf, …

▭ stands for the *softkey* applicable ([F1] - [F6], optionally preceded by [f] or [g]),

[***key***] represents an arbitrary labelled key of your *WP43* or a [f]- or [g]-shifted label on its keyboard,[315] and

name is the *name* of an arbitrary *item* (be it a function, digit, symbol, routine (i.e. global label), *variable*, or *system flag*; note a new *name* must follow the rules stated on p. *60*).

Such a *name* may be…

a) keyed in spelling it via [α] … [ENTER↑],[316] or

b) picked from CATALOG or any other *menu*, or

c) called by accessing the respective key/label on the keyboard.[315]

Just pressing [ENTER↑] where a *name* is expected will be interpreted as an *empty name* (echoed **NULL**). This will clear the assignment at the location specified (returning to its *startup default*).

[315] See p. 319 for assigning [ENTER↑], [EXIT], [←], [▲], and [▼].

[316] Here, upper and lower case letters will be checked, so your *WP43* will regard **Var1** and **VAR1** as being different *name*s. Superscripts and subscripts are not discriminated from normal symbols, so **data1T** and **data$_1^T$** are seen as equal *names* (but the first may ease typing and the latter reading for you). Compare p. 60.

Assigning Your Favorite Functions

Here is how you can tailor the surface of your *WP43* according to your individual preferences:

ASN ***name*** [***key***] will assign this named *item* to [***key***] in *user mode*. It will throw an error if a *name* keyed in doesn't exist.

ASN ***name*** ▢ in particular will assign this *item* to the respective *softkey* of the *user menu* displayed at the time you press this ▢ (preceded by f or g, if applicable), replacing the label shown there before. It will throw an error if the target is not within a *user menu*.

Each user assignment will hold until it is either overwritten by a new assignment or an *empty name* is entered (→ previous page).

All user assignments will be visible and accessible in *user mode* only (→ 324ff) – except the *items* assigned to the 3 *user menus* provided (MyMENU, MyPFN, and Myα, see below).

Example 1:

Let's assign the statistical sample standard error to, e.g., g + CC (this location is labelled [x̄] on the keyboard). There are 3 ways to do this as specified below (printing all keystrokes necessary):

a) f ASN g α ▼ S M ENTER↑ g [x̄]

b) f ASN g CATALOG FCNS S M s_m g [x̄]

This way will work always – even if you remember neither where the target function is stored nor how it is spelled exactly. It will be demonstrated step by step starting overleaf.

c) f ASN g STAT s_m g [x̄]

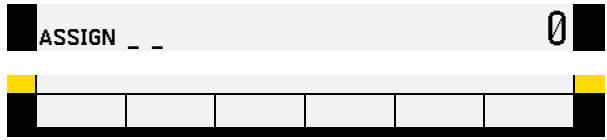
On the other hand,

f ASN ENTER↑ g [x̄]

will reset g-shifted CC to factory default [x̄] as explained at the bottom of previous page.

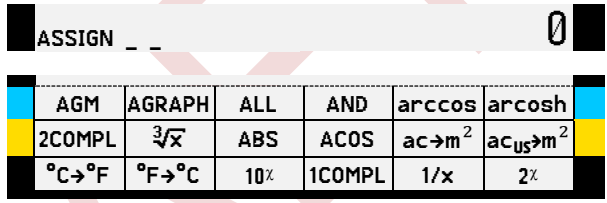
We will walk you through solution b) here, starting with a clear *stack* (press **0** [f] [FILL] if necessary). Only the *menu section* and the command echo row will be shown in the following since all action will take place there:

[f] [ASN]



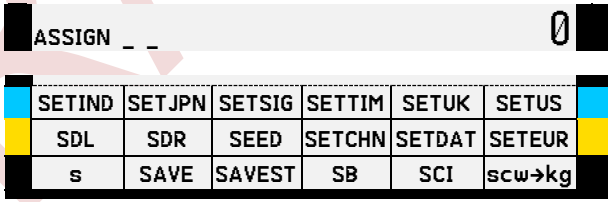
[f] [CATALOG]

FCNS



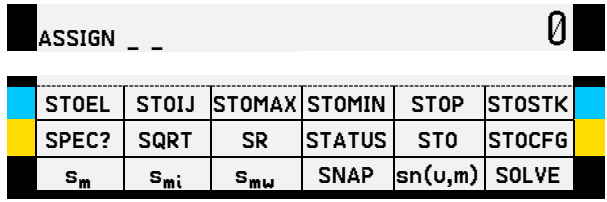
This is the top *view* of CATALOG'FCNS. Now enter the 1st letter of the requested command:

[S]

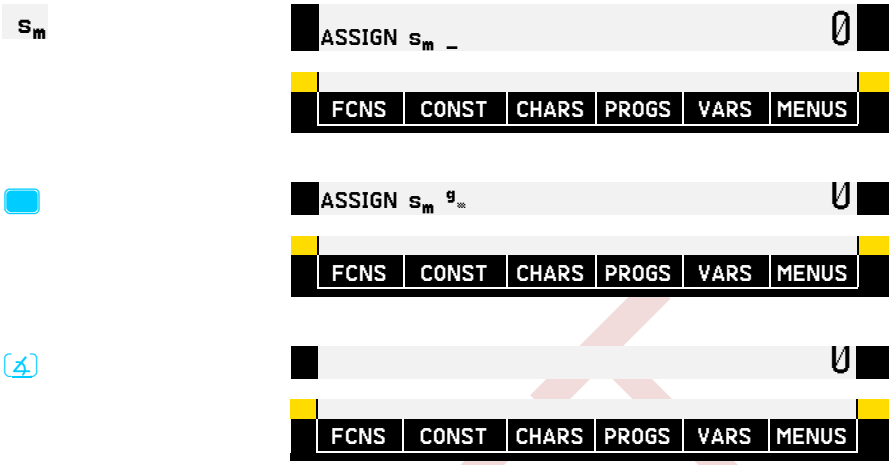


Quickly entering the 2nd letter helps substantially:

[M] (if you realize you waited too long before pressing [M], just wait another few *seconds*, then type [S][M] quickly here instead)

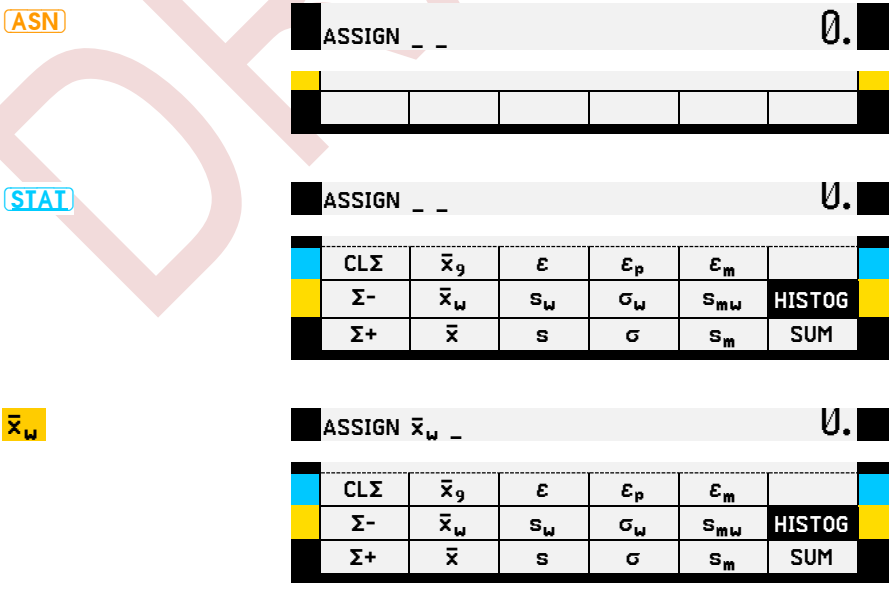


Now, press [F1] for the function to be assigned (the *menu view* will change):

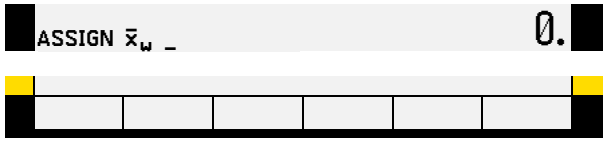


… and this assignment is done (compare above). This last *menu view* will stay on screen until it is EXITed explicitly or another *view* or *menu* is called. You can verify your assignment by setting *user mode* via USER and pressing ⍺ which will be echoed s_m now – the function ⍺ will, however, stay accessible via CATALOG FCNS ⍺ always.

Example 2 (in *SDC*):
Assign the weighted arithmetic mean to the 1st key in MyMENU:



USER



Pressing USER exited all *menus* being open and brought MyMENU on the screen (which is empty still – note the grid indicating an empty *menu* in display). Now, press

F1

… and $\bar{x}_w$ is where we want it to be.

Summarizing, ASN STAT $\bar{x}_w$ USER F1 did this assignment.

MyMENU will appear whenever all other *menus* are exited,[317] i.e. …

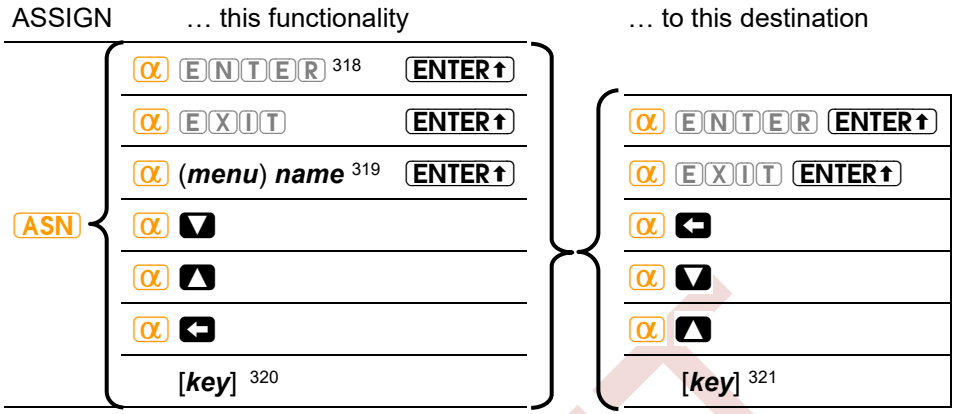
- either after you left all applicable *submenus* and pressed EXIT in the parent *menu*
- or after you pressed EXIT longer than 1 *second* (calling **EXITALL**).

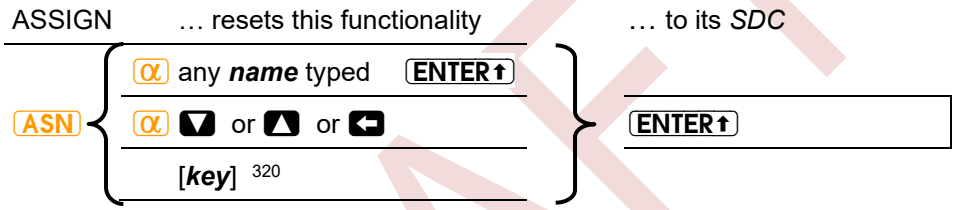
MyMENU will then stay displayed until another *menu* is called. This holds for your *WP43* being in *user mode* or not (see below).

Thus, filling MyMENU may well be the very first step of customizing your *WP43.* You may, for instance, put the 6 trigonometric functions into the unshifted row of MyMENU and will have them almost always at hand.

As mentioned above, some keys and the *menus* need special treatment in assigning. Thus, the general rule for **ASSIGN** reads as listed in the table overleaf (the last row of either column contains the standard case dealt with so far):

[317] In *AIM*, Myα will appear instead of MyMENU, and will stay displayed until another *menu* is called. The same applies to *PEM* and MyPFN by analogy.





Creating Your Own Menus

ASN USER ***new_menu_name*** ENTER↑ will define a new *user menu*.

➡ ASN USER turns on *AIM* so you can immediately enter the new *menu name* (following the rules of p. 60).

Example:
To create a *menu* **Favfun** for your favorite functions, just enter:
ASN USER F ▼ A V F U N ENTER↑

[318] This is to assign the functionality of ENTER↑ to another location. ASN α turns on *AIM*. Spelling just ENTER (without ↑) is sufficient here.

[319] The *name* (or *menu name*) must be spelled.

[320] May be also 🟧 or 🟦 plus a key, or may open a (possibly *multi-view*) *menu* containing the functionality to be assigned. In latter case, press the *softkey* pointing to this source.

[321] May be also 🟧 or 🟦 plus a key, or may open a *user menu* containing the destination to be assigned. In latter case, press the *softkey* pointing to this destination.

The new *menu name* will be inserted in CATALOG'MENUS (note ASN will throw an error if the *menu name* specified turns out being defined already). The new *menu* itself will be created with 18 blank entries – its size is fixed. Though feel free to put *submenus* in it (but no undefined ones). You may fill it now:

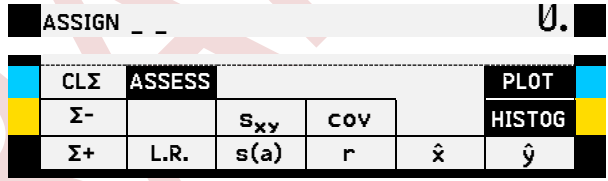
Example:

Assign the y-forecasting function to the 4th key in **Favfun** (assuming you did not define any other *menu* starting with 'Fa' before).

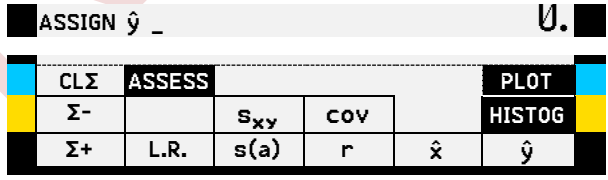
Also the solution of this case will be shown step by step: It starts with the last display of last paragraph since MyMENU stays on screen as long as no other *menu* is called, and we assigned 1 function to it just in previous chapter.

ASN

FIT

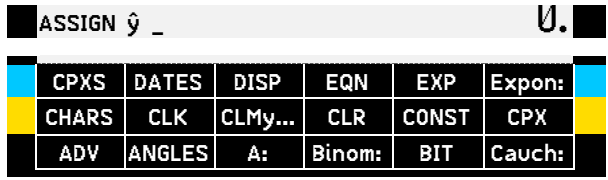


ŷ

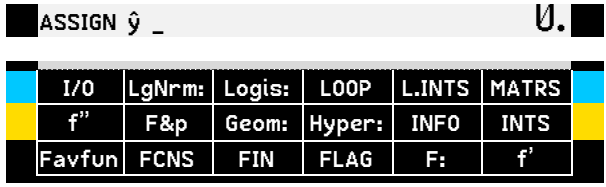


CATALOG

MENUS

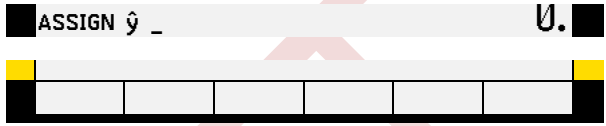


Here you see the first *view* on all the *menus* defined on your *WP43*. Now, enter F and the *view* jumps to the corresponding position in this *submenu*:



Press F1 now

Favfun



Since Favfun was just created above there is nothing to be seen in the *menu section* of the display yet. Pressing F4, however, you will get

Summarizing, ASN STAT ▲ ŷ CATALOG MENUS F Favfun F4 did the job here.

➡ Favfun will remain on screen until it is EXITed explicitly or another *menu* or USER is called.

Assigning Your Favorite Characters

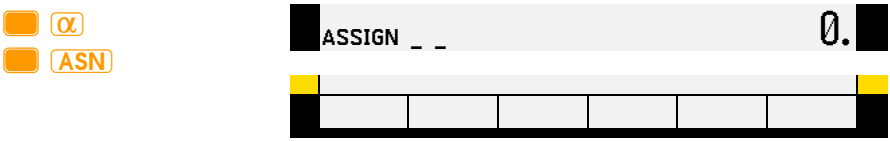
Being in *alpha input mode* (*AIM*), …

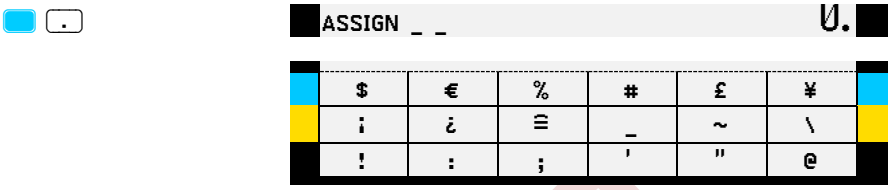
ASN ***character*** [***key***] will assign the character specified to [***key***]. You can pick the character to be assigned from the alpha keyboard or an arbitrary alpha *menu* as introduced on p. 209.

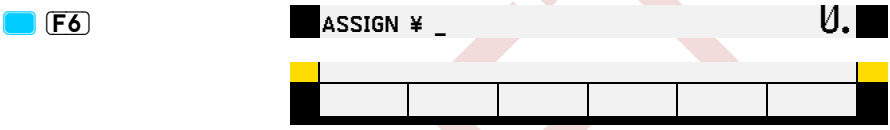
[***key***] may be any legal label location except 🟧, 🟦, α, F1 - F6, ENTER↑, or EXIT. The assignment will become valid when *AIM* is called in *user mode* or when *user mode* is called in *AIM*.

Example (showing all keys to be pressed, incl. prefixes):

Assign the *Yuan* or *Yen* symbol ¥ (contained in α●) to the 3rd key in Myα (assume *SDC* once again):







Pressing [USER] exited all *menus* being open at that time so Myα could slip on the screen with its *startup default* content.

Summarizing, [ASN] [.] [¥] [USER] [F3] did this assignment.

Purging User-Defined Items

Also user-defined *items* (*menus*, *variables*, and programs) are contained in CATALOG. Needing memory space, you can delete such *items* by **DELITM** easily. It lives in CLR. Calling it will show this *menu*:

CLR DELITM PROGS … ▢ allows for deleting the program (starting with the *global label*) selected; → CLP.

CLR DELITM VARS … ▢ allows for deleting the *variable* selected.

CLR DELITM MENUS … ▢ allows for deleting the *menu* selected.

For global deletions, CLPall, DelVall, and DelMall are provided, asking for confirmation for good reasons. – Predefined *labels*, *variables*, or *menus* can't be deleted.

Launching Your Personal *UI* (*User Mode*)

USER toggles *user mode*. Therein, your (user) assignments become valid wherever applicable. Compare previous chapters. *User mode* gives you unexcelled freedom for creating and applying your personal calculator *UI*. Enjoy and play with the opportunities you have: remember you may reassign everything except 🟧, 🟦, α, and F1 … F6 (you may call ASSIGN also in *user mode*).

We recommend leaving also ENTER↑, EXIT / ON,[322] CATALOG, and OFF untouched.

WARNING: Do not remove any necessary functionality from the user keyboard by overwriting it (but you may move it).

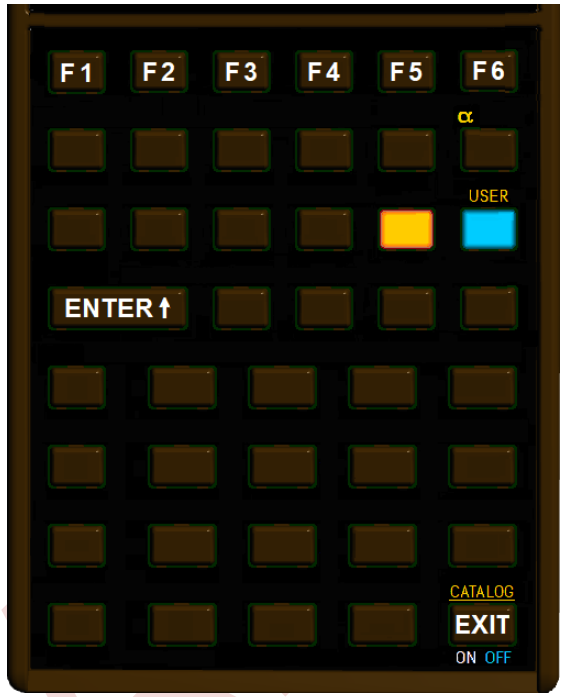
Example:

We recommend assigning USER to another location before assigning anything to 🟧-shifted 🟦 – else you will block your return from *user mode*.

In case of emergency, a *hard reset* (using the RESET hole) may be your only escape – returning you from *user mode*, but erasing all your precious programs, data, and settings in *RAM.* Only what you saved in *FM* (incl. the data on the *FD*) will survive. Thus, we recommend checking consequences meticulously prior to assigning any function.

[322] EXIT and ON are permanently linked.

All your assignments are at your own risk.



Pressing any function key (optionally preceded by 🟧 or 🟦) will show you a preview of the operation currently assigned to this location, left in T row – if you realize you have chosen the wrong key, simply keep it depressed until it falls back to NOP after 1 s. This ***preview and fallback*** applies to all key functionalities except 0 … 9, ., (+/-) in numeric entry,) E, ⬅, ▲, ▼, and F1 … F6.[323] Pressing EXIT will fall 'back' to EXITALL exiting all *menus* (→ *IOI*).[324] 🟧 and 🟦 are echoed but will not fall back. *Preview and fallback* are particularly helpful in *user mode*, when the function called at a particular location may differ from the label seen on the keyboard.

Once you have reached a steady *UI*, store it in a *RoV* using STOCFG (or save it in a *state file* using SAVEST) together with the other settings mentioned on p. 78. Printing keyboard overlays for your personal layouts may pay well, especially if you reassign just functions appearing on the faceplate (so you can place all of them on 1 overlay).

Overlays cover the faceplate entirely and are fixed in 3 slots provided on either side of the keyboard rim (→ *ReMA A* for their locations and dimensions). STOCFG, SAVEST, and overlays are especially beneficial if you plan

[323] On the *HP-42S*, *preview and fallback* were absent also for PRGM, ASSIGN, STO, RCL, XEQ, SHOW, and OFF.

[324] EXIT is the one and only key on your *WP43* changing to a real function (instead of NOP) when held down.

having further alternative *UIs* – you can load any of them via **RCLCFG** or **Load**… when you need it.[325] Think of creating a dedicated *configuration,* e.g., for working with *short integers,* bringing *Boole's* operations on the faceplate. This way you may easily avoid expenses for other calculators.

If, however, you should ever get lost in your multiple *UIs* and *calculator states*, look for **U** top right in the *status bar* – and press USER (or wherever you have moved this functionality meanwhile) which will return you immediately to the *SDC* of your *WP43* as you know it from the time you saw it first.[326]

We sign off wishing you long lasting joy and benefit working with your very own, personalized *WP43.*

[325] **RCLCFG** will throw an error if you attempt recalling something unless a *configuration*. We recommend keeping suitable records what you stored where.

[326] If you want to get rid of any outdated user layout and free the memory allocated to it, simply clear the *register* or delete the *variable* where it is stored, → *322f*. The freed memory will be returned to the pool.

APPENDIX 1: KEY RESPONSE TABLE

Here you find all direct inputs explained per keystroke, top left to bottom right of your *WP43*. For each key in row ***R***, its unshifted function is listed first, then its 🟧- and 🟦-shifted function, if applicable.

Most keys change functionality in *AIM* (→ 208ff), hence the 'alpha' meanings are listed thereafter with labels printed in darker fields and descriptions on light grey, if applicable. Remember *AIM* is also used for entry of equations.

Generally, see the *IOI* for more about any operation mentioned below.

R	Keystrokes	Meaning	
1		Such an **unshifted *softkey*** (F1 - F6) calls the *item* displayed in the bottom *menu row* of the display …	… in the corresponding column 1 - 6 and executes it. It will do nothing if no *item* is displayed at this position (→ 33f).
	🟧	A **shifted *softkey*** calls the *item* displayed in the golden or blue *menu row* …	
	🟦		
2	1/x	Inverts the number or the *matrix* ***x*** (→ 36 and 172).	
	a b/c	Enters *fraction display mode*, showing *reals* as *proper fractions*. With active *fraction display mode*, toggles between *proper* and *improper fractions* (→ 128ff).	
	U→	*Menu* of unit conversions (→ 304ff).	
	A	Enters the Latin letter **A** or **a**.	
	🟧 A	*Catalog* of all Latin letters provided, also accented ones.	
	🟦 A	Enters the Greek letter **Α** or **α**.	

R	Keystrokes	Meaning
2	EXP	*Menu* containing y^x, x^2, x^3, roots, logarithms, and hyperbolic functions (→ 33).
	#	If pressed trailing integer input, defines its base. Else converts a closed x into a *short integer* of the base specified (→ 187ff), or extracts the integer or fractional part of x.
	BIT	*Menu* containing *Boole's* operations (**AND**, **OR**, **NOT**, etc.) as well as further bit manipulating commands.
	B	Enters the Latin letter **B** or **b**.
	#	Enters the character **#**.
	▭ B	Enters the Greek letter **Β** or **β**.

2	TRI	*Menu* containing trigonometric and hyperbolic functions and their inverses (→ 30).
	d.ms	If pressed trailing numeric input, enters an *angle* in *degrees*, *minutes*, and *seconds* (i.e. sexagesimal notation). Else sets *angular display mode* (*ADM*) to sexagesimal angles (→ 132ff).
	∡→	*Menu* of angular conversions (→ 143).
	C	Enters the Latin letter **C** or **c**.
	(	Enters an opening parenthesis.
	▭ C	Enters the Greek letter **Γ** or **γ**.

2	ln	Returns the natural logarithm of x (→ 36 and 86).[327]
	.d	If pressed trailing numeric input, enters a *date* (→ 185). Else either leaves fraction display mode (→ a b/c above), if applicable, or converts … • an *integer* to a *real* (→ 151), • a sexagesimal *angle* to a decimal number (→ 145), • a sexagesimal *time* to a decimal number (→ 207).

[327] Either of the number x or of all *elements* of the *matrix* x.

R	Keystrokes	Meaning
	lg	Returns the decadic logarithm of x (compare ln).[327]
	D	Enters the Latin letter **D** or **d**.
	)	Enters a closing parenthesis.
	🟦 D	Enters the Greek letter Δ or δ (delta).
2	e^x	Raises **e** to the power of x (→ 36 and 86). [327]
	h.ms	If pressed trailing numeric input, enters a sexagesimal *time*. Else converts x to such a *time* (→ 185f).
	10^x	Raises **10** to the power of x (→ 36 and 86). [327]
	E	Enters the Latin letter **E** or **e**.
	🟧 e^x	Enters e^(∎) with the input cursor between the parentheses.
	🟦 E	Enters the Greek letter Ε or ε (e-psilon).
2	$\sqrt{x}$	Returns the square root of x (→ 36 and 81). [327]
	α	Sets *AIM* for entering characters (→ 208ff).
	α.FN	*Menu* of functions operating on *text strings* (→ 213f).
	F	Enters the Latin letter **F** or **f**.
	🟧 $\sqrt{x}$	Enters √(∎) with the input cursor between the parentheses.
	🟦 F	Enters the Greek letter Φ or φ (phi).
3	STO	Stores x in the destination specified (→ 57ff).
	ASN	Assigns an *item* to a key or *softkey*, also in *AIM* (→ 315ff).
	SAVE	Saves all your data in the backup file (→ 253) on the *FD* from where you may recover them entirely or partially.
	G	Enters the Latin letter **G** or **g**.
	🟦 G	Enters the Greek letter Γ or γ (gamma).

R	Keystrokes	Meaning
3	RCL	Recalls a stored object into **X** (→ *57ff*). If pressed in **RBR**, recalls the object at the bottom display row or enters a corresponding program step (→ *289f*) and leaves **RBR**.
	RBR	Calls the *register browser* (→ *289f*).
	VIEW	Views the destination, i.e. displays its address and contents directly below the *status bar* until next keystroke (→ *63*).
	H	Enters the Latin letter **H** or **h**.
	CH	Enters the Latin letters **CH** or **ch**.
	▭ H	Enters the Greek letter **Χ** or **χ** (chi).

3	R↓	Rolls the *stack* contents 1 *level* down (→ *43*).
	R↑	Reverts R↓
	CPX	*Menu* of commands operating on *complex numbers* like **conj**, **cross**, **dot**, and **Re↔Im** (→ *152ff*).
	I	Enters the Latin letter **I** or **i**.
	▼	Makes next character a subscript (if applicable).
	▭ I	Enters the Greek letter **Ι** or **ι** (iota).

3	CC	Closes, composes, cuts, or converts a complex x, depending on context (→ *152ff* and *339*).
	|x|	Returns the absolute (unsigned) value of x (→ *36*) or the magnitude of x. [327]
	∠	Returns the phase of x. [327]
	J	Enters the Latin letter **J** or **j**.
	|x|	Enters $\lvert\,\blacksquare\,\rvert$ with the input cursor between the bars.
	η	Enters the Greek letter **Η** or **η** (ēta).

R	Keystrokes	Meaning
3	🟠	*Prefix* to reach a golden function label. Pressing 🟠 twice will cancel it.
	SNAP	Dumps the current screen into a file on the *FD*, i.e. takes a snapshot or screenshot. → 257 and the *IOI* for more.

3	🔵	*Prefix* to reach a blue function label. Pressing 🔵 twice will cancel it.
	USER	Toggles *user mode* (→ 324f).

4	ENTER↑	Context sensitive key, also in *AIM*, → 339
	FILL	Fills all *stack registers* with x
	DROP↓	Drops x from the *stack* (i.e. forgets x) (→ 43).

4	x≷y	Swaps the contents of **X** and **Y** (→ 43).
	x≷	Swaps x and the contents of the destination specified.
	STK	*Menu* of *stack* related operations (drops, swaps, and shuffles, → 39ff).
	K	Enters the Latin letter **K** or **k**.
🟠	x≷y	Enters the character ≷.
🔵	K	Enters the Greek letter **K** or **κ** (kappa).

4	+/−	If pressed during input of mantissa or exponent, changes its sign (→ 26). Else multiplies x times −1.
	Δ%	Returns $100\ {}^{(x-y)}/_{y}$. Leaves y unchanged; → 85f.
	FIN	*Menu* of financial functions (i.e. various % functions and *TVM*, → 294ff and 342ff).

R	Keystrokes	Meaning
	L	Enters the Latin letter **L** or **l**.
	+/-	Enters the character ±.
	L	Enters the Greek letter Λ or λ (lambda).

4	E	Allows entering an exponent of 10 for convenient entry of very big or very small numbers (→ 26).
	DSP	Changes the number of digits displayed while keeping the format selected (**FIX**, **SCI**, **ENG**, or **ALL**).
	DISP	*Menu* of commands for numeric display formatting like **FIX**, **SCI**, **ENG**, and **ALL** (→ 79ff).
	M	Enters the Latin letter **M** or **m**.
	↑	Makes next character a superscript (if applicable).
	M	Enters the Greek letter Μ or μ (mȳ).

4	←	Context sensitive key, also in *AIM* → 342.
	↶	Undoes the last command executed (→ 56).
	CLR	Calls a *menu* containing commands for clearing (→ 71f).

5	/	If there is a pending question like **Are you sure?**, enters **N** for 'no'. Else divides y by x. For *matrices*, multiplies y times x^{-1}.
	||	Returns $\left(1/x + 1/y\right)^{-1}$.
	MOD	Returns y modulo x, → 193f.
	N	Enters the Latin letter **N** or **n**.
	/	Enters the character /.
	N	Enters the Greek letter Ν or ν (nȳ) .

R	Keystrokes	Meaning
5	7	Enters the digit **7**.
	EDIT	Opens x for editing.
	FIT	*Menu* of operations for *2D* sample statistics: **Σ+**, **Σ-**, **CLΣ**, various covariances, as well as functions for curve fitting, and *MSA* (→ *99ff*).
	O	Enters the Latin letter **O** or **o** (looking like Greek o-mikron).
	f 7	Enters the character **7**.
	g O	Enters the Greek letter **Ω** or **ω** (o-mega).
5	8	Enters the digit **8**.
	USB	Activates the *USB* line to your *PC*.[328]
	MODE	*Menu* of operations for setting *ADM* (→ *132*) or rounding mode and maximum denominator (→ *128ff*). Contains also the link to *DMCP*.
	P	Enters the Latin letter **P** or **p**.
	f 8	Enters the character **8**.
	g P	Enters the Greek letter **Π** or **π** (pi).
5	9	Enters the digit **9**.
	LBL	Enters a label for the *current step* in *PM*.
	RTN	Returns to the caller (→ *218ff*).
	Q	Enters the Latin letter **Q** or **q**.
	f 9	Enters the character **9**.
	g Q	Works like unshifted O above.

[328] This works on the calculator exclusively; thus this label is invisible on the *Simulator*.

R	Keystrokes	Meaning
5	XEQ	If there is an open question like `Are you sure?`, confirms it; else – if in *PEM* – inserts a call to the subroutine with the label specified; else (i.e. in *RUM*) calls the routine with the label specified and starts executing it (→ *218ff*).
	GTO	Goes to the specified location in *PM*.
	FLAG	*Menu* of *flag* commands, most useful in *PEM* (→ *218ff*).
	:	Enters a colon (**:**).
	QU	Enters the Latin letters **QU** or **qu**.

6	X	Multiplies y times x.
	x!	Returns the factorial of x (or $\Gamma(x+1)$ for non-integer x). → *21*, *36*, and *96*.
	PROB	*Menu* containing combinations, permutations, the *gamma function*, random number generators, and all the probability distributions supported (→ *96ff*).
	R	Enters the Latin letter **R** or **r**.
	X	Enters the character **×** or **·**.
	R	Enters the Greek letter **Ρ** or **ρ** (rho).

6	4	Enters the digit **4**.
	Σ	*Menu* of statistical sums accumulated (→ *124*).
	STAT	*Menu* of operations for *1D* and *2D* sample statistics: **Σ+**, **Σ-**, **CLΣ**, various means and measures for scattering, as well as functions for histograms (→ *99ff*). Contains also a link to FIT (→ above).
	S	Enters the Latin letter **S** or **s**.
	4	Enters the character **4**.
	S	Enters the Greek letter **Σ** or **σ** (sigma).

R	Keystrokes	Meaning
6	5	Enters the digit **5**.
	R←	Calls →REC, converting polar coordinates ***r*** (in $\mathbf{X}$) and ***ϑ*** (in $\mathbf{Y}$) to rectangular coordinates ***x*** and ***y*** (→ 145ff).
	→P	Calls →POL, converting rectangular coordinates ***x*** and ***y*** to polar coordinates ***r*** (in $\mathbf{X}$) and ***ϑ*** (in $\mathbf{Y}$). → 22f and 145ff.
	T	Enters the Latin letter **T** or **t**.
	5	Enters the character **5**.
	T	Enters the Greek letter **Τ** or **τ** (tau).
6	6	Enters the digit **6**.
	⏱	Calls the timer application (a.k.a. stopwatch, → 292ff).
	CLK	*Menu* of time and date commands (→ 202ff).
	U	Enters the Latin letter **U** or **u**.
	6	Enters the character **6**.
	ϑ	Enters the Greek letter **Θ** or **ϑ** (theta).
6	▲	Context sensitive key, also in *AIM*, → 341.
	≡▲	Moves the *PP* 1 step back (→ 218ff). Will repeat with $2Hz$ when pressed longer than $0.5s$.
	SF	Sets the flag specified.
7	−	Subtracts ***x*** from ***y***.
	INTS	*Menu* providing the digits **A** … **F**, operations on *integers*, and *integer sign mode* settings. This *menu* is most useful with *short integers* (→ 187, 191ff, and also BIT above).
	PART	*Menu* containing **FP**, **IP**, **sign**, **DECOMP**, etc.

R	Keystrokes	Meaning
	V	Enters the Latin letter **V** or **v**.
	orange −	Enters a minus sign.
	≈	Opens the *menu* of mathematical symbols.

7	1	Enters the digit **1**.
	ADV	*Menu* of advanced operations for solving arbitrary programmed equations, finding roots, integrating, differentiating, computing sums and products (→ *262ff*).
	EQN	*Menu* of all equations currently defined (→ *265ff*). Allows for editing as well as for interactive solving, integrating, and differentiating.
	W	Enters the Latin letter **W** or **w**.
	orange 1	Enters the character **1**.
	ψ	Enters the Greek letter Ψ or ψ (psi).

7	2	Enters the digit **2**.
	MATX	*Menu* of *matrix* operations (incl. $[M]^{-1}$, $\lvert M\rvert$, $[M]^T$, `cross`, `dot`, and the *Matrix Editor*, → *162ff*).
	X.FN	*Menu* of advanced mathematical (extra) functions. → *ReM*.
	X	Enters the Latin letter **X** or **x**.
	orange 2	Enters the character **2**.
	blue X	Enters the Greek letter Ξ or ξ (xi).

7	3	If there is a pending question like `Are you sure?`, enters **Y** for 'yes'. Else enters the digit **3**.
	π	Recalls the first 34 digits of the number **π**.
	CONST	*Catalog* of fundamental physical, astronomical, mathematical, and surveying constants (→ *298ff*).

R	Keystrokes	Meaning
	Y	Enters the Latin letter **Y** or **y**.
	🟧 3	Enters the character **3**.
	🟦 Y	Enters the Greek letter **Υ** or **υ** (y-psilon).

7	▼	Context sensitive key, also in *AIM*, → 341.
	≡▽	Moves the *PP* 1 step forward (→ 218ff). Will repeat with $2Hz$ when pressed longer than $0.5s$.
	CF	Clears the flag specified.

8	+	Adds x to y.
	LOOP	*Menu* containing **INC**, **DEC**, and the related loop control commands **ISG**, **DSE**, etc. (→ 233f).
	TEST	Opens the *menu* of binary tests (→ 230ff).
	Z	Enters the Latin letter **Z** or **z**.
	🟧 +	Enters the character **+**.
	🟦 Z	Enters the Greek letter **Ζ** or **ζ** (zeta) .

8	0	Enters the digit **0**.
	I/O	*Menu* of I/O-related operations (→ 253f).
	🖨	*Menu* of commands for printing, empty for the time being.
	?	Enters a question mark.
	🟧 0	Enters the character **0**.
	🟦 🖨	Enters the printer character (🖷).

R	Keystrokes	Meaning
8	[.]	Usually enters a decimal radix mark (→ 26). If pressed twice in numeric input, allows for entering a fraction directly (→ *75f* and *128ff*). In *register* or *flag* addressing, [.] heads a local address (→ *61ff*).
	SHOW	Shows the number (or string) ***x*** with its maximum displayable precision (or length) until next keystroke.
	INFO	*Menu* of commands returning system information (→ the *IOI* and *ReMS 2*), being most useful in *PEM* (→ *218ff*).
	[,]	Enters a comma.
	[f] [,]	Enters a point.
	[g] [.]	Opens a character *menu* of punctuation marks etc.

8	R/S	Context sensitive key, → *341*.
	P/R	Toggles *PEM* and *RUM* (→ *24* and *218ff*).
	P.FN	*Menu* of dedicated programming functions, being of most use in *PEM* (→ *218ff*).
	[0]	Enters a blank space character.
	[1]	Enters an exclamation mark.
	[g] [0]	Free xxx

8	EXIT	/ ON : Context sensitive key, also in *AIM*, → *340*.
	CATALOG	*Catalog* of everything, be it provided by us or created by you (functions, *variables*, *menus*, programs, constants, etc.). → *ReMS 2* for the structure and contents of CATALOG. Also learn there how to pick a specific item from it the fast way.
	OFF	If in *PEM*, inserts the command `OFF` behind the *current step* (→ *220*). Else turns your *WP43* off.

Context Sensitive Keys

7 context sensitive keys need longer explanations – find them below, sorted alphabetically. If any of these keys is pressed, your *WP43* will run top down through a sequence of specific tests – for the first test becoming true, your *WP43* will execute the corresponding operation and return, waiting for your next input.

Key	Condition(s)	Meaning
CC	**X** contains an open (input) number, → 26	For ~~POLAR~~, CC closes input, checks, and saves it as *real* part of a forthcoming *complex number*, then waits for your input of its *imaginary* part. Else CC closes input, checks, and saves it as magnitude, and waits for your phase input. → 152ff.
	X contains a closed *complex number*, *vector*, or *matrix*	For ~~POLAR~~, CC splits ('cuts') x into its *real* and *imaginary* parts, returning the *real* part in **Y** and the *imaginary* part in **X**. Else CC splits x into its magnitude r and phase ϑ, returning r in **Y** and ϑ in **X**.
	X and **Y** contain 2 closed *reals*	Interprets y and x either (for POLAR) as magnitude and phase, or (for ~~POLAR~~) as *real* and *imaginary* parts. CC combines y and x like a dyadic function, composing 1 *complex number* x.
	X and **Y** contain 2 closed *real matrices* of identical dimension	Returns 1 *complex matrix* x, working in analogy to previous row.
	Else	Throws an error.
ENTER↑	Waiting for parameter input	Closes pending command input and executes said command.
	Asking for confirmation	Confirms the question.
	In TIMER	Honored as described on pp. 292f.
	…	

Key	Condition(s)	Meaning
	In numeric input or *AIM*	Closes input and pushes it either … • on the *stack* (→ *37f* and *43*) or • in the current matrix element (→ *168*) or • in the pending program step.
	In *PEM*	Inserts **ENTER↑** behind the current program step.
	Else	Copies x into Y.
EXIT / ON	*WP43* turned off	Turns your *WP43* on.
	Waiting for numeric, text, or parameter input	If an input *menu* is open, closes this *menu*. Else if waiting for parameter input, cancels the pending command. Else closes input (and *AIM*, if applicable).
	TI displayed (→ *74f*)	Clears *TI* returning to the calculator state as it was before this *TI* was thrown.
	Asking for confirmation	Denies the question.
	In **RBR**, **STATUS**, **TIMER**	Leaves the application (→ *289ff*).
	In a *sub-menu*	Leaves the current *sub-menu* returning to its parent *menu*.
	In a *menu* or *browser*	Leaves the current *menu* or *browser* without executing anything, returning to the status of your *WP43* as it was before.
	⌛ after special commands	Aborts the execution of a running **FACTOR**, **NEXTP**, **SOLVE**, or ∫.
		Stops executing the running program after completing the current program step. will be lit until next keystroke.
	In *PEM*	Leaves *program entry mode* like P/R.
	Pressed > 1 s	Calls **EXITALL**.
	Else	Does nothing.

Key	Condition(s)	Meaning
R/S	In TIMER	Starts or stops the timer without changing its value (→ 292f).
		Acts like EXIT does above.
	In *AIM*	Inserts a blank character (see above).
	In *PEM*	Enters STOP after the current program step.
	Else	Runs the *current routine* (→ 218ff) or resumes its automatic execution starting with the step after the *current step*.
▲ or ▼	After STO or RCL	Honored as described on pp. 62ff.
	In RBR, STATUS, TIMER	Honored as described on pp. 289ff.
	In ASSESS	Honored as described in the *IOI*.
	In ASSIGN	Honored as described on pp. 315ff.
	A	▼ sets lower case and continues testing.
	α	▲ sets upper case
	In EQN	▲ goes to next or … ▼ to previous equation, if applicable.
	In a *multi-view menu*	▲ goes to next or … ▼ to previous *view* in the current *menu*.
	In *PEM*	▲ goes to previous or … ▼ to next program step. Will repeat with 2*Hz* when pressed > 0.5 s.
	In *RUM*	Browses the *current routine* with… ▲ going to previous program step or … ▼ executing the *current* step and proceeding to next step (→ 229). Does nothing for empty *PM*.

Key	Condition(s)	Meaning
⬅	Open command parameter input	Deletes the last character entered. If none is left, ⬅ cancels the pending … …command like EXIT.
	Open (alpha-) numeric input	… input (→ 26).
	TI displayed	Acts like EXIT does above.
	Asking for confirmation	Denies the question.
	In TIMER	Resets the timer (→ 292f).
	In *PEM*	Deletes the *current* program step.
	Else	Calls the command CLX.

Virtual Keyboards

Virtual keyboard in *TAM* for comparisons, STO and RCL, addressing *flags*, etc. (→ 61ff).

Virtual keyboard in *TAM* for the commands **#** and **→INT** (→ *185f*).

Virtual keyboard in *TAM* for addressing *labels* (with **GTO**, **LBL**, **XEQ**, etc.; → *226*).

Virtual keyboard for inserting characters in *AIM*, e.g., for texts and equations (→ 209, especially for explanation of the 3 *alpha menus* and the Greek letters supported by your *WP43*).

There are further *virtual keyboard*s effective in **FBR** and **RBR** (→ 289ff) and **TIMER** (→ 292ff).

Menu Index

This lists all *menu* labels printed on the keyboard, sorted alphabetically. Find a short description of each *menu* on the page indicated – proceed from there for further information.[329] The 3 *menus* accessible exclusively in *AIM* are printed on light yellow background – CATALOG may be called in all modes:

[329] Full contents of each *menu* (and its *submenus*, if applicable) are found in *ReMS 2*.

APPENDIX 2: OPERATOR PRECEDENCE

Since your *WP43* will always execute just 1 operation at a time (→ 52) it doesn't have to care for operator precedence. Hence it is your job to control the sequence of operations you present to your *WP43*. There are common rules and conventions in mathematics dealing with that – you have learned them in school. Here is just 1 **example** for affirmation and/or reminding:

$$1 - 2 \cdot 3^4 : 5 + \sin 2\left(6 - \sqrt[3]{7^2}\right) \cdot 8! + ln\left(-9^{2^3} \cdot 45^{(6/7)}\right)^2$$

or, written for another part of this world needing more space:

$$1 - 2 \times 3^4 \div 5 + \sin 2\left(6 - \sqrt[3]{7^2}\right) \times 8! + ln\left(-9^{2^3} \times 45^{(6/7)}\right)^2$$

This expression may be solved the following way, for instance, using your *WP43* in *SDC*:

9 ENTER↑ 2 ENTER↑ 3 EXP y^x	
y^x +/−	returns -9^{2^3}; note the arguments automatically fill in correctly.
6 ENTER↑ 7 / 45 $x\gtrless y$ y^x	calculates $45^{(6/7)}$.
× x^2 ln	solves the rightmost (4th) term.
7 x^2 $\sqrt[3]{}$ 6 $x\gtrless y$ − 2 × TRI sin	solves the sine.
8 x! ×	solves the 3rd term.
+	adds it to the 4th.
3 ENTER↑ 4 EXP y^x 2 × 5 /	solves the 2nd term.
−	subtracts it from said sum.
1 +	returns the overall solution 3 300.988 666 858 076

The colors indicate the 3 *stack registers* employed for this solution (→ 42ff). Note $x\gtrless y$ is used twice to swap arguments.

APPENDIX 3: FURTHER APPLICATIONS OF TVM

First, let's use *TVM* to solve a very old problem, before cash-flows and pocket calculators were even thought of:

Example:

The inventor of chess was asked by his king for a wish as reward. He is said to have requested 1 grain of rice on the 1st field, 2 grains on the 2nd, 4 on the 3rd, 8 on the 4th, etc. until the 64th field. The king is reported to have laughed about that modest wish. How much rice had the king to lay out in total after all?

Solution:

FIN	TVM End		since grains are laid out at the end of each field,
1	PV	1.00	grain in the 1st field,
64	n_{PER}	64.00	fields in total
100	i%/a	100.00	% interest rate per field,
1	per/a	1.00	for obvious reasons.
0	PMT	0.00	no rice before field 64.
	FV	$-1.844\,674\,407\,370\,96\times10^{19}$	grains had to be laid out. How much is that?

Nowadays, 1 grain of rice weighs about 65 mg. At the time of this legend, rice was smaller and lighter, let's assume 10 mg per grain. So

$$9.2 \times 10^{18} \times 20 \times 10^{-6}\ \text{kg} \approx 184 \times 10^{12}\ \text{kg} = 184\ \text{Gt}\ (gigatons),$$

meaning enough for the inventor, his wife, their kids and grandkids … including their families … and their villages … for all their lifes by far (the total amount is a major multiple of the global rice production of today).

It is told the king never again asked an inventor for a wish. Other sources report he forced the inventor to count his reward.[330]

Consonant with the *cash flow convention* on p. 295, throughout *TVM* pictures, amounts received are represented by arrows pointing up, money laid out (paid,

[330] There are easier ways to compute said total amount of rice, but this is the easiest using *TVM*. Calculating exactly, the inventor should get 18 446 744 073 709 551 615 grains of rice. Assume counting 5 grains per *second*, how long will it take to count them all?

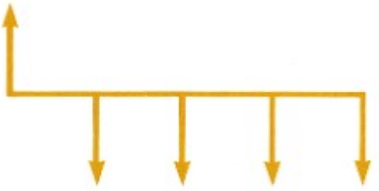
invested) by arrows pointing down. Various types of financial problems can be sketched like this then:[331]

Generalized Net Cash Flow Diagrams and Terminology

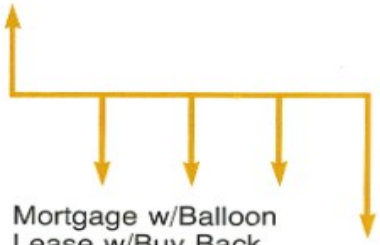
(Note that diagrams involving payments may be represented with payments at the beginning or end of the period.)

Compound Growth
Savings Account
Appreciation

Savings Plan
Sinking Fund
Pension Fund
Annuity (series of payments)



Mortgage
Lease
Direct Reduction Loan
Direct Reduction Installment
Amortization
Annuity



Mortgage w/Balloon
Lease w/Buy Back
Lease w/Residual
Annuity

All pictures as well as the text printed blue throughout this appendix are quoted from the *HP-27 OH*. All calculations are executed in **FIX 2**. Enjoy the boundary conditions of that time gone for long – those were the days ...

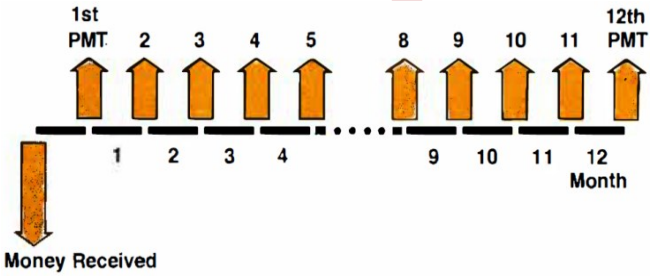
[331] *TN*: You can use this picture and the two chapters following as a tiny dictionary of financial American English which may be hardly understood otherwise.

(Above, the word "with" is abbreviated "w/" although this doesn't save any space at all. Abbreviamania …)

Ordinary Annuities (a.k.a. Payments in Arrears)

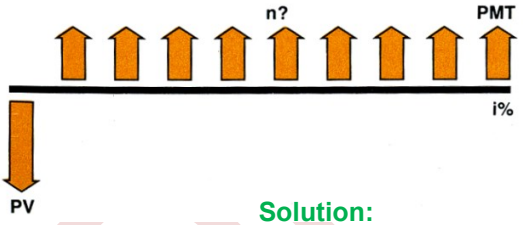
An *annuity* is a series of equal payments made at regular intervals. The time between annuity payments is called the payment interval or payment period. If your payment is due at the end of each payment period, it's called an *ordinary annuity* or *payment in arrears*. Examples of ordinary annuities are a car loan (where you drive away now and pay later) or a mortgage (where the payments start one month after you get your loan).

The time/money relationship for an ordinary annuity with monthly payments for a year would look like this → [332]



Example for finding the number of periods for an ordinary annuity:

Through an insurance fund, you have accumulated $50 000 for your retirement. How long can you withdraw $3 000 every 6 months (starting 6 months from now) if the fund earns 5% per annum compounded semiannually?



Solution:

FIN TVM End		withdrawals are due at the end of each period,
0 FV	0.00	value at the end,
−50000 PV	−50 000.00	principal (i.e. capital) for your …
3000 PMT	3 000.00	payments you withdraw semiannually,
5 ENTER↑ 2 / i%/a	2.50	% semiannual interest rate,
1 per/a	1.00	since you want to know the number of semiannual withdrawals.
n_{PER}	21.83	your savings will last for almost 11 years this way.

[332] *TN*: "Money received" by the bank or insurance – it meant the amount you lay out at the beginning.

Example 1 for finding the interest rate for an ordinary annuity:

What is the annual interest rate (a.k.a. *APR* for *annual percentage rate*) on a 2-year, $1 775 loan with $83.65 monthly payments? (Continue with the settings of previous case):

Solution:

12 **per/a**	12.00	months per year,
2 **x** n_{PER}	24.00	periods in total,
−1775 **PV**	−1 775.00	principal (capital),
83.65 **PMT**	83.65	payment;
i%/a	12.11	% *APR*.

Borrowers are sometimes charged fees related to the issuance of a mortgage, which effectively raises the interest rate. Given the basis of the fee charge, the true annual percentage rate may be calculated.

Example 2 for finding the interest rate for an ordinary annuity:

A borrower is charged 2 *points* for the issuance of his mortgage. If the mortgage amount is $50 000 for 30 years, and the interest rate is 9% per year, with monthly payments, what annual percentage rate is the borrower paying? (1 *point* is equal to 1 % of the mortgage amount.) (Continue with ***FV*** = 0 …)

Solution:

First, compute the payment amount which is based on $50 000

9 **i%/a**	9.00	annual interest rate,
12 **per/a**	12.00	months per year,
30 **x** n_{PER}	360.00	periods in total,
50000 **PV**	50 000.00	principal (capital);
PMT	−402.31	payment (will be reused).
RCL PV **.98** **x** **PV**	49 000.00	effective amount received,
i%/a	9.23	% effective *APR*.

What's really happening? For a mortgage with fees, the borrower is making payments on the original loan amount, which corresponds with the initial calculation of the payment amount. If you borrow $10 000, but are immediately charged $500 in fees, you really only receive $9 500. But, your payments are

based on $10 000. With fees, then, you're really paying the same for less money, which generates the need to compute the true *APR*.

Example for finding the payment amount for an ordinary annuity:

Find the monthly payment amount on a 30-year, $52 000 mortgage at 9.75% annual interest rate. (Continue with the settings of previous case, so ***per/a*** and n_{per} are specified correctly already.)

Solution:

52000 PV	52 000.00	mortgage,
9.75 i%/a	9.75	% annual interest rate;
PMT	−446.76	monthly payment.

A common financial occurrence is an annuity that has a large payment at the end. The last payment – usually considerably larger although it could also be smaller than the others – is called a *balloon payment* or *balloon*.

By subtracting the present value of the balloon payment from the loan amount, the problem effectively becomes "What is the monthly payment on a direct reduction loan?"

Example (finding the payment for an ordinary annuity with *balloon*):

Yellowstone Sam is heading north, and will invest in an $8 000 dog sled and team. His loan specifies 60 monthly payments at 10% with a *balloon payment* in the 60th month of $3 000. What will his monthly payments be?

Solution:

12 per/a	12.00	months per year,
60 n_{PER}	60.00	payment periods in total,
10 i%/a	10.00	% annual interest rate;
−3000 FV	−3 000.00	future value of *balloon*,
0 PMT	0.00	no payments first hand,
PV	1 823.37	present value of *balloon*;
8000	8 000.00	gross value of loan amount,
x≷y − PV	6 176.63	net present value of loan amount less *balloon*,
0 FV PMT	−131.24	monthly payment.

Sam will have to pay 10 840.40 US$ in total. Sounds expensive.

Example for finding the present value of an ordinary annuity:

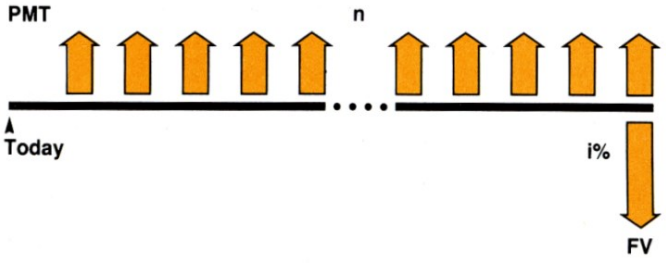
Thus, *Yellowstone Sam* decides to purchase a snowmobile. He plans to pay $80 per month for 3 years, and he's willing to pay 10% annual interest. How much can he afford to pay for the snowmobile? (Continue…)

Solution:

12 per/a	12.00	months per year,
3 × n_{PER}	36.00	payment periods in total,
10 i%/a	10.00	% annual interest rate;
−80 PMT	−80.00	monthly payment,
PV	2 479.30	price he can pay for the snowmobile. It will cost him 2880 US$.

With loan calculations, you generally solve for *n*, *i*, *PMT*, or *PV*. There is another type of ordinary annuity called a "*sinking fund*", where you make payments at regular intervals into a fund to discharge a debt (for example, to pay off a bond issue at maturity). With *sinking fund* calculations, you solve for *n*, *i*, *PMT*, or *FV* (how much you will have in the fund at a future date).

Sinking fund payments start at the end of the 1st period, like sketched here. →



This is different from opening a savings account with a starting deposit today. Savings are annuity due calculations and will be described later in this section.

Example for finding the future value of an ordinary annuity:

A $100 000 bond is to be discharged by the sinking fund method. If, starting 6 months from now, you deposit $3 914.75 twice a year into a sinking fund that pays 5% compounded semiannually, will you be able to pay off the bond in 10 years?

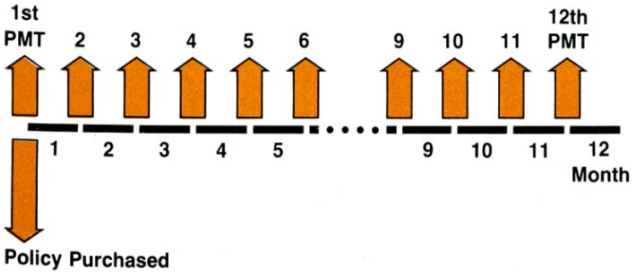
Solution:

2 per/a	2.00	halves per year,
10 × n_{PER}	20.00	payment periods in total,
5 i%/a	5.00	% annual interest rate;
0 PV	0.00	start value,
−3914.75 PMT	3 914.75	semiannual deposit,
FV	100 000.95	balance of the fund after 10 years – it will just make it!

Annuities Due (a.k.a. Payments in Advance)

With some annuities – like insurance premiums or a lease – the payment is due at the beginning of the month. This is called an *annuity due* because the payment falls at the beginning of the payment period. Other terms are *payments in advance* or *anticipated payments*.

An *annuity due* with monthly payments for a year – say, a car insurance policy[333] – looks like sketched here.

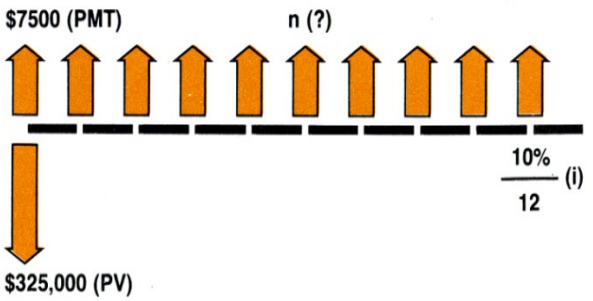


Notice that with an annuity due, you have a payment right away at the beginning of the first interval (with an ordinary annuity, your payment is not due until the end of the first period, but you also have a payment at the end of the entire term).

The following calculations all deal with *annuity due* problems, e.g. savings, insurance, leases, and rents.

[333] *TNG*: *Policy* entspricht hier einer *Police*.

Example 1 for finding the number of periods for an annuity due:



Given an investment possibility of $325 000 that will immediately produce rental income of $7500 per month, how long must the investment be held to yield 10% per annum?

Solution:

FIN TVM Begin		payments are due at begin of each period,
12 per/a	12.00	months per year,
10 i%/a	10.00	% annual interest rate,
−325000 PV	−325 000.00	investment;
0 FV	0.00	final balance;
7500 PMT	7500.00	monthly rental income;
n_{PER}	53.43	months.

Example 2 for finding the number of periods for an annuity due:

If you deposit $50 a month in a savings account that pays 6% interest, how long will it take to reach $1 000? (Continue with as many settings of previous case as possible.)

Solution (***p*** and ***per/a*** are taken over):

6 i%/a	6.00	% annual interest rate,
0 PV	0.00	start balance,
1000 FV	1 000.00	future value;
−50 PMT	−50.00	monthly payments to the bank.
n_{PER}	19.02	months.

Example for finding the interest rate for an annuity due:

Equipment worth $12 000 is leased for 8 years with monthly payments in advance of $200. The equipment is assumed to have no salvage value at the end of the lease. What yield rate does this represent?

Solution:

12 per/a	12.00	months per year,
8 ⓧ n_{PER}	96.00	payment periods in total,
12000 PV	12 000.00	start value of equipment,
−200 PMT	−200.00	payments;
0 FV	0.00	salvage value after 8 years.
i%/a	13.07	% annual yield.

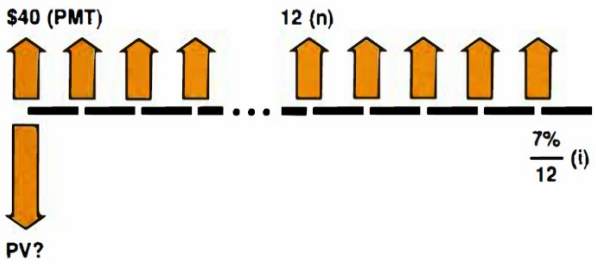
Example for finding the payment amount for an annuity due:

The owner of a building presently worth $70 000 intends to lease it for 20 years at the end of which time he assumes the building will be worthless (i.e., has no residual value).[334] How much must the quarterly payments (in advance) be to achieve a 10% annual yield?

Solution:

4 per/a	4.00	quarters per year,
20 ⓧ n_{PER}	80.00	payment periods in total,
10 i%/a	10.00	% annual target yield,
−70000 PV	−70 000.00	present value of the building;
0 FV	0.00	residual value after 20 years.
PMT	1982.27	quarterly payments.

Example for finding the present value for an annuity due:



The owner of a downtown parking lot has achieved full occupancy and a 7% annual yield by renting parking spaces for $40 per month payable in advance.

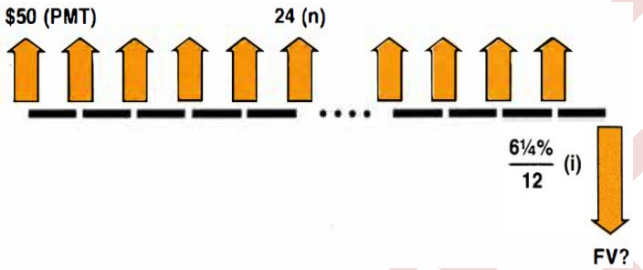
Several regular customers want to rent their spaces on an annual basis. What annual rent, also payable in advance, will maintain a 7% annual yield rate?

[334] *TN*: Feel free to deduct building quality or the present state of this (chipboard?) building from this problem setting, whatever you prefer. Buildings live longer elsewhere.

Solution:

12 per/a	12.00	months per year,
12 [335] n_{PER}	12.00	payment periods in total,
7 i%/a	7.00	% annual target yield,
40 PMT	-40.00	monthly payments,
0 FV PV	-464.98	equivalent annual payment.

Example for finding the future value for an annuity due:



If you can afford to deposit $50 per month in an account with 6 ¼ % interest compounded monthly, how much will you have 2 years from now?

Solution: [336]

12 per/a	12.00	months per year,
2 [x] n_{PER}	24.00	payment periods in total,
6.1.4 i%/a	6.25	% annual target yield,
50 PMT	50.00	monthly payments;[337]
0 PV FV	-1 281.34	balance after 2 years.

[335] You have to key in these digits here again although they are present in **X** already – else *TVM* will try to solve for n_{PER}.

[336] *TN:* The signs or arrow directions are debatable here, in my opinion. Seems *HP* changed their point of view.

[337] For translating the examples of this appendix to 2026, you should multiply all monetary amounts given times 5.88.

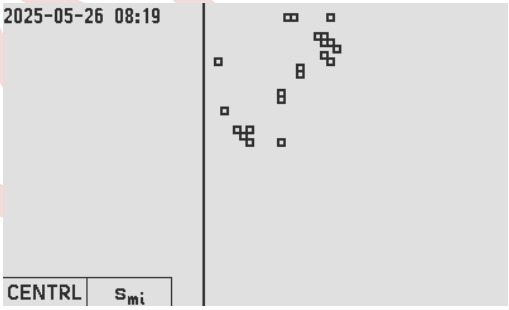
APPENDIX 4: MORE ABOUT PLOT, ASSESS, AND 2D DATA

You are free to apply **PLOT** also to sets of data points not representing pairs of closely correlated measurements as were described in *OMS 2*. The following real-world **example** is a set of daily high and low temperatures recorded in *degrees Fahrenheit* for the first 25 days of May 2025 at Laredo (Texas, *USA*):

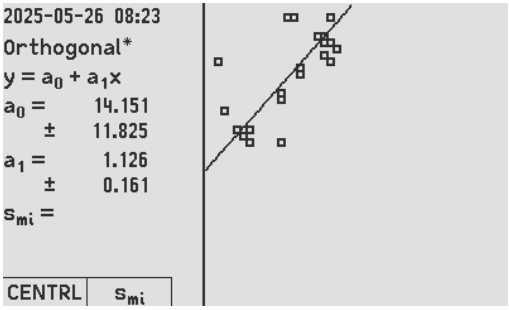
97	92	85	85	96	101	93	86	87	87	90	98	105
74	71	66	71	74	79	71	65	64	66	62	61	72

105	102	101	101	98	102	99	102	101
73	77	78	79	79	77	78	78	78

With these data keyed into STATS by **Σ+**, **PLOT** will display:

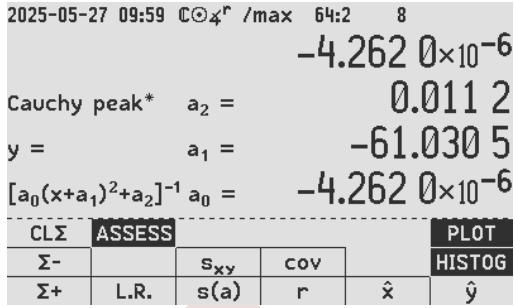


… and **CENTRL** will return:

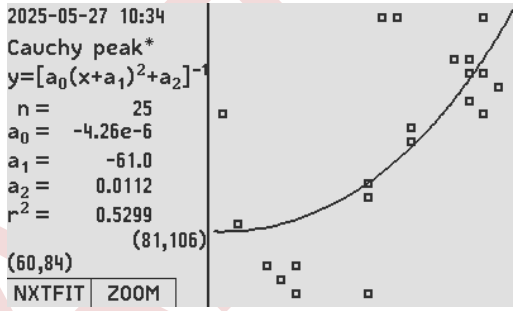


Note that no s_{mi} is computed just since this set contains less than 30 data points – your *WP43* cannot know it would also make absolutely no sense here, that assessment is your task.

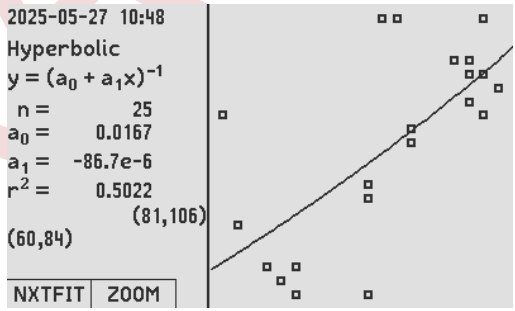
Being at it, you can as well check the correlation between highs and lows. If you allow your *WP43* to apply all fit models (**BestF 000**) and call **L.R.**, it will return:



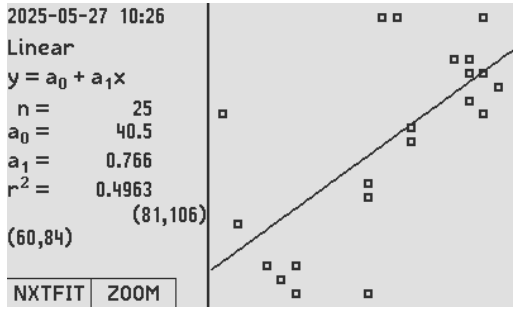
… which you had hardly expected, I guess. Let's call **ASSESS** to check the fit:



Though this fit model doesn't make any sense for your data. Just browse the results for all the other models via **NXTFIT** or ▼ and find the hyperbolic delivering the greatest r^2 of all 2-parameter models for your data:



But its results are actually not far away from linear, so you might choose this model for your publication (although its coefficient of determination isn't really breathtaking):



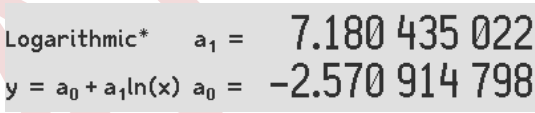
And we can also help *Pia* eventually assessing her bike gearing. Remember (→ 85):

Gear	1	2	3	4	5	6	7	8	9	10	11	12
d / r_c	1.660	1.915	2.165	2.490	2.873	3.247	3.734	4.330	4.979	5.858	6.639	7.660

Enter the gears and the respective distances per crank revolution into STATS:

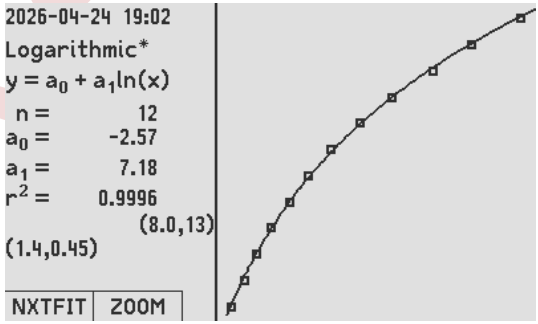
FIT CLΣ
1 ENTER↑ 1.66 Σ+
2 ENTER↑ 1.915 Σ+
…
12 ENTER↑ 7.66 Σ+

▼ BestF 0 ENTER↑
▼ L.R. returns



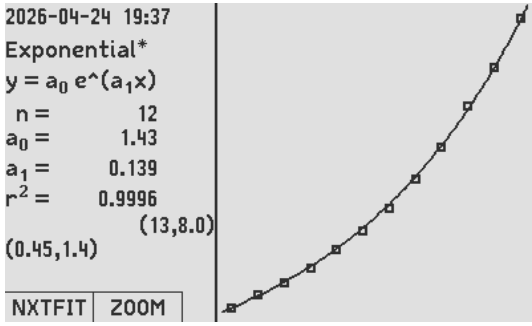
Let's look at the data plotted together with the curve:

▭ ASSESS displays →



… showing an excellent correlation coefficient. *Pia* could be satisfied with her choice – but she wants to see the diagram with ***x*** and ***y*** swapped. Oh dear! Though not worth a discussion … just enter

EXIT
MATX INDEX VAR STATS
1 ENTER↑ 2
▼ C⇄C swaps ***x*** and ***y***
FIT L.R.
▭ ASSESS displays →



APPENDIX 5: MORE GRAPHICS

In *OMS 2* and *App. 4*, graphic output via **ASSESS** and **PLOT** was shown. Furthermore, your *WP43* features the elementary graphic commands **CLLCD**, **AGRAPH**, **PIXEL**, and **POINT**. The first 3 are inherited from *HP-42S*; the latter 3 are stored in P.FN'P.FN2.

CLLCD clears all pixels with $x \geq \mathrm{IP}(x)$ and $y \geq \mathrm{IP}(y)$, i.e. the entire screen northeast of the point specified. Input of [**0**; **0**; …] will erase the whole screen.[338]

AGRAPH ***s*** puts a vertical pattern specified by a 64-bit *short integer* in the source ***s*** to the *LCD*.[338] The point with coordinates $x = \mathrm{IP}(x)$ and $y = \mathrm{IP}(y)$ is the bottom pixel of this column (valid inputs are $0 \leq x < 399$ and $0 \leq y < 175$). **AGRAPH** increments ***x*** automatically by 1 which is handy (→ instance below). So one 64 px high graphic row starting in column 0 may require 400 such *integers* – the more blank space is found in this row the less *integers* are needed for specifying it.

PIXEL sets the pixel with coordinates $x = \mathrm{IP}(x)$ and $y = \mathrm{IP}(y)$. The origin, i.e. pixel (0; 0) is located in the bottom left corner of the screen, the pixel (399; 239) top right.

POINT sets a square of 3×3 pixels around the coordinates $x = \mathrm{IP}(x)$ and $y = \mathrm{IP}(y)$. This is significantly better visible than just a single pixel.

Since most keyboard commands refresh the screen immediately, graphics should be generated by routines. **CLLCD**, **PIXEL**, and **POINT** are self-explanatory. **AGRAPH** may need a little demonstration:

Example:

This little program draws an open square with its bottom left corner at (150; 100). It may be provided on your *WP43* already. Press

GTO **PROG** **PICTURE**
P/R and read (or key in):

[338] This functionality differs from the one known from *HP-42S*.

```
LBL 'PICTURE'
   REM 'initializing'
   FF FF FF FF FF FF FF FF16
   STO 00
   C0 00 00 00 00 00 00 03 16
   STO 01
   62
   STO 02
   100
   ENTER↑
   150
   CLLCD
   INC X
   AGRAPH 00
   AGRAPH 00
LBL 01
   REM 'drawing loop'
   AGRAPH 01
   DSE 02
   GTO 01
   REM 'finalizing'
   AGRAPH 00
   AGRAPH 00
   RTN
```

set up 2 pattern registers for vertical …

… and horizontal framing

initialize a loop counter

erase the screen NE of (150; 100)

draw left frame
… 2 pixels wide

loop: draw 1 point of top and bottom frame
loop: decrement the loop counter
loop: if > 0 then run this loop again

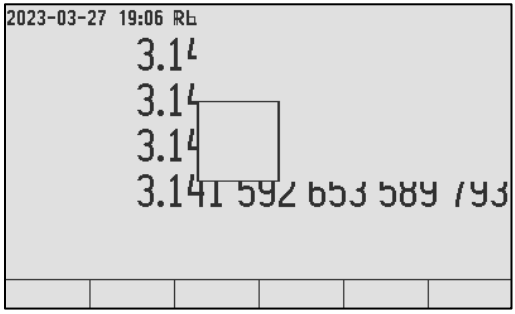
else draw right frame like above

Now do this:

P/R leave *PEM*

π FILL fill the screen

XEQ PROG PICTURE
and you will get a picture like this:



➡ The *status bar* may overwrite the top 20 pixels. Calling *menus* or pressing EXIT will overwrite the *menu section* – the whole graphic, however, will disappear without any trace when you call a function like ENTER↑, updating the *stack*. Add **SNAP** to your routine if you want a more permanent output of your graphic (→ 257).

Another **example** (you might find it provided in your *WP43* as well) demonstrates what you can do with this limited graphic command set:[339]

GTO PROG SPIRAL

P/R and read (or key in):

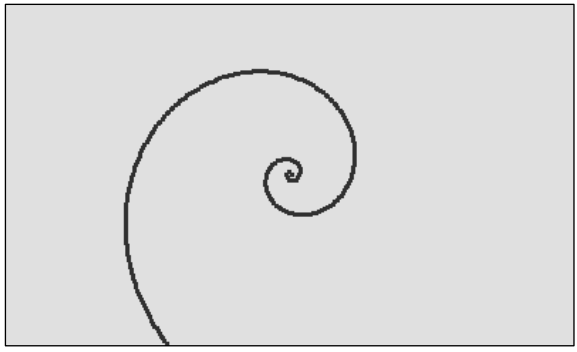
```
LBL 'SPIRAL'
   RCL I
   x=0.?          is there a nonzero value i provided?
   1.             default for i
   STO I
   RCL J
   x=0.?          is there a nonzero value j provided?
   0.3            default for j
   |x|            ensure positive j
   STO J
   RAD
   LocR 04        allocate 4 local registers to this routine
   1
   STO .00        initialize the counter r.00
   0
   ENTER↑
   CLLCD          clear the entire LCD (i.e. everything 'northeast' of (0; 0))
   REM 'start of drawing loop'
LBL 01
   RCL .00
   500.
   /              start with 0.002 in first run
   ENTER↑
   ENTER↑         [r.00/500,  r.00/500,  r.00/500,  …]
   sin
   x⇄y
   RCL× J         [j×r.00/500,  sin(r.00/500),  r.00/500,  …]
   e^x
   ×              [e^(j×r.00/500)×sin(r.00/500),  r.00/500,  …]
   RCL× J
   STO .01        [j×e^(j×r.00/500)×sin(r.00/500),  r.00/500,  …]
   x⇄y
   ENTER↑         [r.00/500,  r.00/500,  j×e^(...))×sin(…),  …]
   cos
   x⇄y
   RCL× J         [j×r.00/500,  cos(r.00/500),  j×e^(...))×sin(…),  …]
```

[339] Kudos to *Jaco*.

```
   e^x
   ×          [e^(j×r.00/500)×cos(r.00/500),  j×e^(...))×sin(...),  ...]
   RCL× I
   STO .02    [i×e^(j×r.00/500)×cos(r.00/500),  j×e^(...))×sin(...),  ...]
   →POL
   STO .03    store the radius
   234        limit value
   x≤? Y      if radius y is beyond the limit
   RTN          then return
   RCL .01        else continue with y
   120
   +          add the center coordinate for y
   ROUNDI     for an integer number of pixels
   RCL .02    recall x
   200
   +          add the center coordinate for x
   ROUNDI     for an integer number of pixels
   SF 'IGN1ER' ignore 1 error
   POINT        while drawing the next 3x3 px. point at (x; y)
   CF 'IGN1ER' watch errors thereafter again
   PAUSE 00   update display
   RCL .03    recall the radius
   1/x
   SDL 03     multiply times 1000
   IP
   INC X
   STO+ .00   add this to r.00
   GTO 01     run this drawing loop again until above limit is reached
END
```

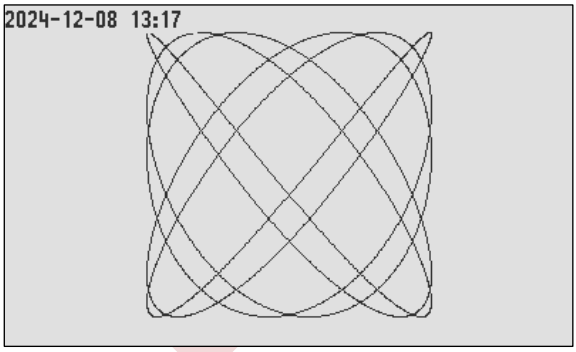
Now press [P/R] to leave *PEM*, then enter

[XEQ] [PROG] SPIRAL and wait (for ~60 s with power supplied by the battery only). You will get a picture like this:



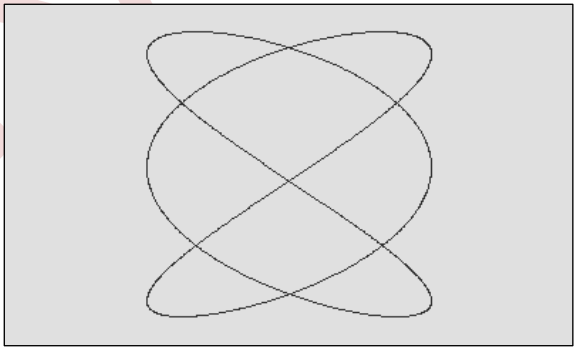
Another graphic program **example** provided is **LISSAJ**. It is fed by the *registers* **I**, **J**, and **K**.

With ***i*** = 5, ***j*** = 6, and ***k*** = 3,
XEQ **PROG** **LISSAJ** will return this:



Date and time will be displayed while drawing since **LISSAJ** will take longer than 1 *minute* with these input values on your *WP43*. You can accelerate it by setting FASTFN.

With ***i*** = 3, ***j*** = 2, and ***k*** = 1 you will get this picture instead:



Check the code (53 steps taking 121 *bytes* in total) for the reasons. Feel free to enter also non-integer values for ***i***, ***j***, or ***k***.

You will find more programs in `testPgms.bin`. Feel free to try them.

APPENDIX 6: SOUND

Beyond **BEEP** and **TONE** which are known from *HP-42S*, *WP43* features 6 new commands operating on the buzzer built in:

BUZZ allows for playing a tone of defined frequency (in *hertz*) and duration (in *milliseconds*); → *ReMA A* for the buzzer specs;

PLAY plays a sequence of **BUZZ** tones stored in a 2- or 3-column matrix of arbitrary size, each tone specified by its frequency and duration (and optionally its volume);

VOL sets the buzzer to a defined volume (1 … 11), while

VOL+ increments the buzzer volume by 1 step and

VOL- decrements it;

?VOL returns the buzzer volume currently set.

All buzzer commands work on the calculator hardware only. Please see the *IOI* for more information about them and their parameters.

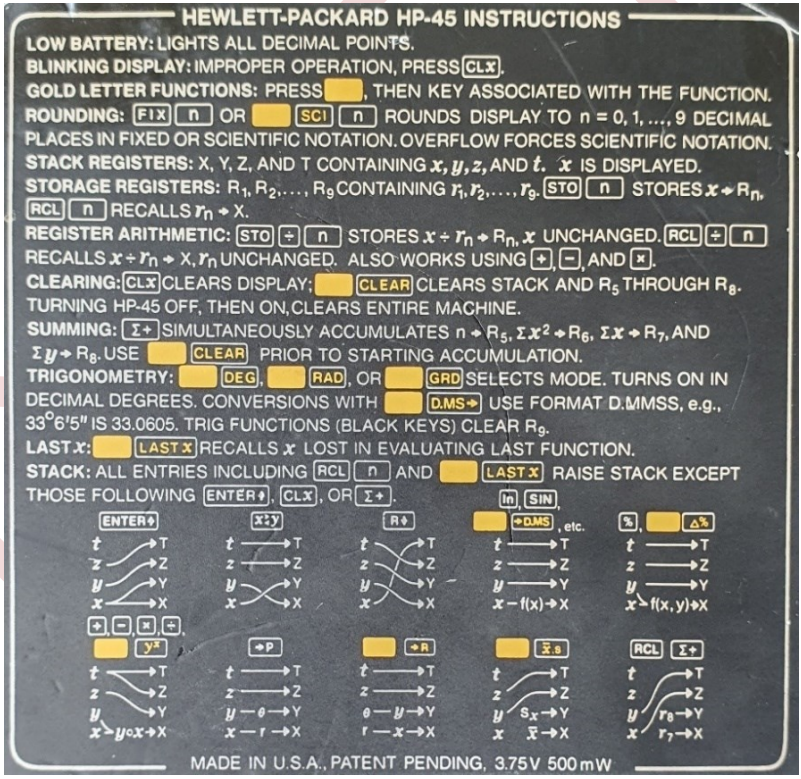
Enjoy playing!

APPENDIX 7: POWER SUPPLY

Your *WP43* is powered by a single *CR2032* coin cell (3 V). It is not rechargeable. Insert it with + facing up (away from the board). With a good battery installed, your *WP43* may be also powered through its *USB* port – running at even higher speed then. When the battery becomes low, replace it with a new quality cell (→ *17f* and *ReMA G*).

WARNING: Removing the battery for longer than some *seconds* may erase all your data and programs in *RAM* – thus, [SAVE] them before!

See here what sufficed for explaining the basic functionality of *HP's* first 'advanced scientific' calculator, the *HP-45* of 1973, on its back:



Though it featured only 59 built-in functions, 9 volatile storage registers including 4 statistical sums, x̄ and s – neither *menus*, *catalogs*, browsers, *data types*, *variables*, regressions, diagrams, advanced operations, applications, keystroke programming, nor keyboard customizing – but **your *WP43* provides them all and much more**. And you can even expand its functionality by including your own routines.

APPENDIX 8: TIME LINE OF QUOTED MANUALS

HP-35 OM	1972
HP-45 OH	1973
HP-55 OH, *HP-21 OH*, *HP-25 OH*	1975
HP-27 OH, *HP-67 OHPG*	1976
HP-97 OHPG, *HP-32 OH*, *HP-33 OH*	1978
HP-34C OHPG	1979
HP-41C/41CV OHPG, *Solving Problems With Your HP Calculator*	1980
HP-16C OH, *HP-15C AFH*	1982
HP-15C OH	1985
HP-22S Science Student Applications, *HP-27S OM*, *HP-42S OM*, *HP-42S Programming Examples and Techniques*	1988
HP-21S OM	1990
HP-15C OH and *AFH* reprints	2011, 2023

→ 24, f. 22, for where you can get these manuals.

For comparison:

In 1948, a mechanical 4-function pocket (!) calculator (the *Curta*) sold for 125 US$, equivalent to 1709 US$ of 2026.

In 1976, an electronic pocket calculator featuring *Continuous Memory* was a breathtaking innovation (you could use it as a notebook, too!); and a grand total of 72 built-in functions, 8 *GP registers*, and 49 merged program steps sufficed for professional engineers and scientists

doing their work – and of course they did for students striving for their Ph.D. But note you had to program linear regressions, correlations, and forecasting manually if and when you needed them, e.g., for calibrating – the respective routine suggested by *HP* took 44 precious steps (i.e. almost all). Also note that 200 US$ of 1976 correspond to an investment of 1166 US$ in 2026.

APPENDIX 9: RELEASE NOTES

	Date	Release notes
0	29.11.12	**Official project start** with first publication of the *43S* concept and its layout on the *MoHPC* (*https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685*). Though internal discussions started in September 2011, and there are even 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 set up based on the one of *WP 34S*. Passed to *Jake Schwartz*, *Eric Smith*, and *Richard Ottosen* for first information. See the vivid discussions starting in February 2014 in this thread *https://www.hpmuseum.org/forum/thread-593.html* (it was stopped at 10.1.16 but still is among the 3 threads most viewed on that forum after over 9 years).
0.2	3.10.15	Update based on *Jake's* feedback and further thoughts, distributed to *Eric*, *Jake*, *Marcus von Cube*, and *Paul Dale*.
0.3	21.3.16	Split the manual in 3; moved LBL onto the keyboard, renamed **STOM** to **STOCFG**, **RCLM** to **RCLCFG**, **SERR** to **sm**, and **SERR$_w$** to **smw**; refined the *Key Response Table*. Passed to *Michael Steinmann* for information.
0.4 … 0.28.2	14.7.25	Look up the *ReM* for the full *Release Notes*. ***WP43* pilot units produced by *SwissMicros* were distributed in October 2022.**
0.28.3	15.9.25	Added EDIT and **!** to the bezel, moved USB and DMCP. Expanded *App. K*. Elaborated about angles. Refined and corrected.
0.29	3.1.26	Refined. More about angles, constants, and the *Integrator*.
0.29.1	21.2.26	Added 2 new error messages. Refined, in particular *OMS 1, ReMS 1, App. G* and *N*, and the font tables. Got rid of 3 colors. Corrected.
0.30	1.8.26	Renamed most commands of INFO from **xxx?** to **?xxx**. Introduced **M.C⇶C**. Refined the fonts reducing abuse of *UTF* codes. Added examples in *OM* and *ReM* and a chapter to *App. I*. Got rid of 2 more colors in *OM* and 1 in *ReM*. Rearranged chapters in *OMS 2*. Refined *App. H*. Split *OMS 2* shifting *Sections 3*ff up. Removed the print functions. Inserted lots of links. Updated, refined, and corrected. ATTENTION: The manuals are ahead of the software!

INDEX

The index following lists names, special terms, and keywords used in this manual. Furthermore, it points to the most prominent of the over 200 vintage and new examples included, marked '(ex.)'.

You will find all the individual *items* provided on your *WP43* explained in *ReMS 1*; since this will also point you to further explanations, if applicable, particular **ITEMS** are listed below only if they are extensively covered in this *OM*. The *Table of Contents* is recommended as well – chapter headings are not repeated in the index.

For *living persons*, only their prenames are listed below unless they published under their full names; *historic* or *fictional characters* (e.g., from *HP's* vintage examples) are quoted fully. Locations, terms, and topics appearing in examples are printed in the corresponding color. For the original names of the ancient Greek philosophers and scientists mentioned in this *OM,* → *208*, f. 229.

* * *