

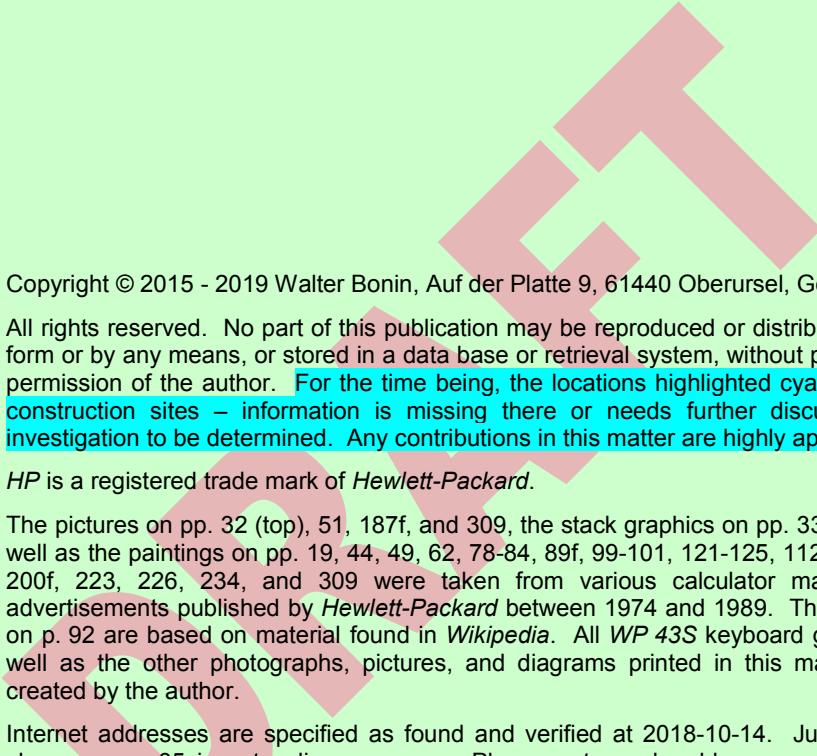


WP 43S OWNER'S MANUAL

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

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

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



Copyright © 2015 - 2019 Walter Bonin, Auf der Platte 9, 61440 Oberursel, Germany

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

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

The pictures on pp. 32 (top), 51, 187f, and 309, the stack graphics on pp. 33 and 36 as well as the paintings on pp. 19, 44, 49, 62, 78-84, 89f, 99-101, 121-125, 112, 149, 192, 200f, 223, 226, 234, and 309 were taken from various calculator manuals and advertisements published by *Hewlett-Packard* between 1974 and 1989. The diagrams on p. 92 are based on material found in *Wikipedia*. All *WP 43S* keyboard graphics as well as the other photographs, pictures, and diagrams printed in this manual were created by the author.

Internet addresses are specified as found and verified at 2018-10-14. Just the map shown on p. 85 is not online anymore. Please note such addresses may change without notice at any time.

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

Printed in the USA

ISBN-13: 978-172950098-9

ISBN-10: 172950098-6

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

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

*Statement of Corporate Objectives
Hewlett-Packard*

DRAFT

TABLE OF CONTENTS

Welcome!	8
Print Conventions and Common Abbreviations	12
Section 1: Getting Started	14
Problem Solving, Part 1: First Steps	16
How the Keyboard is Organized	23
How to Enter Common Numbers (and How to Edit Them)	25
How to Enter and Execute Commands	26
Menus – Items à la carte	27
Clearing and Resetting Your WP 43S	29
Problem Solving, Part 2: Elementary Stack Mechanics	30
Looking Closer at the Automatic Stack	37
Problem Solving, Part 3: The Stack in Advanced Calculations	40
Special Tricks, #1: Top Stack Level Repetition	47
Special Tricks, #2: LASTx for Reusing Numbers	49
Error Recovery: , , and	50
Addressing and Manipulating Objects in RAM	52
Addressing Tables	58
Indirect Addressing – Working with Pointers	61
Store and Recall Arithmetic	61
Section 2: Dealing with Various Objects and Data Types	64
Some Display Basics	64
Supported Data Types	65
Recognizing Calculator Settings	70
Getting Special Information: RBR, STATUS, VERS, etc.	73
Localising Numeric Output	74
Real Numbers: Changing the Display Format	76
Real Numbers: Squares and Cubes and their Roots	78
Real Numbers: Percent Change	79

Real Numbers: Logarithms and Powers	81
Real Numbers: Hyperbolic Functions	88
Real Numbers: Probabilities – Factorials, Combinations, Permutations, and Distributions	89
Real Numbers: From Probability to Statistics – Accumulating Data, Calculating Means, Standard Deviations, and Confidence Limits; Curve Fitting, Forecasting, and Checking Dices	93
Real Numbers: Some Industrial Problems Solved	102
Real Numbers: Summary of Functions	111
Angles and Trigonometric Functions	117
Mixed Calculations: Coordinate Transformations in 2D, Flight Directions, Courses over Ground, etc.	120
Angles: Summary of Functions	126
Integers: Input and Displaying	127
Integers: Bitwise Operations on Finite Integers	131
Integers: Arithmetic Operations	134
Integers: Overflow and Carry with Finite Integers	136
Integers: Summary of Functions	139
Rational Numbers (Fractions)	141
Times	144
Dates	146
Complex Numbers: Introduction	148
Complex Numbers Used for 2D Vector Algebra	152
Complex Numbers: Summary of Functions	155
Vectors and Matrices: Introduction and Input	157
Vectors and Matrices: Displaying and Editing Larger Objects	162
Vectors and Matrices: Complex Stuff	166
Vectors and Matrices: Calculating	167
Vectors and Matrices: Solving Systems of Linear Equations	172
Vectors and Matrices: Eigenvalues and Eigenvectors	173
Vectors and Matrices: Dealing with Statistical Data	178
Alpha Input Mode: Introduction and Virtual Keyboard	181

Alpha Input Mode: Entering Simple Text and More	183
Combining Alpha Strings and Numeric Data	185
Working with Alphanumeric Strings	186
Section 3: Programming	189
Recording a New Routine	191
Labels	195
Editing a Routine	197
Running a Routine from the Keyboard (also for Debugging)	199
Subroutines: Running a Routine from another Routine	200
Automatic Testing and Conditional Branching	201
Loops and Counters	204
Programmed User Interaction and Dialogues	208
Solving Differential Equations	210
The Programmable Menu (MENU)	215
Basic Kinds of Program Steps	217
Deleting Programs	218
Serial Input and Output of Data and Programs	218
Local Data	218
Flash Memory (FM)	219
Section 4: Advanced Problem Solving	221
Programmable Sums	221
Programmable Products	222
Solving Quadratic Equations	223
General Equations	224
The Interactive Solver for Arbitrary Equations	226
The Interactive Solver for Expressions Stored in Programs	230
Using the Solver in a Program	233
Numeric Integration of Equations	234
Interactively Integrating Expressions Stored in Programs	237

Using the Integrator in a Program	239
Differentiating Equations	241
Interactively Differentiating Expressions Stored in Programs	243
Computing Derivatives in a Program	244
Nesting Advanced Operations	245
Section 5: Two Browsers, two Applications, and two Special Menus	248
The Browsers RBR and STATUS	248
The Timer Application	251
The Time Value of Money (TVM)	253
Constants	257
Unit Conversions	263
Section 6: Creating Your Very Personal WP 43S	268
Assigning Your Favourite Functions	269
Creating Your Own Menus	274
Browsing and Purging Menus, Variables, and Programs	276
Assigning Special Characters	277
User Mode	279
Appendix 1: Operator Precedence	281
Appendix 2: Key Response Table	282
Appendix 3: Further Applications of TVM	296
Ordinary Annuities (a.k.a. Payments in Arrears)	296
Annuities Due (a.k.a. Payments in Advance)	300
Appendix 4: Power Supply	304
Appendix 5: Release Notes	305
Index	307

WELCOME!

Dear user, now you hold your very own *WP 43S* in your hands. Congratulations! It is a true pioneer: the **very first entirely community-designed and -built RPN pocket calculator.**¹

All the hardware, firmware, and user interface of your *WP 43S* were thoroughly thought through, discussed, designed and assembled, written and tested by us over and over again. We did this to create a new **fast and compact problem solver like you did never own before** – instant on, fully programmable, customizable, incorporating a state-of-the-art LCD, still comfortably fitting into your shirt pocket, and *RPN* – a serious scientific instrument supporting you in your professional activities! It readily provides several advanced capabilities never before combined so conveniently in a pocket calculator:

- A *Solver* (root² finder) that can solve for any variable in an arbitrary equation.
- A numeric integrator for computing definite integrals.
- Numeric derivation, programmable sums and products.
- A wealth of functions, supporting e.g. real and complex numbers, fractions, integers, dates, times, and text strings.
- Matrix and vector operations, including a comfortable *Matrix Editor*, a solver for simultaneous linear equations, and many more matrix functions useful in real and complex domain.
- Statistical operations, including probability distributions, curve fitting, and forecasting.

¹ *RPN* stands for *Reverse Polish Notation*, a very effective and coherent method for most efficient solutions to complicated problems. It is based on a mathematical logic known as *Polish Notation* (since invented by the Polish logician Jan Łukasiewicz). He placed operators before numbers or variables instead between them as in conventional mathematical notation. See *Section 1* below for more.

The *DM42* of SwissMicros was developed in parallel – it started later and was launched earlier (in 2017). This is due to the fact its design, layout, and firmware are closely linked to the *HP-42S* of *Hewlett-Packard*.

² Translator's note for German readers: *Root* bedeutet hier *Nullstelle*.

- + Base conversions and integer arithmetic in 15 bases from binary to hexadecimal. Bit manipulations in words of up to 64 *bits*.
- + A timer based on a real-time clock.
- + An easy-to-use *menu* system that uses the bottom part of the display to assign the top six keys according to your actual needs.
- + Keystroke programming including branching, looping, tests, *flags*, subroutines, and program-specific local data.
- + A *catalog* for reviewing all *items* stored in memory – be they provided by us or defined and programmed by you.
- + A keyboard layout and *menus* you can customize. You can save various custom layouts on-board and recall them one by one as you need them. Keyboard overlays are supported.
- + The ability to run programs written for the *HP-41Cx*, *HP-42S*, and *WP 34S* calculators.
- + Battery-fail-safe on-board backup memory for all your data (*registers*, variables, *menus*, programs, layouts, and mode settings).
- + A micro *USB* socket allowing for external auxiliary power supply as well as for transmitting your programs to a computer, so you can edit, debug, and test them on your *WP 43S* emulator there, and return them thereafter.
- + An infrared port for immediate recording results, calculations, programs, and data using an *HP 82240A/B Infrared Printer*.

Your *WP 43S* provides the most ample function set ever seen in an *RPN* pocket calculator, presumably in any pocket calculator at all:

- + A full set of scientific functions, including *Euler's Beta* and *Riemann's Zeta*, *Lambert's W*, the *error function*, *Bessel functions* of first kind, *Bernoulli* and *Fibonacci numbers*, as well as the *Chebyshev*, *Hermite*, *Laguerre*, and *Legendre* orthogonal polynomials (no more need for carrying heavy printed tables or running computer software for this matter).
- + Fifteen probability distributions: *Gaussian*, *normal*, *Student's t*, *chi-square*, *Fisher's F*, *Poisson*, *binomial*, *geometric*, *hypergeometric*,

Cauchy-Lorentz, exponential, logistic, Weibull, log-normal, and more.

- Over fifty fundamental physical constants as accurate as used today by national standards institutes such as *NIST* or *PTB*, plus a selection of important constants from mathematics, astronomy, and surveying.
- More than ninety conversions, mainly from old *British Imperial* to universal *SI* units and vice versa.

Furthermore, your *WP 43S* features lots of space for your data, programs, and ideas:

- Your choice of four or eight *stack registers* and up to 107 global general purpose *registers*, each taking one object of arbitrary *data type* (i.e. a matrix, a string or a number of arbitrary kind).
- Named variables – as many as memory can hold. Also each such variable can take one object of arbitrary *data type*.
- 112 global user *flags*.
- Up to 10 000 program steps in RAM, up to 20 000 program steps in flash memory.
- Sixteen local *flags* and up to 100 *local registers* per program allowing e.g. for recursive programming.
- A multi-line, high-resolution, alphanumeric display with adjustable contrast, allowing for showing crisp results, *menus*, mathematical symbols, natural matrix display, Greek and extended Latin letters.

WP 43S is the result of an international collaboration of two teams: *SwissMicros* (<https://www.swissmicros.com/>) – *Michael Steinmann* (from Switzerland) and *David Jedelsky* (from Czechia) – created the hardware, *Martin Lorang* (from France), *Calum Mackay* (from Britain), *Paul Dale* (from Australia), *Gert Menke* and me (from Germany) designed the user interface and wrote the software.³

³ The firmware of your *WP 43S* is based to a large extent on the experience *Paul* and me gained with the *WP 34S RPN Scientific* calculator, being on the market since 2011. We started the *WP 34S* project in 2008. You find specific information about it

As our *WP 34S* and *WP 31S* were before, also *WP 43S* is a hobbyist's project though. It was presented and discussed on the forum of the *Museum of HP Calculators* (see <http://www.hpmuseum.org/>) from 2012 to 2016 and on <https://forum.swissmicros.com/> from 2017 on.⁴ Prototypes of the SwissMicros hardware were publicly shown first on the HHC2016 conference in Nashville (USA) and on the Allschwil Meeting 2016 in Switzerland. We thank the participants of said meetings and all other members of the international community who contributed their ideas, put their votes, and lent their support throughout this project. We greatly appreciate your contributions!

We baptized our baby in honor of the *HP-42S* of 1988, the most powerful *RPN* pocket calculator available before. May it be a worthy and valiant successor of the *HP-42S* – though we would have appreciated and even preferred *Hewlett-Packard* making it (the company *HP* as we knew it in the 70's and 80's of last century). In any way, *WP 43S* stands in the tradition of almost 50 years of *RPN* pocket calculators.

We carefully checked all aspects of *WP 43S* to the best of our ability. Thus we hope *WP 43S* is free of severe bugs. This cannot be guaranteed, however, so we promise to continue improving *WP 43S* whenever necessary. Should you discover any strange results, please report them to us. If they turn out being caused by internal bugs, we will correct the firmware and provide you with an update as soon as possible. Just as we did since 2011, we will continue maintaining short response times.

Enjoy!

Walter Bonin

and its derivative, the *WP 31S*, at <https://sourceforge.net/projects/wp34s/> and the links mentioned there. Both these calculators are based on *HP*'s hardware.

⁴ If you are interested in the long and winding road how your *WP 43S* got the shape and layout you're facing now, you find almost all the information there. Please see also the *Release Notes* in App. 5 at the end of this manual.

Print Conventions and Common Abbreviations

- Throughout this manual, standard text font is Arial. Emphasis is added by underlining or **bold** printing. Calculator COMMANDS and MENUS are generally called by their names, printed in capitals in running text (*menus* underlined). **Quoted text is printed blue** (as well as **translator's footnotes**). **Specific terms, titles, trademarks, names or abbreviations** are printed in italics, hyperlinks in blue underlined italics. The latter will beam you to its target in the original .pdf file – it cannot work in a printed copy for obvious reasons, thus such a link generally refers to a page number, to the Table of Contents, or to a fully specified external address.
- Bold italic Arial letters such as *n* are used for variables; bold normal letters for constant **sample values** (e.g. labels, numbers, or characters).
- Courier is employed for **file names** and describing numeric formats.
- Times New Roman regular letters are for unit symbols (except in formulas, since the formula editor refuses to do this). Italics of this font are for *unit names*.
- Times New Roman bold capitals are used for **REGISTER ADDRESSES**, lower case bold italics for *register contents*. So e.g. the value *y* lives in *register Y* and *r45* in *R45*. Overall *stack* contents are generally quoted in the order [*x, y, z, ...*]. We keep the term *register* for the space where an individual object is stored, although the actual size of such a *register* may vary widely following the size of the object stored therein.
- This **[KEY]** font is taken for references to calculator keys, including **SOFTKEYS** in general. For shifted operations like **GTO** or **LBL**, the respective color is used. **Alphanumeric** and numeric calculator outputs (like 1.234×10^{-56} or $7,089 \cdot 10^{-12}$) are printed as you see them on the calculator screen.
- We will use decimal points in most parts of this manual (but you may set your *WP 43S* to commas as well, of course). Although that point

is less visible than a comma, ‘comma people’ seem to be more tolerant against points used as radix marks than vice versa (based on the number of complaints read so far).

All this holds unless stated otherwise locally.

The following abbreviations, listed alphabetically here, are used throughout this manual – find detailed information about the respective terms starting at the locations referred to below, if applicable:

AIM = *alpha input mode* (see p. 181).

HP = *Hewlett-Packard*.

IOI = *Index of all Items* in the *WP 43S Reference Manual*.

LCD = liquid crystal display.

OH = *Owner’s Handbook*.

PEM = *program-entry mode* (see p. 189).

RAM = random access memory, allowing read and write operations.

ReM = *WP 43S Reference Manual*.

RPN = *Reverse Polish Notation* (see footnote 1 on p. 8 and p. 30).

SI = *Système international d’unités*, a coherent system of units of measurement agreed on internationally and adopted by almost all countries on this planet.⁵

TAM = *temporary alpha mode* (see p. 55).

Further abbreviations are listed in the *Index* on pp. 307f. A few more may be used and explained locally.

⁵ Only Liberia, Myanmar, and the USA are not participating yet. We do not know what they are afraid of – and they obviously do not know what they miss. If you should not know *SI* yet, turn to https://en.wikipedia.org/wiki/International_System_of_Units. See also the chapters about *Constants* and *Unit Conversions* in Section 5.

SECTION 1: GETTING STARTED

At its heart, your *WP 43S* is an extremely powerful, versatile problem-solving tool. It allows you to solve even very elaborate mathematical problems in either of two different ways:

Manual problem solving: Using the calculator's *RPN* logic system, you can manually work step-by-step through the toughest problems while seeing intermediate answers 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 *WP 43S* can remember any sequence of keystrokes you entered, and it can then run it repeatedly as often as you need this sequence. This simple programming paradigm is particularly useful in providing answers to repetitive problems that require different data 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 *WP 43S* and can be exchanged with your computer or laptop.

If you know how to deal with a good old *HP RPN* scientific calculator, you can start using your *WP 43S* right away. Browse this manual to learn about some fundamental design concepts that put your *WP 43S* ahead of previous scientific pocket calculators.

On the other hand, if this is your first *RPN* calculator, we recommend reading Sections 1 and 2 of this manual thoroughly. This will enable you to easily solve problems manually benefitting from this unique logic system implemented. Once learned, the *RPN* logic system forms a long lasting, reliable basis of your work.

Most commands work on your *WP 43S* as they did on its antecessors, in particular the *WP 34S* and the *HP-42S*. This manual is designed to supplement your prior knowledge, focussing on all the new features of your *WP 43S* and providing the necessary information about them. This *OM* (and the *ReM*) may include also some formulas and technical explanations; though they are not intended to replace textbooks on mathematics, statistics, physics, engineering, or programming.

The following text starts with presenting the keyboard of your *WP 43S*, so you learn where you will find what you are looking for. It continues demonstrating basic calculation methods, the memory of your *WP 43S* and addressing objects therein.

Section 2 covers the display and indicators giving you feedback about what is going on. Furthermore, the various *data types* supported by your *WP 43S* are presented and demonstrated comprehensively there.

Programming your *WP 43S* (as shown in *Section 3*) follows field proven concepts known from successful previous pocket calculators up to and including the *HP-42S* and *WP 34S*.⁶

Sections 4 and *5* present advanced functionalities implemented in your *WP 43S*. You will find everything about the opportunity of customizing your *WP 43S* according to your very personal preferences in *Section 6*.

Additionally, this *Owner's Manual* is supplemented by a *Reference Manual*. Its major part is taken by the *IOI*, i.e. an index of all available operations, what they do, and how to call them. It also contains full information about all *menus* provided. The *ReM* closes with appendices covering special topics, e.g. memory management, advanced mathematical functions implemented, and a *WP 43S* emulator for your computer. In the *ReM*, you will also find instructions for keeping your *WP 43S* up-to-date whenever new firmware revisions will be released. Continue using these two manuals for reference.

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

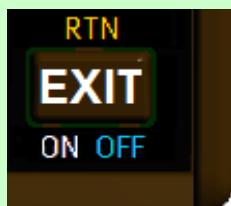
Your *WP 43S* is designed to support you solving problems. But it is just a tool (although a very powerful one): it can neither think for you nor check the sensibility of a problem you apply it to. Thus, please do not blame us nor your *WP 43S* for errors you may make. We are not liable for any of your results.

Gather information, think before and while keying in and calculating, and check your results: these tasks will remain your responsibilities always.

⁶ Getting an *HP-42S Owner's Manual* in addition may be beneficial. It was printed in 1988 and is still available at low cost – together with complete information about all the other vintage *HP* calculators built since 1968 – on media distributed by the *Museum of HP Calculators* (see <http://www.hpmuseum.org/cd/cddesc.htm>).

Problem Solving, Part 1: First Steps

Start exploring your *WP 43S* by turning it on: Press its bottom right key



– notice that **ON** is printed below that key. Doing this the very first time, you will get a display like this:



For adjusting display contrast, hold down **ON** and press **+** or **-** repeatedly until it suits your needs.

For turning your *WP 43S* off again, press the blue key

g (notice a little **g** appearing top left in the display), then press **EXIT** (which has **OFF** printed below it). Since your *WP 43S* features *Continuous Memory*, turning it off and on does not affect the information it contains (there is no “All Clear” at power up). To conserve battery power, your *WP 43S* will shut down automatically some five minutes after you stopped using it – turn it on again and you can resume your work right where you left off.

This works as on preceding pocket calculators (like an *HP-42S* or *WP 34S*). Your new *WP 43S*, however, looks more colorful than an *HP-42S* and cleaner than a *WP 34S*. This is due to your *WP 43S* featuring two prefix keys – offering you up to four functions per calculator key – and *menus*.

Looking at an arbitrary one of the 37 labeled keys, white print is for its *primary* function. For additional (*secondary*) functions, golden and blue labels are printed on the *key plate* above 35 keys, and grey characters are printed bottom right of 29 keys.

Simply press the corresponding key for accessing a function printed white. For a golden or blue function, first press the *prefix* **f** or **g**, then the corresponding key; thus, they are also called **f**- or **g**-*shifted* functions.

For better readability, we refer to keyboard labels using dark print on white from here on (like **EXIT** or **1/x**). And referring to *secondary* (or *shifted*) functions (like **RTN** or **OFF**), we will omit the *prefix* most times since redundant by color print.

Take the key **x**, for **example**. Pressing...

- **x** alone will execute a multiplication,

- **f** + **x** will call a *menu* of matrix operations stored in **MATX** (note each label printed underlined on your *WP 43S* refers to a *menu*). **EXIT** lets you exit the *menu* again.

- **g** + **x** will call **x!** calculating the *factorial* – e.g. press **8** **x!** and you will get $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 40\,320$.

- The grey letter **R** will become relevant when entering alphanumeric data.

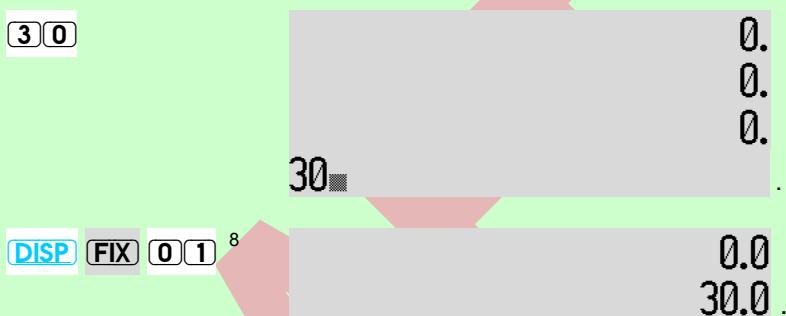


Note all the labels printed on the keyboard of your *WP 43S* are explained top left to bottom right in *Appendix 2* on pp. 282ff.

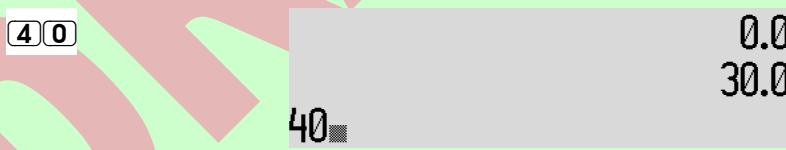
Time for a little problem solving **example**:

Turn your *WP 43S* on again if necessary. Press . Your display will show in each of its four rows.

Now, let's assume you want to fence a rectangular patch of land, 40 yards long and 30 yards wide.⁷ You have already set the first corner post (A), and also the second (B) in a distance of 40 yards from A. Where do you set the third and fourth corner posts (C and D) to be sure that the fence will form a proper rectangle? Simply key in:



Note the cursor vanished from the bottom row and the number 30 is adjusted to the right, indicating input being closed for this number now, so the next number can be entered:



⁷ Most of you are fond of *SI*. Despite this fact, we use old *British Imperial* units here so our US-American readers can follow. But this example will work with *meters* instead of *yards* as well.

⁸ **DISP** is reached by first pressing then . A menu will appear at the bottom edge of the screen: press the top left of your *WP 43S* for **FIX**. Then enter **01** telling your *WP 43S* it shall display just 1 decimal digit (this rounding affects the display only; internally, everything is handled with full precision always). See *Section 2* for more about output formatting.

Generally, we will print no more than one display row containing just zero from here on for space reasons.

(press **g** + **5**)

s =

r =

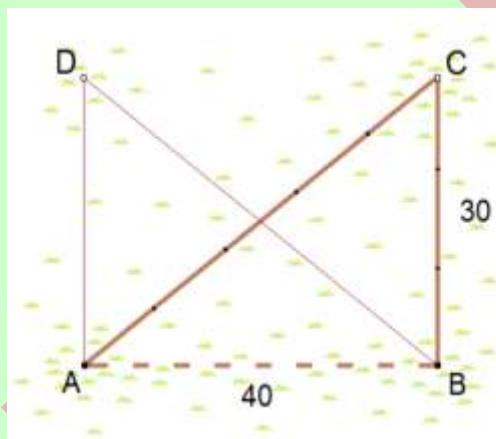
0.0

36.9°

50.0.

All you need is the number in the bottom row,⁹ a friend, and 80 yards of rope now:

Ask your friend to hold both ends of the rope firmly for you, take the loose loop and walk away as far as you can – when the loop is stretched, mark that position on the rope and return to your friend. Ask him or her to hold this point of the rope as well, fetch the two loose loops and walk again as far as possible – when the loops are stretched, mark both positions on the rope. Return again; hand over the two new points and walk once more, now with four loose loops. After marking as before, your rope will show marks every 10 yards.



Nail its one end on post A and its other end on B, fetch the loose loop and walk 5 marks away as calculated. As soon as both sections of the rope are tightly stretched, stop and place post C there. You may set post D the same way on the other side.

This method works for arbitrary rectangles, whatever other distances may apply (you will need a tapeline in the general case). As soon as you press

your WP 43S does the necessary calculation of the diagonal automatically for you. You just provide the land, posts, rope, hammer and nails. And it will be up to you to set the posts!

Another introductory **example** (basically quoted from the *HP-25 OH* though it needed some updating following progress in research in the meantime – in 1975, only 12 moons of Jupiter were known):

⁹ Forget the number displayed above for the time being.



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 3.141 5..., and d is the diameter of the sphere.

Ganymede, one of Jupiter's 79 moons, has a diameter of 5262 km. To use your WP 43S to manually compute the area of Ganymede, you can press the following keys in order:

5 2 6 2 5 262

diameter of Ganymede

x² 27 688 644

square of the diameter

π 3.1

the constant π (rounded to 1 decimal as set above)

x 86 986 440.6

area of Ganymede (in km^2 , i.e. square kilometers).

If you wanted the surface areas of each of Jupiter's 79 moons, you could repeat the above procedure 79 times. 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 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 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.

P/R

switch to *program-entry mode*.

GTO . .

go to the point of program memory where free space begins.

LBL **α** **A** **ENTER↑**

x^2
 π
X

opening step: your program will be named **A** for obvious reasons – for **A** just press **1/x** here as you see a grey **A** printed next to it.

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

RTN

EXIT

closing step. Finally,

exits *program-entry* and returns to *run mode*.

So a program on your *WP 43S* consists of an opening **LBL** step and a closing step framing the keystrokes you need for solving the respective 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 PROGS A (meaning ‘execute program A’).

(As soon as you release **XEQ**, a *menu* will appear at the bottom edge of the display. Press the **▲** key under **PROGS** (it is the second from left) and the *menu* will change. Now press the **▲** key directly under **A** and you are done.)

When you press **XEQ PROGS A** the sequence of keystrokes you recorded is automatically executed by the calculator, giving you the same answer you would have obtained manually:

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

5 2 6 2

5 2 6 2

Ganymede’s diameter

XEQ PROGS A

86 986 440.6

its surface area – as you calculated manually above. So you know 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 PROGS A**. For example, 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 PROGS A

41 693 486.7

its surface area;

3 1 2 2

3 122 ■

Europa's diameter

XEQ PROGS A

30 620 739.2

its surface area;

4 8 2 1

4 821 ■

Callisto's diameter

XEQ PROGS A

73 017 025.3

its surface area;
etc.

Programming your *WP 43S* is *that* easy! It remembers a series of key-strokes and then executes them automatically when you press **XEQ** ...¹⁰

There is no need memorizing a complicated formula after you keyed it in once – your *WP 43S* can remember it for you (and provides space for dozens more). Furthermore, you can even define shortcuts to your favorite routines by customizing the keyboard of your *WP 43S*.

The early portions of this handbook show you how easy it is to manually use the power of your *WP 43S*; while in *Section 3: Programming* you will find a complete guide to *WP 43S* calculator programming. Even if you have used other pocket calculators ..., 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 *WP 43S* features that make programming painless. When you see the simple power of your *WP 43S*, you'll become an apostle just as have some millions of *RPN* calculator owners before you.

¹⁰ Program **A** as recorded above is a very short one. You may store far longer sequences of keystrokes in program memory – the overall procedure of storing and running programs, however, will remain unchanged. Just the center part of the program (containing only three steps above) will grow.

Programming is comprehensively covered in *Section 3* of this manual.

How the Keyboard is Organized

You may have already recognized labels on your WP 43S are printed grouped according to their purposes. Beyond numeric input and general mathematics, there are five larger groups:



f, **g**, and the *menus* in particular allow for easily accessing a multiple of the 43 primary functions this keyboard can take.

Before showing the operation of your *WP 43S* in detail, let's return to our little introductory problem solving examples for four general remarks:

1. We presume you have graduated from an US High School at minimum, passed Abitur, Matura, or an equivalent graduation. So we will not explain basic mathematical rules and concepts here. Please turn to respective textbooks.
2. In four decades of scientific pocket computing, a wealth of sample applications has been created and described by different authors – more and better than we can ever invent ourselves. We do not intend to copy all of them; instead, we recommend the media mentioned on p. 15 once again: they contain almost all the user guides, application handbooks, and manuals published for vintage *HP* calculators in two heroic decades beginning with their very first calculator, the *HP-9100A* of 1968. Be assured that all computations described there for any scientific calculator can be done on your *WP 43S* – most of them significantly faster and in a more elegant way. Nevertheless, we included more than 130 examples in this manual to support you in learning your new tool.
3. There is absolutely no need to enter units in your calculations: just stay with a coherent set of units while calculating and you will get meaningful results within this set.¹¹ If you need to convert special inputs into units being part of such a set or you require results expressed in particular units, however, **U→** or **L→** will help (see *Section 5*, pp. 263ff).
4. Although we entered just integer numbers for both edges of our little patch of land, your *WP 43S* calculated the diagonal using *real numbers*. This allows for decimals in input and output as well. Alternatively you may enter fractions such as e.g. $6 \frac{1}{4}$ if you need them. Your *WP 43S* features also more *data types* – we will introduce them to you from p. 64 on.

Before, let's demonstrate how to enter common numbers in your *WP 43S* and how to deal with them in typical calculations. Therefore, return to *startup default* display format by entering **DISP ALL 00**.

¹¹ A quick and simple unit analysis is strongly recommended before starting a calculation. The big advantage of *SI* is that it is the largest coherent set of units available on this planet. Unit prefixes in *SI* simply represent powers of 1 000 (cf. the *ReM*).

How to Enter Common Numbers (and How to Edit Them)

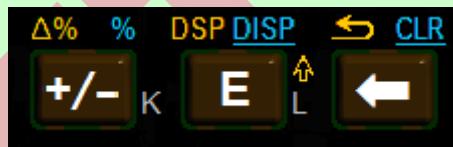
Number entry is as straightforward as typing: for 12.3456, for instance, simply press **1 2 . 3 4 5 6** and you will see

12.345 6

in *startup default* format.

You may enter more than twenty digits at once; all digits keyed in are echoed immediately in the bottom row (note the automatic gap inserted after each and every group of three digits for easier reading). Any digit(s) mistyped may be erased individually by **◀** and can be entered correctly thereafter.

For entering negative numbers such as -7.8, key in **7 +/- . 8** or **7 . +/- 8** or **7 . 8 +/-** – pressing **+/ -** changes the sign of the number being put in. Only negative signs will be displayed.



For a huge figure such as the age of the universe in years as we know it today, just enter **1 3 . 8 2 E 9** which is echoed

13.82×10⁹

in '*mantissa plus exponent*' format. **E** stands for '*enter exponent*'. Note your WP 43S allows for a naturally readable display instead of showing you cryptic machine formats like **1382E9**.

During numeric input, your keystrokes are generally just echoed in the bottom numeric row. Input is closed and released for interpretation by a command – e.g. by **ENTER↑**. Here, this will change the display to the equivalent:

13 820 000 000. .

Note the number moved to the right (cf. p. 18). Closed numbers in the bottom row may be cleared at once by pressing **◀** (calling CLX). This puts **0** in said row, and subsequent input will overwrite this **0** then.

Really tiny numbers such as the typical diameter of an atom (i.e. 0.000 000 000 1 m – with ten zeroes leading the digit 1) are entered in full analogy to huge numbers: **E +/- 1 0** will do here and will be displayed as

0.000 000 000 1 in *startup default* format.

By the way, this may be shown significantly more compact as

1. $\times 10^{-10}$ or even

1, $\cdot 10^{-10}$ with other display settings
(as treated in Section 2).

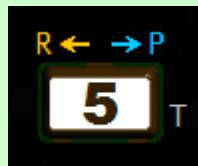
Note you did not have to enter **1 E +/− 1 0** here: if there is no numeric input heading **E**, **1** is assumed for the mantissa per default. And pressing **+/-** after **E** will change the sign of the exponent – if you want to change the sign of the mantissa, press **+/-** before entering **E** or after closing the entire input.

There are also other numeric *data types* like *integers*, *times*, or *dates* available on your WP 43S – these (and more) are covered in Section 2 together with more output formats provided.

How to Enter and Execute Commands

Just press the sequence of keystrokes required to access the label calling the command you wish to be executed (cf. p. 17). Pending input will be echoed at left end of the top numeric row in the *LCD* until the command is completed. Therein, pending *prefixes* **f** or **g** will be indicated by **f** or **g** for visual feedback, if applicable; these characters will be replaced by the name of the command accessed as soon as it can be decoded.

For many commands, **f** or **g** will be the only echo you will see during command input since the next keystroke may well terminate command entry already (as observed with **g →P** above, for example), call and execute the command, and display the result.

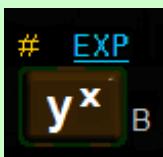


Some commands, however, require trailing parameters and will thus stay in the top numeric row for longer. STO and RCL are commands of this kind, and there are many more (see pp. 58f and the *ReM*).

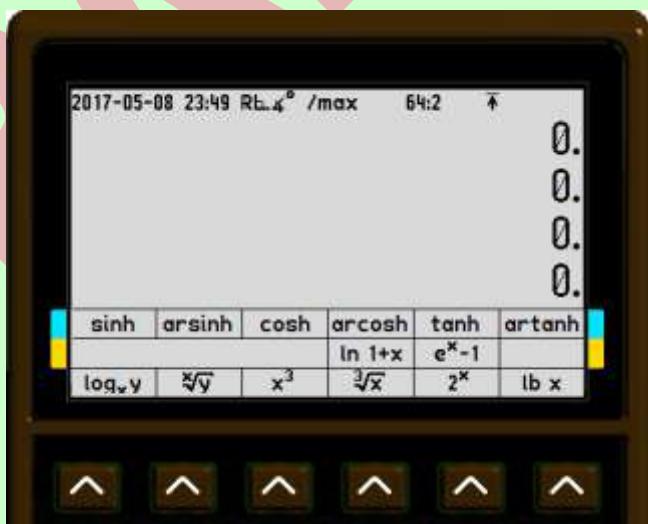
Menus – Items à la carte

Your *WP 43S* features more than 750 operations. Obviously, the keyboard offers far too little space for showing all of them. Hence most commands live in *menus*. In addition to commands, also arbitrary digits, characters, programs, variables, or *submenus* defined may be stored in a *menu*: we collectively call these *menu items* or simply *items*. By employing *menus* we can keep the keyboard clean.

Your *WP 43S* features 29 *menus* on its keyboard, printed underlined for easy recognition.¹² In alphabetic order, these are ADV, BITS, CATALOG, CLK, CLR, CNST, CPX, DISP, EQN, EXP, FIN, FLAGS, INFO, INTS, I/O, LOOP, MATX, MODE, PARTS, PROB, P.FN, STAT, STK, TEST, TRI, U_→, X.FN, a.FN, and ∠_→. Open any *menu* by simply accessing its label. This will cause the lower part (i.e. the *menu section*) of the calculator screen displaying the respective *menu view*.



Example: Press **g EXP**: then EXP will open and cause the following *menu view* appearing:



As long as this view is displayed, simply press, for example...

- the rightmost for lb x (the binary logarithm),
- and the fourth for ln 1+x,
- and the second for arsinh.

¹² We print them this way throughout this manual for the same reason. Note, however, they are stored in your *WP 43S* without that underline and hence will be displayed also in *menu views* without it.

- If you press **f** and the leftmost **▲**, nothing will happen since there is no label displayed there – no operation is connected to this **f**-shifted **▲**.

Thus, we may also print **In 1+x** to refer to **f** + **In 1+x** if we want to indicate the access path to this **f**-shifted **▲** in a compact way. In analogy, a blue background may be printed for a **g**-shifted **▲** (like **arsinh** here), and grey background for an unshifted **▲** (like **lb x** here).

Generally, whenever a *menu* is called, its top *view* will be displayed in the *menu section*. Any *menu view* may contain up to 18 *items*:

- up to six assigned to the unshifted top row of keys,
- up to six to the **f**-shifted (note the golden stripes framing the *LCD* there), and
- up to six to the **g**-shifted (note the blue stripes).

For calling a specific *item* contained in such a *view*, use the corresponding **▲** preceded by **f** or **g** if applicable (this access is called a *softkey* from here on).

Any predefined menu may contain more than just one *menu view*. This will be indicated by a dashed line limiting the *menu section* on the screen. Whenever such a *multi-view menu* shows up, **▲** will advance to the next *view* and **▼** will return to the previous one, changing the labels displayed. Because *multi-view menus* are circular, also pressing **▲** repeatedly will return to the first *view* after all other available *views* were displayed. Thus, for a menu containing just two *views*, both **▲** and **▼** will display the next *menu view*.

Any menu view will stay constant – granting easy direct access to up to 18 functions displayed in the menu section – until you leave it (e.g. via ▼, ▲, or EXIT) or call another menu.

To indicate the access path via a *menu* and the corresponding *softkey*, we will generally print the background colors as explained above in the example from here on. Note that *submenus* contained in a *menu* are displayed **inverted** without the underline. Pressing **EXIT** in a *submenu* will bring you back to its parent *menu* (containing the label of said *submenu*).

Clearing and Resetting Your WP 43S

There are several ways you can remove obsolete information from your WP 43S. The most basic one is using – you have just learned about it on p. 25. Almost all other clearing commands are contained in CLR:

CLX	Clears <i>stack</i> ¹³ register X (i.e. sets it to zero)	CLSTK	Clears all <i>stack registers</i>
CLS	Clears all statistical <i>registers</i>	CLREG	Clears all global and local g.p. <i>registers</i> ¹⁴
CLP	Clears <i>current program</i>	CLPALL	Clears all programs
CF	Clears the <i>flag</i> specified	CFALL	Clears all user <i>flags</i>
CLMENU	Clears the <i>programmable menu</i>	CLCVAR	Clears all variables of the <i>current program</i>
CLALL	Clears <u>all</u> programs and data (<i>variables</i> , <i>flags</i> , and <i>registers</i> including the <i>stack</i>) ¹⁵	RESET	Resets your WP 43S to <i>startup default</i> (excluding flash memory)

For your reference, *startup default* settings are:

2COMPL, ALL 0, CLK24, CPXi, DEG, DENANY,
DENMAX 9999, DSTACK 4, GAP 3, J/G 1752-01-01,
LinF, LocR 0, LZOFF, MULTx, PROFRC, RDX.,
RECT, RM 0, SCI0VR, SSIZE4, TDISP -1, WSIZE 64,
and Y.MD.

Turn to p. 74 to learn about commands for formatting the display to your preferences; turn to the ReM for more information about the other commands mentioned above.

¹³ Learn more about the *stack* in next chapter.

¹⁴ G.p. = general purpose. Find more about these *registers* at the end of this section. Note that variables, *stack* and statistical *registers* are not touched by CLREG.

¹⁵ Note display formats as well as other user settings and assignments will remain constant. Only RESET clears everything except flash memory (see Sections 3 & 6).

Problem Solving, Part 2: Elementary Stack Mechanics

Most of the commands your WP 43S provides are mathematical operations or functions taking and returning *real numbers*. *Real numbers* (or shortly *reals*) are numbers like 1.1 or -2.34 or 3.141 592 653 59 or 5.6×10^{-7} . Note that integers like 3, 12 345, or -121, as well as fractions like $\frac{3}{32}$ or $\frac{37}{7}$ are mere subsets of *reals*.

Depending on the particular command you choose, it may operate on one, two, or three such numbers to generate its result. In spite of the hundreds of functions available, you will find your WP 43S functions simple to operate by using a single, all-encompassing rule:

When you press a function key, your WP 43S immediately executes the function assigned to it.¹⁶

One-number (*monadic*) functions: Many functions provided on your WP 43S operate on one number only.

Example:

Enter **.49**

.49

and press **[x]**. You will get

0.7

Pressing **[x²]** returns

0.49 again.

Seven *monadic* functions more are seen on the keyboard of your WP 43S, starting top left:

- the reciprocal **[1/x]**,
- the logarithms **[ln]** and **[lg]**, the exponentials **[e^x]** and **[10^x]**,
- the function **[+/-]** (multiplying closed numbers by -1), and
- the factorial **[x!]**.

Examples:

6 4 [1/x]

returns

0.015 625 ;

¹⁶ If said function requires parameters it will execute with parameter input completed.

$\boxed{7} \text{ } \boxed{10^x}$ returns $10\ 000\ 000$;
 $\boxed{\lg}$ returns 7 ;
 $\boxed{e^x} \text{ } \boxed{x^2} \text{ } \boxed{\ln}$ returns 14 ;
 and $\boxed{6} \text{ } \boxed{x!} \text{ } \boxed{+/-}$ returns -720 ,

since $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$.

Generally, *monadic* functions replace the value (called x) displayed in the lowest numeric row on the screen by the function result $f(x)$ (e.g. $f(x) = x!$ in last example). Everything else on screen stays as it was.

Your WP 43S features also *monadic* functions operating on other data than common (*real*) numbers and/or returning different output. These functions work the same way: x will be replaced by $f(x)$.

Check the *I/O* for the lot of *monadic* functions provided (more logarithmic, exponential, root, trigonometric, and hyperbolic functions, unit conversions, etc.).

Two-number (*dyadic*) functions: Some of the most popular mathematical functions operate on two numbers and return one. Think of the four basic arithmetic operators + and –, \times and $/$.

Example:

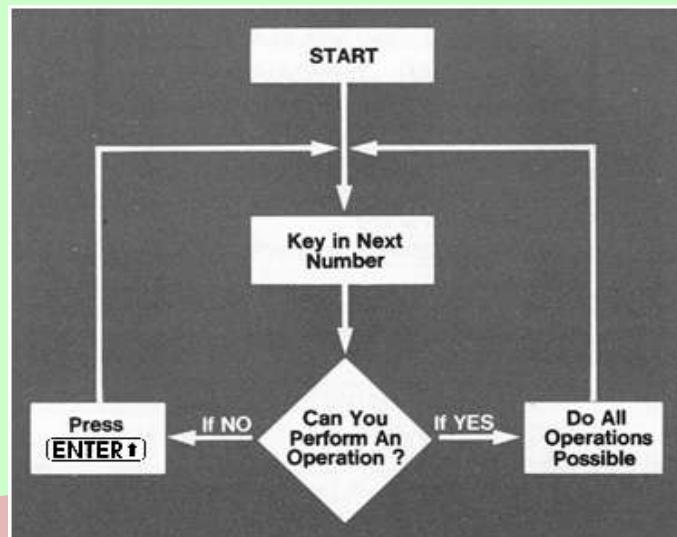
Assume owning an account of 1 234 US\$ and taking 56.7 US\$ away from it. What will remain? One easy way for solving such a problem works as follows:

On a piece of paper	→	On your WP 43S
Write down the 1 st number: 1234		Key in the 1 st number: 1 2 3 4
Start a new row.		ENTER↑
Write down the 2 nd number: 56.7		Key in the 2 nd number: 5 6 . 7
Draw a line below. _____		Subtract: -
Subtract: 1177.3		1177.3

This is the essence of *RPN*:

**Provide the necessary operands;
then execute the requested operation
by pressing the corresponding function key.**

HP itself explained the *RPN* method using a very compact picture.¹⁷ And a major advantage of *RPN* compared to all other calculator operating systems known is that it sticks to this basic rule – always.¹⁸



As the paper holds your operands while you are calculating manually, a place holding your operands is required on your *WP 43S*, too. The *stack* does this job. Think of it like a pile of *registers*, a work space for your calculations.¹⁹

¹⁷ This picture is copied from the brochure ‘**ENTER↑** vs. **=**’ of 1974.

¹⁸ This rule applies for functions regardless of the objects they operate on. This way of writing operations is called *postfix* notation since the operator is entered behind the operands (hence *RPN*, cf. footnote 1). It suits electronic calculating very well; and it eases work for human brains, too – see further below.

Some people might claim that the above global rule strictly holds for *RPL* only. *RPL* (meaning *Reverse Polish Lisp*) is a programming language and notation developed from *RPN* in the 1980's. Maybe those people are even right. In my opinion, however, *RPL* strains the *postfix* principle beyond the pain barrier, exceeding the limit where it becomes annoying for human brains. Not for everybody, of course, but also for many scientists and engineers. Thus we stick to classic *RPN* on the *WP 43S* as we did on the *WP 34S* and the *WP 31S*.

¹⁹ See next chapters for more about the *registers* provided also beyond the *stack*.

Stack register	name	contents
D		d
C		c
B		b
A		a
T		t
Z		z
Y		y
X		x

Display

Bottom up, these *registers* are traditionally called **X**, **Y**, **Z**, and **T**, optionally followed by **A**, **B**, **C**, and **D** on your *WP 43S*.²⁰ New input is always entered in **X**, and at least *x* is displayed always – *y*, *z*, and *t* may be. So you may see one to four *stack registers* on the screen at once.

ENTER↑ separates two input numbers by closing the first number *x* and copying it into **Y**, so **X** can take

a new number then without losing information (cf. above). The contents of the upper *stack registers* are lifted out of the way before. In a 4-register stack, *z* is copied into **T** and *y* into **Z** before *x* will be copied into **Y**.

This is the classical function of **ENTER↑** from the *HP-35* of 1972 until the *HP-42S* ceased production in 1995. **ENTER↑** affects all *stack registers*, and the previous content of the top register gets lost. It is often said **ENTER↑** ‘pushes *x* on the *stack*’ (although it pushes *x* under the *stack* in our picture).

Press	Contents	Location
ENTER↑	<i>t</i> <i>z</i> <i>y</i> <i>x</i>	(lost) → T → Z → Y → X

Let's look at our account example again, putting it in a *stack diagram*:²¹

²⁰ This optional 8-register stack was launched by Paul Dale and me with the *WP 34S* in 2011. The *WP 31S* features it as well. See the further text for its advantages.

²¹ We will generally use plain bold text denoting numeric input from here on for print space reasons.

The *stack diagram* is presented here for a traditional 4-register stack. At the beginning, some data may be present in the registers **Y**, **Z**, and **T**, remaining from earlier operations. These data are not relevant for this calculation, so we left them aside here; and in further *stack diagrams* we will also omit all the *stack registers* not containing any data relevant for the particular calculation, for sake of clarity and print space.

T					
Z					
Y		1 234	1 234		
X	1 234	1 234	56.7		1 177.3

Input **1234** **[ENTER↑]** **56.7** **[-]**

After having entered the second number (**56.7**, the new x), pressing **[-]** subtracts this x from the first number (**1234**, now in Y) and puts the result $f(x, y) = y - x = 1177.3$ in X. This procedure applies to most functions featured:

**Put the operands on the stack,
then execute the operation $f(x, \dots)$,
and the result will be displayed.²²**

A large part of mathematics is covered by such *dyadic* functions and combinations of them. Let's look at a chain calculation:

Example:

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

This is as a combination of six *dyadic* functions: two subtractions, an addition, a multiplication, an exponentiation, and a division.

And this is how that problem is solved on your WP 43S, starting top left in the formula, and what happens on the stack during its solution:

Y		12.3	12.3		
X	12.3	12.3	45.6	-33.3	

Input **12.3** **[ENTER↑]** **45.6** **[-]**

You will have recognized that this first parenthesis in the numerator was solved exactly as demonstrated in our little account example above. Now proceed to the second parenthesis:

²² This completely eliminates the need for an **[=]** on the keyboard.

Z		-33.3	-33.3		
Y	A -33.3	78.9	78.9	-33.3	
X	78.9	78.9	1.2	80.1	-2 667.33
	78.9	(ENTER↑)	1.2	[+]	[x]

It is solved like the first. Note in the first step of this sequence, however, the prior result (of first parenthesis) is lifted automatically (A) to Y to avoid overwriting it with the next number keyed in. This move is called *automatic stack lift*.

Actually, such an *automatic stack lift* works as if (ENTER↑) was pressed before the first digit of the new input number (i.e. before 7 here). Also an *automatic stack lift* affects all *stack registers*, and the previous content of the *top register* gets lost again. *Automatic stack lifting* is standard on *RPN* calculators, reduces the number of keystrokes necessary, and will not be indicated from here on anymore.²³ Remember you generally need pressing (ENTER↑) just for separating two consecutive numbers in input – see the flow diagram at top of p. 32.

Due to *automatic stack lift*, there is also no need for clearing your WP 43S before starting a new calculation – old results are just lifted when new input is entered. In consequence, we need neither a (C) nor a (D) and solve calculations with a minimum number of keystrokes.

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

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

²³ Of all commands provided on your WP 43S (more than 750), there are just 4 disabling automatic stack lift: ENTER, CLX, Σ+, and Σ-. Some reasoning:

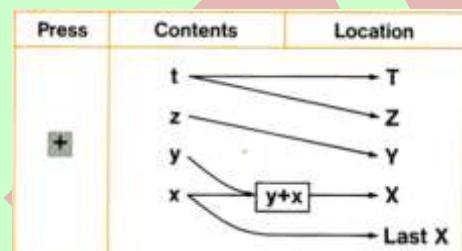
- After (ENTER↑), you generally want to key in the consecutive number.
- CLX (called by (CLX) or (C)) is for clearing X to make room for a corrected value instead of the one deleted (and we do not want an extra zero on the stack).
- And regarding (Σ+) and (Σ-), please see the chapters about statistical functions in Section 2 for the reasons.

	-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	-249.43...
3.4	ENTER↑	5.6	-	7	y^x	/

Note the *automatic stack lift* when entering **7** affects two intermediate results now. Thus, everything is well prepared for the exponentiation in the penultimate step and the final division of the numerator (in **Y**) by the denominator (in **X**). Voilà!

Following this example, you have seen the five most popular *dyadic* functions in action: **+**, **-**, **×**, **/**, and **y^x** . Your *WP 43S* provides many more *dyadic* functions: you find **RMDR** (cf. pp. 134f) next to **/** and the others in various *menus*.

As you have observed several times now, the contents of the *stack registers* drop whenever a *dyadic* function is executed. Like the *automatic stack lift* mentioned above, also this *automatic stack drop* affects all *stack registers*:



x and *y* are combined giving the result $f(x, y)$ loaded into **X**; then *z* drops to **Y**, and *t* to **Z**; since nothing is available above *t* on a 4-register stack for dropping, the top *register* content is repeated (cf. p. 39; 'Last X' will be covered on p. 49).

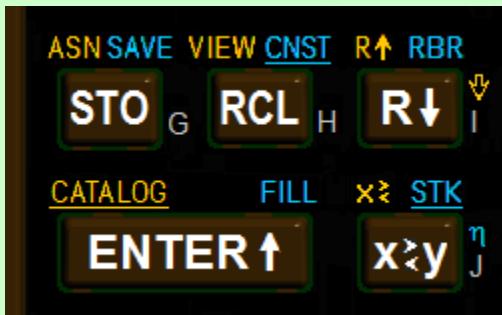
There are also a few **three-number (triadic) real functions** featured (e.g. →DATE and %MRR). Executing such a function replaces *x* by $f(x, y, z)$; then *t* drops into **Y** and so on, and the content of the top *stack register* is repeated twice (cf. p. 39). All *triadic* functions are found in *menus* on your *WP 43S*.

And some real functions (e.g. DECOMP or DATE→) operate on one number but return two or three. Other operations (such as RCL or SUM) do not consume any *stack input* at all but just return one or two

objects. Then these extra objects will be pushed on the *stack*, taking one *register* each (see p. 39).

Looking Closer at the Automatic Stack

For understanding the genius of *RPN*, we shall look a bit closer to the functions operating on the *stack*. In addition to the one-, two-, and three-number (*monadic*, *dyadic*, and *triadic*) functions explained above, there are some dedicated *stack* and *register* commands:



The control operations **[STO]**, **[RCL]**, **[ENTER↑]**, **x↔y**, **R↓**, **R↑**, and **x↔z** are known from previous *RPN* calculators for decades. They are all found within this small area of the keyboard, together with **[VIEW]**, **[RBR]**, and **[FILL]**.

Your *WP 43S* contains even more *stack* and *register* commands: CLSTK, CLREGS, DROP, DROPy, RCLCFG, RCLS, STOCFG, STOS, t↔, y↔, z↔, η↔, and SSIZE... (most of them stored in **STK**).

And it offers a choice of four or eight *stack registers* (as introduced with the *WP 34S* in 2011). Set the *stack size* according to your preferences via **[STK] SSIZE4** or **SSIZE8**. In consequence, the fate of *stack* contents depends on the particular operation executed as well as on the *stack size* set at execution time.

Operations on the traditional 4-register *stack* work as known from vintage *HP RPN* calculators since the *HP-45*. On the optional 8-register *stack* of your *WP 43S*, everything works in analogy – just with more *registers* available for intermediate results so you will hardly ever run into a *stack overflow* (cf. p. 44).

Please see on the next two pages what **[ENTER↑]**, **[FILL]**, **[DROP]**, **[DROPy]**, **x↔y**, **R↓**, **R↑**, **[RCL]**, and further representative functions do in detail on *stacks* of either size:

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

For example, $\text{x}\leftrightarrow\text{y}$ takes the initial *stack* contents as listed in the third column left and swaps the contents of the *registers* X and Y. Depending on the problems you solve and the way you proceed, you may sometimes find that x and y should be swapped before executing e.g. - , / , or $\text{\textit{y}^x}$.

Manually clearing the entire *stack* can be done by O FILL most easily. A dedicated command CLSTK is provided for backward compatibility and program space saving: press CLR CLSTK to call it.

Now you also know why $\text{ENTER}\uparrow$ and the *stack* rotation command $\text{R}\uparrow$ show an arrow pointing up while $\text{R}\downarrow$ shows an arrow down. $\text{R}\downarrow$ and $\text{R}\uparrow$ may come handy for reviewing *stack registers* else unseen (or use the register browser RBR for that – see Section 5).

		Stack contents after executing ...								
	Stack register	Assumed initial contents								
			RCL L ²⁴	S ²⁵	X ²⁶	+	DATE ²⁸	DATE ²⁹		
T	$t = 4.$	3.	2.	4.	4.	4.	4.	2.		
Z	$z = 3.$	2.	1.1	3.	4.	4.	4.	20.		
Y	$y = 2.$	1.1	s_y	2.	3.	3.	4.	10.		
X	$x = 1.1$	last x	s_x	1.21	3.1	3.1	1-02-03	1.		

D	$d = 8.$	7.	6.	8.	8.	8.	8.	6.
C	$c = 7.$	6.	5.	7.	8.	8.	8.	5.
B	$b = 6.$	5.	4.	6.	7.	8.	8.	4.
A	$a = 5.$	4.	3.	5.	6.	7.	7.	3.
T	$t = 4.$	3.	2.	4.	5.	6.	6.	2.
Z	$z = 3.$	2.	1.1	3.	4.	5.	5.	20
Y	$y = 2.$	1.1	s_y	2.	3.	3.	4.	10
X	$x = 1.1$	last x	s_x	1.21	3.1	3.1	1-02-03	1

RCL L²⁴ represents the vintage command LASTx (see p. 49 for more). Note that the previous contents of the top stack register are lost when **ENTER↑** or **RCL** are executed. Functions like S²⁵ or DATE²⁸ will even cost the contents of two or three stack registers, respectively. We

²⁴ This represents an arbitrary function pushing one object on the stack.

²⁵ This represents an arbitrary function pushing two objects on the stack.

²⁶ This represents an arbitrary *monadic* function.

²⁷ This represents an arbitrary *dyadic* function.

²⁸ Assume DATE²⁸ is called in *startup default* mode (i.e. Y.MD). It represents an arbitrary *triadic* function here.

²⁹ Assume 1.102 or 1-10-20 in X initially here and *startup default mode* set, cf. Sect. 2.

recommend mitigating the effect of such losses by setting the *stack* to eight *registers*. – Please see the *IOI* for information about the other commands mentioned.

Problem Solving, Part 3: The Stack in Advanced Calculations

Using the *stack* as described above, *RPN* eliminates the need for an **=** key as well as for any parentheses keys! See the following **example** for further demonstration, showing a more elaborate formula than above. Find below what can be used for solving it step by step and the corresponding *stack* diagrams. Choose **MODE RAD** and start with the red 7:

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

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...
	7	ENT↑	4	/	5	ENT↑	6	/	-	

								0.25...	0.25...
					0.25...	0.25...		0.25...	12.4
0.91...			0.25...	6.5	6.5	0.25...	12.4	3	3
3.14...	2.87...	0.25...	6.5		6.5	5.9	12.4	3	7

π **x** **TRI sin** **6.5** **ENT↑** **5.9** **+** **3** **ENT↑** **7**

0.25...		0.25...							
12.4	0.25...	2.94...	0.25...				27.6...		
0.42...	2.94...	1.7	5.00...	5.25...	27.6...	3.5	24.1...		

/ **y^x** **1.7** **x** **+** **x^2** **3.5** **-**

This was the solution of the entire denominator. Let's continue with calculating the numerator now, basically following the same procedure, i.e. calculating from inside out:

					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...

7.6 ENT↑ .8 x 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 √x - x² -

	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...

PARTS |x| .3 y^x 1 + x>y / √x 2 +

Even solving this formula requires only four *stack registers*.³⁰ Note there are no pending operations – each operation is executed individually, one at a time, allowing **perfect control of each and every intermediate result.**³¹

³⁰We admit we were cautious seeing this formula and chose SSIZE8 before starting calculation here.

In the fifth step of last diagram, we have got the complete result for the numerator in X. And the result for the denominator is in Y now whereto it silently traveled during all the other calculations (see above). In the following step, we swap x and y to put the operands in proper order for division of the numerator y by the denominator x.

³¹Thus, operator precedence is your job. Look up App. 1 for confirmation or reminder.

Note 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 – this is called ‘*exploratory math*’ and may well occur more frequently in your professional work than evaluating textbook formulas.³²

Experienced *RPN* calculator users have determined that by **starting every problem at its innermost number or parenthesis** and working outwards, you maximize the efficiency and power of your calculator.

If you had solved the formula on p. 40 starting with the numerator of the root straight ahead, stubbornly calculating from left to right, you would have needed six *stack registers* for the complete solution instead of four (the colors in the record below represent the top *stack register* involved in each step):

1 [ENTER↑] 30 [ENTER↑] 7 [/] 7.6 [ENTER↑] .8 [x] − 4 [y^x]
5.1 [fx] 6 [ENTER↑] 5 [/] − [x²] − [x²] [fx] .3 [y^x] [+]
[RAD] [π] [ENTER↑] 7 [ENTER↑] 4 [/] 5 [ENTER↑] 6 [/] − [x] [sin]
1.7 [ENTER↑] 6.5 [ENTER↑] 5.9 [+] 3 [ENTER↑] 7 [/] [y^x] [x]
[+] [x²] 3.5 [−] [/] [fx] 2 [+]

Admittedly, this way is not very smart though you see it is viable.

There are, however, some problems where four *stack registers* will just not suffice regardless of the way you tackle with them:

Example:

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

This highly symmetric formula lacks an unambiguous ‘inside’, so it does not matter where we start solving it. Let’s begin with the numerator:

³² Most probably you would not execute the step [PARTS] |x| above when calculating manually since you see immediately x is positive. In an automatic evaluation of such a formula, however, this step is important unless you know in advance a negative intermediate result will not occur there.

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	+	x	3	

				51	51						
51	51		51	7	7	51					
3	3	51	7	11	11	7	51			170	
3	4■	7	11■	11	6■	17	119	170	5■		
ENT↑	4	+	11	ENT↑	6	+	x	+	+	5	

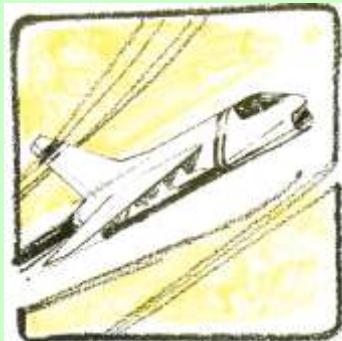
				170	170						170
170	170		170	-2	-2	170			170	-44	
5	5	170	-2	10	10	-2	170	-44	13		
5	7■	-2	10■	10	12■	22	-44	13■		13	
ENT↑	7	-	10	ENT↑	12	+	x	13	ENT↑		

A				170	170						
T	170		170	-44	-44	170					
Z	-44	170	-44	27	27	-44	170				
Y	13	-44	27	15	15	27	-44	170			
X	14	27	15■	15	16■	31	837	-881	0.049 94...		
	14	+	15	ENT↑	16	+	x	-	/		

If you had set your WP 43S to four stack registers, however, as all of HP's pocket calculators provided so far, the last stack diagram would have deviated since register A could not be loaded automatically then:

T	170	170	170	-44	-44	-44	-44	-44	-44	
Z	-44	170	-44	27	27	-44	-44	-44	-44	
Y	13	-44	27	15	15	27	-44	-44	-44	
X	14	27	15■	15	16■	31	837	-881	0.049 94...	
	14	+	15	ENT↑	16	+	x	-	/	

Then it would return a wrong result due to *stack overflow* in step 4 above and following repetition of the wrong contents. Note this is possible – and there is (and will be) no warning message since your *WP 43S* cannot know what you still need or what may be discarded without a problem.³³ Thus, we recommend setting *SSIZE8* to play safe.



We will close this chapter with another real-life **example**:

For decades, solving the following formula for the *Mach* number of an airplane as a function of its calibrated airspeed (*CAS*) in *knots*³⁴ (here: 350) and pressure altitude (*PA*) in *feet*³⁵ (here: 25 500) was used for demonstrating the simplicity and coherence of RPN:

$$\sqrt{5 \left(\left[\left\{ \left(1 + 0.2 \left[\frac{CAS}{661.5} \right]^2 \right)^{3.5} - 1 \right\} \{ 1 - 6.875 \times 10^{-6} \times PA \}^{-5.2656} + 1 \right]^{0.286} - 1 \right)}$$

Solve it like this:

```
350 [ENTER↑] 661.5 [÷] [x²]
.2 [×] 1 [+] 3.5 [yˣ] 1 [−]
6.875 [E] [+/-] 6 [ENTER↑] 25500 [×] [+/-]
1 [+] 5.2656 [+/-] [yˣ] [×]
1 [+] .286 [yˣ] 1 [−]
5 [×] [√]
```

resulting in 0.84, i.e. 84% of the speed of sound. You need only three *stack registers* for solving this.

³³ Assuming you begin your calculations with a clear *stack*, you could think of writing a little routine checking the contents of the top *stack register*, and displaying a warning if (and when) this *register* deviates from zero. Though that routine will turn useless at this very moment since the top *stack register* contents will then be repeated and will thus stay nonzero further on. See previous page and trick #1 three pages below.

³⁴ The *knot* is an ancient *British Imperial* unit of velocity surviving in aviation business.
 $1 \text{ knot} = 1 \text{ nautical mile/hour} = \frac{463}{900} \text{ m/s} \approx 0.5144 \text{ m/s} \approx 1.85 \text{ km/h}$

³⁵ The *foot* is another unit taken from that heap of pre-modern units made obsolete by *SI* for long (see U in Section 5). We quote this *Mach-number* formula without having it checked.

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

With an eight-register stack as provided on your *WP 43S*, you will be on the safe side, even dealing with the most advanced mathematical expressions you will meet in your professional life as a scientist or engineer.³⁶

Let's quote a paragraph of the *HP-25 OH* once more, just replacing all the strings '*HP-25*' by '*WP 43S*' in it:

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* (*Reverse Polish Notation*).

It is this unique system that gives you all these calculating advantages whether you're writing keystrokes for a *WP 43S* program or using the *WP 43S* under manual control:

- *You never have to work with more than one function at a time.* The *WP 43S* 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 keystrokes 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.

³⁶ Of course, constructing an example leading to *stack overflow* even for eight *registers* is trivial. But first of all this will be exactly that: a constructed example – no real world formula. And last not least, we assume there will be still an intelligent person operating the calculator, solving from inside out as recommended above.

RPN takes a few minutes to learn. But you'll be amply rewarded by the ease with which the *WP 43S* solves the longest, most complex equations. With *RPN*, the investment of a few moments of learning yields a lifetime of mathematical bliss.

And calculations with other *data types* (see *Section 2*) follow the same simple rules. So at the bottom line, we recommend:

**Choose SSIZE8 and
let your *WP 43S* care for the arithmetic³⁷
while you care for the mathematics!**

³⁷ Why do we display only four *stack registers* on your *WP 43S*, not more?

The reason is simple: Once you have accustomed to *RPN*, you know the way it deals with your data on the *stack*. Always. Watching the entire *stack* mechanics reliably working all the time, however, does not carry any valuable information then and will become boring or even distracting very soon.

Actually, the overwhelming majority of *RPN* pocket calculators displayed *x* only although there were *Y*, *Z*, and *T* quietly acting unseen. Users were doing all sorts of tricks on that *stack* – just tracking *y*, *z*, and *t* in their minds. Even *HP*'s *RPL* calculators did not display more than four *registers* (although they featured an 'infinite' *stack*).

Assuming people's mental abilities did not deteriorate in general in the last decades, displaying more than four *stack registers* carries no lasting benefit. This holds especially since the odds for *stack overflow* in real world calculations are reduced to zero when you follow our recommendation above.

On the other hand, if you feel distracted by the screen showing more than necessary, you may cut the number of *stack registers* displayed to three, two, or even just one (using *DSTACK*), letting your brains compete with the ones of users in the last fifty years. Free space will flow in the display top down – *x* will always be displayed directly above the menu section. We count on your abilities and are very confident you will succeed!

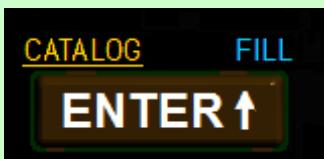
Special Tricks, #1: Top Stack Level Repetition

Whenever a *dyadic* or *triadic* function is executed, the *stack* will drop and the content of its top *register* will be repeated as illustrated on p. 39. You may employ this *top stack register repetition* for some nice tricks.

See the following compound interest calculation:³⁸

Example:

Assume your bank pays you 3.25% p.a.³⁹ on an amount of 15 000 US\$; what would be your status after 2, 3, 5, and 8 years?



Here, you are interested in currency values only, so set the display format by **DISP FIX 2**. This causes the output being rounded to cents (internally, numbers are kept with far higher precision):

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	1.0325		1.03	15 000		15 990.84		16 510.55		17 601.17		19 373.66

Below the table, the values 1.0325 and 15000 are shown again, followed by the word 'FILL'. To the right, there are two columns of three 'x' symbols each, with the labels 'after 2 years', '3 years', '5 years', and '8 years' positioned below them.

Each multiplication consumes *x* and *y* for the new product put in **X**, followed by *z* dropping to **Y**, and *t* copied to **Z**. Due to *top stack register repetition* the interest rate is automatically kept as a constant on the stack, so the accumulated capital value computation becomes a simple series of **x** strokes.

Debt calculations are significantly more complicated – so avoid debts whenever possible! In the long run, it is better for you and the economy. Nevertheless, you can cope with such calculations as well using your *WP 43S* (see Section 5).

³⁸ Translator's note for German readers: *Compound interest* = Zinseszins.

³⁹ Those were the times, my friend... !

Another application making use of top *stack register* repetition is the *Horner scheme* for calculating polynomials. It tells:

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \\ &= (\dots (a_nx + a_{n-1})x + \cdots)x + a_0 \end{aligned}$$

Example:

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

This problem can be rewritten to

$$\{ [(-3x + 5.2) x - 2.1] x + 6.4 \} x + 7$$

and is easily solved this way (with the display set to **DSP 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	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
.908	0.9	-3	-2.7	5.2	2.3	2.1	0.1	6.4	5.9	12.9	
.908	FILL	3	+/	x	5.2	+ x	2.1	- x	6.4	+ x	7

These calculations were demonstrated here for a *4-register stack*; they work with an *8-register stack* as well. **FILL** loads the entire *stack* always – be it 4 or 8 *registers* deep – it saves you from hitting **ENTER↑** multiple times.

Special Tricks, #2: LASTx for Reusing Numbers

Your WP 43S loads x into the special register **L** (for ‘Last x ’) automatically just before a function is executed – as previous RPN calculators did (cf. the picture on p. 36). What is the benefit for you?



Example (from the HP-15C OH of 1987):

Two close stellar neighbors of Earth are Rigel Centaurus (4.3 light-years away) and Sirius (8.7 light-years away). Use the speed of light, c ($2.997\ 92 \times 10^8$ meters/second, or $9.460\ 73 \times 10^{15}$ meters/year), to figure the distances to these stars in meters.

Solution:

4.3	4.3	
ENTER ↑		4.3
9.46073 [E] 15	$9.460\ 73 \times 10^{15}$	
[X]		4.1×10^{16}
8.7	8.7	
RCL [L]		9.5×10^{15}
[X]		8.3×10^{16}

RCL [**L**] is reached by pressing **RCL**, then [**E**]; note the grey **L** printed bottom right of [**E**].



Result: Rigel Centaurus has a distance of 4.1×10^{16} m (or 4.1×10^{13} km) to our planet, Sirius 8.3×10^{16} km.

So, recalling the last x via **RCL** [**L**] may save you from keying in lengthy numbers more than once. It also allows for reusing intermediate results without the need for storing them explicitly.⁴⁰

⁴⁰ There are only very few commands changing x but not loading **L**. Those are mentioned explicitly in the *IOI*. See the *ReM*.

Previous RPN calculators used LASTx also for error recovery. Your WP 43S provides **G** fulfilling this task easier – see next chapter.

Error Recovery: , **EXIT**, and

Nobody is perfect – errors will happen although you are equipped with such a powerful tool. Stay cool – your *WP 43S* allows you undoing the last command executed, restoring the calculator state exactly as it was before the error occurred.

1. If you got an **error message** in response to your function call, press or **EXIT**; this will erase that *temporary message* and return to the state before that error happened (cf. pp. 65 and 294f). Then do it right!
2. If you have erroneously called a **wrong function**, just press to undo it immediately. recalls the entire calculator state as it was before the last operation was executed; then resume calculating where you were interrupted.⁴¹



Example:

Assume – while you were watching an attractive fellow student or collaborator – you pressed inadvertently instead of in the second last step solving the lengthy formula on p. 40. Murphy's Law! But luckily there is absolutely no need to start that calculation all over again – that error is easily undone as follows:

Z					
Y	numerator		num		
X	denominator	num × den	den	num / den	correct result
	Fine so far.	Oops!	Undo	Resume	

As you have rightfully expected, works for an *8-register stack* as well. So don't worry – be happy!

⁴¹ operates on everything but program memory. And note will not revert any operations you confirmed explicitly (like RESET). And will undo the very last operation only, not more – i.e. = .

Previous RPN calculators used LASTx for error correction – works easier.



THE HP-35, THE WORLD'S FIRST HANDHELD SCIENTIFIC CALCULATOR orbited in space aboard three manned *Skylab* missions (1973-74). Solar research figured prominently among the wide assortment of experimental research conducted. Used as a backup to on-board computers, the HP-35 calculated predocking rocket burns necessary to align the Apollo Command Module with *Skylab*. In addition, the HP-35 helped *Skylab* crews aim their telescopes at stars in attempts to measure ultraviolet radiation.



Now, that's almost all you have to know about number crunching for the time being – calling commands and real calculations on the *stack*. It did suffice for high flying applications already – cf. the picture.

Note the following alternative for accessing commands: you can call every operation, wherever it may be stored, by entering its name:

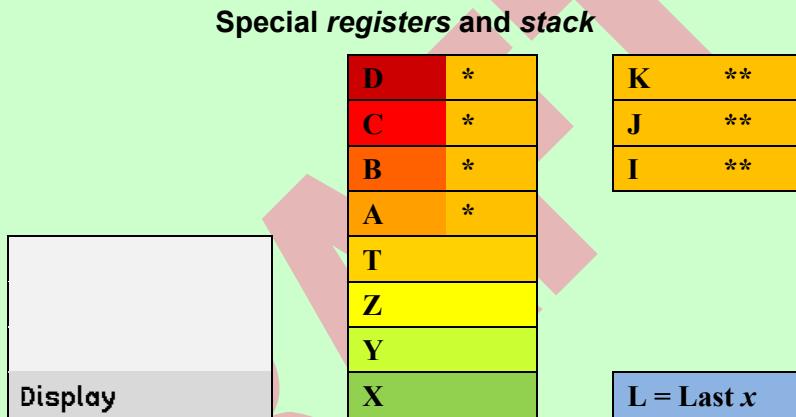
1. Press **[XEQ] [α]**.
2. Key in the name of the operation wanted using the grey letters on the keyboard. Use either case.
3. Press **[ENTER↑]**. Your input will be checked.
 - a. If the operation specified exists then it will be checked for required parameters;
 - i. if there are any, you will be prompted for them; then the function will be executed.
 - ii. else the function will just be executed.
 - b. else an error (**No such function**) will be thrown.

There are, however, far more places than just the *stack* where you may store and save your data in your *WP 43S*. Let's present them to you.

Addressing and Manipulating Objects in RAM

You have learned about the *stack* providing work space and temporary storage during your calculations. For long term storage, feel free to employ other *registers*, variables, *flags*, and program memory. This and the following chapters will tell you how to use them.

The two pictures below and overleaf show the entire address space of your WP 43S. Depending on the way you configure its memory, a subset of all these addresses will be accessible for you.



Depending on the *stack size* you choose, either **T** or **D** will be the top *stack register*; **A – D** will be allocated for the *8-register stack* if required. *Registers I, J, and K* may carry parameters of statistical distributions (see pp. 89ff); **I** and **J** will be also used in matrix editing (see pp. 157ff), and **K** is the default *alpha register* for some special operations (pp. 215f). Unless required for these purposes, **A, B, C, D, I, J, and K** are usable as additional global general purpose *registers*.

For numeric addressing of *registers* or *flags*, see the tables on pp. 58ff. Addresses ≥ 112 are used for local data (see pp. 218f).

Statistical data are accumulated in a set of dedicated summation *registers* (like in the *WP 34S* and *31S* before). You may enter your gathered statistical data value by value, point by point, or in a single matrix all at once (see *Section 2* for more).

General purpose registers

R.99 = R211
R.98 = R210
R.97 = R209
...
...
R.02 = R114
R.01 = R113
R.00 = R112

Local reg.s

R99
R98
R97
...
...
R02
R01
R00

Global reg.s

Program steps

0000
0001
0002
...
...
9997
9998
9999

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

...

Like the *stack*, also each other *register* can hold any object you store therein – more than just a common number (you will learn about these other objects in *Section 2*). – General purpose *registers* are also beneficial storing intermediate results for repeated use.

Example: Solve

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

Solution:

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

1.09 [ENTER] 1.78 [/]

[x^2] 3 + STO K

i.e. 3.374 98...

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

[\sqrt{x}]

solves the first factor of the expression,

[RCL] K ln

solves the numerator,

[\times]

multiples,

[RCL] K TRI cos

solves the second part of the denominator,

[/]

divides by it,

4 [/]

divides by four finally, returning 0.559 63...

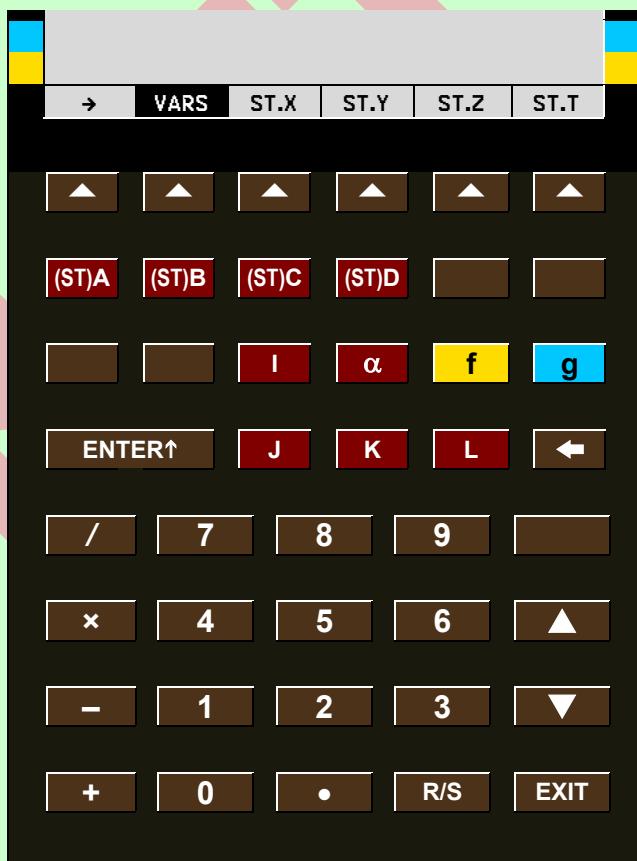
That's it – solving this expression has become really easy this way.

Flags are elementary *items* having only two states, *set* and *clear*. You may think of them as switches being either on or off. Some user *flags* have special purposes or effects. The system will set *flags* **B** ('big') and **C** ('carry') for *finite integers* (see pp. 117ff) like the *overflow* and *carry flags* of *HP-16C*; some integer operations also read the *carry flag* **C**. *Flag I* is set by CPXRES and cleared by REALRES for allowing or blocking complex results of real functions (see p. 70). *Flag D* allows for special (i.e. infinite or non-numeric) results without throwing an error, **T** sets the print line to '*tracing mode*'. All three *flags* **I**, **D**, and **T** are user-settable, and the system only reads them. Since *flags* are generally most useful in programming, they will be dealt with in *Section 3*.

☞ You will not get 10 000 program steps and 212 registers and 127 flags together at the same time certainly – see the ReM, App. B, for the reasons and for resource management.

Let's return to *registers* and variables: During input processing in memory addressing, e.g. while entering parameters for storing, recalling, swapping, copying, clearing, or comparing, you will not need all the labels presented on the keyboard. Just 29 keyboard labels plus the *prefixes* will do instead. The calculator mode supporting these 29 labels exclusively is called *temporary alpha mode (TAM)*. As shown in examples on the next pages, it may be automatically set in cases as outlined above.

Entering *TAM*, the operational keyboard is temporarily reassigned as pictured here. This kind of picture is called a *virtual keyboard* since it may deviate from the physical (or real) keyboard of your WP 43S. In such a picture, **dark red background** is used to highlight changed key functionality. White print denotes *primary* functions also on *virtual keyboards*, such as the left key in row two directly accessing (*stack*) register A in *TAM*.⁴²



⁴² What is printed white on your physical WP 43S, on the other hand, is called a *default primary* function.

Also all other lettered *registers* can be called directly – the *stack registers* X, Y, Z, and T via unshifted *softkeys*. And accessing numbered *registers* stays as easy as can be.

is for local addresses here (see pp. 218f).

Variables are named storage locations. As well as each *register*, also each variable can hold any type of data (see Section 2). Variables already defined at execution time will show up in the submenu VARS in alphabetical order – so you can select the variable of your choice by pressing the respective *softkey*. You can also access them via α (or even create new variables this way – see pp. 58f for how to do this).

Note that you will not need **f** or **g** except for *softkeys*. These may be context sensitive in TAM. If a comparison (e.g. $x < ?$) is called,⁴³ the **f**-shifted row will look like this:

x < ? __					
0.	1.				
→	VARS	ST.X	ST.Y	ST.Z	ST.T

This allows for directly comparing x with the numbers 0 or 1 (cf. p. 58).

If **STO** or **RCL** is called, on the other hand, the **f**-shifted row will look like this instead:

STO __					
Config	Stack			Max	Min
→	VARS	ST.X	ST.Y	ST.Z	ST.T

This allows for storing and recalling all your specific settings easily via **STO Config** and **RCL Config**, respectively (s. p. 75).

STO Stack stores the entire *stack* in a block of four or eight *registers* (depending on *stack size set*), **RCL Stack** recalls it. And **Max** (or **Min**) lets you work with the maximum (or minimum) of x and

⁴³ Comparisons are most useful in programming – see Section 3.

the contents of the source automatically (see the *IOI*). You may press **▲** as shortcut for **f Max** and **▼** for **f Min** here.

For all other operations requiring trailing parameters, the *menu* will stay with a single row of *softkeys* as pictured on p. 55.

If you just want to look up the current contents of a storage location without disturbing the stack, however, use **VIEW**. For inspecting a row of various *registers* instead, take **RBR**; call **STATUS** for checking the status of all user *flags* (both latter commands are explained in Sect. 5).

Example:

DISP FIX 5

VIEW K

returns

$K = 3.374\ 98$

0.000 00
0.000 00
0.559 63

... as expected from previous example.

Note the view into *register K* is displayed adjusted to the left immediately below the *status bar*.

TAM will be terminated as soon as sufficient characters are entered for the respective operation. You may delete pending parameter input keystroke by keystroke using **◀** and correct it if necessary – or just abort the pending command by **EXIT**; the latter will leave *TAM* immediately, automatically returning to the mode set and *menu* displayed before, if applicable.

- ☞ You are granted unlimited access to all the global *registers* and *flags* allocated; there are no safety constraints like ‘*memory protection*’ on your *WP 43S*. You are the sole and undisputed master of its memory! Thus, it is also your responsibility to take care of it – keep suitable records to avoid inadvertently overwriting or deleting your precious data (in *Section 3*, you will learn about a method preventing your programmed routines from interfering with data of other programs).

Addressing Tables

1 User input		TEST $x = ?$, $x \neq ?$, $x > ?$, $x \geq ?$, $x \leq ?$, $x < ?$, or another comparison alike			
Echo		OP _ ? (with TAM set), e.g. $x <_ ?$			
2 User input	$0.$ or $1.$	<i>Stack or lettered register</i> $ST Y$, ..., I , J , K , or variable defined ⁴⁴	<i>Register #</i> $\boxed{0} \boxed{0} \dots$ $\boxed{9} \boxed{9};$ $\boxed{.} \boxed{0} \boxed{0} \dots$ $\boxed{.} \boxed{9} \boxed{9}$ if the respective register is allocated.	\rightarrow opens indirect addressing	α ⁴⁵ turns on <i>alpha input mode</i> (see pp. 181ff) for a (new) variable name
Echo	OP n ? e.g. $x = 0?$	OP? x e.g. $x \geq ? ST.Y$	OP? _ e.g. $x \neq ? r23$	OP? → _	OP? ‘ _
		Compares x with the number 0 .	Compares x with the content of stack register Y .	Compares x with the content of $R23$.	See overleaf for more about indirect addressing.
		OP? ‘xx’ e.g. $x > ? ‘Stp1’$			Variable name (see overleaf for more) OP? ‘xx’ e.g. $x > ? ‘Stp1’$

Compares x with the content of the variable called 'Stp1'.
 Press **g TEST** $x > ? \alpha S \downarrow T P f 1$ **ENTER↑** for this.

⁴⁴ It is recommended calling variables being already defined via **VARS**.

⁴⁵ Note you can skip pressing **f** here, as you might have noticed on the *virtual keyboard* above already.

1 User input	RCL , STO , VIEW , xΣ , yΣ , zΣ , tΣ , DEC , DSE , DSL , DSZ , INC , ISE , ISG , ISZ , ALL , FIX , SCI , ENG , DSP , bit and flag commands, etc.			
Echo	OP _ (with TAM set), e.g. RCL _ ⁴⁶			
2 User input	<i>Stack or lettered register</i> ST X , ..., K , or variable defined ⁴⁴	<i>Register or flag number</i> or number of bit(s) or decimals (see overleaf for valid ranges)	opens indirect addressing	α ⁴⁵ turns on alpha input mode (see pp. 181ff) for a (new) variable name
Echo	OP x e.g. SF K	OP nnn e.g. SCI 10	OP → _	OP _
3 Input	Sets flag 111.	<i>Register #</i> 00 ... 99 ; •00 ... •99 if the respective register is allocated.	<i>Stack or lettered register</i> ST X , ..., K , or variable defined ⁴⁴	<i>Variable name</i> (up to 7 characters incl. at least one letter, name must be unique) ⁴⁷ OP 'xx' e.g. INC 'Zähler'

Please find explanations overleaf.

⁴⁶ For plain **RCL** and **STO**, any of the operators **+**, **-**, **×**, **/**, **▲**, or **▼** may precede step 2 here. Entering such an operator twice will cancel it, so e.g. **RCL / /** equals **RCL**. See below for more about this store and recall arithmetic.

Such operators are not allowed in **RCL Config** (calling RCLCFG), **STO Config** (calling STOCFG), **RCL Stack** (calling RCLS), and **STO Stack** (calling STOS), however.

⁴⁷ If a variable with this name is not defined at execution time yet, it will be automatically created, containing zero initially.

- STO \rightarrow 45** stores (i.e. copies) x in the location where $r45$ is pointing to (see p. 61).
- $x \leftarrow \rightarrow L$** swaps x and the content of the *register* where L is pointing to.
- INC 'Zähler'** increments the variable called 'Zähler'.

Clearing an individual *register* or variable is most easily done by storing zero in it. Deleting a variable from memory is demonstrated on pp. 276f.

Valid number range ⁴⁸	
<i>Registers</i>	0 ... 99 for direct addressing of <u>global numbered registers</u> .099 for direct addressing of <u>local registers</u> 0 ... 211 for indirect addressing (≤ 111 without local <i>registers</i>)
<i>Flags</i>	0 ... 99 for direct addressing of <u>global numbered flags</u> .015 for direct addressing of <u>local flags</u> if allocated 0 ... 127 for indirect addressing (≤ 111 without local <i>flags</i>)
Decimals	0 ... 15
Integer bases	2 ... 16
Bit numbers	1 ... 64
Word size	1 ... 64 bits

⁴⁸ Specifying low numbers (and numeric addresses), you may key in e.g. **5 ENTER↑** instead of **0 5**. Remember some *registers* and *flags* may also be addressed using letters.

Indirect Addressing – Working with Pointers

Parameters for many functions can be specified using *indirect addressing*. I.e. rather than entering the parameter itself as part of the instruction, you supply the *register* or variable pointing to the actual parameter.

Example:

Assume $x = 12.3$, $r0 = 45.67$, and $r12 = 8.9$. Then...

STO **0** **0**

RCL **→** **0** **0**

will return **8.9** since (at the time this command is executed) **R0** is containing **12.3** and thus is pointing to **R12**. And now...

FLAGS **SF** **→** **X**

will set flag 8, while...

DISP **FIX** **→** **X**

will display **8.900 000 00** showing 8 decimals.

Since the content of the *register* specified is used as a pointer to the *register* wherfrom we want to read (or whereto we want to write), this method is called indirect addressing. Each and every *register* of your WP 43S can be used for indirect addressing.⁴⁹ And each and every register can be accessed this way (also the *stack*). Indirect addressing can be most beneficial in programs when the parameter for a function is calculated (see Section 3).

Store and Recall Arithmetic

As mentioned in footnote 46, arithmetic (or two conditional picks, i.e. max or min) can be performed upon the contents of *registers* or variables by pressing **STO** or **RCL** followed by the respective operator key (**+**, **-**, **×**, **/**, **▲**, or **▼**) trailed in turn by the address or name of the storage space.

⁴⁹ Several vintage calculators, on the opposite, featured just a single dedicated *register* for indirect addressing. See the HP-34C or HP-15C, for instance.

Example for store arithmetic:

123.4

STO **-** **K**

closes numeric input and subtracts **123.4** from **k**. The difference is stored in **K**. The stack remains unchanged.

Alternatively, the same result could also be achieved by the sequence

123.4

RCL **K**

x \geq y

-

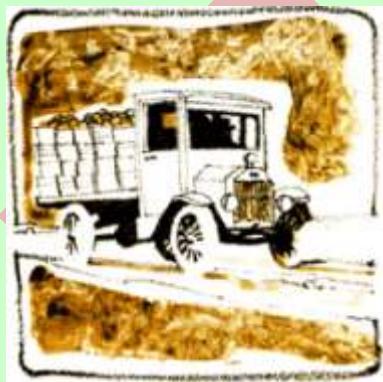
STO **K**

RCL **L**

but that is far clumsier and would cost one *stack register* in addition.

The **general rule for store arithmetic** reads:

$$\text{new contents of register or variable} = \text{old contents of register or variable} \left\{ \begin{array}{l} + \\ - \\ \times \\ / \end{array} \right\} x$$



Example (from the *HP-67 OH and Programming Guide* of 1976):

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?

Solution:

DISP **FIX** **2**
25 **ENTER** **↑** 27 **+**
19 **+** 23 **+**
55 **x**

Total of Monday's & Tuesday's tonnage

94.00

5 170.00

Gross amount for these days

STO 0 1	5 170.00	Take R01 for accounting
2 %	103.40	Deduction for these days
STO - 0 1	103.40	Subtracted from the total in R01
26 ENTER ↑ 28 +	54.00	Wednesday's tonnage
57.5 ×	3 105.00	Gross amount for Wednesday
STO + 0 1	3 105.00	Added to the total in R01
3 %	93.15	Deduction for Wednesday
STO - 0 1	103.40	Subtracted from the total in R01
RCL 0 1	8078.45	Snopes' total net income from his tomatoes

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. The rest of the **stack** as well as the contents of **R23** remain unchanged.

The same result could also be achieved by the sequence

78.91

RCL 2 3

/ but that replaces one step by two and costs one **stack register**, too.

The **general rule for recall arithmetic** reads:

$$\text{new } x = \text{old } x \left\{ \begin{array}{l} + \\ - \\ \times \\ / \end{array} \right\} \text{contents of register or variable}$$

Stack-wise, store and recall arithmetic work like *monadic* functions. Note these functions may operate on each and every **register** or variable provided, also on the **stack** and even on **L**. Indirect addressing may be used as well. See pp. 210ff for more examples and the **I/O** for further details.

Although these techniques have been more important in times when program memory was very limited, they may be still beneficial today.

SECTION 2: DEALING WITH VARIOUS OBJECTS AND DATA TYPES

Some Display Basics

The *LCD* is your window to your *WP 43S* – there you see what is going on and what the current results are. Going top down, you find ...

- the *status bar*,
- space for up to four rows of *standard numeric output*, and
- the *menu section* for displaying up to three rows of *softkeys* (cf. pp. 27f).



1. The **left** side of the **top numeric row** is also used for the output of **VIEW** (see p. 57) as well as for echoing command input until completed, i.e. until all the required command parameters are entered and the command can be executed.⁵⁰ Prefixes **f** and **g** will be displayed (using **f** and **g**) until they are resolved. And you may edit any pending operation by **⬅** or cancel it by **EXIT** (cf. p. 57).

If you pressed **f** or **g** erroneously, however, recovery is as easy as follows:

- **f f** = **g g** = NOP
- **g f** = **f** and **f g** = **g**

⁵⁰ This information will be shown at this screen position even if only one *stack register* is displayed (see e.g. matrix handling further below and DSTACK in the *I/O*).

2. The **left** side of the **second numeric row** is used for displaying an error message, if applicable.⁵⁰ Then, the command input will stay in the top numeric row until the error message disappears.
3. The **left** side of the **third numeric row** may be used for any additional (temporary) output information heading *y*, if applicable.⁵⁰
4. The **left** side of the **bottom numeric row** is used for...
 - a. echoing numeric input (cf. p. 25). Note the row can take up to 21 digits, sign, and radix mark in *startup default* display format. You may edit pending input character by character by . Numeric input will be checked and interpreted as soon as it is completed and closed, according to the calculator settings at closure time.
 - b. echoing alphanumeric input as described on pp. 181ff.
 - c. displaying any additional (temporary) output information heading *x*, if applicable.

In *run mode*, any information exceeding the plain contents of the *stack registers X, Y, and Z* is **temporary information**;⁵¹ it will vanish with the next keystroke you enter: pressing or will just clear the *temporary information* returning to the pure display of *x, y*, and *z* (if you chose DSTACK>2) – any other key will be executed in addition.

Supported Data Types

Your *WP 43S* can do more for you than just calculating with *real numbers*. Like the *HP-42S* before, it can deal with various *data types*; It can handle *integer*, *real* and *complex numbers* as well as *angles*, *times*, and *dates* in various formats.⁵²

⁵¹ If you choose less than two *stack registers* to be displayed (see DSTACK), temporary information will nevertheless show up at the positions mentioned above, wherever applicable. And operations resulting in multiple output rows will display their entire output independent of the DSTACK setting.

⁵² Furthermore, your *WP 43S* can deal with alphanumeric character strings, real and complex vectors and matrices – these *data types* are covered in dedicated chapters further below in this section. All the *data types* provided are listed in the ReM.

But how shall your WP 43S learn about the particular meaning of your input? Some **examples** will explain (showing the bottom numerical row in *startup default* display format):

Input	Display	Meaning
1234567 [ENTER↑]	1 234 567	<i>Infinite and finite integers</i> of various bases, see pp. 111ff
1234567 [#][H]	1 23 45 67 ₁₆	
1000110111 [#][2]	10 0011 0111 ₂	
901.23.4567 [ENTER↑]	901 23/4 567 >	Fraction, see pp. 141ff
12340.56789 [ENTER↑]	12 340.567 89	Real numbers, see pp. 76ff
12 [E] 345 [ENTER↑]	12. ₁₀ 345	
12.3 [CC] 4.56 [ENTER↑]	12.3 +i×4.56 12.3 4 4.56°	Complex numbers in rectangular or polar notation; mantissa plus exponent format is viable as well; see pp. 148ff
123.45678901 [d.ms]	123°45'67.89"	Sexagesimal angle; see pp. 117ff for such angles and other angular formats
1.2345678901 [h.ms]	1:23:45.678 901	Sexagesimal time, see pp. 144f
01.020304 [.d]	0001-02-03	Date, see pp. 146f

Some of these input data may be read and displayed differently depending on particular mode settings. *Startup default* displays are printed in light blue, further widespread formats in grey fields overleaf.

	RDX.	RDX,
GAP 4	1 2340.5678 9	1 2340,5678 9
GAP 3	12 340.567 89	12 340,567 89
GAP 0	12340.56789	12340,56789
MULT×	$12 \cdot 10^{345}$	$12, \cdot 10^{345}$
MULT·	$12 \cdot 10^{345}$	$12, \cdot 10^{345}$
MULT×, CPXi	$12.3 + i \cdot 4.56$	$12,3 + i \cdot 4,56$
MULT·, CPXi	$12.3 + i \cdot 4.56$	$12,3 + i \cdot 4,56$
MULT×, CPXj	$12.3 + j \cdot 4.56$	$12,3 + j \cdot 4,56$
MULT·, CPXj	$12.3 + j \cdot 4.56$	$12,3 + j \cdot 4,56$
	$12.3 \triangleleft 4.56^\circ$	$12,3 \triangleleft 4,56^\circ$
	$123^\circ 45' 67.89''$	$123^\circ 45' 67,89''$
	1:23:45.678 901	1:23:45,678 901
	1:23:45.678 901 a.m.	
	Y.MD	D.MY
	0001-02-03	01.02.0304
		M.DY
		01/02/0304

Obviously, your *WP 43S* allows for interpreting and displaying your input very flexibly. And it allows you immediately recognizing the various *data types* and format settings looking at the screen.

Now, how can you use data of various types in calculations? The matrix below lists in its first column eleven *data types* your *WP 43S* supports. Furthermore, it shows what will happen when you combine various objects: an object of the *data type* as indicated in one of the

lean columns at right (y) plus or minus an object of the *data type* in column 1 (x) will result in an object of the *data type* at the intersection (thus, wherever a resulting *data type* number is printed at the intersection, the corresponding combination is legal for addition or subtraction).

Data type and meaning		y											
x		1	2	3	4	5	6	7	8	9	10	11	12
1 \mathbb{Z} Integer of infinite precision		1	2	3	4	5	6	7	-	-	1	11	12
2 \mathbb{R} Real number		2	2	3	4	5	6	7	-	-	2	11	12
3 \mathbb{C} Complex number (\mathbb{L} or \odot)		3	3	3	-	-	-	7	-	-	3	12	12
4 Angle (in various formats) ⁵³		4	4	-	4	-	-	7	-	-	4	4	-
5 Time interval (in H..MS)		5	5	-	-	5	-	7	-	-	-	5	-
6 Date (in various formats)		6	6	-	-	-	1 ⁵⁴	7	-	-	-	6	-
7 Alphanumeric string ⁵⁵		-	-	-	-	-	-	7	-	-	-	-	-
8 Real matrix or vector		-	-	-	-	-	-	7	8	9	-	-	-
9 Complex matrix or vector		-	-	-	-	-	-	7	9	9	-	-	-
10 Integer of finite precision ⁵⁶		1	2	3	4	-	-	7	-	-	10	11	12
11 DP real number		11	11	12	4	5	6	7	-	-	11	11	12
12 DP complex number		12	12	12	-	-	-	7	-	-	12	12	12

Example:

A *complex number* (*data type* 3) plus or minus a *real number* (*data type* 2) will result in a *complex number*.

⁵³ Angular output will be tagged according to the *angular display mode* chosen.

⁵⁴ A date minus a date will return an integer number of days. There are no other arithmetic operations on dates.

⁵⁵ In additive operations on alpha strings, such a string must be present in **Y** at beginning. Adding corresponds to appending x (converted to a string according to the display format set if necessary) to string y . Adding a matrix will append its abbreviation (e.g. [3x4 \mathbb{C} matrix], see below). Subtractions from strings are not allowed.

⁵⁶ If such integers of different bases are combined by an arithmetic operation, output will be an integer of the base given in **Y**.

The following matrix shows the resulting *data types* of products and ratios in the same way:

	An object y of <i>data type</i> ...											
	1	2	3	4	5	8	9	10	11	12		
... times an object x of the <i>data type</i> below returns a product of the <i>data type</i> printed at the intersection.												
1 \mathbb{Z} Integer of infinite precision	1											
2 \mathbb{R} Real number	2	2										
3 \mathbb{C} Complex number	3	3	3									
4 Angle	4	4		-	-							
5 Time interval	5	5		-	-	-						
8 Real matrix or vector	8	8	9	-	-		8					
9 Complex matrix or vector	9	9	9	-	-		9	9				
10 Integer of finite precision	1	2	3	4	5	8	9	10				
11 DP real number	11	11	12	4	5	8	9	11	11			
12 DP complex number	12	12	12	-	-	9	9	12	12	12		
... divided by an object x of the <i>data type</i> below returns a ratio of the <i>data type</i> printed at the intersection.												
1 \mathbb{Z} Integer of infinite precision	1/2	2	3	4	5	8	9	1	11	12		
2 \mathbb{R} Real number	2	2	3	4	5	8	9	2	11	12		
3 \mathbb{C} Complex number	3	3	3	-	-	9	9	3	12	12		
4 Angle	-	-	-	-	-	-	-	-	-	-		
5 Time interval	-	-	-	-	-	-	-	-	-	-		
8 Real matrix ⁵⁷	8	8	9	-	-	8	9	8	8	9		
9 Complex matrix ⁵⁷	9	9	9	-	-	9	9	9	9	9		
10 Integer of finite precision	1	2	3	4	5	8	9	10	11	12		
11 DP real number	11	11	12	4	5	8	9	11	11	12		
12 DP complex number	12	12	12	-	-	9	9	12	12	12		

⁵⁷ The matrix x must be invertible. Dividing by x is equivalent to multiplying times x^{-1} .

Explicit type conversions are available where necessary in addition:

A single object of <i>data type</i> ...											
1	2	3	4	5	6	10	11	12	... will be converted in an object of the <i>data type</i> below by the command printed at the intersection.		
Z	R	C	a	t	d	fi			1	Z	Infinite integer
-	IP	-	-	-	-	-	IP	-	2	R	Real number
→REAL	-	→REAL (.d)				→SP	-		3	C	Complex number
→INT (#)	-	-	-	-	-	-	→INT	-	10		Integer of finite prec.
-	→DP	-	-	-	-	-	-	-	11		DP real number
-	-	→DP	-	-	-	-	-	-	12		DP complex number

You cannot enter *DP* numbers directly – use →DP. *DP* numbers are displayed like

123.456 78 or 123,456 78

– watch the radix marks. *Single precision* can be regained letting →SP operate on a *DP* number.

Recognizing Calculator Settings

As demonstrated above, radix marks and gap settings are recognized in the numeric display immediately; so are date and time display modes (Y.MD / D.MY / M.DY and CLK24 / CLK12) in the time string left within the *status bar*. Also *program-entry mode* (PEM) is easily recognized (see pp. 189ff).

Further modes and system states as well as many settings for specific *data types* are indicated in the *status bar*: The following specific characters may appear trailing the time string there, listed below from left to right in various groups – indicators shown in *startup default* are printed in a light blue field again:

Indicator	Set by	Deleted by	Explanation, remarks
C	CPXRES, SF I	REALRES, CF I	CPXRES allows complex results of <i>real number</i> calculations, like $\sqrt{-1}$. Else a domain error would be thrown in such cases (cf. ReM, App. C).
R	REALRES, CF I	CPXRES, SF I	
L	RECT	POLAR	Rectangular or polar format chosen for displaying <i>complex numbers</i> .
Ø	POLAR	RECT	
4°	DEG	setting any other <i>ADM</i>	Current <i>angular display mode (ADM)</i> setting: decimal <i>degrees</i> , <i>gradians</i> or <i>gon</i> , <i>radians</i> , <i>multiples of π</i> , and <i>sexagesimal degrees</i> .
4g	GRAD		
4r	RAD		
4π	MULTπ		
4"	d.ms		
/max or /2345	DENANY	Can only be modified by DENMAX	Fraction display settings. The current value of the maximum displayable denominator is shown behind the fraction bar (<i>startup default</i> and maximum is 9 999, displayed as /max). DENFAC and DENFIX set a specific character trailing said maximum displayable denominator in the <i>status bar</i> .
/2345f	DENFIX	DENFAC, DENANY	
/2345x or /2345-	DENFAC	DENFIX, DENANY	
64:1	1COMPL	setting any other <i>integer sign mode (ISM)</i>	<i>Finite integer</i> settings. First two digits tell the <i>word size</i> , the character after the colon the <i>ISM</i> . <i>Startup default</i> is 64 bits (the maximum settable) and 2's complement. <i>Carry & Overflow</i> may trail the <i>ISM</i> but are only lit if set.
64:2	2COMPL		
64:U	UNSIGN		
64:S	SIGNMT		

Indicator	Set by	Deleted by	Explanation, remarks
A	α , aON; if α is set	pressing EXIT in AIM unless in a menu, aOFF	<i>Alpha input mode (AIM) is set. Upper (for A) or lower (for α) case letters can be entered now.</i>
α	if A is set		
	program waiting for user input	program running	Will also be lit if a program is stopped by EXIT or R/S – then it will be cleared by next keystroke.
	see remarks	WP 43S idling	Flashes while a program is running; steady while a function is executing.
	top of program memory	else	Program pointer at step 0000.
	timer running in background	idle timer	See the TIMER (or stopwatch) application on pp. 250f.
	serial I/O in progress	idle communication line	See Serial Input and Output of Data and Programs on pp. 218f.
	data are being sent to printer		
	[UM]	[UM]	Toggles <i>user mode</i> (see pp. 279f).
	low battery	battery voltage > 2.5 V	Low battery will reduce processor speed automatically. Your WP 43S will shut off when voltage drops < 2.0 V.

The *startup default* configuration is indicated in the *status bar* like this:

2017-05-08 23:49 RL4° /max 64:2

On the other hand, choosing CLK12 time format (or M.DY), CPXRES, fraction display settings with a four-digit DENMAX and DENFIX, unsigned *finite integers*, *Carry* and *Overflow* set, having a program

waiting for input with *A/M* set, timer and printer running in background, *user mode* set, and a low battery will be reflected in the following *status bar*:

05/08/17 11:49pm CLE4r /3546f 64:u⁰ A

Note also and might show up at right end of the *status bar* (see App. D in the *ReM* for more).

Getting Special Information: RBR, STATUS, VERS, etc.

Some commands use the display in a special way. These operations are listed below:

1. The *Matrix Editor* is described comprehensively on pp. 157ff.
2. **FLAGS** **STATUS** returns free space available, memory currently used, and user *flags* set (see pp. 248f).
3. **RBR** allows for browsing the contents of all *registers* currently allocated (see pp. 249f).
4. **TIMER** calls the timer or stopwatch application (see pp. 250ff).
5. **FBR** browses all the characters defined in the fonts provided.

Further commands throw *temporary information* as defined on p. 65:

1. **ERR** and **MSG** display the corresponding error message. See the *IOI* and App. C of the *ReM* for more.
2. Some commands like **r**, **VIEW**, **x**, and **y** return results headed by text.
3. Commands returning two or three values (like **→P**, **R↔**, **DATE→**, **DECOMP**, **Ȑx**, **s**, **L.R.**, **SUM**, **M.DIM?**, **RCLIJ**, **Σ+** and **Σ-**) label their respective output rows accordingly (cf. e.g. pp. 18 and 103f).
4. A few far-reaching commands (like **CLALL**, **CLPALL**, **RESET**) ask you for confirmation before executing. Answer **Y** by pressing **3** or **N** by pressing **7**; also **ENTER↑** and **XEQ** will be interpreted as

'yes', while **EXIT** and **QUIT** will be interpreted as 'no'; any other keystroke will be ignored. Note that such explicitly confirmed actions cannot be undone by **UNDO**.

5. **VERS** generates a string showing version and build of the firmware running on your *WP 43S*:

WP 43S v0.1 b0123 by Pauli, Walter & Martin

WHO works in a similar way.

Localising Numeric Output

You can summon display preferences for *real numbers*, *times*, and *dates* all at once according to your region's customs and practices using dedicated commands (all contained in DISP). In the table printed overleaf,



- **Grouping** states the digit group interval – after *n* digits a narrow blank is displayed;⁵⁸
- **JG** states the year the Gregorian calendar was introduced in the particular region, typically replacing the Julian calendar (or national calendars in East Asia).⁵⁹

Note that some mode settings (determining the configuration of your *WP 43S*) and screen formats may be stored collectively at one location.

⁵⁸ This follows ISO 31-0. As far as we know, the *WP 43S* is the first pocket calculator displaying numbers this way as internationally agreed on. Previous calculators featuring limited displays had to use e.g. points or commas as crutches since they could not display narrow blanks.

Chinese counting and traditional mathematics work with powers of 10 000 while (originally Indian, then Persian, then) European counting and mathematics work with powers of 1 000. Thus, Chinese count 一 (= 1), 十 (= 10), 百 (= 100), 千 (= 1 000), 万 (= 10 000), 十万 (= 10 × 10 000), 百万 (= 100 × 10 000), 千万 (= 1 000 × 10 000), 亿 (= 10⁸), 十亿 (= 10⁹), 百亿 (= 10¹⁰), 千亿 (= 10¹¹), etc. The command **GAP 4** takes care of this notation while **GAP 3** formats the European way.

⁵⁹ Officially, the Gregorian calendar became effective in October 1582 in the catholic world. Many countries switched not earlier than in 1752, others joined even later. See the *IOI* for the command J/G. Note, however, there are still also other calendars in use, e.g. in the Muslim world. See also the chapter *Dates* below.

These are *stack height*, *display contrast*, *complete decimal display settings* (see next chapter), *angular display mode*, date and time display settings, parameters of integer and fraction display modes, curve fit model chosen, rounding mode, and the status of *flags A ... I*. STOCFG puts this information in the *register* you specify.⁶⁰ RCLCFG recalls such configuration information and will set your WP 43S accordingly.

	Radix mark ⁶¹	Grouping	Time	Date ⁶²	JG	Remarks
SETCHN	RDX.	GAP 4	CLK24	Y.MD	1949	
SETEUR	RDX,	GAP 3	CLK24	D.MY	1582	Applies to South America as well (and to the area of the former Soviet Union, Vietnam, Indonesia, and South Africa, but with deviating JG's).
SETIND	RDX.	- ⁶³	CLK24	D.MY	1752	Applies to India, Pakistan, Bangladesh, & Sri Lanka.
SETJPN	RDX.	GAP 3	CLK24	Y.MD	1873	
SETUK	RDX.	GAP 3	CLK12	D.MY	1752	Applies also to Australia ⁶⁴ and New Zealand.
SETUSA	RDX.	GAP 3	CLK12	M.DY	1752	

⁶⁰ Actually, it stores even more. See Section 6.

⁶¹ See <http://upload.wikimedia.org/wikipedia/commons/a/a8/DecimalSeparator.svg> for a world map of radix mark use. Looks like an even score in this matter. Thus, the international standard ISO 31-0 allows either a decimal point or a comma as radix mark, and requires a narrow blank as unambiguous separator of digit groups.

⁶² See http://upload.wikimedia.org/wikipedia/commons/0/05/Date_format_by_country.svg for a world map of date formats used. The international standard ISO 8601 states 24h for times and Y.MD for dates. This combination is commonly used in East Asia (see SETCHN and SETJPN).

⁶³ GAP 3 is set here but proper formatting would require separators every two digits over thousand. Think of *lakh* = 10^5 and *crore* = 10^7 . Actually, an amount of 50 cr. ($=5 \times 10^8$) reads 50,00,00,000 in Indian newspapers.

⁶⁴ 24h is taking over in the UK, so SETIND will work there then as well.

People using radix commas generally employ multiplication dots while those using radix points need a cross for multiplication to avoid misunderstandings. This latter convention causes further ambiguities in vector multiplication (see pp. 167ff).

Real Numbers: Changing the Display Format

As mentioned above, the ‘usual’ numbers you calculate with (decimal numbers or measured values) are *reals*. Any number you enter containing one **.** and/or an **E** is interpreted by your *WP 43S* as a *real number* unless there is additional information given (cf. pp. 65f). The majority of functions provided by your *WP 43S* operate on *real numbers*.

As soon as input of such a number is closed, its mantissa will be displayed right adjusted as far as possible (cf. p. 25). *Startup default* display format (ALL 0) shows all digits of the number if less than 17 are needed to do so; narrow spaces are inserted to separate digit groups. Your *WP 43S* will automatically turn to SCientific format (i.e. mantissa plus exponent, cf. pp. 25f) if more than 16 digits are needed.⁶⁵

Besides ALL and SCI, there are two more numeric output formats, FIX and ENG. ENG looks almost like SCI but the exponent will always be a multiple of three corresponding to the S/I unit prefixes – thus it is called the ENGineer’s notation (see the examples below and the ReM).

You can choose the switch point from ALL to SCI or ENG by specifying a positive parameter for ALL (telling up to how many zeros you want to see before the output shall be switched):

Example:

Input:

-700

[1/x]

Display:

-700■

-1.428 571 428 571 429×10⁻³

⁶⁵ No matter what display format or notation you select, these rounding options affect the display only. Your *WP 43S* continues using its full precision (typically 16 digits) internally always.

DISP	ALL	0 3	-0.001 428 571 428 571
10	/		-1.428 571 428 571 429×10 ⁻⁴
▲	ENG0VR		-142.857 142 857 142 9×10 ⁻⁶
▼	ALL	0 4	-0.000 142 857 142 857
10	/		-14.285 714 285 714 29×10 ⁻⁶
▲	SCI0VR		-1.428 571 428 571 429×10 ⁻⁵

And you can specify the number of decimals you want to see with SCI, FIX, or ENG (note the parameter of FIX and SCI specifies the number of decimals to be displayed while the parameter of ENG specifies the total number of digits within the mantissa minus one):

Format	Startup default display format (ALL 0)	FIX 5	SCI 5
Input 107.12345678 ENTER↑	107.123 456 78	107.123 46	1.071 23×10 ²
1/x 2 x 1.867 004 725 311 852×10 ⁻²	1.867 004 725 311 852×10 ⁻²	0.018 67	1.867 00×10 ⁻²

With FIX, the radix mark stays at the FIXed position defined; it floats in the other formats.⁶⁶ See more examples of displays varying according to popular choices for GAP, decimal radix mark, and multiplication symbol (compare with the examples shown on pp. 65ff.⁶⁷):

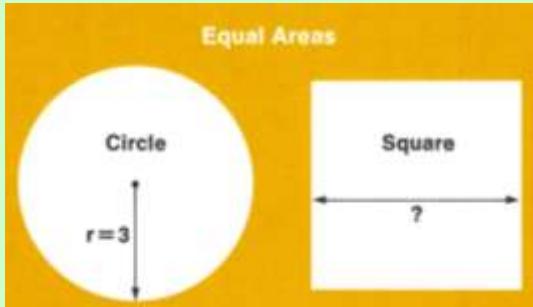
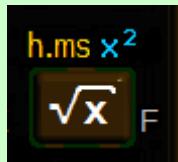
Format	FIX 3	ENG 6
Input 89012345678.9 ENTER↑	-89 012 345 678.900	-89.012 35×10 ⁹
	-89 012 345 678,900	-89,012 35×10 ⁹
	-890 1234 5678.900	-89.0123 5×10 ⁹
	-89012345678.900	-89.01235×10 ⁹

⁶⁶ Deviating from previous calculators, output of $\times 10^0$ is suppressed on your WP 43S.

⁶⁷ You will find nearly all functions for *real number* display format control in DISP: FIX, SCI, ENG, ALL, GAP, rounding, radix mark settings, and more. Please see the ReM.

Real Numbers: Squares and Cubes and their Roots

You find \sqrt{x} and x^2 on the keyboard of your WP 43S, while $\sqrt[3]{x}$ and x^3 are stored in EXP (cf. p. 27). The following example using these functions contains some of the most popular problems of antique mathematics:



What size square has the same area as a circle whose radius is 3 arbitrary units? 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:

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 $A_S = 4\pi r^2$.
And the volume of a cube is $V_{cu} = a^3$, while its surface is $A_{cu} = 6a^2$.
Thus,

DISP FIX 0 1

3 x² π x returns 28.3 for the area of the circle. Then
√x returns 5.3 for the edge length of the square.

Furthermore,

3 EXP x³ π x

4 x 3 / returns 113.1 for the volume of the sphere.

Then

³√x

returns 4.8 for the edge length of the cube with same volume. Thus,

x² 6 x

returns 140.3 for the surface of the cube.

Finally,

3 x² π x 4 x returns 113.1 for the surface of the sphere.

Actually, there was no necessity for calculating this last surface in this case – why?

Another problem, found in a calculator manual of 1976:



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.⁶⁸ *Donn* clears the mountain tops safely. How much did the volume of the balloon increase?

Solution:

Since the volume of a sphere is $V = \frac{4}{3}\pi r^3$, the difference of two such volumes is $\Delta V = \frac{4}{3}\pi(r_2^3 - r_1^3)$. One decimal shall do again.

8.25 [EXP] x^3
7.5 x^3 [−]
 π [×]
4 [×] 3 [/]

returns

561.5

139.6

438.7

584.9 m^3 for the volume increase.

Real Numbers: 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?

⁶⁸ In the *HP-21 Owner's Handbook*, the balloonist *Ike Daedalus* had to increase the radius from 25 to 27 feet – in 1975. Sometimes some progress was observable.

Solution:

$$7.5 \times^3 421.9$$

$$8.25 \times^3 561.5$$

$\Delta\%$ returns 33.1 % increase.

Another Example:

How about designing an almost optimum bicycle gearing? Feel free to choose sprockets and gear clusters to your liking.

Solution:

As long as drag may be neglected, an optimum gearing will show equal velocity ratios between subsequent gears (or uniform increase of distances per crank revolution). There are several ways you can reach this, depending on the number of sprockets chosen at front and rear. One inexpensive way is employing three front sprockets of 48, 36, and 24 teeth and getting a standard seven-gear cluster featuring 13, 15, 17, 20, 23, 26, and 30 teeth at the rear. This will result in the following distances travelled per crank revolution (d/cr in meter) for a 26" MTB:

Gear	1	2	3	4	5	6	7	8	9	10	11	12 ⁶⁹
Front	24				36			48				
Rear	30	26	23	20	26	23	20	23	20	17	15	13
d/cr	1.66	1.92	2.17	2.49	2.87	3.25	3.73	4.33	4.98	5.86	6.64	7.66
$\Delta\%$	-	15.7	13.0	15.3	15.3	13.2	14.8	16.1	15.0	17.7	13.3	15.4

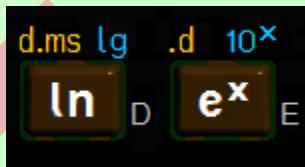
Assuming you bike with 60 rpm constantly, such a bicycle will run with velocities between 6 and 28 km/h. Employing also some statistical

⁶⁹ Note that you will get just 12 out of 21 theoretically possible gears this way. This is due to gear overlaps; and you will want to avoid extreme chain skew. On the other hand, if you are planning for a recumbent bicycle, this restriction might not apply anymore. Then you may think about a combination of three sprockets with a seven-gear cluster leading to 17 different, usable gears following the so-called "half-step and granny" scheme; speed increase in half-step range is 9% per gear step; this gearing covers velocities from 5 to over 40 km/h. For pictures, graphics, diagrams, tables, specifications, and further information about gearing bicycles, please order "*Die Fahrradschaltung*" (144 pages written in German) of the same author – just contact me.

functions provided on your *WP 43S* (i.e. $\Sigma+$, \bar{x} , and s explained on pp. 93ff), you will determine a mean speed increase per gear step of $(15.0 \pm 1.4)\%$, being quite uniform and convenient for town and country.

Real Numbers: Logarithms and Powers

Your *WP 43S* features two logarithmic functions on its keyboard and two more in EXP (cf. p. 27):



- [ln] calculates the *natural logarithm* of x , i.e. the logarithm of x to the base e (being Euler's constant, see CNST). Thus, [ln] inverts [ex].
- [lg] returns the (*common*) *decadic logarithm*, i.e. the logarithm of x to the base 10. [lg] inverts [10^x].⁷⁰
- [lb x] calculates the *binary logarithm*, i.e. the logarithm of x to the base 2. [lb x] inverts [2^x].
- [LOG_{xy}] is the most general of these four functions: it returns the logarithm of y to the base x . [LOG_{xy}] can be used to invert [y^x].

In the operating manual of the very first pocket calculator of the world featuring transcendental functions, the *HP-35* of 1972, is printed just a single example concerning this then new class of pocketable functions:

Example:

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:⁷¹

$$\frac{\text{altitude}}{[\text{feet}]} = 25\,000 \times \ln\left(\frac{30}{\text{pressure}/[\text{inches of Hg}]}\right)$$

⁷⁰ You may be used to the calculator label LOG for the decadic logarithm; though this is a mathematically ambiguous notation, so we avoided it.

⁷¹ Emphases in these quoted examples added by me.

Solution:

DISP **FIX** **0 0** should suffice.

30 **ENTER↑** **9.4** **/**

In

25 **E** **3** **X** returns **29 012.**

[We suspect that you may be on Mt. Everest (29 028 feet).]

Note the concise and factual style of this text. The *HP-35* was a calculator made by engineers for engineers, and the manual was alike. Said example was reprinted in the *HP-45 OH* of 1973. It underwent slight modifications before reappearing in 1975:



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*. Although the exact relationship of pressure and altitude is a function of many factors, *Eugobanks* knows that **an approximation** is given by the formula ...

This problem remained in the subsequent calculator manuals though the explorers changed for unknown reason. The picture above was added in 1976, and not every snowstorm was worth mentioning anymore. In 1978, however, a switch of units reached our explorers in the Himalayas – and also the weather and the methods changed:

Example (in *Solving Problems with Your Hewlett-Packard 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:



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

Where is *Wallace Quagmire*?

Solution:

760 **ENTER↑** 238 **/**

In

7620 **X** returns **8 847.**

Quagmire appears to be near the summit of Mt. Everest (8848 m).

And it seems neither he nor his barometer returned from this expedition since this example did not show up in the *HP-41C OH and Programming Guide* of 1980 anymore. Perhaps there was something wrong with the recalibration of his instrument?⁷²

By the way, the altitude approximation formula for standard SI units reads:

$$\frac{\text{altitude}}{[\text{m}]} = 7\,620 \times \ln \left(\frac{1\,013}{\text{pressure}/[\text{mbar}]} \right)$$

Beyond the barometric scale, there are more logarithmic scales used in science and engineering, e.g.

- in astronomy for assessing the brightness of stars or
- in chemistry for the power of acids (pH);
most popular may be
- the *decibel* (dB) in acoustics and electronics (see U→ on pp. 263f) and
- the so-called *Richter scale* for magnitudes of earthquakes.⁷³

⁷² Maybe this is the reason why the last three countries on this planet do not switch to SI – do they fear the recalibrations inevitably necessary for their measuring equipment?

⁷³ This name is still popular in the news although not quite true anymore. The actual moment magnitude scale for earthquakes differs but is still logarithmic.



Example:

One of the strongest earthquakes observed recently was the one causing the devastating tsunami in the Indian Ocean in December 2004. This earthquake had a magnitude of 9.1. Another one in March 2011 – with a magnitude of 9.0 – led to the Fukushima nuclear accident. Compare with the ‘great San Francisco earthquake’ of 1906, having had a magnitude of 7.9.

Solution:

The formula for comparing the energies released in two different earthquakes (with their magnitudes known) reads

$$\frac{E_2}{E_1} = 10^{1.5(M_2 - M_1)}$$

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

9.1 **ENTER↑** 7.9 **[−]**
1.5 **[x]** **[10^x]** returns 63. and
9 **ENTER↑** 7.9 **[−]**
1.5 **[x]** **[10^x]** returns 45. .

So the energy released in said Japanese earthquake in 2011 was 45 times greater than the so-called ‘great San Francisco earthquake’. And said earthquake in the Indian Ocean was even 63 times more intense.

Taking into account that published magnitudes of earthquakes never show more than one decimal, we did not lose anything real setting the WP 43S to FIX 0 here.

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.

Example:

What difference in magnitude is going to cause double destruction?

Solution:

Rewriting the formula above results in $\Delta M = \frac{2}{3} \lg \left(\frac{E_2}{E_1} \right)$. Thus, for double destruction we need a magnitude difference of

DSP 0 1

2 lg 2 x 3 / equalling 0.2 only.

But there are also friendlier applications of logarithms:

Example:

How many bits are required if the unsigned integer number 3.7×10^9 shall be the maximum to be handled by a microprocessor?

Solution:

3.7 E 9 lb x returns 31.8, so 32 bits shall suffice.

If we had a tristate logic, however,

RCL L 3 logxy returned 20.1, so 21 cells would suffice.

Providing y^x , your WP 43S also allows for raising any positive *real number* to an arbitrary real power, as well as any negative *real number* to an arbitrary integer power, all returning real results. See e.g. the *Mach number* formula shown on p. 44.

In combination with $1/x$, y^x also provides a simple way to extract roots:

Example (with startup default settings):

What is the fifth root of 17 ?

Solution:

This is equivalent to $17^{1/5}$, so 17 ENTER↑ 5 1/x y^x will do. This solution path may be faster executed than 17 ENTER↑ 5 g EXP 5√y.

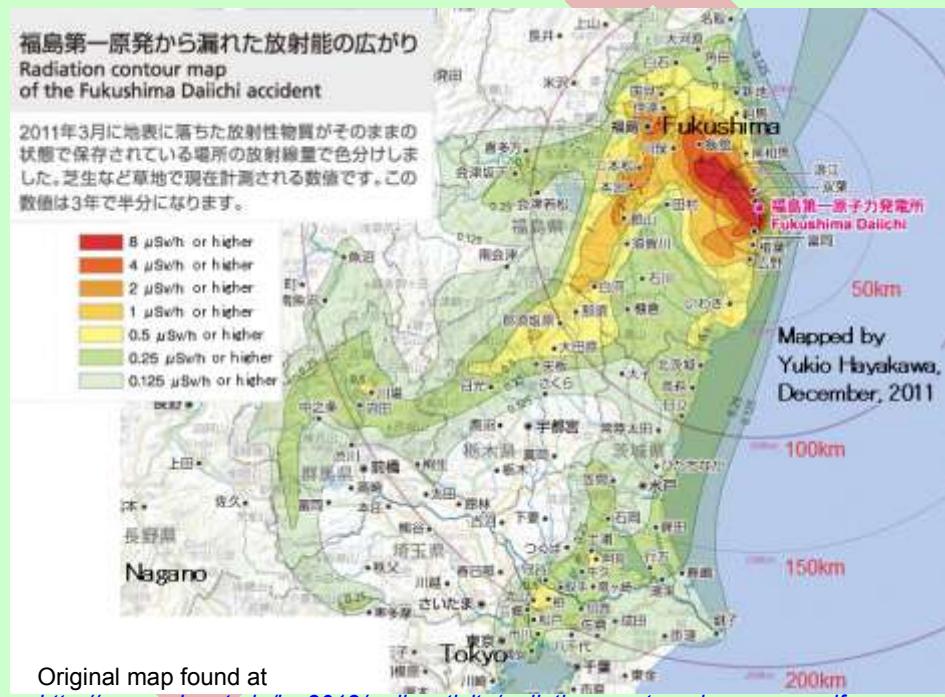
Both keystroke sequences will return 1.762 340 347 832 317, however.

Let's return to Fukushima for a final and (alas!) more down-to-earth application of powers and logs:

Example:

Locations in a distance of 30 km to the nuclear plant being devastated by the tsunami in March 2011 showed radioactivity in the soil of some $1\text{--}3 \text{ MBq/m}^2$ corresponding to an annual radiation dose of 4 mSv in 2013 (cf. the map below). Assume this was mainly caused by ^{137}Cs ; this radioactive Caesium isotope has a half-life of 30.2 years.⁷⁴

To the best of our knowledge today, an unborn child must not receive more than 1 mSv before birth. So when will it be reasonably safe to let the evacuated inhabitants of the villages in that area return to their homes finally?



Solution:

Assuming there will be no further nuclear accident in that area and no special measures will be taken, the isotopes set free will stubbornly

⁷⁴ With a probability of 94%, ^{137}Cs decays emitting an electron of up to 512 keV and a γ -ray of 662 keV. These facts are just for your information – they do not affect the calculation below.

decay following the inevitable laws of physics. Having had a (radio-) activity a_0 at time zero, the activity a at an arbitrary time t will be

$$a = a_0 \times 2^{-(t/T_{1/2})}. \text{ Hence, } t = T_{1/2} \times \ln\left(\frac{a_0}{a}\right).$$

1 mSv in nine months corresponds to an annual dose of $\frac{4}{3}$ mSv.
Well, 4 divided by $\frac{4}{3}$ equals 3, and

DISP FIX 00
3 EXP lb x
30.2 x returns **48.** years.

So you can recommend reproductive people shall rather not live in that area earlier than 2061. Elderly inhabitants may return far sooner.⁷⁵

Quite similar considerations apply to nuclear waste of power plants – at the bottom line, there are many tons of radioactive material produced decaying with half-lives exceeding thousand years; and this means you

⁷⁵ Note that different limits are considered ‘reasonably safe’ for the public by different national authorities. By nature, all such limits are arbitrary to some extent since we talk about probabilities here, and there are no step functions in probability but smooth transitions (cf. the chapter after next chapter). Furthermore, still a large fraction of world-wide knowledge about damage caused by radiation in human bodies in the long range is based on extrapolation of experiences collected since 1945 following two large-scale events in Japan. A third experiment was started in 1986 in the USSR – Belarus and the Ukraine have to bear the consequences. Mankind knows of the physics of radioactivity for some 120 years only so far, that’s little!

If an annual dose of 1 mSv would be the limit (as it is in some industrial nations), this would mean that reproductive people shall keep out of said area for 60 *years* instead. Note there are further risks linked to agriculture there – they are beyond the scope of this simple example though.

Please note this example covers a worst case scenario. Actually, radioactivity will be washed to deeper layers of soil with time, reducing the activity seen at the surface. And there are mitigation efforts in the area (many km²): at some places the contamination was washed off houses and trees, and the top layers of soil were removed, storing them in big black plastic bags ‘elsewhere’. Success of these efforts 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.

None of these efforts and effects, however, can ever reduce the given physical half-lives of the radioactive isotopes set free and spread in this nuclear accident.

have to ‘put them away’ safely for really long times – a task kept under wraps for decades but not solved by waiting so far.⁷⁶

The formula above is a nice example of a mathematically simple law of physics linking science and society quite closely.

Real Numbers: Hyperbolic Functions

Your *WP 43S* provides three hyperbolic functions and their inverses in the g-shifted row of EXP (cf. p. 27):

sinh	<i>Hyperbolic sine of x given in radians.</i>
arsinh	<i>Inverse hyperbolic sine returning radians.</i>
cosh	<i>Hyperbolic cosine of x given in radians.</i>
arcosh	<i>Inverse hyperbolic cosine returning radians.</i>
tanh	<i>Hyperbolic tangent of x given in radians.</i>
artanh	<i>Inverse hyperbolic tangent returning radians.</i>

Hyperbolic functions tell us something about hanging ropes, cables, chains, and the like. We found the following **example** in the *HP-32 OH*⁷⁷ though we modified the problem a bit:

In *Upper Lagunia*, a tram⁷⁸ carries tourists between two peaks in the *Baruvian Alps* that are the same height and 437 meters apart. How long

⁷⁶ Surprise! Mankind has absolutely no experience with locking something away for several thousand years. Note it must be tagged properly for the same time.

Sometimes you might meet people talking about ‘transmuting’ that entire long-living radioactive material by converting it to isotopes with significantly shorter half-lives by some nuclear reactions (never met anybody being more specific in this matter so far). If that would be physically possible for all that material, the energy needed for that transmutation process would easily outweigh the energy ‘produced’ by nuclear power plants. And as a matter of fact, the companies who made profits with those plants for decades are very reluctant in definitely solving the waste problem they created.

⁷⁷ This was *HP*'s first pocket calculator featuring hyperbolic functions. It was launched in 1978. Note the *SR50* of *Texas Instruments* (*HP*'s arch rival in those years of the so-called ‘calculator wars’) provided hyperbolic functions four years earlier already.

⁷⁸ Translator’s note: British readers might frown here at least.

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°.

Solution:



The travel time is given by the formula $t = \frac{d}{v} \times \frac{\tan \alpha}{\text{arsinh}(\tan \alpha)}$

Let's start with

DISP FIX 2 since we do not need more decimals displayed.

43 tan

ENTER↑

duplicates this intermediate result on the stack since we need it twice, in numerator and denominator.

EXP arsinh [7]

437 [x]

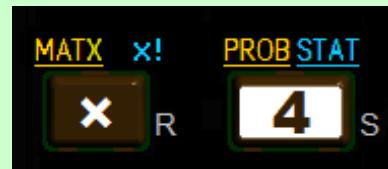
returns **489.30** m for the length of the cable.

135 [z]

returns **3.62**, i.e. a bit more than 3 ½ minutes.

Real Numbers: Probabilities – Factorials, Combinations, Permutations, and Distributions

Besides the keyboard commands **x!** and **Δ%**, you find a lot of probability and statistical operations in your *WP 43S*, going far beyond the *Gaussian distribution*. It contains all the preprogrammed functions implemented in *WP 34S* and more – presumably the maximum set available in a pocket calculator world-wide. These operations are stored in the adjacent menus PROB and STAT.



PROB includes the functions for *combinations* and *permutations*, complementing the factorial on the keyboard.

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* $8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 8!$

8 **x!** returns 40 320 for the total number of arrangements possible.



Example (continued):

The photographer looks through his viewfinder (in 1978) 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):

8 **ENTER** 5

PROB C_{yx} returns 56 for the number of sets.

Example (continued):

Again, there are different arrangements feasible. How many pictures of different widget arrangements are possible within these limits?

Solution:

The number of possible arrangements is $5 \times 4 \times 3 \times 2 \times 1 = 5!$ according to the statement above. Thus,

5 **x!** returns 120 for that number. And...

x 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). It can be obtained in one step by keying in

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

Furthermore, **PROB** contains ten continuous and five discrete *distributions* for calculating probabilities, confidence intervals, etc.⁷⁹ These functions share a few features:

- Discrete distributions (*Poisson*, *binomial*, *negative binomial*, *geometric*, and *hypergeometric*) are confined to integers. Whenever your WP 43S sums up a *probability mass function* (*PMF*) $p(n)$ to get a *cumulated distribution function* (*CDF*) $P(m)$, it starts at $n = 0$. Thus,

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

- Continuous distributions (*Cauchy*, *exponential*, *logistic*, *log-normal*, two kinds of *normal*, *Fisher's F*, *Student's t*, *Weibull*, and *chi-square*) operate on *real numbers*. Whenever your WP 43S integrates a function, it starts at left end of the integration interval. Thus, integrating a continuous *probability density function* (*PDF*) $f(x)$ to get a *CDF* works as

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

⁷⁹ In a nutshell, discrete statistical distributions deal with “events” governed by a known mathematical model. Such statistical events may be persons entering a store, radioactive nuclei decaying, faulty parts appearing, etc. The *PMF* then tells 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, but not more.

For doing statistics with continuous statistical variables – e.g. the heights of three-year-old toddlers – similar rules apply: Assume we know the applicable mathematical model; then the respective *CDF* gives the probability for their heights being less than an arbitrary limit, for example less than 1 m. And the corresponding *PDF* tells how these heights are distributed in a sample of let's say 1 000 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.

Translator's note for German readers: *PMF* und *PDF* entsprechen der *Wahrscheinlichkeitsdichte*, *CDF* der *Verteilungsfunktion* bzw. *Wahrscheinlichkeitsverteilung*.

- Many frequently used continuous *PDFs* look more or less like the ones plotted in the upper diagram at right. The lower diagram shows their corresponding *CDFs*, using the same scale and colors.

Typically, any *CDF* starts with a slope of almost zero, becomes steeper then, and runs out with its slope returning to zero. This holds even if the respective *PDF* does not look as nicely symmetric as the sample *normal distributions* plotted here.

Thus, obviously you will get the most precise results for the *CDF* on its left side using P . On its right side, however, where P slowly approaches 1, the *error probability* $Q = 1 - P$ is more precise. Thus, also Q is computed in your *WP 43S* for each distribution, independently of P . The definitions are:

- for discrete distributions:

$$Q(m) = \sum_{n=m}^{\infty} p(n)$$

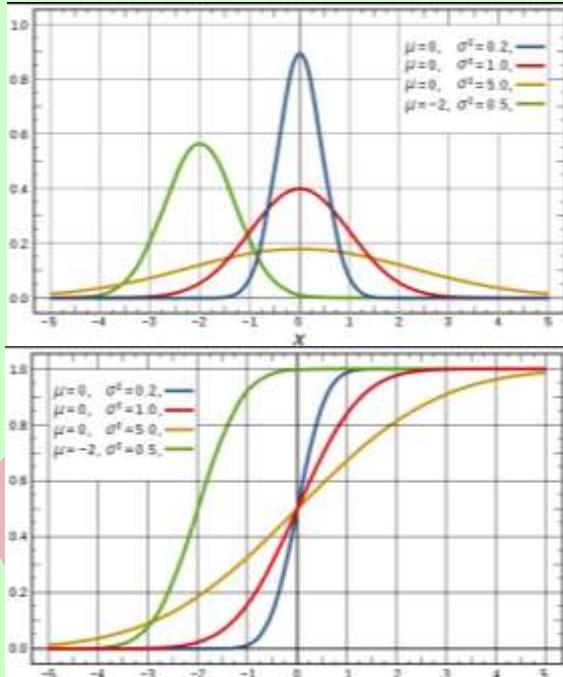
- for continuous distributions:

$$Q(x) = \int_x^{\infty} f(\xi) d\xi$$

- On your *WP 43S*, with an arbitrary *CDF* named ***XYZW*** (returning P) you will find the names

XYZW_u used for the function returning Q (also known as upper tail probability), if applicable,

XYZW⁻¹ used for the inverse of the *CDF* (the so-called *quantile function* or *QF*), and



$XYZW_P$ used for its *PDF* or *PMF*.

This naming convention holds for the *binomial*, *Cauchy* (a.k.a. *Lorentz* or *Breit-Wigner*), *exponential*, *Fisher's F*, *geometric*, *hypergeometric*, *log-normal*, *logistic*, *negative binomial*, *normal*, *Poisson*, Student's *t*, and *Weibull* distributions. *Chi-square* and *standard normal (Gaussian)* distributions are named differently for reasons of mathematical tradition. See PROB, e.g. on p. 104.

Find examples for calculating with distributions in the two chapters following now.

Real Numbers: From Probability to Statistics – Accumulating Data, Calculating Means, Standard Deviations, and Confidence Limits; Curve Fitting, Forecasting, and Checking Dices

There is also a wealth of commands for sample and population statistics in STAT, applicable in one or two dimensions. After clearing the summation registers by **CLΣ** initially, use **Σ+** to accumulate your experimental data (typically counted or measured values) as on previous calculators; weighted data require the weight in **Y**, pairs of data or coordinates of data points shall be entered in **X** and **Y** as usual. **Σ-** is provided for easy data correction.

The data analysis functions are found in STAT as well: e.g. *arithmetic mean* \bar{x} , *sample and population standard deviations* s and σ , and *standard error* s_m (also known as *standard deviation of the mean*).

Example:⁸⁰

Archibald is champion of the *Golden Bow*, his archers club. In his standard exercise, aiming at a black target disk of 1.5 m diameter at a distance of 50 m, his arrows scatter symmetrically around the center of

⁸⁰ Many of our customers live in a country where long range weapons play a significantly greater role than in most civilized societies, hence this explanatory example. Please note we refrained from using firearms here, though our resistance was strained almost to the limit.

the target showing quite a small variance. Actually, *Archibald's* statistics tells his arrows have a standard deviation (*SD*) of 1 *foot* at that distance. Assume his shots are distributed normally around the center of the disk, how often must he walk further than 50m to collect an arrow?

Solution:

DISP FIX 2
1.5 ENTER↑ 2 /
1 [U] x: feet→m
STO 0 1
/
+/- PROB Φ: Φ(x)
2 [x]

0.75, the radius of the target disk.
0.30, 1 *foot* in *meters*, Archibald's *SD*.
stores this value for later re-use.
2.46, the disk radius measured in Archibald's standard deviations.
 6.93×10^{-3} , and
0.01. Thus, Archibald has to collect an arrow in the green only once in 100 shots on average.

Example (continued):

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

Solution:

DISP FIX 3
15 [1/x]
2 /
Φ⁻¹(p)
+/- .75 x>y /
STO 0 0
RCL 0 1
Δ%

we want to see more precision here.
0.067, i.e. about 7% of Bill's arrows miss the target disk.
0.033 ~3% on either side.
-1.834, the corresponding lower limit of the standardized normal distribution.
0.409 m = Bill's *SD*. Just store it since we will need that again soon:

Note that the *SD* of Archibald's arrows is just...
-25.47 % better than Bill's, but his rate of misses is more than 10 times less.

There are applications of this methodology in industry as well, where the scattering or variation of a process (e.g. in production) is compared with its tolerance limits. Resulting from such comparisons, so-called *capability indices* are computed, directly linked to the amount of scrap to be expected in the process investigated. Please consult applicable literature and standards – look for *process capability*.

On the other hand, we may continue with our example as is, leading you over the border to advanced statistics.

Example (continued):

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 disk. Only two missed with one of them scratching the very edge of the disk. Cheers! But is this just a lucky chance success (within the usual scattering of results to be expected) or probably a consequence of his extra training efforts?

Solution:

Calculate *Bill's* new SD:

1.5 [ENTER] 50 [/] 0.030

2 [/] 0.015 = 1.5% misses on either side.

$\Phi^{-1}(p)$ -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 previous SD?

It is better (based on a *confidence level* of 95%) if it is lower than the 95% *confidence limit* of his old SD. We assume his old SD (s_o) was computed based on 60 shots. Then the formula for the single-sided lower 95% *confidence limit* of this old SD reads:

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

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

59 [STO] J stores the degrees of freedom.

.95 **PROB** $\chi^2:$ $(\chi^2)^{-1}$ calls the inverse chi-square, returning

77.931 .

/

0.757

\sqrt{x}

0.870

RCL **X** **0** **0**

0.356 m for σ_L .

Looks like Bill's training really made a difference!

Well ... with 95% confidence. If we had required 99% confidence instead, the lower *confidence limit* had been 0.337 m (you can easily verify this now) – then Bill's new weekend result would have been an insufficient indicator for a *significant* improvement.⁸¹

STAT contains also operations for curve fitting, featuring different regression models (linear, exponential, logarithmic, and power functions – see the *ReM*), their parameters, the forecasting functions \hat{x} and \hat{y} , and the *coefficient of correlation* r . The fit model applied will be displayed heading numeric output after any command related to fitting (i.e. after CORR, COV, L.R., s_{XY} , \hat{x} , and \hat{y}). And after **L.R.**, even the generic formula of the regression model applied will be shown (see examples below).

The command **BESTF** tells your *WP 43S* to select the regression model fitting your data resulting in the largest absolute *coefficient of correlation* (i.e. approx. 1). Then, an elevated asterisk (*) will trail the name of the fit model automatically chosen this way. Like with all other auto-functionality, you should know what you are doing here.

Example (from the *HP-27 OH* of 1976):

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 rock⁸² from various levels of

⁸¹ Statistics may cause that you might have more doubts than without – but such is life: doubts increase with knowledge. Only dumb people have no doubts.

Generally, standard *confidence limits* and *levels* (also those defined for indicating *significant differences*) may depend on the country or industry you are working in. Note the term *significant* is well defined in statistics – this definition may deviate from common language. Be sure to check the applicable valid standards before blindly copying the exemplary calculations demonstrated in this manual.

⁸² I hope not! A pebble would have done as well if not better.

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.⁸³

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

Unlike Galileo, you are equipped with a WP 43S; so what can you learn from this experiment? Let's look what we may find:

DISP **FIX** **4**

STAT

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

CLΣ

30 **ENTER↑** 2

Σ^+ returns

0.355 9
Data point 001
30.000 0
2.000 0

Note that Σ^+ took x and y , added them to the statistical sums, incremented the count of data points, and gives you feedback (also note this output contains *temporary information* as explained on p. 65). Your next input after Σ^+ will overwrite x :⁸⁴

50 **ENTER↑** 2.5 Σ^+

90 **ENTER↑** 3.5 Σ^+

130 **ENTER↑** 4 Σ^+

150 **ENTER↑** 4.5 Σ^+

⁸³ The raw data really do not look very plausible, and actually it is dubious whether Galileo made such experiments using the *Tower of Pisa* at all, but at least HP believed that its calculator users would believe in that story in 1976.

⁸⁴ Remember Σ^+ disables stack lift. Though note that accumulation of 2D data will slowly overwrite the stack.

Data point 005

150.000 0
4.500 0

OrthoF					
LinF	ExpF	LogF	PowerF		BestF
L.R.	r	s _{xy}	cov	ŷ	ŷ̂

BestFinstructs the *WP 43S* to pick the curve fitting model matching these data best (as explained above).**L.R.**

Power*	a ₁ =	150.000 0
y = a ₀ x ^{a₁}	a ₀ =	1.994 0
		7.722 6

Your *WP 43S* chose power regression as the model fitting these given data best. Let's check the *correlation coefficient*:**r**

returns

Power*

0.997 6

This is an almost perfect correlation. The equation expressing the experimental results best is hence $h \approx 7.72 \times t^{1.99}$ (with t measured in *pulses* and h in *Pisan feet*). Galileo could not know around 1600 yet, but we know today that $h = \frac{1}{2} g t^2$. The task to determine the size of a *Pisan foot* and Galileo's heartbeat is left for the reader.

In addition, we found the following linear regression **example** in various calculator owners' handbooks of 1976 - 78. It reads typical for the thinking at that time:

Big Lyle Hephaestus,⁸⁵ owner-operator of the *Hephaestus Oil Company*, wishes to know the slope and y-intercept of a least squares line for the

⁸⁵ Maybe his ancestors emigrated from Greece; *Hephaistos* is the ancient Greek god of fire and forging (and perhaps of underground natural resources as well?).



consumption of motor fuel in the United States (of America⁸⁶) against time since 1945. He knows the data given in the table:

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

Solution:⁸⁷

Hephaestus could draw a plot of motor fuel demand against time. However, with his WP 43S, Hephaestus has only to key the data into the calculator using the **$\Sigma+$** key, then press **L.R.**.



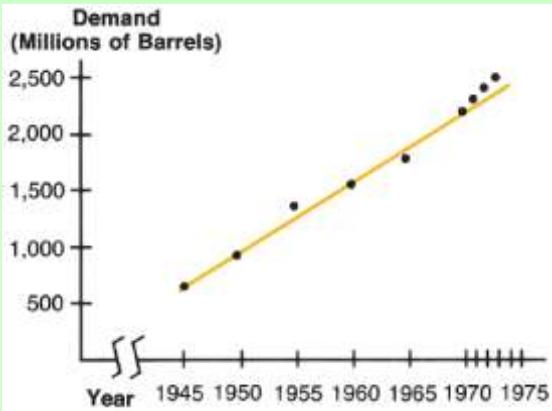
Your WP 43S chose linear regression as the model fitting these given data best. Let's check the *correlation coefficient*:

DSP 0 2 **r** returns Linear* **0.99**

We can concur with this choice. With these data in the statistical registers and such a good correlation, even extrapolations may make sense (mathematically).

⁸⁶ Differentiating from los Estados Unidos Mexicanos, for example.

⁸⁷ Looks like it was a time before CEOs and large staffs became fashionable.



Example (continued):

If Hephaestus wishes to predict the demand for motor fuel for the years 1980 and 2000, he keys in the new x values and presses \hat{y} . Similarly, to determine the year that the demand for motor fuel is expected to pass 3 500 million barrels, Hephaestus keys in 3 500 (the new value for y) and presses \hat{x} .

1980	\hat{y}	returns	Linear*	2 808.63
2000	\hat{y}	returns	Linear*	4 031.85
3500	\hat{x}	returns	Linear*	1 991.30

These were forecasts (extrapolations) of the demands in 1980 and 2000 at that time.
– the demand was expected to pass 3.5 billion barrels in 1992.

Another example from the HP-27 OH:

The *chi-square statistic* measures the goodness of fit⁸⁸ between two sets of frequencies. It's used to test whether a set of observed frequencies differ 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.

⁸⁸ Meaning: How good do both sets match?

Let's demonstrate the application of *chi-square statistic*⁸⁹ using the following problem, presuming *startup default* settings of your WP 43S:



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:

Expected frequency is $120/6 = 20$ for each number here. For calculating χ^2 , just enter:

25	[ENTER]	20	[-]	χ^2	25.
17	[ENTER]	20	[-]	χ^2 [+]	34.
15	[ENTER]	20	[-]	χ^2 [+]	59.
23	[ENTER]	20	[-]	χ^2 [+]	68.
24	[ENTER]	20	[-]	χ^2 [+]	84.
16	[ENTER]	20	[-]	χ^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 the *significance level* applicable (here 5%). As you have seen above already, also this function is provided in your WP 43S. Note that a significance level of 5% is equivalent to an error probability of 5% and a *confidence level* of 95%. Simply key in:

5	[STO]	J	5.	for the degrees of freedom;
.95	[PROB]	$\chi^2:$	(χ^2) ⁻¹	11.

⁸⁹ Do not confuse this χ^2 with the χ^2 distribution mentioned in previous chapter. The latter will be used in this example as well. Unfortunately, both chi-squares are called and written equally usually. It looks like the naming commission was inattentive here at the crucial time.

Since 5 is less than 11, χ^2 is small enough to conclude that this dice is fair (with 95% confidence).

Real Numbers: Some Industrial Problems Solved

To get an idea of further real-life opportunities covered by your *WP 43S* and of some constraints inherent to statistics, see the sample applications shown below. All of them are demonstrated employing the traditional 4-register stack but will work with the 8-register stack as well.

Application 1 (scrap rate, confidence limits):

Assume you own a little tool shop, produce axis pins in series, and want to know the quality of the parts you produce. You drew a *representative sample* of pins (all being nominally equal parts!) and precisely measured their real sizes using a proper instrument. How can you know your batch will be ok?

This is easy, based upon analysis of this sample.

Example:

Ten turned pins drawn from a batch produced on a precision lathe, diameters measured: 12.356, 12.362, 12.360, 12.364, 12.340, 12.345, 12.342, 12.344, 12.355, and 12.353. From earlier large scale investigations, you know that diameters from this production process follow a *Gaussian (or normal) distribution*.

Now you should just know your objective:

- Do you want to know what pin diameters you will get in your batch? Statistics cannot tell you about all of them but it will tell you where to find almost all (e.g. 99%) of them.

Example (continued):

Accumulate the measured values:

DISP FIX 3
STAT CLΣ

0 [ENTER↑] 12.356 Σ+

Data point 001

0.000

12.356

Continue this way for the remaining measured sample data:

12.362 Σ+ 12.360 Σ+ 12.364 Σ+ 12.340 Σ+
12.345 Σ+ 12.342 Σ+ 12.344 Σ+ 12.355 Σ+
12.353 Σ+

Data point 010

0.000

12.353

By knowing these measured pin diameters are drawn from a *Gaussian* process, you get the best estimates for the mean and standard deviation of your batch by pressing

⋮

ȳ =

0.000

ȳ =

12.352

[STO] ⌂

s

s_y =

12.352

s_x =

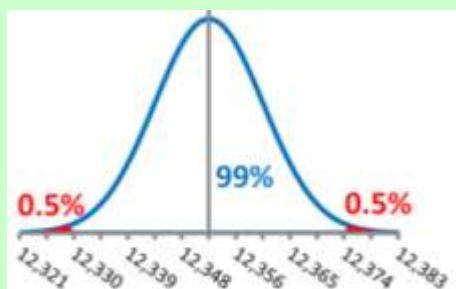
0.000

0.009

[STO] ⌂

We stored \bar{x} and s_x for the next steps already.

Now, if 99% of a batch is found inside some arbitrary symmetric limits of a *Gaussian* process then 0.5% will be out on either side since the *Gaussian* distribution is symmetric around its mean.



Based on the ten pins analyzed, you may expect 0.5% of all pins with diameters less than

.005 PROB

					0.005	
	Binom:	Geom:	Hyper:	NBin:		Poiss:
	Norml:	Lgnrm:	Cauch:	Expon:	Logis:	Weibl:
	t:	$\Phi:$	C_{yx}	P_{yx}	F:	$\chi^2:$

f Norml:

Normlp	Norml	Normlu	Norml ⁻¹
--------	-------	--------	---------------------

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% beyond either limit, this indicates your process may be running out of control.

Assume the pins shall have a nominal diameter of 12.35. Then – based on this sample analysis – you can safely commit to hold a tolerance of ± 0.05 (you will hardly produce any scrap). If your customer would try, however, to force you to accept a tighter tolerance of ± 0.02 , you must expect some losses:

12.35 ENTER↑

.02 -

Norml

x≥y

.02 +

Normlu

+

12.350

12.330

= lower limit.

0.006

= lower scrap = 0.6%.

12.350

12.370

= upper limit.

0.021

= upper scrap = 2.1%.

0.026

= total scrap.

What will hurt you even more than these 2.6% scrap you must expect will be the need to establish a very precise sorting tool or machine to ensure only good pins will go to your customer. Stay firm (if you can afford it) and refuse that customer request to constrict your tolerance limits – it may well be you cannot afford becoming weak here.

- Are you interested in the mean pin diameter of your batch? Do you want to know both upper and lower limits confining the mean with a probability of e.g. 95%?⁹⁰ So you know how much space you must provide to store a stack of e.g. 50 pins? Then determine the applicable mean value and the size of its variation. Use them to find said limits.

Example (continued):

Since we have got a sample out of a *Gaussian* process, the arithmetic mean is applicable, the standard error tells its variation, and *Student's t* is required. For the latter, we need its *degrees of freedom*. Press

STAT	▼	n	10.000
1	-	STO J	9.000
▲	s_m		0.003

Having 95% inside means having 2.5% outside at either end (cf. the diagram above). Thus generally one must take 0.025 and 0.975 as arguments in two subsequent calculations using the QF to get both 95% limits below and above the sample result:

.025	PROB	t: t⁻¹(p)	-2.262
x			0.006
STAT		̄x	12.352
x>y	R↓		12.352
x>y			0.006
-			12.346 = lower limit.

⁹⁰ The value of 95% is called the *confidence level* of this calculation. In this example, you calculate the 95% *confidence limits* for the mean value. Also 99% are frequently applied. Be sure to check the applicable valid standards before copying the example calculations here. Of course, you are free to apply other confidence levels wherever they fit your needs.

Translator's note for German readers: *Confidence limit* entspricht der *Vertrauensbereichsgrenze*, *confidence level* dem *Vertrauensniveau*.

⁹¹ **̄x** returns \bar{x} and \bar{y} as shown above. Only \bar{x} is interesting in this example, however, so pressing **x>y R↓** moves \bar{y} quickly out of the way. In a program, **DROPy** will be a better alternative since it leaves the *stack order* as is.

RCL L

2 **X** **+**

⁹²

0.006 = last x .

12.358 = upper limit.

Now you know where you can expect the future mean diameter of such batches. Hence a stick being

50 **X**

617.900 long inside will suffice for holding 50 pins in 97.5% of all cases.

12.346 and 12.358 are the 95% *confidence limits* of the mean calculated above. So here is a chance of 2.5% that the mean will be < 12.346 and an equal chance that it will be > 12.358. These chances are an inevitable consequence of the fact that you know something about a small *sample* only (drawn out of a large *population*), but want or have to tell something about said total population.⁹³ If you cannot live with these uncertainties or the widths of the confidence limits, do not blame statistics but collect more or more precise data instead.

Application 2 (quick measuring system analysis):

Your colleagues in R&D have specified particle accelerator beam pipes made of a special stainless steel shall have a magnetic susceptibility ≤ 0.01 . How can you verify whether or not the susceptibility meter available in your lab is sufficiently precise to control the series production of those tubes?

Solution:

1. Collect 30 samples of material covering the susceptibility range you are interested in.⁹⁴ Mark each sample unambiguously (e.g. by a number).

⁹² The upper *confidence limit* can be calculated this easy way since $t^{-1}(p)$ is symmetric. Else it would have been necessary to repeat the above calculation (except the last two steps) with an input value of 0.975.

⁹³ Statisticians call these chances ‘probabilities of a type I error’ or ‘probabilities of an error of the first kind’.

Translator’s note for German readers: Type I error entspricht dem Fehler 1. Art.

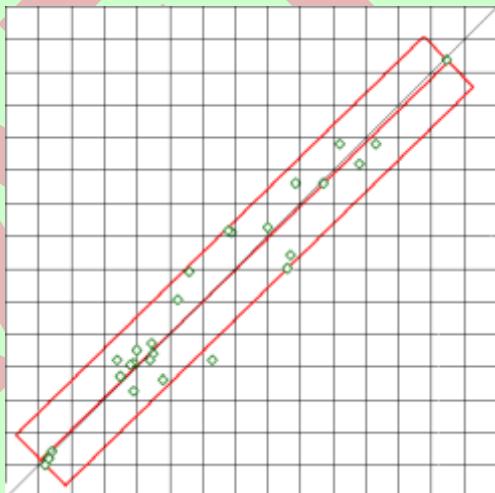
⁹⁴ This range could extend from 0 to about 0.015 here (nature allows for positive susceptibilities only). Note there is no requirement to know the exact susceptibilities of your samples beforehand – they shall just fall in said range and must be robust

2. Use the measuring instrument under investigation to measure all samples carefully under controlled conditions. Record as many decimals as possible. Write each measured value in a table next to the respective sample number.
3. Measure a second time, but following another sample sequence. Don't allow for looking at the values measured in step 2! Hiding the data measured in step 2 will be helpful. Write each second measured value behind the first one of the respective numbered sample.
4. Now clear the statistical registers via **CLΣ**. Then enter all 30 pairs of values using **Σ+**. The first measured value shall be y , the second x – thus, the input will be

mv1 **ENTER↑** **mv2** **Σ+**

for each sample (alternatively, you can enter your statistical data into a matrix and then accumulate all points at once – see pp. 178f).

5. It is recommended to plot these 30 points. The diagram shall look like an ant trail around the center line $y = x$ (see an exemplary plot here).⁹⁵



enough to stay constant in your measurements. No need for any investment in expensive gauges here – scrap may well do.

⁹⁵ Even pencil plotting on quadrille paper will do. Since it is important here, we might think about implementing some basic scatter plot abilities in this calculator. See App. D of the ReM.

Go to an expert in metrology if your diagram should deviate fundamentally from the one pictured here.

In my four decades professional experience, such correlation diagrams are the most powerful though easy tools to assess the quality of real-life (e.g. industrial) processes. Even manually drawn clean correlation diagrams will support your decisions far better than just staring at numbers. The resolution required for showing all measured points clearly separated from each other might, however, exceed the capability of such a small screen by far. The calculations will be correct nevertheless.

6. Identify the most distant point (x, y) from said line. Remove it from the data set using $\Sigma-$.
7. Let your WP 43S fit a straight line through the remaining 29 points and compute $c_0 = \frac{T}{30s_x s_y} \sqrt{\frac{s_x^2 + r^2 s_y^2}{1 - r^2}}$ with T being the width of the tolerance zone you want to control:

STAT **OrthoF**

r **STO** **0** **0**

s

R↓

x

R↑

1 **RCL** **-** **0** **0**

s

.01

select the orthogonal linear fit model.

get the *coefficient of correlation* and store its square.

get s_x^2 .

roll it out of the way.

get $r^2 s_y^2$.

return s_x^2 from the top *stack register* and calculate the numerator

and the denominator.

this is the second factor now.

divide by $30 s_x s_y$.

this returns c_0 for our T now.

If you get $c_0 \geq 1$ then this measuring device may be used for controlling this tolerance zone under these conditions (i.e. it is a *capable* instrument for this control job) – else look for a more precise instrument, better measuring conditions, or a wider tolerance.

Application 3 (significant changes):

Assume you have taken a sample out of an arbitrary industrial production process at day 1. Then you have changed the process parameters, waited for stabilization, and have taken another sample of same size at day 2 (there may well have been a longer time interval between both sampling days). Being serious, you have thoroughly measured and recorded a critical quantity (e.g. a characteristic dimension) for each specimen investigated at both days. Now: do the two samples show any *significant difference*?

The following simple three-step test is well established. It may easily save yourself some unwanted embarrassments in your next presentation or after your next publication:

1. Accumulate your sample data. Then let your WP 43S compute the means and *standard errors* for both samples, and their *normalized distance* $d = |\bar{x} - \bar{y}| / \sqrt{s_x^2 + s_y^2}$. If you are working with four stack registers, this calculation could look like the following:

STAT **S_m**
x² **x̄y** **x²** **+** **fx**
̄x
- **PARTS** **|x|**
x̄y **/** **STO D**

returns both standard errors in X and Y.
so this is the entire denominator.
returns both \bar{x} and \bar{y} .
thus, this is the numerator
and this is d .

Now prepare the next two steps:

STAT **▼** **n**
1 **-**
STO I

recall the number of samples measured.
calculate the *degrees of freedom f*
and store them.

2. Let your WP 43S calculate the critical limit t_{cr} of *Student's t* for *f degrees of freedom* and a probability of 97.5% now:

.975 PROB t_c

as mentioned above, the requested QF lives in PROB. It takes the degrees of freedom stored in I to get t_{cr} .

If $d < t_{cr}$ now, then the test indicates the difference between both samples being due to random deviations only. Congratulations – you have got a robust process regarding the parameters you changed!

Else continue.

3. Let your WP 43S compute a new critical limit t_{cs} for *f* and 99.5%:

.995 t⁻¹(p) get t_{cs} .

If $d \geq t_{cs}$ now, then the 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 based on the information provided – your samples may contain too little data or your measurements were not precise enough or the process is scattering too far etc. Though do not let your audience lead you in temptation: stay silent or mumble something like “investigation in progress”!⁹⁶

Application 4 (operating characteristics):

Assume you draw a sample of 20 parts out of a batch of 100 parts and check the sample thoroughly. What is the probability P to find at least one defect in the sample if the overall probability for a defect in this batch is 5%, 2%, or 1%?

This is a textbook example for applying the hypergeometric distribution. $P(n \geq 1)$ equals $100\% - p(n = 0)$. Thus, the solution is as simple as this:

DISP	FIX	3
100	STO	I
20	STO	K
0.05	STO	J
0	PROB	Hyper: Hyper
1	x \geq y	-
0.02	STO	J
0	Hyper	
1	x \geq y	-
0.01	STO	J
0	Hyper	
1	x \geq y	-

returns 0.319
returns 0.681
returns 0.638
returns 0.362
returns 0.800
returns 0.200

Even with 5% defects in the batch the odds are only some 2 out of 3 that any defect is found in such a sample at all!

⁹⁶ This test goes back to *DGQ (Deutsche Gesellschaft für Qualität)*. It assumes your samples are drawn from a *Gaussian* process which is frequently the case in real life (but needs to be verified).

You will find more examples of statistical applications in the manuals of various vintage *HP* calculators, especially of the *HP-27*.

Many more statistical functions are collected in STAT (e.g. covariances, means and standard deviations for weighted data, *geometric means* and *scattering factors*, and all your accumulated sums) – just look them up there and check the respective entries in the *IOI*.

Furthermore, we strongly recommend you consult a good statistics textbook for more information about statistical methods in general, the terminology used, and the mathematical models provided.

Real Numbers: Summary of Functions

The majority of the functions your *WP 43S* features are for calculations operating on *real numbers*. It provides many more than the numeric functions shown on pp. 29ff, 30f, and 78ff in various applications. See all functions listed below:

- General mathematics:
 - *Monadic* functions:

\sqrt{x} & x^2 , $\sqrt[3]{x}$ and x^3 , 2^x and $\log x$, 10^x and $\log_{10} x$, e^x and $\ln x$, $\sin x$, $\cos x$, $\tan x$, and their inverses work as demonstrated above and you learned in school (see also pp. 117ff for more information about angular I/O),
 e^{x-1} and $\ln(1+x)$ return more accurate results with $x \approx 0$,
 $\text{ceil } x$ returns the smallest integer $\geq x$,
 $\text{floor } x$ returns the greatest integer $\leq x$,
 $\text{SDL } n$ shifts digits left by n decimal positions, equivalent to multiplying x times 10^n ,
 $\text{SDR } n$ shifts digits right by n decimal positions, equivalent to dividing x by 10^n ,
for $\sinh x$, $\cosh x$, $\tanh x$, and their inverses see pp. 88f,
 $\text{+/-} x$ returns $x \times (-1)$ for closed input (a.k.a. ‘unary minus’), and
 $(-1)^x$ returns $\cos(\pi x)$ for non-integer x .

- *Dyadic* functions:

[+], **[−]**, **[×]**, **[/]**, **[y^x]**, and **[$\sqrt[3]{y}$]** work as was shown above and you learned in school; use ...

idiv for integer division

(e.g. **7.8 [ENTER]↑ 3.2 [INTS] idiv** returns 2),

(**idivr** additionally returns the remainder in Y),

log_xy for the logarithm of y for the base x

(e.g. **625 [ENTER]↑ 5 [EXP] log_xy** returns 4),

max (or **min**) for the maximum (or minimum) of x and y;

RMDR returns the remainder of y/x (s. pp. 134f for examples),

MOD returns y mod x (see p. 135 for examples), and

|| returns $\left(\frac{1}{x} + \frac{1}{y}\right)^{-1}$ for $x \times y \neq 0$ and 0

else, being handy in electrical engineering.



- *Triadic* functions:

xMOD returns $(z \cdot y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$, and

^MOD returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$

(e.g. **73 [ENTER]↑ 55 [ENTER]↑ 31 [INTS] ^MOD** returns 26).

- Isolating parts of numbers:

EXPT for the exponent of x and **MANT** for its mantissa,

FP (or **IP**) for the fractional (or integer) part of x,

|x| for the absolute value of x, and

SIGN for the *signum* of x; thus, SIGN returns 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$ or non-numeric data.

- Rounding:

RDP n rounds x to n decimal places in FIX format

(e.g. **1.234 567 89E-95 RDP 99** will return **1.2346×10⁻⁹⁵**),

ROUND rounds x using the current display format (like RND did on HP-42S),

ROUNDI rounds x to next integer ($\frac{1}{2}$ rounds to 1), and
RSD n rounds x to n significant digits.

- Conversions:

→P converts rectangular *coordinates* to polar ones (cf. pp. 18f), while **R↔** converts vice versa.

Angular and time conversions are covered on pp. 117ff and 141f.
For *unit conversions* see pp. 263ff.

- Boole's algebra:

AND, **NAND**, **OR**, **NOR**, **XOR**, **XNOR**, and **NOT** operate on *real numbers* like these operations did in the HP-28S, i.e. x and y are interpreted before executing the operation. Zero is 'false' (= 0); any other number is 'true' (= 1).

Thus, e.g. **13.5 [ENTER]** **-7.2 [BITS]** **AND** returns 1.

- Probability and statistics (unless introduced and explained on pp. 89ff already):

Γ(x) calculates the *Gamma function*,

InΓ returns the natural logarithm of the *Gamma function*, allowing also for calculating really great factorials:

Example: What is $5432!$?

Solution: Remember $\Gamma(x + 1) = x!$ So, entering

5433 [X.FN] InΓ

10 [In] [/]

returns 17 931.480 374 010 87 as the decadic logarithm of the result; now calling

PARTS FP [10^x] returns some 3.023 for its mantissa.

Thus, $5\ 432! \approx 3.023 \times 10^{17\ 931}$.

RAN# returns a (pseudo) random number between 0 and 1, and
SEED stores a seed for RAN#.

Find additional information about all the statistical and probability functions provided on your WP 43S in the ReM.

- Percentages:

calculates $\frac{xy}{100}$, leaving y unchanged (so you can easily calculate another percentage of the same base after CLX).⁹⁷

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** returns 310. US\$ for the sales tax;
 returns 6 510. US\$ for the total cost.

If the dealer gives you a 10% discount on the car, what will your total cost be?

6200 **ENTER↑**
10 returns 5 580. US\$ for the discounted price;
5 returns 5 859. US\$ for the total cost.

calculates the percentage of change from y to x , returning $100 \frac{x-y}{y}$, leaving y unchanged again for the same reason.

You can use 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 and what percent is your margin?

21.99 **ENTER↑** **26.5** returns 20.5 % markup.
26.5 **ENTER↑** **21.99** returns -17.0, i.e. 17 % margin.

%MRR calculates the mean rate of return in % per period, i.e. $100 \left(\sqrt[z]{\frac{x}{y}} - 1 \right)$ with y = present value, x = future value after z periods,

⁹⁷ Actually, that's the real benefit of the function . Every engineer or scientist should be able to calculate e.g. 17.5% of 3456 faster via **3456** **ENTER↑** **.175** though.

%T calculates $100 \frac{x}{y}$ (called “% of total”),⁹⁸

%Σ returns $100 \frac{x}{\sum x}$, and

%+MG calculates a sales price by adding a margin of $x \%$ to the cost y ; 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 186.4.⁹⁹

- Advanced mathematics (see the *ReM, App. J* for comprehensive information about the functions listed below):

- 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,

NEXTP the next *prime* number greater than x ;

sinc returns $\frac{\sin(x)}{x}$ for $x \neq 0$ and 1 for $x = 0$ (*input* is converted to *radians* before calculating – see pp. 117ff),

W_p returns the principal branch of *Lambert's W* for given $x \geq -1/e$, **W_m** the negative branch of it,

W⁻¹ returns x for given W_p (≥ -1), and

ζ(x) *Riemann's Zeta function*.

⁹⁸ I still wait for somebody convincing me of the use of this financial function. Please see also **%+MG** and the next footnote.

⁹⁹ Each and every engineer or scientist will be able to produce the very same result significantly faster via 221.82 **ENTER↑** .81 **X** (sometimes, you really might get the impression that financial people are mathematically slightly more challenged and thus need a little extra push ... emmh ... support sometimes – see also **%T** above).

Call H_n (or $H_{n\mu}$) for the *Hermite polynomials* for probability (or physics),
 L_n (or $L_{n\alpha}$) for *Laguerre's (generalized) polynomials*,
 P_n for the *Legendre polynomials*, and
 T_n (or U_n) for the *Chebyshev polynomials of 1st (or 2nd) kind*.

- *Dyadic* functions:

AGM returns the *arithmetic-geometric mean*,
 $J_y(x)$ the *Bessel function of first kind* and order y ,
 $\beta(x,y)$ *Euler's Beta function*,
 $\ln\beta$ the natural logarithm of *Euler's Beta function*,
 γ_{xy} the *lower incomplete gamma function*,
 Γ_{xy} the *upper incomplete gamma function*,
 $I\Gamma_p$ and $I\Gamma_q$ return the *regularized gamma function* (one of two kinds).

- *Triadic* function:

I_{xyz} returns the *regularized beta function*.

Angles and Trigonometric Functions

For dealing with *angles* on your WP 43S, you may choose out of five *angular display modes (ADM)* featured: DEG, RAD, GRAD, MULT π , and D.MS, all stored in MODE.¹⁰⁰ Angles are entered as *real numbers*. They are interpreted according to the current *ADM* as indicated in the *status bar* by $\frac{d}{d}$, $\frac{r}{r}$, $\frac{\pi}{\pi}$, $\frac{g}{g}$, or $\frac{''}{''}$ (cf. p. 70) as soon as a function expecting angular input is called.

Exception: Sexagesimal angles must be entered in the format dddd.dmmsspp – with dddd standing for integer *degrees*, mm for *angular minutes*, ss for *seconds*, and pp for hundredth of *seconds* – terminated by **d.ms**.



Example:

Entering **12.3454321 d.ms** returns $12^\circ 34' 54.32''$.

There are some functions (e.g. ARCSIN) operating on *real numbers* and returning *angles*. The returned values will be automatically tagged according to the current *ADM*. Assume FIX 3 and RDX. are set for the following **examples**:

In ADM5 TRI arccos will return ...
$\frac{r}{r}$	1.047 r
$\frac{\pi}{\pi}$	0.333 π
$\frac{^o}{^o}$	60.000 o
$\frac{''}{''}$	60 $^\circ$ 0' 0.00 $^{''}$ ¹⁰¹
$\frac{g}{g}$	66.667 g

¹⁰⁰ Translator's note: This traditional calculator notation is misleading in German at least: DEGrees on your WP 43S mean "Grad", while calculator GRADs are generally called "Gon" in Continental Europe.

¹⁰¹ Note there are no leading zeroes in the angular *minutes* and *seconds* sections. And this display format can neither take nor display anything smaller than 0.01''. On the other hand, it will display down to that fraction always and cannot be shortened.

Whenever you see a number formatted alike on your WP 43S you know it is an *angle*.

Other functions presume their inputs being *angles*, e.g. SIN. Decimal inputs are generally interpreted as *angles* of the current ADM.

Fourteen angular conversions are provided, all accessible via $\text{L}\rightarrow$:

From ... to ...	sexages. degrees	decimal degrees	radians	grads/ gon	multiples of π	current ADM
sexagesimal degrees	—	$4^{\circ} \rightarrow 4''$	—	—	—	$4 \rightarrow 4''$
decimal degrees	$4'' \rightarrow 4^{\circ}$	—	$4^r \rightarrow 4^{\circ}$	—	—	$4 \rightarrow 4^{\circ}$
radians	—	$4^{\circ} \rightarrow 4^r$	—	—	—	$4 \rightarrow 4^r$
grads/gon	—	—	—	—	—	$4 \rightarrow 4^g$
mult. of π	—	—	—	—	—	$4 \rightarrow 4\pi$
current ADM	$4'' \rightarrow 4$	$4^{\circ} \rightarrow 4$	$4^r \rightarrow 4$	$4^g \rightarrow 4$	$4\pi \rightarrow 4$	—

Example:

DISP FIX 5

MODE MUL π

Choose *multiples of π* as ADM and 4π will appear in the *status bar* and stay there for the time being.

300 $\frac{1}{x}$

$\text{L}\rightarrow 4 \rightarrow 4^r$

$4^r \rightarrow 4^{\circ}$

$4^{\circ} \rightarrow 4''$

$4'' \rightarrow 4$

0.003 33

So $\pi / 300 \dots$

0.010 47 r

are 0.010 47 radians

0.600 79 $^{\circ}$

or exactly 0.6°

0°36' 0.00"

or 36 angular minutes

0.003 33 π

equivalent to $\pi / 300$ still.

Note $4 \rightarrow 4^r$ ‘knew’ it had to convert from *multiples of π* since this function expects angular input and took the current ADM setting into account. And $4'' \rightarrow 4$ converted to *multiples of π* since this is the current ADM still. Angular output of operations is tagged generally and will stay tagged.

You have learned about trigonometric functions in school. Thus, we just demonstrate their use with *angles* with one example.

Example (found in the HP-25 OH):¹⁰²

Lovesick sailor Oscar Odysseus dwells on the island of *Tristan da Cunha* ($37^{\circ}03'S$, $12^{\circ}18'W$), 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* ($15^{\circ}55'S$, $5^{\circ}43'W$), use the following formula to calculate the great circle distance that *Odysseus* must sail in order to court her.

Solution:

The formula for the great circle distance d in *nautical miles* is:¹⁰³

$$d = 60 \times \arccos[\sin(B_s)\sin(B_d) + \cos(B_s)\cos(B_d)\cos(L_d - L_s)]$$

with B_s and L_s being the latitude and longitude of the start (*Tristan da Cunha*) and B_d and L_d being the latitude and longitude of the destination (*St. Helena*).¹⁰⁴ Hence, with the numbers inserted, this formula reads:

$$\begin{aligned} d = 60 \times \arccos & [\sin(37^{\circ}03'S)\sin(15^{\circ}55'S) \\ & + \cos(37^{\circ}03'S)\cos(15^{\circ}55'S) \\ & \times \cos(5^{\circ}43'W - 12^{\circ}18'W)] \end{aligned}$$

Set the appropriate number of decimals and calculate from inside out, remembering the trigonometric functions assume their input being in the current *ADM* as indicated in the *status bar*.

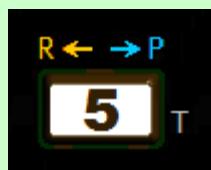


TRI	cos	0.99
15.55	STO 0 1	cos
x		0.96
37.03	STO 0 0	cos
x		0.96
RCL 0 0	sin	0.80
RCL 0 1	sin	0.60
x		0.76
+		0.27
arccos		0.17
.d		0.93
60	x	21°55'24.66"
	returning	21.92
		1 315.41 nmi that Odysseus must sail to visit Penelope.

Mixed Calculations: Coordinate Transformations in 2D, Flight Directions, Courses over Ground, etc.

Two functions are provided for converting polar or rectangular coordinates in two dimensions. Input and output data are in *stack registers X* and *Y* here.

$\rightarrow P$ converts 2D Cartesian coordinates *x* and *y* to polar magnitude or radius *r* in *X* and angle *θ* in *Y*.



Example (assuming *startup default* settings):

Convert $(x, y) = (6, 4.5)$ to polar. Two decimals shall do, and *ADM* shall be *decimal degrees*.

Solution:

DISP FIX 2
MODE DEG
4.5 ENTER↑ 6 $\rightarrow P$ returns

<i>θ</i> =	36.87°
<i>r</i> =	7.50

i.e. a vector of magnitude 7.5 pointing up right from the origin with an angle of some 37° to the positive x -axis.

R \leftrightarrow converts 2D polar magnitude or radius r in X and angle θ in Y to Cartesian coordinates x and y . Both functions honour the *ADM* settings and tags as described in previous chapter.

Example (continued):

Convert the returned angle of the conversion executed above to radians, and then convert the resulting coordinates (r, θ) to rectangular.

Solution:

x \gtrless y

L \rightarrow **4 \rightarrow 4r**

x \gtrless y

R \leftarrow returns

7.50
36.87°
7.50
0.64r
4.50
6.00

$y =$
 $x =$
as expected.

Note angular input can range from $-\infty$ to $+\infty$; angular output, however, is confined to -180° to $+180^\circ$ or its equivalents, e.g. $-\pi$ to $+\pi$ in radians.

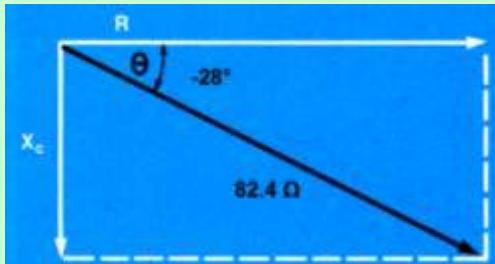


Example (triggered by the HP-67 OH and Programming Guide):

In an electronic circuit designed for alternating current, an overall impedance of $82.4\ \Omega$ is measured, and voltage lags current by 28° . Replacing said circuit by an equivalent containing just a resistor and a capacitor in series, what would be the resistance R and the capacitive reactance X_C therein?

Solution:

The values measured correspond to an impedance vector of magnitude 82.4 pointing down right at an angle of -28° to the positive x -axis. R is



its component parallel to the x -axis, and X_c is its perpendicular component parallel to the y -axis:

DSP 0 1

-28 ENTER↑ 82.4

R↔ returns

$y =$

-38.7

$x =$

72.8

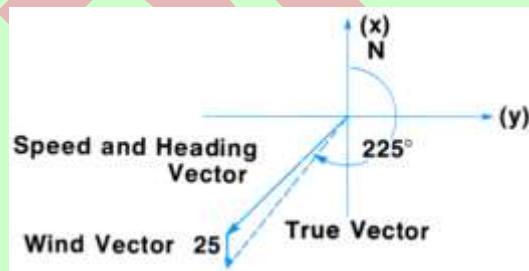
i.e. a resistance of 72.8 Ω and a reactance of 38.7 Ω .

By the way, you can use **P→** and **R↔** also to convert 3D cylinder coordinates to Cartesian and vice versa, since z is kept unchanged.

Having learned about **P→** and **R↔** as well as about **Σ+** and **Σ-**, we can profit from combining these functions. Here is an example:

Example (from the HP-21 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:

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 summation registers. 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 corresponds with navigational convention.)

CLR**CLΣ****DISP****FIX [2]****225 [ENTER] 125****R⁻¹**

returns

clears the summation registers.

y =

-88.39

x =

-88.39**STAT****Σ+****180 [ENTER] 25****R⁻¹**

returns

y =

0.00

x =

-25.00**Σ+****SUM****→P**

returns

θ =

-142.06°

r =

143.77**x²y****360 +** returns**217.94°**

(we have to change the angle to become positive for being in line with navigational convention).

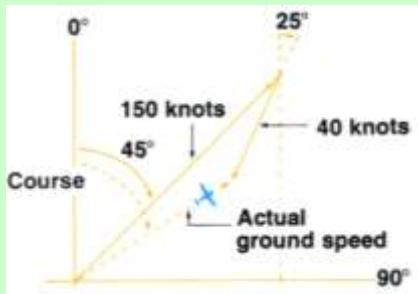
So, Mr. Sweeney is actually flying at 143.77 *knots* on a course of 217.94° over ground. Note we will demonstrate an alternative way for solving this kind of 2D vector problems on p. 152.

A similar example appeared first in the *HP-55 OH* of 1975 and was copied then for some years. We quote the respective text from the *HP-33 OH* of 1978:



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*?

Start of solution:

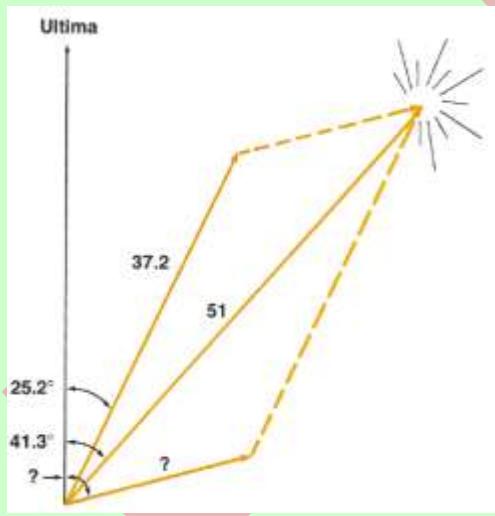
Method 1: The course and ground speed are equal to the difference of the two vectors.

Method 2: Taking into account that a bearing of 25° equals a heading of $25^\circ + 180^\circ = 205^\circ$, the corresponding

headwind vector may be added (cf. the *HP-97 OH and Programming Guide* of 1978).

We leave it to you to solve this problem using $\boxed{\Sigma+}$ and $\boxed{\Sigma-}$ (but give you the results for crosschecking: 51.94° and 113.24 knots).

Additionally, here is an advanced problem from a universe far, far away:



Example from the *HP-32 OH* of 1978:¹⁰⁵

Federation starship *Felicity* has emerged victorious from a furious battle with the starship Θανάτος¹⁰⁶ from the renegade planet *Maldek*. However, its automatic pilot is kaput,¹⁰⁷ and its main thrust engine is locked on at 37.2 meganewtons (MN) directed along an angle of 25.2° from the star *Ultima*.¹⁰⁸ Consulting the ship's star map, the navigator reports a hyperspace entrance vector of

¹⁰⁵ Note the first episode of *Star Wars* was launched in 1977.

¹⁰⁶ Translator's note: This is the ancient Greek word for 'death', pronounced like 'Tunnatoss' in English but like 'Thanatos' in Spanish, Italian, French, German, and Finnish, for example. Actually, they printed *Thanatos* in the English handbook.

¹⁰⁷ Yes, I know this is spelled 'kaputt', but they printed this German word this way in said manual. Oh, why can't the English learn to spell?

¹⁰⁸ Latin for 'the last' (female). – By the way, the plane of action in 3D space seems to be defined sufficiently by *Felicity*, *Ultima*, and said entrance (hopefully, its center) here – this problem was reprinted unmodified in the *HP-34C OH* one year after.

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?

Solution:

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 \leftrightarrow** , and their difference is calculated using **$\Sigma+$** and **$\Sigma-$** . This difference is recalled to the **X**- and **Y-registers** using **SUM**. Then, these rectangular coordinates of the auxiliary engine thrust vector are converted to polar coordinates using **$\rightarrow P$** .

CLR **CLΣ** clears the summation *registers*.

41.3 **ENTER↑** **51** hyperspace entrance vector

R \leftrightarrow returns $y =$ **33.66**
 $x =$ **38.31**

STAT **$\Sigma+$** adds x and y of the hyperspace entrance vector to the summation *registers*.

25.2 **ENTER↑** **37.2** main engine thrust vector

R \leftrightarrow returns $y =$ **15.84**
 $x =$ **33.66**

$\Sigma-$ subtracts x and y of the main engine thrust vector from the summation *registers*.

SUM recalls the summation *registers* **Σx** and **Σy** :

$\Sigma y =$ **17.82**
 $\Sigma x =$ **4.65**

$\rightarrow P$ returns $\theta =$ **75.36°**
 $r =$ **18.42**

meaning the auxiliary engine shall be set at 18.42 MN and an angle of 75.36° from *Ultima*.¹⁰⁹

¹⁰⁹ See previous footnote – a proper error calculation would be appreciated.

Real adepts of vector algebra may prefer subtracting the main engine thrust vector first and adding the hyperspace entrance vector second. This will work as well although the count of ‘data points’ will become negative once – simply don’t bother.

Those who are looking for an extra challenge can compute now how flat the crew of *Felicity* will become within *seconds* after the auxiliary engine is ignited.

Angles: Summary of Functions

The number of functions operating on and with *angles* are quite limited. See all functions listed below:

- General mathematics:
 - *Monadic* functions:
 - `sin`, `cos`, and `tan` operate on *angles* and return reals,
 - `arcsin`, `arccos`, and `arctan` operate on reals and return *angles*,
 - `±` returns $x \times (-1)$ for closed input (a.k.a. ‘unary minus’).
 - *Dyadic* functions:
 - `+`, `-`, `×`, and `÷` work as specified in the matrices on pp. 68f,
 - `max` (or `min`) return the maximum (or minimum) of x and y .
- Rounding:
 - `ROUND` rounds x using the current display format (cf. pp. 111f),
- Conversions:
 - `→P` converts rectangular *coordinates* to polar ones (cf. pp. 18f), while `R→` converts vice versa. See the examples on pp. 120ff.
Angular conversions are covered on pp. 117ff.

Integers: Input and Displaying

Any number (e.g. a counted value) you enter without using \square or E is regarded as an integer by your *WP 43S* (see pp. 65f). It allows for integer computing in fifteen bases from binary to hexadecimal.

Any single number displayed without an exponent or any punctuation on your *WP 43S* is an integer (see the examples following). And such numbers will stay integers as long as they are exclusively combined with other integers and only integer functions operate on them. Else they will be converted to other data (see the matrices on pp. 68f and Section 3 of the *ReM*).

Note that a closed integer in x will be converted automatically...

- to a *real number* by $.d$, DSP , or by $\text{DISP ALL, FIX, SCI, or ENG}$ (see pp. 76f),
- to a *real number* displayed as a fraction by $a b/c$ or d/c (see pp. 141ff), and
- to a sexagesimal *time* (see p. 141) by $h.ms$.

There are two kinds of integers provided by your *WP 43S*: integers of infinite precision and integers of finite precision. We will call them *infinite* and *finite integers* – this is not exact but saves print space.

Infinite integers are useful e.g. for numeric tasks. If you enter an arbitrary number just without using \square or E , it is taken as an *infinite integer* of base 10. For example,

11111 x^2 returns

123 454 321

Note the number is adjusted to the right again when closed, though no radix mark is displayed. Large infinite integers ($> 10^{21}$) will be displayed using the small font, very large ones ($> 10^{42}$) will be shown with an exponent instead of their least significant digits; nevertheless, all digits are kept internally.

This is all you need to know about entering and displaying *infinite integers* – see pp. 134ff for further information about calculating with them.

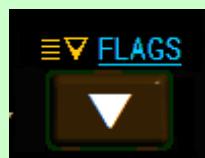
Finite integers feature a finite word size (up to 64 *bits*) and are especially useful for computer logic and system design. Your *WP 43S* contains the entire set of dedicated integer and bit manipulation operations of the *WP 34S*, exceeding the function set of the *Computer Scientist's HP-16C*.



Finite integers are entered with trailing **# base** (= 2...16). For decimal and hexadecimal integers, you may use **#D** instead of **#10** and **#H** instead of **#16**.

Open INTS (see its top view displayed here) for the digits **A** ... **F** required for numeric input in bases >10. If you do not specify a base (just do not press **#**, **D**, **E**, or **CC** in input) you will get an integer of the same base you entered before as long as you did not enter another *data type* in between.¹¹⁰

Word size and *integer sign mode (ISM)* settings are indicated in the *status bar* using a format *ww:x*. Therein, *ww* denotes the *word size* and *x* is 1 or 2 for 1's or 2's complement, respectively, *u* for *unsigned*, or *s* for *sign-and-mantissa mode* (cf. p. 71); these *ISM*'s control the handling of negative numbers. *Carry* and *overflow* – if set – will be shown as **c** or **o** or **g**, respectively, trailing *ISM* in the *status bar*. Generally, *carry* and *overflow* behave like they did on *HP-16C* or *WP 34S*. They correspond to *flags* (cf. p. 53) – if you want to set, clear, or check them individually, use the commands in FLAGS.



¹¹⁰ Illegal digits in input (e.g. **2** in base 2 or **B** in base 10) can be detected no earlier than said input is completed, so an error will be thrown then. You can key in more than current *word size* can take – also this will be checked when input is closed.

Example:

Enter INTS ▲ WSIZE 1 2

LZON

This setting allows seeing all bits at a glance easily.

147 ENTER↑

Enters 147 (base 10)

2

Converts decimal 147 to binary.

1COMPL

and you will see ¹¹¹

0000 1001 0011₂ and – after $\pm/-$ – 1111 0110 1100₂.

Obviously $\pm/-$ in 1COMPL inverts every bit, equivalent to NOT here.

Return to the original number via $\pm/-$, press 2COMPL, and you will get

0000 1001 0011₂ and – after $\pm/-$ – 1111 0110 1101₂.

Note the negative number equals the inverse + 1 in 2COMPL.

Return via $\pm/-$ again, press SIGNMT and you will see

0000 1001 0011₂ and – after $\pm/-$ – 1000 1001 0011₂.

Negating a number will just flip the top bit in SIGNMT (hence the name of this mode).

Return via $\pm/-$ once more, press UNSIGN and you will get

0000 1001 0011₂ and – after $\pm/-$ – 1111 0110 1101₂.

Note the 2nd number looks like in 2COMPL, but in addition an *overflow* is set here – see the ¹¹² in the *status bar* trailing the ISM.¹¹² Thus, pressing $\pm/-$ will not suffice anymore for returning to the original number here; you must also clear the *overflow flag* by CF B explicitly (see p. 53).

¹¹¹ Note the separator space automatically inserted every four bits for easy reading.

¹¹² This needs explanation, since changing signs has no meaning in unsigned mode per definition. Thus, $\pm/-$ should be illegal here or result in no operation at least. “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).

Unfortunately, however, $\pm/-$ in unsigned mode was allowed by the designers of the HP-16C and implemented as shown above, so we follow that implementation for sake of backward compatibility though frowning.

As you have seen, positive numbers stay unchanged in all those four modes. Negative *finite integers*, on the other hand, are represented in different ways. Therefore, taking a negative integer in one mode and switching to another one will lead to different interpretations.

Example:

The fixed bit pattern representing

-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.

Keeping the mode and changing bases will produce different views of the constant bit pattern as well.

Example:

Compare the outputs for different bases in $12:2$:

-147_{10} equals...

[2] $1111\ 0110\ 1101_2$,

[3] $-12\ 110_3$,

[4] $33\ 12\ 31_4$, or

[5] $-1\ 042_5$.

You may have noticed that the displays for bases 2 and 4 look similar, presenting all twelve bits to you, while in the other bases a signed mantissa is displayed instead. There are also different separator intervals; they are fixed for *finite integers* unless **GAP 0** is set by you. These different display formats (and more) take into account 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 probably gain most interest in dealing with different number representations and calculating therein, where base 10 is the common reference standard.

Let's look to bigger words now:

Example (continued):

Enter INTS ▲ WSIZE 6 4

UNSIGN

LZOFF

◀ 9 3 A 1 4 C 6 # H (see the menu shown on p. 128).

Then your *WP 43S* will display

9 3A 14 C6₁₅

In binary representation, this number will need 28 digits and would look like

$1001\ 0011\ 1010\ 0001\ 0100\ 1100\ 0110_2$.

Obviously, your WP 43S cannot display a binary number of this size in a single row (no pocket calculator can as far as we know). Look what it does instead – enter **#** **2** for converting x to binary and you will see:

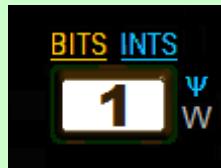
1001 0011 1010 0001 0100 1100 0110₂

using the small

font provided (see the *ReM, App. D*). If leading zeros were turned on via LZON, all 64 bits would be displayed in a special font:

Integers: Bitwise Operations on Finite Integers

Your *WP 43S* carries all the bitwise operations you may know from the vintage *HP-16C*, plus some more. You find them in [BITS](#). Generally, bits are counted from right to left, starting with number 1 for the least significant bit. This convention is important for specifying bit numbers in the operations BC?, BS?, CB, FB, and SB.



The following **examples** deal with 8-bit words showing leading zeros for easy reading. Set

INTS **WSIZE 8**
LZON
10110011 # 2 STO 0 0

So **1011 0011₂** is the common initial number for the operations presented in the table below. For seven shift and rotate functions, you find schematic pictures here like they were printed on the backplane of the HP-16C. The 'C' in the box stands for the *carry* bit indicated in the *status bar*, if applicable..

Operation	Schematic picture if applicable	E. g.	Output
Clear Bit		CB 5	1010 0011 ₂
Flip Bit		FB 6	1001 0011 ₂
Set Bit		SB 7	1111 0011 ₂
Negate		NOT (\neg)	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

Operation	Schematic picture if applicable	E. g.	Output
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 UNSIGNED: ¹¹³ 0001 0110 ₂ in SIGNMT: 1000 0110 ₂

Now let's also look at the bitwise *dyadic* functions. We will continue using 8-bit words displayed as above for the following **examples**:

Common input		Y	0110 1011 ₂
		X	1011 1001 ₂
Operation		Output	
BITS	AND	Λ	0010 1001 ₂
	NAND	¬Λ	1101 0110 ₂
	OR	∨	1111 1011 ₂
	NOR	¬∨	0000 0100 ₂
	XOR	∨	1101 0010 ₂
	XNOR		0010 1101 ₂

¹¹³ The picture for ASR correctly describes this operation for 1's and 2's complement modes only. In all modes of the HP-16C, however, ASR 3 equals a signed division by 2³, hence the different results for the latter two modes shown above. The other bitwise operations are insensitive to /SM setting. Turn to the /OI for further details.

See the *I/O* for these and further commands operating on bit level on integers (LJ and RJ, MASKL and MASKR, #B, and the tests BS? and BC?). Most of them are found in BITS.

Finally, note that no such operation will set an *Overflow*. *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 four basic arithmetic operations (+, -, ×, and /), the first three work with both kinds of integers as they do with *real numbers*; the only difference lies in precision: up to 64 digits precision for *finite integers* in binary representation on your WP 43S or even infinite precision for *data type 1*. Take INT as a multiplication times -1, and y^x as repeated multiplication. Depending on input parameters and mode settings, the *overflow* or *carry flags* may be set in such an operation (see pp. 137ff).

Divisions, however, must be handled differently in integer domain since the result cannot feature a fractional part here. Generally, the formula

$$\frac{a}{b} = (a \text{ div } b) + \frac{1}{b} \times \text{rmdr}(a; b)$$

applies; therein, the horizontal bar denotes real division, div represents integer division, and rmdr stands for the remainder of the latter. While remainders for positive parameters are simply found, remainders for negative dividends or divisors may lead to confusion sometimes. The formula above, however, is easily employed for calculating such remainders (also for *real numbers* – see the first row of the **examples** here):

$$\frac{25}{7} = 3 + \frac{1}{7} \times 4 \quad (\text{and for a real case: } \frac{25}{7.5} = 3 + \frac{1}{7.5} \times 2.5)$$

$$\frac{-25}{7} = -3 + \frac{1}{7} \times (-4) \quad \Rightarrow \quad \text{rmdr}(-25; 7) = -4$$

$$\frac{25}{-7} = -3 + \frac{1}{-7} \times 4 \quad \Rightarrow \quad \text{rmdr}(25; -7) = 4$$

$$\frac{-25}{-7} = 3 + \frac{1}{-7} \times (-4) \quad \Rightarrow \quad \text{rmdr}(-25; -7) = -4$$

In general, $\text{rmdr}(a; b) := a - b \times (a \text{ div } b)$ applies.

Unfortunately, there is a second function doing almost the same: it is called mod. With the same pairs of numbers as above, it returns:

$$\begin{aligned}\text{mod}(25; 7) &= 4, \\ \text{mod}(-25; 7) &= 3, \\ \text{mod}(25; -7) &= -3, \\ \text{mod}(-25; -7) &= -4.\end{aligned}$$

So mod returns the same as rmdr only if both parameters have equal signs. The general formula for mod is a bit more sophisticated than the one above:

$$\text{mod}(a; b) := a - b \times \text{floor}\left(\frac{a}{b}\right) \quad \text{with e.g.} \quad \text{floor}\left(\frac{25}{7}\right) = 3 \quad \text{and} \\ \text{floor}\left(-\frac{25}{7}\right) = -4.$$

By the way, this formula applies to *real numbers* as well. So it may be used straightforwardly for calculating e.g.

$$\text{mod}(25.3; -7.5) = 25.3 - (-7.5) \cdot (-4) = -4.7.$$



These four functions are called IDIV, RMDR, MOD, and FLOOR in your WP 43S for obvious reasons. Beyond them, there are more integer operations in INTS, such as ×MOD and ^MOD (see p. 139 for an example).

Furthermore, many exponential and logarithmic operations, x^2 and \sqrt{x} , x^3 and $\sqrt[3]{x}$, COMB and PERM as well as SIN, COS, and TAN operate on integers, too. Note some of them will stay in integer domain while others may or will return *real numbers*. See the IOI and Section 3 of the ReM for further information.

Integers: Overflow and Carry with Finite Integers

There are conditions where *overflow* or *carry* will be touched in arithmetic operations on *finite integers* on your WP 43S.

Note there is a maximum and a minimum integer displayable for each word size and /SM setting – let's call them I_{max} and I_{min} .

Example:

For four-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 various modes. And whenever *overflow* or *carry* will be lit in this course, we will clear them (using CF B or CF C) before continuing.

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
...
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				

For comparison, we start another turn from 1 but decrementing by 1:

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
$1111_2 \text{ } c$	15	$1111_2 \text{ } c$	-1	$1110_2 \text{ } c$	-1	$1001_2 \text{ } c$	-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	$0111_2 \text{ } o$	7	$1000_2 \text{ } o$	-0
0111_2	7	$0111_2 \text{ } o$	7	0110_2	6	1001_2	-1
...	...	0110_2	6
0010_2	2	0010_2	2	0001_2	1		
0001_2	1	0001_2	1				

The most significant bit is #3 in 4:s and #4 in all other modes here.

With these results, I_{max} , and I_{min} , the general rules for setting and clearing *carry* and *overflow* in ISMs are as shown here:

Operation	Effect on Carry	Effect on Overflow
Shift and rotate	As shown on pp. 132f.	None.
ABS	None.	Clears o (but sets it for $x = I_{min}$ in 2COMPL).
+, RCL+, STO+, INC, etc.	Sets c if there is a <u>carry out of</u> the most significant bit, else clears c.	Sets o if the result exceeded $[I_{min}; I_{max}]$, else clears o.

Operation	Effect on Carry	Effect on Overflow
-, RCL-, STO-, DEC, etc.	<p>Sets c in a subtraction $m - s$</p> <ul style="list-style-type: none"> in 1COMPL or 2COMPL if the binary subtraction causes a <i>borrow</i>¹¹⁴ <u>into</u> the most significant bit, in UNSIGN if $m < s$, in SIGNMT if $m < s \& m + s > 0$. <p>Else clears c.</p>	Sets o if the result exceeded $[I_{min}; I_{max}]$, else clears o .
\times , RCL \times , STO \times , $+/-$, $(-1)^x$, x^2 , x^3 , LCM, $x!$, etc.	None.	Thus in UNSIGN, $\boxed{+/-}$ always sets o and $(-1)^x$ does so for odd x .
2^x	<p>Clears c.</p> <p>Sets c only if $x = -1$, or in UNSIGN if $x = \text{wsize}$ or in the other modes if $x = \text{wsize} - 1$.</p>	
y^x , 10^x	Sets c for $x < 0$ (as well as for 0^0), else clears c .	Sets o if the result exceeded $[I_{min}; I_{max}]$, else clears o .
e^x	Sets c for $x \neq 0$, else clears c .	
DBL \times	None.	Clears o .
$/$, RCL $/$, STO $/$, DBL $/$, LB, LG, LN, LOG_{xy} , \sqrt{x} , $\sqrt[3]{x}$, $\sqrt[x]{y}$	Sets c if the remainder is $\neq 0$, else clears c .	Clears o (but sets it for the division $I_{min}/(-1)$ in 2COMPL)

¹¹⁴ See the examples above.

Translator's note: The so-called *borrow* in subtraction is a specialty of the USA. See the subtle methodic differences in manual subtracting explained in http://de.wikipedia.org/wiki/Subtraktion#Schriftliche_Subtraktion. The corresponding English article is less instructive. Both *carry* and *borrow* translate to *Übertrag* in German.

Integers: Summary of Functions

Many of the numeric functions operating on *real numbers* also work for integers. In addition, there are some specialties as shown in the preceding chapters, and beyond:

- General mathematics:

- *Monadic functions*:

x^2 , x^3 , 2^x , and 10^x return integers as you expect,
 \sqrt{x} , $\sqrt[3]{x}$, $\text{lb } x$, and lg return integers if possible,¹¹⁵ and
 +L works as demonstrated on p. 129.

- *Dyadic functions*:

$+$, $-$, \times , and y^x return integers as you expect,
 $\lceil \rceil$ returns the integer part of the solution,
 RMDR returns the remainder of y/x (s. pp. 134f for examples),
 MOD returns $y \bmod x$ (see p. 135 for examples),
 \sqrt{y} and $\log_{10}y$ return integers if possible¹¹⁵
(e.g. $625 \text{ [ENTER} \uparrow 5 \text{ [EXP] } \log_{10}y$ returns 4);
 max (or min) return the maximum (or minimum) of x and y ,
 GCD the Greatest Common Divisor of x and y and
 LCM the Least Common Multiple (remember school?).¹¹⁶

- *Triadic functions*:

$\times\text{MOD}$ returns $(z \cdot y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$, and
 $\wedge\text{MOD}$ returns $(z^y) \bmod x$ for $x > 1$, $y > 0$, $z > 0$
(e.g. $73 \text{ [ENTER} \uparrow 55 \text{ [ENTER} \uparrow 31 \text{ [INTS] } \wedge\text{MOD}$ returns 26).

¹¹⁵ ... for infinite integers. They return the integer part of the solution for finite integers.

¹¹⁶ Translator's note for German readers: GCD entspricht dem ggT und LCM dem kgV.

- Boole's algebra:

`AND`, `NAND`, `OR`, `NOR`, `XOR`, `XNOR`, and `NOT` operate bitwise on *finite integers* as shown on p. 133. They operate on *infinite integers* like in the HP-28S, i.e. x and y are interpreted before executing the operation; zero is 'false' (= 0); any other number is 'true' (= 1), cf. p. 113.
- Bitwise operations are exclusively for *finite integers*:

`CB`, `FB`, `SB`, `ASR`, `SL`, `SR`, `RL`, `RLC`, `RR`, `RRC`, and `MIRROR` work as demonstrated on pp. 131ff.

`LJ` left justifies the bit pattern within its word size,
`RJ` right justifies the bit pattern within its word size,
`MASKL` and `MASKR` create mask words,
`BC?` and `BS?` test if the specified bit is clear or set,
`#B` counts the number of bits set in x .

See the *IOI* for more information about these commands.
- Probability:

`(x!)` returns the factorial,
`Cyx` calculates the number of *combinations*, and
`Pyx` the number of *permutations* (cf. pp. 89f).
- Advanced mathematics (see the *ReM*, App. J for comprehensive information about the first three functions listed below):

`Bn` and `Bn*` return the *Bernoulli numbers*,
`FIB` the *Fibonacci number*, and
`NEXTP` the next *prime* number greater than x .

Many more functions accept integer input but return different, mostly real output. See the *IOI* and *Section 3* of the *ReM* for details.

Rational Numbers (Fractions)

Fractions are handled like in previous *RPN* calculators. In particular, DENMAX sets the maximum allowable denominator (see the *IOP*). On your *WP 43S*, you can work with fractions like on the *HP-32SII* and its successors but with higher precision.

A fraction is **entered** directly by keying in a second radix mark in numeric input (see the examples in the table below). In this case the first radix mark is interpreted as a space, the second as a fraction mark. This way of input is straightforward and logically coherent:

Examples:

Key in:

and get in *startup default* format

1 2 . 3 . 4 **ENTER↑**

$12 \frac{3}{4} =$

1 . 2 **ENTER↑**

1.2 (decimal input)

. 1 . 2 **ENTER↑**

$\frac{1}{2} =$

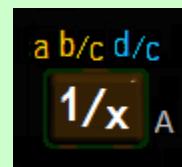
. 1 2 **ENTER↑**

0.12 (decimal input)

1 . . 2 **ENTER↑**

$1 \frac{0}{1} =$ (= $1 \frac{0}{2}$)¹¹⁷

Any closed integer or *real number* in X will be **displayed** as a fraction after **a b/c** or **d/c** is pressed, or after this number is combined with a fraction by an arithmetic operation. If the resulting fraction is exactly equal, slightly less, or slightly greater than the respective *real number*, =, <, or > will trail this fraction, respectively.



¹¹⁷ This display of a pure integer number tells you unambiguously your *WP 43S* is in *proper fraction display mode*. In *improper fraction display mode*, $\frac{1}{1} =$ will be displayed instead. For comparison, note the *HP-32SII* reads 1 . . 2 as $\frac{1}{2}$ – though this is not coherent with its other input interpretations (and does not even save keystrokes but adds confusion only).

Vice versa, any closed number x displayed as a fraction will be shown as a *real number* after **.d**, **DSP**, as well as after **DISP ALL**, **FIX**, **SCI**, or **ENG**. And a closed fraction x will be decomposed to its integer numerator in **Y** and its integer denominator in **X** by **PARTS DECOMP**.

There are two fraction display modes: *proper* and *improper fractions*.¹¹⁸ They are illustrated below. On your *WP 43S*, fraction display can handle numbers with absolute values greater than 10^{-4} ; maximum denominator is 9 999 (greater denominators may be entered but will be reduced as soon as input is closed).

The following example comprises most aspects of fraction display:

Example (with startup default settings):

Enter:

3 **[1/x]**

and you will see:

$3.333\ 333\ 333\ 333\ 333 \times 10^{-1}$

[d/c]

$1/3 >$

since $\frac{1}{3} > 0.333\ 333\ 333\ 333$.

[1/x]

$3/1 =$

since this is exact.

78.40625 **[d/c]** **[=]**

$-2\ 413/32 =$

[x²]

$5\ 822\ 569/1\ 024 =$

2 **[X]**

$5\ 822\ 569/512 =$

Now, press **[a b/c]** for converting this *improper fraction* to a *proper one*.¹¹⁹ You will get

$11\ 372\ 105/512 =$

11 **[/]**

$1\ 033\ 4\ 713/5\ 632 <$ ¹²⁰

¹¹⁸ Translator's note for German readers: *Proper fractions* decken sowohl *echte Brüche* (wie $\frac{3}{4}$) als auch *gemischte Brüche* (wie $2\frac{1}{2}$) ab. Bei *improper fractions* wird der ganzzahlige Anteil nicht herausgezogen, so dass der Zähler größer als der Nenner sein kann.

¹¹⁹ This conversion was newly introduced with the *WP 34S*.

¹²⁰ Display space suffices even for displaying very long fractions.

This fraction is less than the real value, deviating less than 0.5/5 632 from it.

Now, let's reduce the maximum denominator by

64 [MODE] DENMAX

[R↓]

DENFAC

64 0/1 =

1 033 41/49 <

1 033 27/32 > since

DENFAC allows for denominators being factors of DENMAX only (i.e. 2, 4, 8, 16, 32, and 64 here). This last fraction is greater than the real value; the fraction deviates from it by 0.5/32 maximum (and by 0.5/64 minimum – else the display would read 1033 53/64 instead).

Before closing this chapter about numbers displayed as fractions we will not forget mentioning those isolated irrational islands in the vast sea of SI where you may come across dimensions like the following for a real calculator stand, for **example**:

9" × 3 1/2" × 5/8". It goes without saying that your WP 43S will support you also in such harsh environments. Just remember only absolute greenhorns will expect that a tight thin-walled box around this stand will displace

9 [ENTER] 3 [.] 1 [.] 2 [x]
[.] 5 [.] 8 [x]

31 1/2 =

19 11/16 =

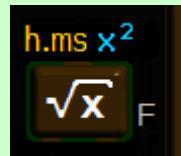
cubic inches of water.

Instead, a magic conversion factor from *cubic inches* to so called *fluid ounces* is required now.¹²¹ And this factor even depends on the country you are in! Though do not despair: in *Section 5* you will learn how to do this magic using your WP 43S – it takes just a little more time and effort than calculating with rational units.

¹²¹ Honestly, unless you grew up in such a place we bet you have assumed *fluid ounces* being a unit of mass, haven't you? Since you have heard of *ounces* once before and just thought ... terribly wrong! Do not think there – you may run into deep troubles easily (though thinking less you may achieve top positions in administration there).

Times

There also is a special *data type* for time calculations on your WP 43S. Sexagesimal *times* are **entered** most easily in the format hhhh.mmssffff terminated by **h.ms** – with hhhh standing for *hours*, mm for *minutes*, ss for *seconds*, and ffff for decimal fractions of *seconds* (these fractions may take more than three digits).



Example:

Enter 5 hours 39 minutes and 7.8642 seconds:

5 **.** 39078642 **h.ms**

This is displayed with *startup default* settings:

5:39:07.864 2

Choosing **CLK** **▲ TDISP** **2** will return

5:39:07

instead, while **CLK** **▲ TDISP** **0** returns

5:39

The latter two formats allow for compact time displays like in clocks.

The colon is the unambiguous indicator for a *time* on your WP 43S. In general, there may be leading zeroes in the *minutes* and *seconds* sections and a settable number of digits after the second colon. You can choose CLK24 or CLK12 display for *time of day*.

Example (continued):

Call TIME in the evening and you might get

21:47

CLK **▲ CLK12**

9:47 p.m.

- ☞ When *time of day* is returned by a function, it will be displayed according to your choice (CLK24 or CLK12) – internally, however, it is stored as standard 24h *time* for further calculations.

You may add and subtract sexagesimal *time intervals* simply using **+** and **-**; and *time intervals* may be multiplied or divided by any integer, rational, or *real number*. If you add a rational or *real number* to such a *time*, it will be automatically converted to a *time* before adding. This applies to subtractions in analogy.

Example (assuming startup default settings):

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 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?

Solution:

CLK ▲ TDISP 0

.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. So you have to leave at 3:13 p.m. the latest.

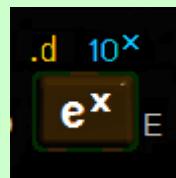
Note your WP 43S returns something looking like a 12h-time here even with *startup default* settings because it cannot know better based on the input given.

You can convert such (closed) *times* to decimal numbers using **.d**, and reconvert the decimal result to sexagesimal *times* by pressing **h.ms**.

There is only one more dedicated *time* command – SETTIM, serving obvious purposes. You may add and subtract *times*; and *times* may be multiplied or divided by any integer, rational, or *real number* – the result will stay a *time*. Compare the matrices on pp. 68f.

Dates

For date calculations, choose one out of three *date display modes* (*DDM*) on your *WP 43S*: Y.MD, D.MY, and M.DY (these mode-setting commands are contained in [CLK](#)). ISO Y.MD is *startup default*. *Date input* is decimal according to the *DDM* chosen and is terminated by [.d](#) (as shown on pp. 65f).



Example:

The 18th of December in 2017 is entered

2017.1218 [.d](#) in Y.MD,

18.122017 [.d](#) in D.MY, and

12.182017 [.d](#) in M.DY.

Alternatively, any *real number* may be converted into a *date* via [CLK](#) $\times \rightarrow$ **DATE**, and any triple of *real numbers* or integers via \rightarrow **DATE** (see p. 39). Input containing more than the necessary digits for a *date* in the *DDM* selected will be rounded.

Vice versa, **DATE** \rightarrow splits a *date* in three integers and pushes them on the stack as shown on p. 39. Note that both **DATE** \rightarrow and \rightarrow **DATE** observe the *DDM* chosen. If you want to extract particular information from a *date* independent of current *DDM*, we recommend using the operations **DAY**, **MONTH**, or **YEAR**.

Like in the *status bar*, closed *date* input or *dates* returned by a function are displayed as in the following **example**:

2017-12-18 in Y.MD,

18.12.2017 in D.MY,

12/18/2017 in M.DY.

So you immediately know the *DDM* chosen from looking at the date format. ALL, DSP, FIX, SCI, or ENG have no effect on *dates*.

[CLK](#) **WDAY** takes a *date* from the *stack* – or a decimal input of e.g. 2013.0504 in Y.MD mode (equivalent to inputs of 4.052013 in D.MY or

5.042013 in M.DY) – and returns an integer indicating the position of this day in the corresponding week, temporarily headed by the name of this weekday:

Saturday

6.¹²²

Expect similar displays after **CLK DATE**.¹²³

There is only one more dedicated *date* command – SETDAT, serving obvious purposes.

Note integers (or the integer parts of rational or *real numbers*) may be added to or subtracted from a *date*, always representing an integer number of days regardless of the *date* format set. And *dates* may be subtracted from *dates*, resulting in an integer number of days (intervals between various *dates* are calculated based on their respective Julian day numbers).

But that's it – these are all the legal operations. Compare the matrices on pp. 68f.

¹²² Translator's note: Numeric output corresponds to Chinese weekdays 1 to 6 directly. For Portuguese weekdays ('segunda-feira' etc.), add one to days 1 to 5.

¹²³ 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, that calendar became effective in October 1582 in the catholic world. Large parts of the world took their time and switched later (see the chapter about *Localizing Calculator Output* above). Note, however, there are still also other calendars in use on this planet, e.g. in the Muslim world.

Dates before the year 8 A.D. may be indicated differently than they were at the time due to the inconsistent application of the leap year rule before then. We count on your understanding and hope this shortcoming will not affect too many calculations.

Note that 8 A.D. should be better written A.D. 8 or even A.D. VIII correctly – quite some false Latin is found in the English language. Nobody, however, counted years this way at that time – around the Mediterranean Sea, it was the year DCCLXI A.V.C. in best case then. Also note the Julian calendar was introduced and became valid not earlier than DCCVIII A.V.C. – before, months were organized differently. Julius Caesar was murdered in DCCIX A.V.C.; calendars may be a sensitive topic.

Complex Numbers: Introduction

So far, we dealt with *real numbers* only (rational and integer numbers are mere subsets of reals). Your WP 43S can do more for you. Mathematicians know of more complex items than *real numbers*; these are called *complex numbers*. If you do not know of them, leave them aside; you can use your WP 43S perfectly without them.

If you know of *complex numbers*, however, note your WP 43S supports many operations in complex domain as well as it does in real domain.

Complex numbers may be **entered** using **CC** (see pp. 293 and 65f). In numeric input, **CC** separates real and imaginary part in *rectangular display mode*.



Examples (assuming startup default settings):

$3 + i \times 4$ is keyed in **3 f CC 4 ENTER↑** while the display (set e.g. to FIX 5) shows in lowest numeric row:

3
3 + i×
3 + i×4
3.000 00 + i×4.000 00

You enter the real part first – **CC** closes it – the imaginary part second.¹²⁴

Input of negative *complex numbers* works in full analogy to *real number* input (cf. p. 25). Following our example above,

$3 - i \times 4$ is keyed in **3 CC 4 +/- ENTER↑**,

$-3 + i \times 4$ is keyed in **3 +/- CC 4 ENTER↑**,

¹²⁴ Entering both parts vice versa would be more like *RPN*: first the imaginary part, then **CC** interpreted as $i \times$, finally the real part to be added. But it was decided differently for the HP-42S already. So we follow tradition here.

For those of you working on the field of electronic engineering, an alternate format is provided employing the letter **j** for the complex unit (the respective formatting commands are called CPXi and CPXj for obvious reasons).

$-31 - i \times 42$ is keyed in **3 1 +/- CC 4 2 +/- ENTER↑**.

Alternatively, use **3 1 CC 4 2 ENTER↑ +/-** here.

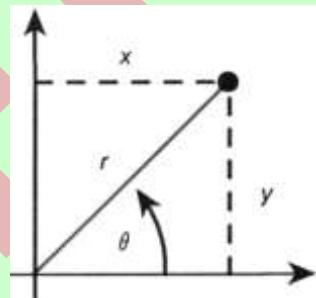
Choosing 'scientific notation', e.g. SCI 5, this last number will be displayed like

-3.100 00 $\times 10^1$ -i \times 4.200 00 $\times 10^1$

Depending on display format set, this may be shown more compact (allowing for one decimal more):

-3,100 000 $\cdot 10^1$ -i \cdot 4,200 000 $\cdot 10^1$

Alternatively to rectangular form, *complex numbers* may be written in polar form as well. With *polar display mode* set (by **MODE POLAR** causing \odot lit in the status bar), the magnitude (or radius) r shall be entered first for a new *complex number* and the angle θ second. Though θ may be entered in any angular notation, often *radians* or *multiples of π* make most sense here; e.g. set **MODE RAD** causing 4^r lit in the status bar.



Example:

With *polar display mode*, *radians*, and FIX 2 set, the *complex number* $(5 ; 1.2^r)$ is keyed in **5 CC 1 . 2 ENTER↑** with the display showing in lowest numeric row:

5
5 $\frac{1}{4}$
5 $\frac{1}{4}$ 1.2
5.00 $\frac{1}{4}$ 1.20^r

Special cases: If a negative magnitude is entered, it is made positive and θ is increased by π and then normalized (i.e. θ will never exceed the interval $(-\pi, \pi]$ in *radians* or its equivalents – cf. p. 121). Large angular input is legal, but the output will be normalized always. If zero is entered for the magnitude, θ will be set to zero as well.

Composing and decomposing a *complex number*: Alternatively to entering a *complex number* directly, it may be generated from two closed *real numbers* provided in X and Y. If L is lit, this process will take x as real part and y as imaginary part composing the *complex number* (like $x + iy$ in textbooks). If \odot is lit on the other hand, it will take x as magnitude and y as angle of the new *complex number* (compare numeric input above). Whenever a *complex number* is returned, your WP 43S will set CPXRES and C will appear in the *status bar* unless set before.

Example:

MODE RECT

REALRE these two entries return to *startup default* settings L and R .

4 **ENTER** ↑ -3 **EXIT**

4.
-3.

EXIT closed input without disturbing the stack.

CC composes a *complex number* out of x and y now, lighting C and returning

4.-i×3.

POLAR turns L to \odot and displays

5.000 000 ↗ 126.869 898°

Vice versa, **CC** may also *cut* a *complex number* x into two *real numbers* in X and Y following the same rules.

Example (continued):

CC returns

$\text{r} = \quad 126.869\ 897\ 645\ 844\ 0^\circ$
 $\text{s} = \quad 5.$

C and \odot remain lit in the *status bar*.

RAD **CC** returns

5.000 000 ↗ 2.214 297^r

RECT	turns to				4. $-i \times 3.$
	and displays		Im =		-3.
	returns		Re =		4.

, , and remain lit in the *status bar*.

Generally, *complex number* outputs follow *real number* formats (see pp. 76ff). The number of displayable decimals, however, may be limited by screen space. If you want to view both parts of a *complex number* in full precision, press , watch, and press again.¹²⁵

Complex results in calculations: As long as you work exclusively with real input, you will get only real results with REALRES set; you can, however, also set CPXRES to allow for complex results. Try **1** and see the different results.

With at least one complex input parameter in arithmetic operations or function calls, your *WP 43S* will set CPXRES automatically (indicated by in the *status bar*).

With input closed for a complex x and RECT chosen, for example,

- will change the signs of both the real and the imaginary part (as shown above) or
- will swap real and imaginary parts.

Press for separating complex input as you do in real domain.

Example:

$(1 + 2i) \times (3 + 4i)$ is entered and solved like this:

¹²⁵ Choosing RECT and MULT· allows for displaying real and imaginary components within $(10^{-999}, 10^{999})$ in SCI 4. It will work in SCI 5 for both components within $(10^{-99}, 10^{99})$. Staying with the *startup default* (i.e. MULTx) instead will cost you one displayed decimal in complex domain.

In *polar display mode*, angles will be normalized to $(-\pi, \pi]$ always, so there will be no space for a power of ten needed for an angle; hence POLAR will allow for SCI 6 within $(10^{-999}, 10^{999})$ regardless of the multiplication symbol chosen, and for SCI 7 within $(10^{-99}, 10^{99})$. See App. D in the ReM.

1 [CC] 2 [ENTER↑]

3 [CC] 4

[X]

returning

1. +i×2.

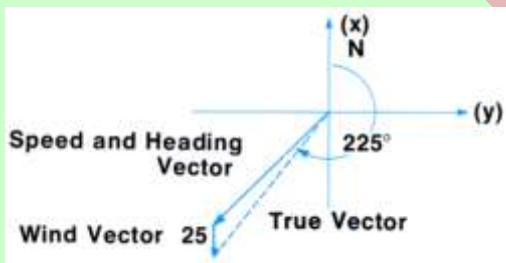
3. +i×4.

-5. +i×10.

Many transcendental functions will operate on *complex numbers* as well (e.g. sin, cos, tan, LN, e^x , y^x , \sqrt{x} , etc.). Please check pp. 155f.

Complex Numbers Used for 2D Vector Algebra

You can use complex domain for 2D vector algebra (see the following examples). The functions $|x|$, +, -, CROSS, DOT, and UNITV wait for you – see CPX and the IOI.



We can, for **example**, compute Mr. Sweeney's ground course (as explained on p. 122) according to the following alternative way:

[DISP] FIX [2]
[MODE] DEG POLAR

125 [CC] 225

125 ↴ 225.

indicated air speed and heading.

[ENTER↑]

125.00 ↴ 225.00°

25 [CC] 180

25 ↴ 180.

for the north wind.

[+]

returns

143.77 ↴ -142.06°

for the resulting vector.

[CC]

returns $r =$

-142.06°

[CC]

$r =$

143.77

[x²y] 360 [+]

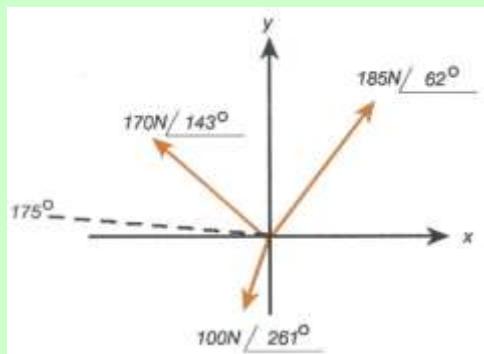
displays

143.77

217.94

(according to convention, the angle must be positive in navigation. – Compare with pp. 122f.)

Some examples more (taken from the *HP-42S Owner's Manual*):



Dot Product of Complex Numbers.

The figure ... represents three two-dimensional force vectors. Use *complex numbers* and add the three vectors. Then use the DOT (dot product) function to find the component of the resulting vector along the 175° line.

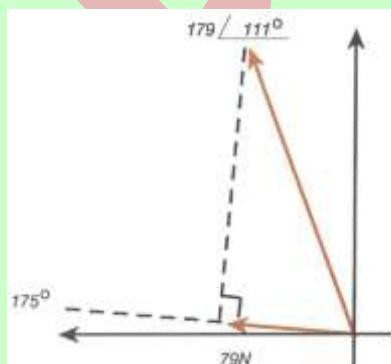
Solution:

MODE DEG POLAR ensure proper ADM and coordinates

170	CC	143	ENTER↑	125.00 ↘ 225.00°
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°
CPX	dot			78.86



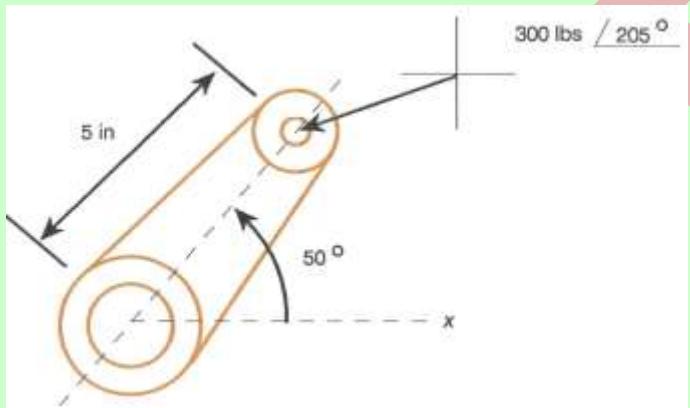
Thus, the resulting vector sum has a component of approximately 79 Newtons in the direction of 175° .

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 WP 43S 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 below, where

$$\vec{M} = \vec{r} \times \vec{F}$$



Note this illustration shows a two-dimensional situation still. Lever and force are both acting in the drawing plane.

Key in the radius vector and the force vector:

5 [CC] 50 [ENTER]

5.00 ↘ 50.00°

300 [CC] 205

300.00 ↘ 205

[CPX] cross

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.¹²⁶

¹²⁶ If the problem you're working requires a true (three-dimensional) vector as a result, use a 1×3 or 3×1 matrix to represent each vector in three dimensions.

Complex Numbers: Summary of Functions

Many of the numeric functions operating on *real numbers* also work for *complex numbers*:

- General mathematics:

- *Monadic* functions:

\sqrt{x} and x^2 , $\sqrt[3]{x}$ and x^3 , 2^x and $\text{lb } x$, 10^x and lg , $\frac{1}{x}$, e^x and \ln , \sin , \cos , and \tan as well as their inverses, \sinh , \cosh , and \tanh as well as their inverses work as usual (see also pp. 117ff for more information about angular I/O),

e^{x-1} and $\ln(1+x)$ return more accurate results with $x \approx 0$, +/- returns $x \times (-1)$ for closed input (a.k.a. ‘unary minus’), and $(-1)^x$ returns $\cos(\pi x)$ for non-integer x .

- *Dyadic* functions:

$+$, $-$, \times , / , y^x and $\sqrt[x]{y}$ work as usual,

$\log_x y$ calculates the logarithm of y for base x ,

dot and cross allow using *complex numbers* for 2D vector computations, and

\parallel returns $\left(\frac{1}{x} + \frac{1}{y}\right)^{-1}$ for $x \times y \neq 0$ and 0 else.

- Isolating and manipulating parts of numbers:

Use RE for the real part of x and IM for its imaginary part,

$\text{RE} \gtrless \text{IM}$ for swapping its real and imaginary part,

$|x|$ for its magnitude and \angle for its angle,

FP for the fractional part of x and IP for its integer part;

SIGN and UNITV return the unit vector of x , and

conj returns its complex conjugate.

- Rounding:

RDP n rounds x to n decimal places in FIX format
(e.g. 1.23456789E-95 RDP 99 will return 1.2346 $\times 10^{-95}$),

ROUND rounds x using the current display format like RND on HP-42S,

RSD n rounds x to n significant digits.

- Advanced mathematics (see the *ReM, App. J* for comprehensive information about the functions listed below):

- *Monadic* functions:

FIB returns the extended *Fibonacci number*,

gd and **gd⁻¹** the *Gudermann function* and its inverse,

sinc returns $\frac{\sin(x)}{x}$ for $x \neq 0$ and 1 for $x = 0$ (*input* is converted to *radians* before calculating – see pp. 117ff),

W_p returns the principal branch of *Lambert's W* for $x \geq -1/e$,

W⁻¹ returns x for given W_p (≥ -1),

Γ(x) calculates the *Gamma function*, and

InΓ returns the natural logarithm of the *Gamma function*.

- *Dyadic* functions:

AGM returns the *arithmetic-geometric mean*,

COMB and **PERM** calculate with complex *Gamma*,

β(x,y) returns *Euler's Beta function*, and

Inβ the natural logarithm of *Euler's Beta function*.

Vectors and Matrices: Introduction and Input

So far, we dealt with just one or two or (seldom) three numbers at once. Your *WP 43S* can do more for you – e.g. manipulate a set of numbers in a column or a row or even an array of four, six, eight, nine, or more numbers simultaneously. Such number columns or rows are called *vectors* and the arrays are called *matrices* by mathematicians. If you do not know of vectors and matrices yet, feel free to set them aside; your *WP 43S* will serve you perfectly without them.

If you know of them, however, note the function set of your *WP 43S* covers vector operations and also allows for adding, multiplying, inverting, and transposing matrices, as well as for editing and manipulating parts of such matrices. It also provides functions for computing *determinants*, *eigenvalues* and *eigenvectors*, and for solving *systems of linear equations*.¹²⁷



Generally, matrices are characterized by their number of rows and columns. We talk of an $n \times m$ matrix if it features n rows and m columns. A *vector* may be regarded as a special matrix featuring one column or one row only.

Example:

A vector $\begin{bmatrix} 4 \\ -5 \\ 6.7 \end{bmatrix}$ and a matrix $\begin{bmatrix} -1 & 12 & 7 \\ 25 & 0 & 3 \end{bmatrix}$ shall be entered subsequently. The *stack* shall be clear at beginning.

Enter **DISP** **FIX** **0** **1**

3 **ENTER↑** **1** **MATX** **NEW** (the leftmost unshifted softkey)

to initialize the 3D column vector (i.e. a 3×1 matrix). See the new matrix in **X** and the top view of **MATX** displayed in the *menu section*:

							0.0
$[0.0 \quad 0.0 \quad 0.0]^T$							
	RNORM	ENORM	STOEL	RCLEL	PUTM	GETM	
	dot	cross	UNITYV	DIM	INDEX	EDITN	
	NEW	$[M]^{-1}$	$ M $	$[M]^T$	SIM EQ	EDIT	

¹²⁷ Its function set is based on the one of *HP-42S*.

For saving screen space, your WP 43S displays each column vector transposed (thus the superscript T trailing it), i.e. in one row instead of one column on the screen. The vector is initialized with all its components being zero. To enter the vector components, press **EDIT** (the rightmost unshifted softkey) and the *Matrix Editor* will appear in the *menu section*:

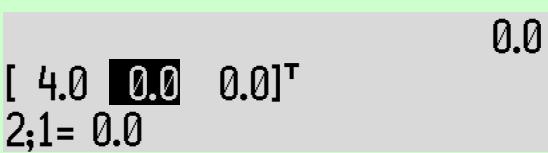


Note the first element of the vector is displayed inverted now indicating the position of the edit cursor. This particular element is shown below in the format set (i.e. FIX 1 here), so we need two rows for X.

Now press **4**



Move the cursor to the next element: →



Continue editing: 5 ↵ → 6.7



EXIT

						0.0
			[4.0 -5.0 6.7] ^T			
RNORM	ENORM	STOEL	RCLEL	PUTM	GETM	
dot	cross	UNITV	DIM	INDEX	EDITN	
NEW	$[M]^{-1}$	$ M $	$[M]^T$	SIM EQ	EDIT	

Note **EXIT** left the *Matrix Editor*, returning to the top view of **MATX**, closes input for the object in **X**, and shifts **x** to the right.

Now, let's initialize the 2×3 matrix via

2 **ENTER↑** **3** **NEW** and begin editing once again by
EDIT

[3x1 Matrix]

$$\begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$1;1 = 0.0$$

Three numeric rows are required for editing x now. The 3×1 matrix in \mathbf{Y} is the 3D vector we just entered before; note any matrix is displayed in this short form (with a \times even for MULT· chosen) in any *stack register* but \mathbf{X} .

Again, all elements of the new matrix start containing zero. Its first element is displayed inverted as the first element of the vector was above. Matrix editing will continue in analogy:

1  →

[3x1 Matrix]
[-1.0 0.0 0.0]
[0.0 0.0 0.0]
1;2= 0.0

$$12 \rightarrow 7 \rightarrow$$

[3x1 Matrix]

$$\begin{bmatrix} -1.0 & 12.0 & 7.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

2;1= 0.0

Entering the last → moved the cursor from the last element of row 1 to the first element of row 2. So you can simply continue:

25 → → 3

[3x1 Matrix]

$$\begin{bmatrix} -1.0 & 12.0 & 7.0 \\ 25.0 & 0.0 & 3.0 \end{bmatrix}$$

2;3= 3

EXIT

0.0

[3x1 Matrix]

$$\begin{bmatrix} -1.0 & 12.0 & 7.0 \\ 25.0 & 0.0 & 3.0 \end{bmatrix}$$

RNORM	ENORM	STOEL	RCLEL	PUTM	GETM	
dot	cross	UNITY	DIM	INDEX	EDITN	
NEW	$[M]^{-1}$	$ M $	$[M]^T$	SIM EQ	EDIT	

Now also this matrix is closed and ready for calculating. Assume you want to multiply it by $\frac{2}{3}$. And we want more than just a single decimal displayed in the result:

DSP 0 3

2 3

0.000

[3x1 Matrix]

[2x3 Matrix]

$$0 \frac{2}{3}$$

Remember input is evaluated not earlier than it is closed.

Press **X** and you will get immediately

0.000
[3×1 Matrix]
[-0.667 8.000 2.667]
[16.333 0.000 1.000]

which are all matrix elements multiplied by $\frac{2}{3}$ at once.

You may store such matrices in any *register* or variable. So let's store our resulting matrix in **R00** – just press **STO 0 0** for this.

You can also create and fill a matrix directly in a variable (i.e. you do not have to create the matrix on the *stack* and store it afterwards).

Example:

Create a matrix $[MA] = \begin{bmatrix} 4 & -3 \\ -2 & 1 \end{bmatrix}$ and fill it without calling it to the *stack*.

2 **ENTER↑** **DIM** **α M A** **ENTER↑** creates **MA** as a 2×2 matrix.

EDITN **VARS** **MA**

[2×3 Matrix]
[0.000 0.000]
[0.000 0.000]
1;1= 0.000
INSR DELR WRAP GROW
← ↑ OLD GOTO ↓ →

4 → 3 [+/-] → 2 [+/-] → 1

[2×3 Matrix]

$$\begin{bmatrix} 4.000 & -3.000 \\ -2.000 & 1.000 \\ 2;2= 1 \end{bmatrix}$$

Now, press **EXIT** and you are done with **MA** – while the screen looks just as before again:

[3×1 Matrix]

$$\begin{bmatrix} 0.000 \\ -0.667 & 8.000 & 2.667 \\ 16.333 & 0.000 & 1.000 \end{bmatrix}$$

RNORM	ENORM	STOEL	RCLEL	PUTM	GETM
dot	cross	UNITYV	DIM	INDEX	EDITN
NEW	[M] ⁻¹	M	[M] ^T	SIM EQ	EDIT

Vectors and Matrices: Displaying and Editing Larger Objects

Whenever **X** contains a matrix, your *WP 43S* will try to show it completely (i.e. display all its elements in the format you chose for *real numbers*). Objects in higher *stack registers* will be indicated in a single row (abbreviated if necessary) or will be shifted out of the display window – but *x* and *y* will stay on the screen at least.

If space turns out being insufficient for showing the complete matrix in the format chosen, your *WP 43S* will switch to the small font.

Example (continued):

DSP 0 5

[3×1 Matrix]

$$\begin{bmatrix} -0.666\ 67 & 8.000\ 00 & 2.666\ 67 \\ 16.333\ 33 & 0.000\ 00 & 1.000\ 00 \end{bmatrix}$$

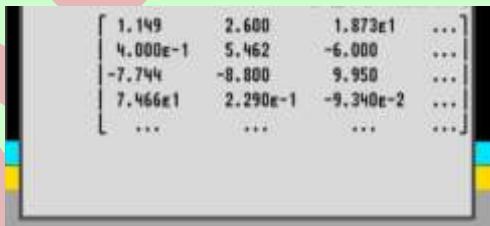
If font switching should not suffice, your *WP 43S* will turn to SCI 3 for the elements of the respective matrix. This allows for showing arbitrary 5×4 real matrices entirely (see the *ReM, App. D*). If a real matrix exceeds 5 rows, its fifth row is displayed filled with ellipses (...); if it exceeds 4 columns, its fourth 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 present stack and is in **X** now. Then the screen will look like this to scale:



Editing such a large matrix will push also **y** from the screen until input is closed again. You can browse the entire matrix regardless of its size.

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 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 *real numbers*. 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:

1.149	2.600	1.873E1	...
4.000E-1	5.462	-6.000	...
-7.744	-8.800	9.950	...
7.466E1	2.290E-1	-9.340E-2	...
...

1;1= 1.149 30

The first matrix element is selected. And the lowest numeric row displays this element in FIX 5 as we had chosen.

Go to the bottom row of this matrix by pressing (or) five times and you will get:

...
-7.744	-8.800	9.950	...
7.466E1	2.290E-1	-9.340E-2	...
3.390E1	-7.940E1	3.436	...
4.880E-2	7.000	5.980	...

6;1= 0.048 80

Now go to the very last element of this matrix by pressing four times:

...
...	9.950	5.450E1	1.700E-1
...	-9.340E-2	2.000	-3.829
...	3.436	9.080	4.256
...	5.980	-6.800E-1	-2.249E1

6;5= -22.492 00

Wherever you are within a matrix, you can replace or modify the currently selected element in two ways:

1. Let an arbitrary *monadic* function operate on the selected element. If you need any *menus* to reach a function, they will temporarily replace the *Matrix Editor menu*; exiting those *menus* will bring you back to the *Matrix Editor menu*.
2. Simply key in a new number replacing the old one.

Example (continued):

Replace the last matrix element by 17.435.

17.435

...
...	9.950	5.450E1	1.700E-1
...	-9.340E-2	2.000	-3.829
...	3.436	9.080	4.256
...	5.980	-6.800E-1	1.744E1

6;5= 17.435

INSR	DELR	WRAP	GROW
←	↑	OLD	GOTO
↓	→		

If you now decide to recover the old element again, however, call **OLD**. This old content is actually not overwritten until you press one of **▲**, **↑**, **▼**, **↓**, **←**, or **→** after entering a new number, or you leave the *Matrix Editor* via **EXIT**.

- ☞ Repeatedly pushing the cursor in one direction (e.g. by **→**) will jump from the first, second, etc. to the last row and then return to the first row in default WRAP mode. If GROW is set instead, another **→** from the very last element of the matrix will add a new row to the matrix.

Example (continued):



...
7.466E1	2.290E-1	-9.340E-2
3.390E1	-7.940E1	3.436
4.880E-2	7.000	5.980
0.000	0.000	0.000

7;1= 0.000

Here, we are done with that matrix for now. So press **EXIT** and you will see again:

[2x3 Matrix]			
1.149	2.600	1.873E1	...
4.000E-1	5.462	-6.000	...
-7.744	-8.800	9.950	...
7.466E1	2.290E-1	-9.340E-2	...
...

Note the first matrix element is not highlighted anymore since you left the *Matrix Editor*.

So matrix editing is easy and straightforward. The *I/O* contains additional information, also about the further commands DELR, INSR, and R \leftrightarrow R showing up in the *Matrix Editor menu*.

Vectors and Matrices: Complex Stuff

Your *WP 43S* supports also complex vectors and matrices, i.e. matrices containing complex elements. They are created and initialized like real objects via **NEW** or **DIM** as explained above. Or you can recall a real matrix and edit it; if you enter one 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. 162):

Create and store a complex matrix $\begin{bmatrix} 5 + 8i & \pi i \\ -2 & 4 - 3i \end{bmatrix}$.

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 **VARS** **MA**
DSP **0** **2**
MATX **EDIT**

since we will not need five decimals for that process.

[6x5 Matrix]
[4.00 -3.00]
[-2.00 1.00]
1,1= 4.00

We can now just enter the new elements there as we have done before:

5 **CC** 8 **→** 0 **CC** **π** **→** **→** 4 **CC** 3 **+/−**
EXIT
STO **0** **1**

$$\begin{bmatrix} 5.00+i \times 8.00 & i \times 3.14 \\ -2.00 & 4.00-i \times 3.00 \end{bmatrix} \quad [6 \times 5 \text{ Matrix}]$$

Since we edited on the stack and stored the resulting new complex matrix at a new location, the old real matrix **MA** is not affected at all.

Compare pp. 138f for the input and formatting of *complex numbers*. Everything else works as it does for real matrices. You see complex vectors and matrices are no complex topic at all for you with your *WP 43S*.

Vectors and Matrices: Calculating

As we have seen on p. 161, your *WP 43S* can multiply a matrix by 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 when a matrix *y* is combined with a scalar *x*. Vice versa, with a scalar *y* and a matrix *x*, additions, subtractions and multiplications will work the same way (remember you cannot divide a number by a matrix). Also *monadic* functions operate on each matrix element in your *WP 43S*, if applicable.

Examples:

With an arbitrary matrix in **X**, pressing...

- **\sqrt{x}** will take the square root of each matrix element individually (if **CPXRES** is set, a real matrix *x* containing at least one negative element will become complex this way).
- **x^2** will square each matrix element individually (use **ENTER↑ x** for squaring the matrix instead);
- **$|x|$** will calculate the absolute value of each matrix element (see **CPX** or **PARTS**; use **MATX ENORM** for calculating the Euclidean norm of the matrix or take **|M|** for getting its determinant instead);
- **$\pm x$** will change the sign of each matrix element.

You can also let the *dyadic* functions **[+]**, **[-]**, **[x]**, or **[/]** operate on two matrices or vectors alone (i.e. *data types* 8 and 9), provided the rules of *linear algebra* are obeyed:

	<i>y</i>	<i>x</i>	Op.	Resulting <i>x</i>
[+]	[<i>m</i> × <i>n</i> Matrix]	[<i>m</i> × <i>n</i> Matrix]	[y] + [x]	[<i>m</i> × <i>n</i> Matrix]
[-]			[y] - [x]	
[x]	[<i>m</i> × <i>n</i> Matrix]	[<i>n</i> × <i>p</i> Matrix]	[y] · [x]	[<i>m</i> × <i>p</i> Matrix]
[/]	[<i>m</i> × <i>n</i> Matrix]	[<i>n</i> × <i>n</i> Matrix]	[y] · [x]⁻¹	[<i>m</i> × <i>n</i> Matrix]

The first row of this table reads as follows: For adding or subtracting two arbitrary matrices, both must be of identical size, and the result will be of the same size as well. The subsequent rows read in analogy.¹²⁸ If either matrix is complex, the result will be complex in most cases as well.

Example (continuation of p. 162):

Multiply the matrices in **R00** and **MA**. Output format shall be FIX 3.

Solution (we omit the *menu section* in the following pictures):

DSP **0** **3**
RCL **0** **0**

[2×2 C matrix]

$$\begin{bmatrix} -0.667 & 8.000 & 2.667 \\ 16.333 & 0.000 & 1.000 \end{bmatrix}$$

¹²⁸ Remember matrix multiplication behaves different than multiplication of scalars. Generally, for two arbitrary matrices A and B of matching sizes, $A \cdot B \neq B \cdot A$. Also note that only square matrices may be squared.

And matrix division is special: it is defined as multiplication of the numerator times the inverse of the denominator. Therefore, X must contain a nonsingular (invertible) matrix here – else you cannot divide by that matrix. Only square matrices can be inverted.

Note the ‘ $2 \times 2 \mathbb{C}$ 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 **MA**:

RCL **VARS** **MA** (or **RCL** **α** **M** **A** **ENTER↑**, if you have defined many variables already – cf. p. 55)

$$\begin{bmatrix} 2 \times 3 \text{ Matrix} \\ [4.000 \ -3.000] \\ [-2.000 \ 1.000] \end{bmatrix}$$

The 2×3 matrix in **Y** now is the one we have recalled from **R00** into **X** before recalling **MA**. We multiply **y** times **x** as usual by

X resulting in

$$\begin{bmatrix} 2 \times 2 \mathbb{C} \text{ matrix} \\ [-79.000 \ 48.000 \ 19.000] \\ [27.000 \ -24.000 \ -11.000] \end{bmatrix}$$

You see that arithmetic operations on matrices are almost as easy as on scalars using your *WP 43S*.

And your *WP 43S* features further matrix operations: **|M|** for computing determinants, **[M]⁻¹** for inverting, **[M]^T** for transposing, **M.LU** for computing the LU decomposition, and two 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 **VARS** **M**

EDIT etc.

[3×2 Matrix]

$$\begin{bmatrix} 1.000 & 2.000 \\ 3.000 & 4.000 \end{bmatrix}$$

$[M]^{-1}$

[3×2 Matrix]

$$\begin{bmatrix} -2.000 & 1.000 \\ 1.500 & -0.500 \end{bmatrix}$$

Thus, the inverted matrix reads $[M]^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}$.

For two **vectors** of identical size, there are two special multiplications provided: DOT and CROSS. DOT will return the dot product, a scalar – exactly what the table above says for $m = p = 1$. CROSS works 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 velocity $\vec{v} = (0.4 \quad 2.8 \quad -1.2) \cdot 10^7$ 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:

Just remember that in cross products, vectors must be entered in proper sequence as written from left to right:

DSP 0 2 since we will not need three decimals for that process.

3 ENTER↑ 1 MATX DIM α V ENTER↑ creates **v** as a 3×1 matrix.

RCL VARS v

EDIT etc.

[2×2 Matrix]

[0.40 2.80 -1.20]

STO **α B** **ENTER↑**

creates **B** as a 3×1 matrix, too.

RCL **VARS** **B**

EDIT etc.

[3×1 Matrix]

[1.30 -0.30 0.70]

x

[2×2 Matrix]

[1.60 -1.84 -3.76]

E 7 **x**

1.6 **E** +/- 19 **x** resulting in

[2×2 Matrix]

[2.56×10^{-12} -2.94×10^{-12} -6.02×10^{-12}]

... newtons, of course.

The total 'length' or absolute value of this force is

ENORM

[2×2 Matrix]

5.14×10^{-11}

Compare with the weight of a proton:

CONST **α M P** **^**

recall the proton mass.

G **^**

recall earth acceleration.

x

5.14×10^{-11}

1.64×10^{-26}

So this is a force ratio of

[*f*]

3.14×10¹⁵

Thus, physicists deliberately neglect gravitational effects in such microscopic calculations.

If you just want to perform elementary vector operations in 2D, however, there are two simple alternatives (known for long from earlier calculators):

1. Enter the Cartesian components of each vector in **X** and **Y** (if necessary, converting its polar components into Cartesian ones by **R↔** before) and use **[Σ+]** for additions or **[Σ-]** for subtractions. Recall the result via **[SUM]**; it may look like this, for example:

**Σy = 1 464.21
Σx = 123.58**

2. Calculate with *complex numbers* (as shown on pp. 148ff). In complex domain, 2D vector multiplication is possible since the commands DOT and CROSS are contained in **CPX** as well. See pp. 152ff for two examples.

Vectors and Matrices: Solving Systems of Linear Equations

Your *WP 43S* can also solve simultaneous linear equations (of the kind $[A] \cdot \vec{X} = \vec{B}$) for you.¹²⁹ To deal with such a system of linear equations, proceed as follows:

¹²⁹ This works the same way as it did on the *HP-42S*. The number of unknowns is only limited by the free memory available in your *WP 43S*.

1. Specify the number of unknowns (e.g. 4) by entering

MATX SIM EQ [4]

Your *WP 43S* automatically creates (if necessary) and dimensions three matrices: MatA, MatB, and MatX. You will see a new *menu* showing up:



2. Press **Mat A**. The *Matrix Editor* will open and you can enter the elements of the 4×4 *coefficient matrix* (see on pp. 157ff how to do this). Close the *Matrix Editor* by **EXIT** to return to the *menu* shown above.
3. Press **Mat B** and enter the elements of the 4×1 *constant matrix* the same way (this is a vector actually).
4. Press **Mat X** to let your *WP 43S* compute the 4×1 *solution matrix* (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** and start over with step 1.

Vectors and Matrices: Eigenvalues and Eigenvectors

An *eigenvalue* 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 having the same dimension as \vec{X} are called the *eigenvectors* of $[A]$.

Your WP 43S can solve such problems for you as well:

Example 1:

We need the eigenvalues of a 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 VARS M
DISP FIX 0 1
MATX EDIT etc.

$$\begin{bmatrix} 2.0 & 1.0 \\ 6.0 & 1.0 \end{bmatrix}$$

STO VARS M

The eigenvalues are the solutions of the *characteristic polynomial* of this problem:

$$(2 - \lambda)(1 - \lambda) - 6 = 0$$

▲ EIGVAL returns

$$M = [2 \times 2 \text{ Matrix}]$$
$$\begin{bmatrix} 4.0 & 0.0 \\ 0.0 & -1.0 \end{bmatrix}$$

being the matrix with the eigenvalues as its diagonal elements. Note this resulting diagonal matrix is pushed on the stack.

Example (continued):

Now, what are the eigenvalues of $[N] = \begin{bmatrix} 3 & 4 \\ -4 & 3 \end{bmatrix}$?

Solution:

▲ EDIT etc.

$$M = [2 \times 2 \text{ Matrix}]$$
$$\begin{bmatrix} 3.0 & 4.0 \\ -4.0 & 3.0 \end{bmatrix}$$

STO **α** **N** **ENTER↑**
▲ EIGVAL returns

$$M = [2 \times 2 \text{ Matrix}]$$

$$N = [2 \times 2 \text{ Matrix}]$$

$$\begin{bmatrix} 3.0 + i \times 4.0 & 0.0 \\ -4.0 & 3.0 + i \times 4.0 \end{bmatrix}$$

Note that although **N** contained only real elements, we get complex eigenvalues here.

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 **VARS** **Q**
EDIT etc.

$$[2 \times 2 \text{ C Matrix}]$$

$$\begin{bmatrix} 0.0 & 0.0 & -2.0 \\ 1.0 & 2.0 & 1.0 \\ 1.0 & 0.0 & 3.0 \end{bmatrix}$$

▲ EIGVAL returns

$$Q = [3 \times 3 \text{ Matrix}]$$

$$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Note one eigenvalue comes twice here. Let's get the eigenvectors of **Q** now – they will be put out as a matrix whose rows are these vectors:

x \approx y returns **Q** into **X**:

[3x3 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 = [3x3 Matrix]
[1.0 0.0 -2.0]
[0.0 1.0 1.0]
[-1.0 0.0 1.0]

STO **♂** **V** **ENTER↑**
X returns

[3x3 Matrix]
[2.0 0.0 -2.0]
[0.0 2.0 1.0]
[-2.0 0.0 1.0]

RCL **L** recalls V.
▲ **[M]-1**
X returns

[3x3 Matrix]
[2.0 0.0 0.0]
[0.0 2.0 0.0]
[0.0 0.0 1.0]

This looks very much like what was returned for the eigenvalues of Q above. Let's check:

DSP **0** **4**
- returns

[2x2 c Matrix]

$$\begin{bmatrix} 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \\ 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \\ 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \end{bmatrix}$$

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 .

Your WP 43S 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.

Note each matrix element can be entered as integer or fraction but is converted to a *real number* following the current display settings as soon as said element is closed:

[3x3 Matrix]

$$\begin{bmatrix} -38.000\ 0 & 6.142\ 9 & 31.500\ 0 & 82.071\ 4 \\ -14.000\ 0 & 2.714\ 3 & 7.000\ 0 & 25.857\ 1 \\ -1.142\ 9 & -2.489\ 8 & 3.428\ 6 & 3.612\ 2 \\ -16.000\ 0 & 3.714\ 3 & 13.000\ 0 & 34.857\ 1 \end{bmatrix}$$

[DSP] [0] [1] shall suffice here:

$$\begin{bmatrix} -38.0 & 6.1 & 31.5 & 82.1 \\ -14.0 & 2.7 & 7.0 & 25.9 \\ -1.1 & -2.5 & 3.4 & 3.6 \\ -16.0 & 3.7 & 13.0 & 34.9 \end{bmatrix}$$

MATX  **EIGVAL**

returns:

-5.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	3.0	0.0
0.0	0.0	0.0	5.0

Note the second eigenvalue is zero here.

RCL 

EIGVEC displays:

4.0	5.0	4.0	5.0
3.0	-2.0	2.0	-4.0
1.0	-4.0	-3.0	5.0
1.0	4.0	3.0	1.0

Generally, your WP 43S solves characteristic polynomials numerically.

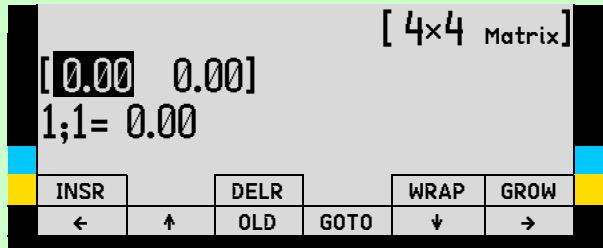
Vectors and Matrices: Dealing with Statistical Data

We mentioned above you can enter 2D statistical data using a matrix as well as keying them in point after point. How is this done?

Let's return to the **application** introduced on p. 106 with its step 4 – remember there were 30 samples measured twice in a special way using the instrument under investigation, resulting in 30 pairs of measured values:

1. Create a 1×2 named matrix and open it for editing:

1 **ENTER** **↑** **2** **DIM** **α M S A** **ENTER** creates **MSA** accordingly.
EDITN **MSA**



2. **GROW** allows the matrix to grow with data entered.
3. Now key in all 30 pairs of measured values. The first value shall be x , the second y – thus, the keystroke sequence will be

mv1 → mv2

for each sample. A subsequent **→** lets the matrix grow by one row for the next data point.

With all points entered, eventually key in

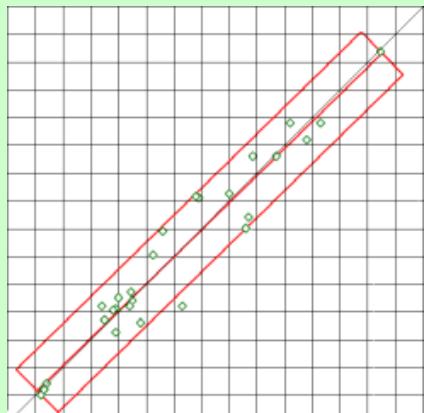
STAT CLΣ

RCL MSA

Σ+

Calling this with a 30×2 matrix in **X** will display the data of the 30th pair in **X** and **Y** and save a copy of the data point matrix in **L**.

4. It is recommended to plot these 30 data points. The diagram shall look like an ant trail around the center line $y = x$ (see an exemplary plot here but check the footnote on p. 107 as well).
5. Identify the most distant point (x, y) from said line. Remove it from the data set using **Σ-**.
6. Let your WP 43S fit a straight line



through the remaining 29 points and compute $c_0 = \frac{T}{30s_x s_y} \sqrt{\frac{s_x^2 + r^2 s_y^2}{1 - r^2}}$

with **T** being the width of the tolerance zone you want to control:

OrthoF
 r STO 0 0
 s

 x
 +
 1 RCL - 0 0
 /
 s x / 30 /
 .01 x

select the orthogonal linear fit model.
 get the *coefficient of correlation* and store its square.
 get s_x^2 .
 roll it out of the way.
 get $r^2 s_y^2$.
 return s_x^2 from the top stack register and calculate the numerator and the denominator.
 this is the second factor now.
 divide by $30 s_x s_y$.
 this returns c_0 for our T now (cf. pp. 106ff).

If you get $c_0 \geq 1$ then this measuring device may be used for controlling the given tolerance zone under these conditions – else look for a more precise instrument, better measuring conditions, or a wider tolerance.

If you want more information about vectors and matrices, their benefits and further applications, please turn to a textbook covering *linear algebra*. See MATX and the /OI for all the vector and matrix commands provided on your WP 43S.

Alpha Input Mode: Introduction and Virtual Keyboard

This mode is designed for text entry, e.g. for keying in messages, prompts, and answers. It is entered via **α** typically. It is also entered automatically whenever **X** contains an alpha string. Within AIM,



1. primary function of most keys will be appending the letter printed bottom right of the respective key to *x* – see the *virtual keyboard* overleaf;
2. The *menu Myα* will pop up immediately in the *menu section*, containing your favorite special characters or groups of them (up to 18 items);¹³⁰
3. prefix **g** leads to homophonic Greek letters.¹³¹

Upper and lower case are set by **▲** and **▼**, respectively, applying also to the letters in Myα and CATALOG'CHARS'aINTL (see pp. 183f).

Wherever a default primary function is not primary anymore in AIM but continues being meaningful, it is reached via *prefix f*. Thus, **f** is required here for appending a digit to *x*, for example. And **f** is also a shortcut to some special characters, like **f R/S** calling **≡** for fast access to the print functions in CATALOG'FCNS.

¹³⁰ For people writing German, for example, Myα might look like pictured overleaf

¹³¹ This will work wherever applicable, with “homophonic” following classic Greek pronunciation. Kudos to Thales, Pythagoras, Heraclitus, Leucippus, Democritus, Aristotle, Archimedes, Euclid (ὁ Θαλῆς ὁ Μιλήσιος, ὁ Πυθαγόρας ὁ Σάμιος, ὁ Ἡράκλειτος ὁ Ἐφέσιος, ὁ Λεύκιππος, ὁ Δημόκριτος, ὁ Αριστοτέλης, ὁ Ἀρχιμήδης ὁ Συρακούσιος καὶ ὁ Εὐκλείδης), and their colleagues for laying the foundations of logics, mathematics, and physics (i.e. ἡ λογικὴ καὶ ἡ μαθηματικὴ τέχνη καὶ ἡ φυσικὴ ἐπιστήμη – note that the first two were called “practical arts” while the latter was called “theoretical science”) as we know them today – starting some 2600 years ago.

And we assigned **Gamma** also to **C** following the alphabet, and **Chi** to **H** since this Latin letter comes next in pronunciation. Three Greek letters require special handling since they lack single-letter equivalents in English: **Psi** is accessed via **g 2** (since looking like **w** in a way), **Theta** via **g U** (following **T**), and **Eta** via **g J**. These three are printed in blue on the keyboard as reminder. **Omicron** is not featured since looking exactly like the Latin letter **O** in either case.

Two prefixes operate exclusively in AIM: **f R↓** makes the next directly keyboard-accessible input character a subscript if provided, while **f E** makes it a superscript. See the yellow arrows printed to the right of these two keys, above **H** and **L**.



And three alpha *menus* become accessible for more letters, punctuation marks as well as mathematical and other symbols (abbreviations are printed in blue on the keyboard as reminder). Look up their contents in *Section 2* of the *ReM*, and also the entire character set provided (in *App. D* therein).

Alpha Input Mode: Entering Simple Text and More

Your WP 43S features a large font for mainly numeric output and a small alphanumeric font for text strings. See here all characters directly evocable through the *virtual alpha keyboard* shown above:

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 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 Ρ Σ Τ Υ Φ Χ Ψ Ω
0 1 2 3 4 5 6 7 8 9 + - × or ÷ / . , ? ☰

and 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

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 δ η

and 0 1 2 3 4 5 6 7 8 9 + -

as well as superscripts α f g h o r T x and 0 1 2 3 4 5 6 7 8 9 + -

The 26 plain Latin letters can be also found in CATALOG'CHARS'A...Z while the 24 Greek letters (plus eleven accented ones) are stored in CATALOG'CHARS'A...Ω.

So you may, for **example**, easily store and display an actual Greek message like

Οι μελλοθάνατοι σε χάρετουν.

Latin-based letters required for international communication live in the menu αINTL (called via **g** **+** in AIM since it contains additional letters; see the ReM for its content).

Actually, we could have written the major part of this manual just using said font. It covers at least 47 languages from Afrikaans to Zhōng-wén (see the ReM, App. D), providing the

means that your display messages or prompts can be easily read and understood by more than 50% of all mankind.

Example:

You can even key in and display Deng Xiaoping's famous and successful slogan

in Pinyin straight ahead.

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 text strings in many more languages using letter sets based on Greek and Latin alphabets.

Your WP 43S supports you in climbing the very first step of politeness and respect by allowing you to adapt the software you write to the language your customers speak - instead of hacking in everything in English or using merely the very meager plain Latin letter set.

Two more *menus* ($\underline{\alpha MATH}$, called via), and $\underline{\alpha \bullet}$, called via contain further mathematical and non-mathematical symbols and marks (cf. the *ReM*):

$\oplus \perp \odot \mathbb{R} \mathbb{C}^{-1 T} * \mathbf{E}_{\mathbb{B}} \otimes \mathbb{Q}_{\mathbb{B}}$ etc.

Pressing **UM** in AIM toggles USERa. Individual characters may be assigned to particular locations on the keyboard or within *menus* in AIM only (see pp. 277ff for how to do that). Such user assignments will become accessible when USERa is set (indicated by **U** and **A** (or **U** and **a**) being both lit in the *status bar*).

AIM is closed by **EXIT** unless pressed in a menu.

Combining Alpha Strings and Numeric Data

Due to the *data type* concept of your WP 43S, adding numeric data to a text string is as simple as pressing **[+]**.

Example:

Assume the two lowest stack registers look like this:

The train will arrive at
23:55

So there is an alpha string in Y and a *time* in X. Pressing **[+]** now will combine x and y, returning

The train will arrive at 23:55

So, x will be converted to a string, taking into account its display format, and then it will be appended to y (cf. the matrix on p. 68). Let's enter a second string now, by pressing **[@]** and the necessary characters:

The train will arrive at 23:55
sharp at Victoria station.

Close AIM by pressing **[EXIT]**:

The train will arrive at 23:55
sharp at Victoria station.

Note this **[EXIT]** just leaves AIM and closes x. Now we have got alpha strings in X as well as in Y, so pressing **[+]** once more will append x to y returning

The train will arrive at 23:55 sharp at
Victoria station.

Once numeric data (like a *time* here) became part of an alpha string, they are fixed and will not vary even if format is changed. Easy, isn't it?

Working with Alphanumeric Strings

Your *WP 43S* provides some commands more for dealing with such strings. You find them all in a.FN:

aLEN? source pushes the length of the source string on the stack.

aPOS? source searches the string in the source for the character or string given in **X**; if a match is found, aPOS? returns the position number where the target was found (counting the leftmost character as position 0) – else it returns -1; previous *x* is saved in **L**.

aRL source rotates the source string by *x* characters to the left.

aRR source rotates the source string by *x* characters to the right.

a→x source converts the leftmost character in the source to the corresponding code, removes this character from the source string, and pushes its code on the stack; if the source is empty, a→x returns zero.

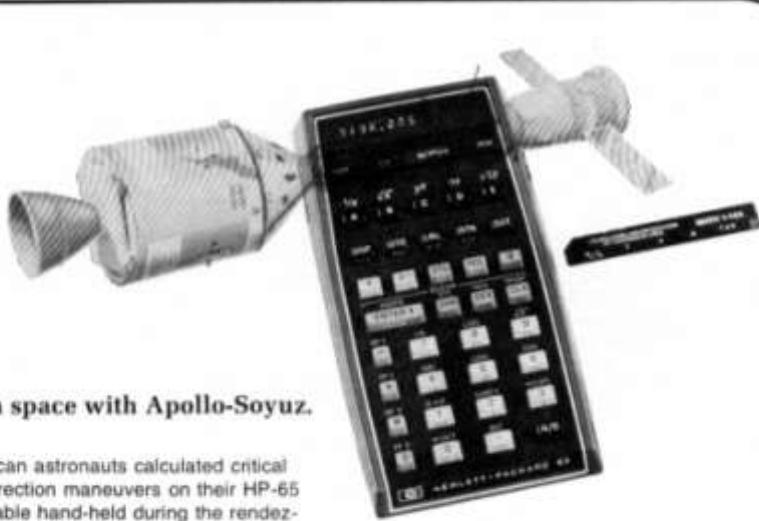
x→a destination converts a character code in **X** to the corresponding character and appends it to the destination; the character code is saved in **L**.

If **X** contains an alpha string, the entire string is appended to the destination.

If **X** contains a matrix, x→a uses each element in the matrix as a character code or alpha string. x→a begins with the first element (1; 1) and continues rowwise (to the right) until reaching the end of the matrix.

aSL source shifts the source string by *x* characters to the left, deleting the *x* leftmost characters from the string.

Nevertheless, do not forget that your *WP 43S* is mainly designed as a programmable calculator. Please turn overleaf to see what can be performed with such a device.



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, 800-642-9862) for the name of the retailer nearest you.

For more information on these products, write to us, Hewlett-Packard, 1504 Page Mill Road, Palo Alto, California 94304.

HEWLETT  **PACKARD**

Sales and service from 372 offices in 65 countries.

Mail to: Hewlett-Packard, 1504 Page Mill Road, Palo Alto, CA 94304
Please send me further information on:

- 1 HP 3000/1000A Logic/Stat Analyzers
 1 HP-43 Hand-Held Programmable Calculator

Name _____

Company _____

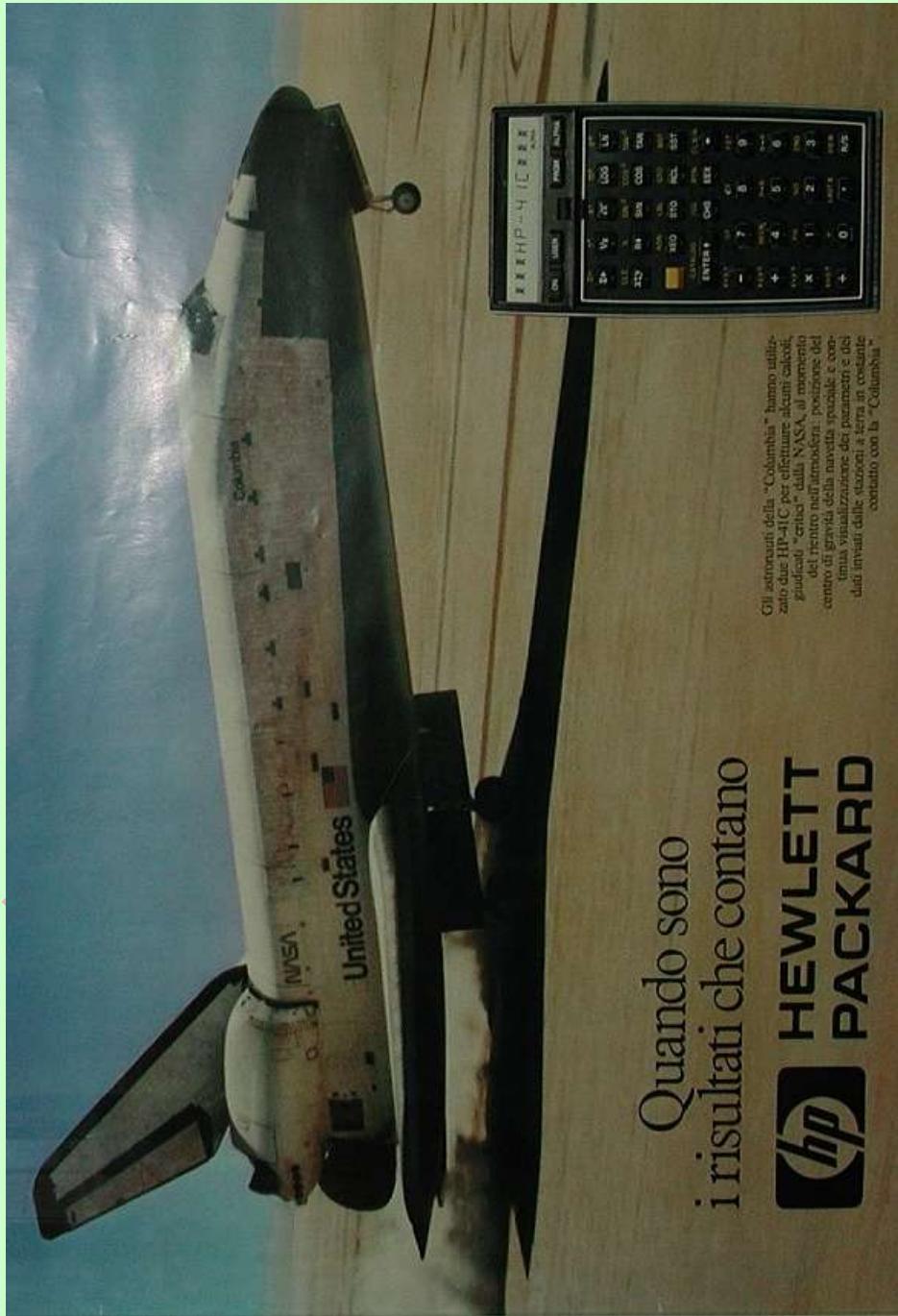
Address _____

City _____ State _____ Zip _____

*Domestic USA price only.

00148

An advertisement of 1975 (above) and another one of 198x (overleaf). Compare the capabilities of the WP 43S in your hands. Imagine the opportunities.



Quando sono
i risultati che contano

**HEWLETT
PACKARD**



Gli astronauti della "Columbia" hanno utilizzato due HP-43S per effettuare alcuni calcoli cruciali: i calcoli della NASA al momento del rientro nell'atmosfera, posizionare del centro di servizio, eletta navetta possibile e avvertire i risultati di posizionamento e dati inviati dalle stazioni a terra ai cosmonauti inviato con la "Columbia".

SECTION 3: PROGRAMMING

Your *WP 43S* is a powerful *keystroke-programmable* calculator. If already this statement makes you smile with delight, this section is for you. Else we will bring the smile on your face by mentioning the following facts:

Your *WP 43S* allows you to store a sequence of keystrokes like you would use them to solve a problem manually; this is to save you time on repetitive calculations (remember the example on pp. 19ff). Once you have written the keystroke procedure (or *routine*) for solving a particular problem and recorded it in your *WP 43S*, you need no longer devote attention to the individual keystrokes that make up the procedure. You can let your *WP 43S* 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 do not have to worry each time about whether or not you have pressed an incorrect key. Your *WP 43S* performs the drudgery, leaving your mind free for more creative work.

And it becomes even better: You may use program memory for storing more than just a single routine. For telling your *WP 43S* where such a routine begins and ends, each one is confined by two steps: it starts with LBL (for LaBeL) and typically ends with RTN (for ReTurN) – compare p. 21. These two steps separate it from other routines you may add for other tasks. And LBL puts a label on your routine so you can find and call it easily when you want it to be executed.

You may structure program memory even more: Put two or more routines together and separate them by END steps from other routines or sets of routines. What we find between two END steps we call a *program*. Programs are the basic building blocks within program memory. Think of the beginning and end of the entire used program memory section containing implicit END steps.¹³² So even with program memory cleared, there will be at least one program within at any time.

Within routines, you may store any sequence of keystrokes (commands, operations, objects). Choose any operation featured – the overwhelming majority of them are programmable. The steps in your

¹³² You cannot see that first END but the last one is visible – as pictured e.g. overleaf.



switch to *PEM* via **P/R**.

routine may also use each and every global *register*, variable, or *flag* provided – there are (almost) no limits. You are the sole and undisputed master of the memory!¹³³

Each such routine itself may contain one or more *subroutines*. Also subroutines start with LBL and typically end with RTN. Actually, subroutines may look exactly like routines. The only difference is that a **subroutine is called from another routine, while a main routine is called from the keyboard.** Thus we do not need to differentiate subroutines from routines further on.

Enough of theory – press **RTN** and The display of your *WP 43S* will change to something like this:

2015-07-15 14:31 RE ₄ ^o /5000 64:2	¶					
0000: LBL 'A'						
0001: x ²						
0002: π						
0003: x						
0004: RTN						
0005: END						
0006:						
R-CLR	R-COPY	R-SORT	R-SWAP	LocR	OFF	
PSTO	PRCL	αOFF	αON	CNST	PUTK	
INPUT	END	ERR	TICKS	PAUSE	P.FN2	

showing the example you entered on pp. 21f (the *status bar* on top will differ according to your time and settings).

¹³³ This freedom has a price: Take care that the routines do not interfere with each other in their quest for data storage space. It is good practice to record the global *registers*, variables, and *flags* a particular routine uses, and to document their purposes and contents for later reference.

Also remember some global *flags* have special effects as well.

An alternative method – using *local registers* and *flags* – will be explained further below.

In the section of the screen used for numeric output so far, the first seven steps in program memory are shown.¹³⁴ Labeled steps and END are ‘outdented’ for visual structuring. The current position of the *program pointer* (the *current step*) is highlighted by inversion; the routine the program pointer is currently in is called the *current routine*; the corresponding program is the *current program*. The *menu section* displays the top view of P.FN.

On the other hand, if you switch to *PEM* for the very first time after unpacking your *WP 43S* (or after resetting it), the display will look like this instead:

Recording a New Routine

Whenever you want to enter a new routine, switch to *PEM* using **P/R** (unless you are already in) and start with pressing **GTO****00**. These keystrokes will bring you to the very end of the used section of program memory, so you can start keying in your new routine right there without interfering with anything coded previously.

Start with LBL giving your routine a unique name (it may be up to seven characters long). Then press the keys as you would do in manual problem solving (cf. pp. 19ff). Each new step will be inserted right after the *current step*. You find

- **LBL** for LaBeLing a routine or a program step following,
- **XEQ** for eXECUting or calling a specific routine,

¹³⁴ There is no routine-specific step counting like in the *HP-42S* or *HP-35S*.

- **RTN** for ReTurNing from a routine called,
- **GTO** for unconditionally Going TO a specified label (i.e. positioning the program pointer to the respective LBL step),
- **R/S** for Running or Stopping the current routine,
- **▲** and **▼** (or **≡Δ** and **≡∇** if there is a *multi-view menu* displayed) for browsing program steps,
- **P/R** for toggling Program-entry and Run mode, and
- **EXIT** for EXITing PEM (returning to *run mode*)

all bottom right on your keyboard as shown on p. 190, together with the *menus* for LOOPs, TESTs, FLAGS, and PARTS. Further programming commands (like END mentioned above) are collected in P.FN. Note that **▲**, **▼**, **≡Δ**, **≡∇**, **P/R**, and **EXIT** are not programmable but useful in programming nevertheless (see also p. 197).

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:

$$\text{base area} = \pi r^2$$

$$\text{volume} = \text{base area} \times \text{height} = \pi r^2 h$$

$$\text{surface area} = 2 \text{ base areas} + \text{side area} = 2 \pi r^2 + 2 \pi r h$$

r [cm]	h [cm]	Base Area [cm ²]	Volume [cm ³]	Surface Area [cm ²]
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 (πr^2), store it for later use, and add the base area to a *register* which will hold the sum of all base areas.
2. Enter **h** and calculate the volume ($\pi r^2 h$). Add it to a *register* to hold the sum of all volumes.
3. Recall **r**. Divide the volume by **r** and multiply by **2** to yield the side area. Recall the base area, multiply by **2**, and add to the side area to yield the surface area. Sum the surface areas in a *register*.

Do not enter the actual data while writing the program-just provide for their entry. These values will vary and so will be entered before and/or during each program run.

Key in the following program to solve the above problem (assuming *startup default* – and we chose named variables instead of *registers*):

```

P/R
GTO .,.
LBL α MK ENTER↑
STO α ▽ R ENTER↑
x²
π
×
STO α BASE ENTER↑
STO + α g SB ENTER↑
VIEW VARS BASE
P.FN INPUT α ▽ H ENT↑
×
STO α VOLUME ENTER↑
STO + α g SV ENTER↑
VIEW VARS VOLUME
RCL VARS r

```

LBL 'MK'	Switch to PEM
STO 'r'	Store radius
x ²	
π	
×	Compute base
STO 'BASE'	
STO+ 'ΣB'	Sum of bases
VIEW 'BASE'	Show base for 1 s
INPUT 'h'	Enter height
×	Compute volume
STO 'VOLUME'	
STO+ 'ΣV'	Sum of volumes
VIEW 'VOLUME'	Show vol. for 1 s
RCL 'r'	

/	/	
2	2	
x	x	
RCL VARS BASE	RCL 'BASE'	Compute side
2	2	
x	x	
+	+	
STO + α g S S ENTER↑	STO 'ΣS'	Compute surface
RTN	RTN	Sum of surfaces
P/R		End of routine
		Leave PEM

Now, let's run the program:

2.5	2.5	Radius of 1 st can
DISP FIX 2		
XEQ PROGS MK	BASE = 19.64	Base of 1 st can
8	8	Height of 1 st can
R/S	VOLUME = 157.08	Volume of 1 st can
	2.50	Surface of 1 st can
4	4	Radius of 2 nd can
XEQ PROGS MK	BASE = 50.27	Base of 2 nd can
10.5	10.5	Height of 2 nd can
R/S	VOLUME = 527.79	Volume of 2 nd can
	164.93	Surface of 2 nd can
4.5	4.5	Radius of 3 rd can
XEQ PROGS MK	BASE = 63.62	Base of 3 rd can
4	4	Height of 3 rd can
R/S	VOLUME = 254.47	Volume of 3 rd can
	364.43	Surface of 3 rd can
RCL VARS ΣB	240.33	Sum of bases
RCL VARS ΣV	133.52	Sum of volumes
RCL VARS ΣS	939.34	Sum of surfaces
	769.69	Sum of surfaces

The preceding program illustrates the basic techniques of programming. It also shows how data can be manipulated in *PEM* and *run mode* by entering, storing, and recalling data (input and output) using **ENTER↑**, **STO**, **RCL**, store arithmetic, and programmed I/O.

If you want to run this routine again for another set of cans, remember to clear (or delete) the variables **ΣB**, **ΣV**, and **ΣS** before.

See the next paragraphs and the *IOI* for comprehensive information about all the commands used in this example and more.

Labels

As mentioned above, each routine or subroutine begins with a LBL step. Structuring program memory and jumping around within is eased by those labels. You may tag labels not only to the first but to any step in your routine – as known from previous programmable pocket calculators. Your *WP 43S* allows for specifying a wide variety of alphanumeric labels as described overleaf.

Whenever a step like e.g. GTO **label** is encountered in *run mode* (with **label** representing an arbitrary label), your *WP 43S* will **search this label** using one of the two following methods:

1. If **label** is plain numeric (00 ... 99), it will be searched forward from the current position of the program pointer. When an END step is reached without finding **label** 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*. So, *local labels* are valid in the *current program* only and may hence be reused in another program.
2. If, however, **label** is an alphanumeric label of up to seven characters of arbitrary case (automatically enclosed in ' like 'Ab1'), searching will start at program step 0000 and cover the entire program memory (first *RAM*, then *flash memory*) independent of the current position of the program pointer. This is the procedure for *global labels*.¹³⁵

¹³⁵ These search procedures for local and global labels are as known from the *HP-41C*.

So, *global labels* can be accessed from anywhere in memory, while *local labels* can only be accessed from within their own program.

Addressing labels, on the other hand, follows the rules given below:

1	User input Echo	[XEQ], [GTO], [LBL], [LBL?], SOLVE, \int, $f'(x)$, $f''(x)$, Π, or Σ OP _ (with TAM set) e.g. GTO _		
2	User input Echo	α ¹³⁶ sets AIM. OP ' _	\rightarrow ¹³⁷ opens indirect addressing. OP \rightarrow _	2-digit numeric (local) label 00 ... 99 OP nn e.g. LBL 07
3	User input Echo	Alphanumeric (global) label (up to 7 chars ¹³⁸) OP 'label' e.g. SLV 'F1μ'	Stack or lettered register X, Y, ..., K OP \rightarrow x e.g. $\int fd \rightarrow ST.T$	Register number 00 ... 99 000 ... 15 ¹³⁹ OP \rightarrow nn e.g. XEQ \rightarrow 44

$\int fd \rightarrow ST.T$ integrates the function given by PGMINT over the variable whose label is on *stack register T* (see pp. 239ff).

XEQ \rightarrow 44 calls and executes the routine whose label is found in **R44**.

Furthermore, GTO provides two special cases: see GTO. and GTO.. in next chapter.

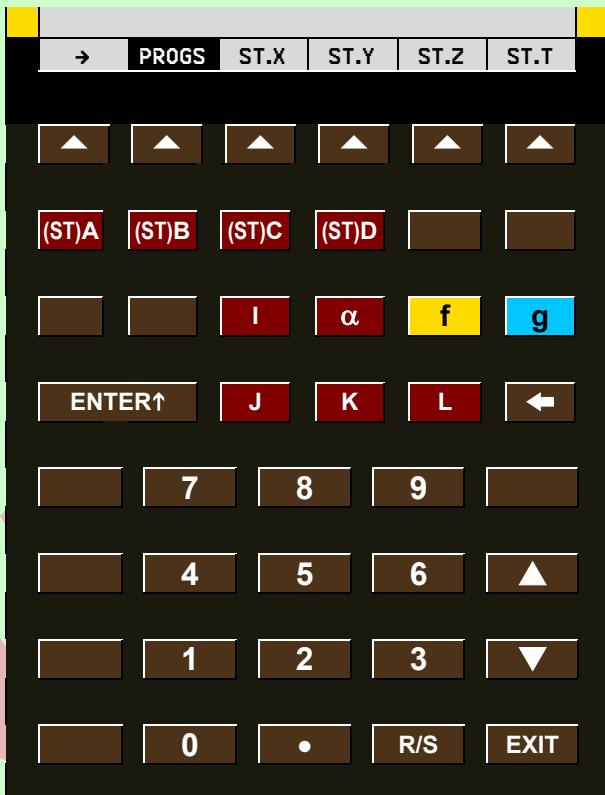
¹³⁶ Note you can skip pressing **f** here. And see the alternative overleaf.

¹³⁷ Works with all these operations except **LBL**.

¹³⁸ Said label must contain at least one letter. The seventh character will terminate entry and close AIM – shorter labels need a closing **ENTER↑**.

¹³⁹ ... if the respective *local register* is allocated. Some lettered *registers* may be dedicated to special applications. Check the memory table in *Section 1*.

And remember *TAM* is set during addressing, so the *virtual keyboard* of your *WP 43S* will work like this:



Note the changed assignment of the second softkey compared to p. 55. So, instead of keying in a longer global label in *alpha input mode*, it may be easier to press **PROGS** and select it from the *menu* of all global labels defined at the time of execution.

Editing a Routine

Whenever you want to edit (correct, expand, modify, etc.) an existing routine, ensure you are in *run mode* – then enter **GTO labl**. This will position the program pointer onto the corresponding LBL step (as explained on p. 195). Then switch to *PEM* using **P/R** and start browsing from this LBL step.

Let's browse program steps in our example routine: press **▼** four times:

2015-07-15 14:52 R _E 4° /5000 64:2	↑
0001: x^2	
0002: π	
0003: x	
0004: RTN	
0005: END	
0006:	
0007:	
R-CLR	R-COPY
PSTO	PRCL
INPUT	END
R-SORT	αOFF
	αON
	CNST
	PUTK
	TICKS
	PAUSE
	P.FN2

Unless you are next to the very beginning of program memory, the program pointer will always be placed in the middle of the *LCD* with three steps displayed above and three steps below of it, if available.

Navigating in program memory, you may execute various actions. If, for example, you want to...

- delete a program step, go to said step (i.e. make it the *current step* by positioning the program pointer on it), then press **◀**; it will vanish and the program pointer will move on the step before (note this deletion cannot be undone);
- insert something, go to the program step before, and then press the key(s) to be inserted after it;
- continue browsing forward, press **▼** (or **≡▼** if a *multi-view menu* is displayed); when reaching the END, browsing will start with the first step again;
- browse backwards, press **▲** (or **≡▲** if a *multi-view menu* is displayed); when reaching program top, browsing will stop;
- go to a particular global label (without inserting a GTO step in the current routine), press **GTO** **█** **label** **ENTER↑**; if you want to go to a local label, press **GTO** **█** **nn** instead;
- start writing a new routine, press **GTO** **█** **█**, then **LBL** ...

That's almost all (cf. the *IOI*). When you are done, press **P/R** or **EXIT** to leave *PEM*, returning to *run mode* again.

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 three alternatives:

1. **Normal execution of the current routine:** Press **RTN** to return the program pointer to the first step of the current routine. Then press **R/S**. This will run the routine, i.e. start automatically executing the following steps until a STOP, a final RTN, or an END will be encountered¹⁴⁰ where it will stop and display *x*.
2. **Normal execution of a selected routine:**¹⁴¹ Press **XEQ** and specify the label of the program you want to execute (or press **CATALOG PROGS** and pick the label from the *menu*). This will move the program pointer to the corresponding LBL step (cf. p. 195) and start automatically executing the following steps until a STOP, a final RTN or an END will be found¹⁴⁰ where it will stop and display *x* (cf. pp. 19f).
3. **Stepwise execution of a selected routine:** Press **GTO** instead. Specify the label of the program you want to execute (or pick it from the *menu*). This will move the program pointer to the corresponding LBL step (cf. p. 195) and wait. Each following program step will then be executed one at a time as you press **▼** (or **≡▼**) for it: pressing **▼** will display the step to be executed, releasing it will execute it. When you reach the end of the current routine, **▼** will return to its first step.¹⁴²

By following this procedure, you will go through the routine the same way as in normal execution but significantly slower – and you may perform additional checks after each program step. This

¹⁴⁰ ... or until you stop it manually by pressing **R/S** or **EXIT** – then execution halts after the current step is completed. For resuming execution, press **R/S** again.

¹⁴¹ This is the standard way to run routines. Furthermore, you can define shortcuts to your favorite routines by customizing your WP 43S as described in *Section 6*.

¹⁴² **▲** (or **≡▲**), on the other hand, moves the program pointer backwards in the current routine without executing anything.

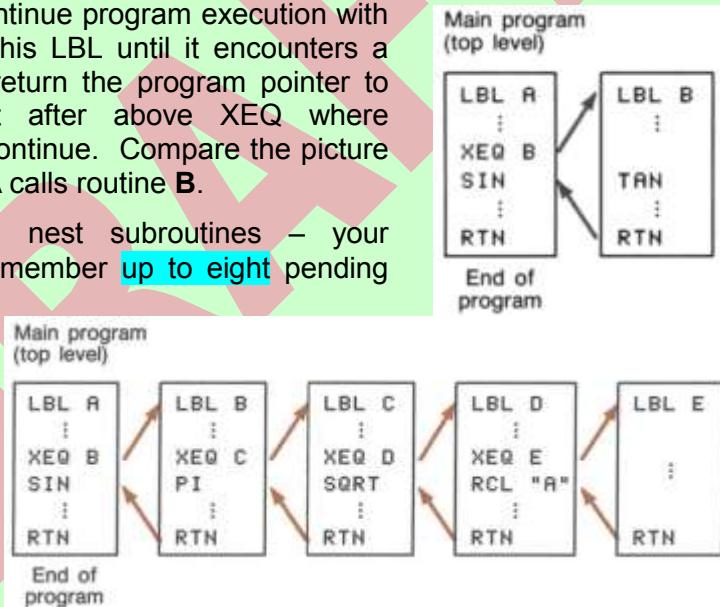
procedure is especially useful in *debugging* a routine (i.e. looking for errors in it).¹⁴³

If an error occurs while a routine is running, it stops immediately at the step generating said error and throws the corresponding error message (see App. C in the ReM for a list of all error messages provided). Press any key to clear this *temporary information*; to view the corresponding program step, 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. 195, go to it, and continue program execution with the step after this LBL until it encounters a RTN; this will return the program pointer to the step right after above XEQ where execution will continue. Compare the picture where routine A calls routine B.

You can also nest subroutines – your WP43S can remember up to eight pending return locations. But all of them will be lost for the current program if you should alter the program pointer while execution of this program is stopped; pressing **R/S** or **REV** or **V**, however, will not cause a loss of return locations.



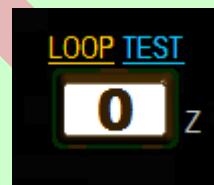
¹⁴³ Watch that your additional checks in debugging, if applicable, do not alter the status of your WP 43S in a way deviating from its status in automatic execution; else you shall compensate.

If you need any of your subroutines elsewhere, you can call it 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 at the beginning of this subroutine must be global.

Automatic Testing and Conditional Branching

So far, we were talking about linear programs running straight from beginning (LBL) to end (final RTN or END). Your WP 43S can do more for you: like keystroke-programmable calculators before, it features a set of binary tests checking various calculator states.

Most of the binary tests provided are collected in the menu TEST. There are also two tests in BITS, and eight tests on *flags* stored in FLAGS. Names of binary test commands contain a '?', most times as their last character.



Generally, binary tests will return **true** or **false** as *temporary message* at left of the lowest numeric row if called from the keyboard; if called automatically from a routine 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" (or "skip if false").¹⁴⁴ Think of the next step after 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
then go to the label **Join** (at step 32);
else swap *x* and *y*
and continue working here.

¹⁴⁴ The one and only exception: KEY? skips if true.

Most binary tests operate on x . They can check its *data type*:

- REAL? tests if X contains a real object (*data type* 2, 8, or 11) and executes the next program step if true, else skips it.
- DBL? tests if X contains a *DP* number (*data type* 11 or 12) ...
- CPX? tests if X contains a complex object (*data type* 3, 9, or 12) ...
- MATR? tests if X contains a matrix (*data type* 8 or 9) ...
- STRI? tests if X contains an alpha string (*data type* 7) ...
- SPEC? tests if x is special (i.e. \pm infinity or ‘Not a Number’) ...
- NaN? tests if x is ‘Not a Number’ and executes the next program step if true, else skips it.

They can check its numeric content:

- INT? tests if x is an integer number (i.e. has a zero fractional part) and executes the next program step if true, else skips it.
- EVEN? tests if x is an integer and even ...
- ODD? tests if x is an integer and odd...
- FP? tests if x has a nonzero fractional part ...
- PRIME? tests if the absolute value of the integer part of x is a prime number ...

They can compare its numeric content with 0, 1, or the content of another source specified (let’s call it s , see also pp. 56 and 58):

- $x < ?$ tests if x is less than s and executes the next program step if true, else skips it.
- $x \leq ?, x = ?, x \neq ?, x \geq ?,$ and $x > ?$ work in analogy to $x < ?$.
- $x \approx ?$ tests if the rounded values of x and s are equal ...

They can check its internal structure:

- BC? tests if X contains a *finite integer*, then checks the bit specified therein and executes the next program step if said bit is clear, else skips it.

- BS? tests if **X** contains a *finite integer*, then checks the bit specified therein and executes the next program step if said bit is set, ...
- 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 ...

General **flag tests** operate on the *flag* specified. It can be either clear or set.

- FC? tests that *flag* and executes the next program step if said *flag* is clear, else skips it.
- FC?C works as FC? but clears the *flag* after testing.
- FC?S works as FC? but sets the *flag* after testing.
- FC?F works as FC? but flips the *flag* after testing (i.e. clears it if it was set or sets it if it was clear).
- FS? tests that *flag* and executes the next program step if said *flag* is set, else skips it.
- FS?C works as FS? but clears the *flag* after testing.
- FS?S works as FS? but sets the *flag* after testing.
- FS?F works as FS? but flips the *flag* after testing.

Finally, there are **special tests**:

- LBL? tests for the existence of the label specified, anywhere in program memory.
- TOP? will return **true** if the program pointer is in the top routine (the leftmost one as sketched on p. 200).
- 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; 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 (see the picture overleaf).
- ENTRY? checks the (internal) entry *flag*. It is set if:
 - any character is entered in AIM, or

- any command is accepted for entry (be it via **ENTER↑**, a function key, or **R/S** with a partial command line).

ENTRY? is useful e.g. after PAUSE.

See the *IOI* for more information about the individual commands contained in TEST, also beyond those mentioned above.

☞ There are further commands also featuring a trailing '?' but returning numbers (e.g. WSIZE?) or codes (e.g. KTyp?) instead of **true** or **false** – you will find these commands in INFO.

As mentioned further above, routines end with RTN (typically) and programs with END. Executing a program, both steps work in a very similar way and show only subtle differences: a RTN immediately after a binary test returning **false** will be skipped – an END will not.

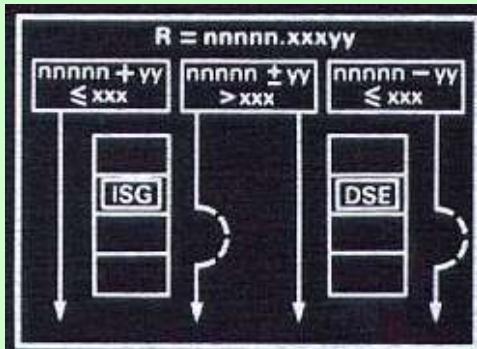
11	12	13	14	15	16
\sqrt{x} 21	y^x 22	TRI 23	ln 24	e^x 25	\sqrt{x} 26
STO 31	RCL 32	R↓ 33	UM 34	? 35	t 36
ENTER↑ 41	x>y 42	+/- 43	EEX 44	◀ 45	
/ 51	7 52	8 53	9 54	XEQ 55	
x 61	4 62	5 63	6 64	▲ 65	
- 71	1 72	2 73	3 74	▼ 75	
+ 81	0 82	. 83	R/S 84	EXIT 85	

Loops and Counters

The commands DSE, DSL, DSZ, ISE, ISG, and ISZ are for controlling loops in routines. They are all contained in LOOP. Each of them Decrements or Increments a counter in a *register* or variable as specified and executes or skips the following program step depending on the result. See the picture overleaf illustrating ISG (Increment and Skip if Greater) and DSE (Decrement and Skip if Equal); it is copied from the back label of a Voyager. With a GTO placed in the skipped step pointing to a label upstream in the same routine you can create

nice loops running until the specified condition is met.

Without such an exit condition you can create an infinite loop – such a routine will run until you stop it manually by **EXIT** or **R/S** or by battery voltage falling below the limit. Note such loops are allowed by the operating system of your **WP 43S**.



Example:¹⁴⁵

Write a little routine to store random numbers in **R25** through **R39**.

Solution:

Initialize the loop counter via **25.039** **STO** **2** **4**.

Reset the program pointer to the start of program memory by **RTN**.
Switch to programming via **P/R** and key in:

LBL **A**
PROB **▲ RAN#**
STO **→ 2 4**
LOOP **ISG** **2 4**
GTO **A**
RTN
EXIT

LBL **'A'**
RAN#
STO **→24**
ISG **24**
GTO **'A'** if $r24 \leq 39$ return to **A**
RTN else return to *run mode*.

Start this program by pressing **XEQ** **α** **A** **ENTER↑**. It will stop with the last random number in display. Check the target *registers* using **RBR**.

Example (continued):

Now, write a routine to sort those fifteen stored numbers so the smallest moves to the *register* with the smallest address.

¹⁴⁵ This example follows an idea of Gene Wright.

Solution:

We will use the so-called ‘bubble sort’ algorithm. Re-initialize the loop counter via **25.039 [STO] 2 4** (r_{24} was changed by program A above). Reset the program pointer to the start of program memory by **[RTN]**. Switch to programming via **P/R** and key in:

```

LBL B
P.FN LocR 2 ENTER↑
LBL 01
RCL 24
STO .000
LOOP INC X
STO .011
FLAGS CF .000
LBL 03
RCL → .000
RCL → .011

TEST x< ? Y
GTO 02
LBL 04
LOOP ISG .000
ISG .001
GTO 03

FLAGS FS? .000
GTO 01
RTN
LBL 02
SF .00
STO → .000
x>y
STO → .011
x>y
GTO 04
EXIT

```

LBL 'B'	
LocR 002	Allocate two local registers.
LBL 01	
RCL 24	Put the start pointer r_{24} into local register 00.
STO .00	
INC ST.X	Increment the pointer and store it in local register 01.
STO .01	
CF .00	Clear local flag 00.
LBL 03	
RCL →.00	Recall the contents of the registers where $r.00$ and
RCL →.01	$r.01$ are pointing to.
x< ? ST.Y	Is $x < y$?
GTO 02	Then go to label 02 – else
LBL 04	...
ISG .00	... increment $r.00$ and
ISG .01	if $r.00 \leq 39$ increment $r.01$ &
GTO 03	if $(r.00 > 39 \text{ or } r.01 \leq 39)$ return to label 03;
FLAGS FS? .000	else check local flag 00:
GTO 01	if set, return to label 01,
RTN	else stop this routine.
LBL 02	
SF .00	Set local flag 00.
STO →.00	Store the smaller value
x>y	where $r.00$ and the greater
STO →.01	where $r.01$ is pointing to.
x>y	Restore the stack and
GTO 04	return to label 04 and ...

Start the program by pressing **XEQ** **α** **B** **ENTER**. 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**.

By the way, note this program allocates two *local registers* for its exclusive use (**R.00** and **R.01**). Furthermore, it uses one local *flag* and four local labels.

The following alternative sorting program is even shorter (kudos to Jean-Marc Baillard for this routine):

```
LBL 'C'
SIGN
LBL 01
RCL L
RCL L
RCL →L
LBL 02
RCL →ST.Y
x> ? ST.Y
GTO 03
x>y
RCL L
+
LBL 03
R↓
ISG ST.Y
GTO 02
x> →L
STO →ST.Z
ISG L
GTO 01
END
```

Just start it by keying in **2** **5** **.** **0** **3** **9** **XEQ** **α** **C**.

Cf. *HP-42S Owner's Manual*, pp. 152 – 154.

Programmed User Interaction and Dialogues

A number of commands are provided for controlling the interaction of programs with you. A program shall output some results to you at least, and it may also ask for your input. In the *I/OI*, 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 hour glass  will start flashing in the *status bar*. While in *manual mode* each command executed may change the display immediately, in *automatic mode* only INPUT, PAUSE, STOP, or VIEW will update the display, and this display will hold until the next such command is encountered or *automatic mode* is left. For programmed I/O, see the following **examples**:

- Take VIEW for displaying intermediate results. Specify any *register* or variable you want as source of information – also **X** is a valid parameter of VIEW. The name of the source will label the output.

 Frequent display updates will slow down program execution, since the anti-flicker logic waits for a complete display refresh cycle before allowing the next update.
- Use
VIEW xyz
PAUSE nn for displaying numeric or alphanumeric output for a defined minimum time interval, specified by PAUSE.
- If you have a printer connected, you may send your program output thereto as well. Turn to pp. 218ff for more about printing.
- Ask ('prompt') for numeric input employing

VIEW xyz

STOP

STO xyz

update display showing the *register* or variable **xyz**, ... and wait for user reaction, finished by **[R/S]**.

 A stop sign  will be displayed in the status bar when the program pointer runs on STOP. Whatever you will key in then will be put into **xyz** when you continue the program by pressing **R/S**.

More elegant is using INPUT for this task:

INPUT xyz does the same in just one step.

- Prompt for alphanumeric input using the following steps:

αON	sets AIM for upcoming input.
RCL xyz	displays the <i>register</i> with the message string.
STOP	waits for your input. Whatever you key in now is appended to <i>x</i> when you continue by pressing R/S .
αOFF	returns to the numeric mode previously set.
STO xyz	stores <i>x</i> to wherever you like.

Again, more elegant is using INPUT for this task:

αON	sets AIM for upcoming input.
INPUT xyz	
αOFF	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.
MVAR 'xy1'	
MVAR 'xy2'	
MVAR 'xy3'	
...	
VarMNU 'Var.In'	creates a <i>menu</i> for the variables defined immediately after 'Var.In' and shows it.
STOP	stops for user interaction.
EXITall	exits the <i>menu</i> when program continues.
RCL 'xy2'	recalls what you need first (it may have been entered in any order).
...	

The label called '**'Var.In'**' here should be located close to the program top. It shall be followed by up to six MVAR steps defining your variables required. When the program encounters

the step VarMNU, it will setup a *menu* for these variables and display it. Here, this would look like

xy1	xy2	xy3
-----	-----	-----

Now, if you want to

- **write** a new value into one of the variables displayed, key the value in or calculate it, then press the corresponding *softkey*.
- **recall** the present value of one of the variables displayed, enter **RCL**, then press the corresponding *softkey*.
- **view** the present value of one of the variables displayed, enter **VIEW**¹⁴⁶, then press the corresponding *softkey*.
- **exit** this *menu*, press **EXIT**.
- **continue** program execution, press **R/S**.

See the *IOI* for more information about the commands mentioned in this chapter and their parameters.

Solving Differential Equations

The following method uses the programmability of your *WP 43S* for solving ordinary second order differential equations, a type frequently occurring in physics.¹⁴⁷

In a **first example**, we will solve the equation of motion for the fall of a parachutist $\frac{d^2f}{dt^2} = a - b \left(\frac{df}{dt} \right)^2$ with earth acceleration $a = 9.81 \frac{m}{s^2} = g$ and b taking care of drag.

¹⁴⁶ Note that the *HP-42S* allowed for just viewing the present value of one of the variables displayed by pressing **f**, then the corresponding *softkey*; we cannot support this on your *WP 43S* since you get three menu rows here.

¹⁴⁷ Turn to the *ReM*, *App. J*, for background information about the method applied here.

Proceeding in small constant time steps Δt , the following set of equations governs the vertical motion of the parachutist (or skydiver):

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

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

$$f_2 = f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t \quad \text{etc.}^{148}$$

Assume start height ($t=0$) is 1 000 m and vertical velocity is zero (i.e. $f(t=0) = f(t_0) = f_0 = 1000$ and $\left(\frac{df}{dt}\right)_0 = 0$). Using named variables Δt , a , b , t , f , and df/dt , the following routine will compute height and velocity of the parachutist as functions of time:

```
LBL 'PFall'
  .5
  STO 'Δt'
  # g
  STO 'a'
  .003
  STO 'b'
  1000
  STO 'f'
  0
  STO 't'
  STO 'df/dt'

LBL 01
  RCL 'a'
  RCL 'b'
  RCL 'df/dt'
  x2
  x
  -
  RCL 't'
```

initialize all variables used
take g out of CNST
assumed realistic drag value for this case
start height
start time and velocity
end of initialization
begin of time loop
 $b \times (df/dt)^2$
 $a - b \times (df/dt)^2$

¹⁴⁸ Note the contents of the rectangular brackets must be ≥ 0 always. Thus, this routine will work for velocities $< \sqrt{a/b}$ only, not for decelerating fast initial movements.

```

x>0 ?
GTO 02
DROP
2
/
GTO 03
LBL 02
DROP
LBL 03
RCLx 'Δt'
STO+ 'df/dt'
RCL 'Δt'
STO+ 't'
RCLx 'df/dt'
STO- 'f'
VIEW 't'
STOP
VIEW 'f'
STOP
VIEW 'df/dt'
STOP
GTO 01
END

```

check time – it will be zero in first run
from 2nd run on go to local label 02

1st run only: forget t

1st run only:

1st run only: $[a - b \times (df/dt)^2]/2$

1st run only: go to common part
from 2nd run on:

from 2nd run on: forget t

common part of time loop begins here again

$[a - b \times (df/dt)^2] \times \Delta t$ (or half of it in 1st run)

calculate the new df/dt

calculate the new time

$df/dt \times \Delta t$

calculate the new f

display new time

display new height

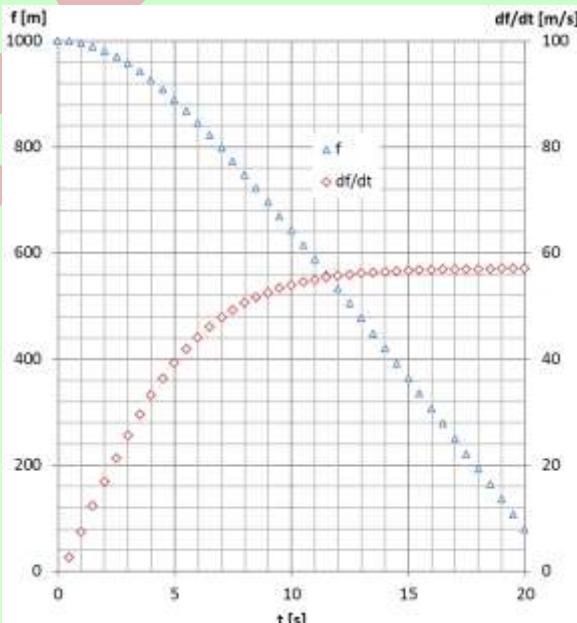
display new velocity

} for plotting next
data point
(t; f)
and
(t; v).

end of time loop

Now, leave PEM and start executing this program via XEQ PROGS

PFall – plotting the points calculated will result in a diagram as printed here. Height decreases following a parabola over time in the beginning but becomes linear later. Note vertical velocity does not increase much anymore after some 12 seconds here, approaching some 60 m/s



with closed parachute. For comparison: with open parachute the velocity limit would be <6 m/s for $b = 0.3$, so the vertical velocity at touchdown will be like falling from a wall 1.65 m high.

In a **second example**, we will demonstrate solving a 2D problem like e.g. finding the orbit of a satellite in the gravitational field of the earth. Here we have a pair of coupled differential equations. This problem is solved as follows:

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

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

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

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

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

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

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

So, here is some crosstalk (a.k.a. coupling) between x and y . Nevertheless, proceeding like we did in the first example above, the following routine will compute the coordinates x and y of the satellite as functions of time.

For ease of handling in a first calculation, we set $GM = 1 = a$ and the start values $x_0 = 1$, $\left(\frac{dx}{dt}\right)_0 = 0$, $y_0 = 0$, $\left(\frac{dy}{dt}\right)_0 = 1$. These ‘variable’ start values shall be entered using INPUT here (cf. p. 209):

```
LBL 'Satell'
  INPUT 'x'
  INPUT 'y'
  INPUT 'dx/dt'
  INPUT 'dy/dt'
  .1
  STO 'Δt'
  1
  STO 'a'
  0
  STO 't'
```

start of variable initialization

initialize the remaining ‘fixed’ start values

(for earth satellites, take GM out of CNST instead)

start at time zero

end of initialization

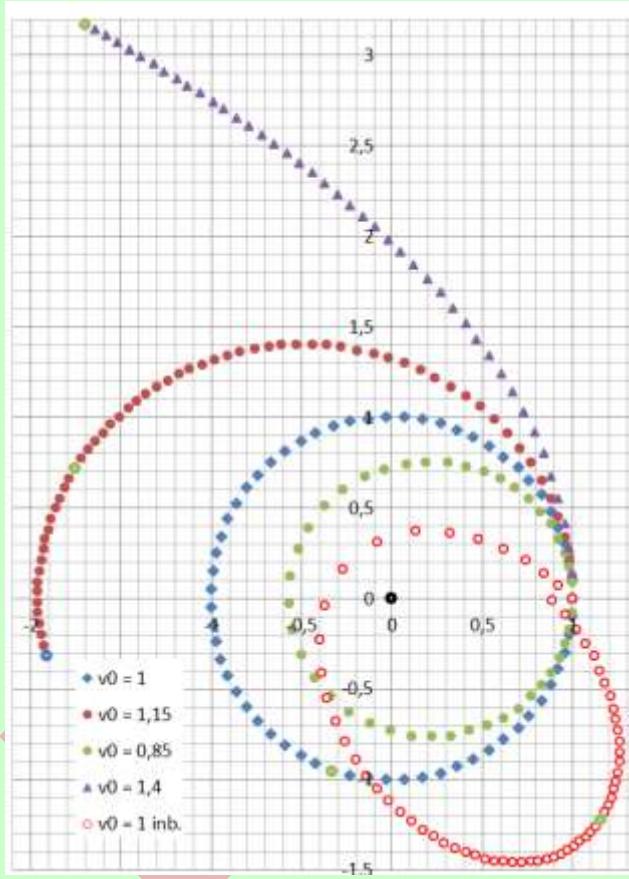
LBL 01
 RCL 'y'
 RCL 'y'
 x^2
 RCL 'x'
 x^2
 +
 -1.5
 y^x
 RCLx 'a'
 x
 RCL L
 RCLx 'x'
 RCL 't'
 $x > 0 ?$
GTO 02
 DROP
 2
 /
 ~~$x \times y$~~
 2
 /
 ~~$x \times y$~~
GTO 03
LBL 02
 DROP
LBL 03
 RCLx ' Δt '
 STO- 'dx/dt'
 DROP
 RCLx ' Δt '
 STO- 'dy/dt'
 RCL ' Δt '
 STO+ 't'
 RCLx 'dx/dt'
 STO+ 'x'
 VIEW 'x'
 STOP
 RCL ' Δt '
 RCLx 'dy/dt'
 STO+ 'y'
 VIEW 'y'

begin of time loop
 y^2
 y^2+x^2
 $(y^2+x^2)^{-1.5}$
 $a(y^2+x^2)^{-1.5}$
 $y a(y^2+x^2)^{-1.5} = -K_y$
 $a(y^2+x^2)^{-1.5}$
 $x a(y^2+x^2)^{-1.5} = -K_x$. Stack is $[-K_x, -K_y, \dots]$ now.
 check time – it will be zero in first run
 from 2nd run on go to local label 02
 1st run only: forget t
 1st run only:
 1st run only: $-K_x / 2$
 1st run only: stack is $[-K_y, -K_x / 2, \dots]$ after ~~$x \times y$~~
 1st run only:
 1st run only: $-K_y / 2$
 1st run only: stack is $[-K_x / 2, -K_y / 2, \dots]$ after ~~$x \times y$~~
 1st run only: go to common part of time loop
 from 2nd run on:
 from 2nd run on: forget t
 common part of time loop begins here again
 $-K_x \times \Delta t$ (or half of it in 1st run)
 calculate the new dx/dt
 bring y to the front
 $-K_y \times \Delta t$ (or half of it in 1st run)
 calculate the new dy/dt
 calculate the new time
 $dx/dt \times \Delta t$
 calculate the new x
 display the new x for plotting
 $dy/dt \times \Delta t$
 calculate the new y
 display also new y for plotting the new point (x, y)

STOP
GTO 01
END

end of time loop, return for next run through it
end of program

Plotting the points calculated for these start values will result in a perfect circle as shown by the blue symbols in the diagram – taking 64 Δt for one orbit. We added some examples with slightly different start velocities for comparison. The green elliptical orbit takes 46 Δt only, the dark red one 116. Green and blue marks in the other curves highlight the positions after 46 and 64 Δt for comparison.



The innermost red ellipse starts with velocity 1 but directed 45° 'NW'. Note this curve does not close due to the perigee speed being too high for the time step Δt chosen. Watch the limits of such computational models always!

The Programmable Menu (MENU)

Your WP 43S has a *programmable menu* which is used to cause program branching. By this, you can create *menu-driven* programs. The MENU function selects the *programmable menu*. The *menu* is displayed when the program stops. You can define each *softkey* in this

menu so that when this key is pressed, a particular GTO or XEQ instruction will be executed. You can even define **[▲]**, **[▼]**, and **[EXIT]**.¹⁴⁹

To define a **softkey** in the *programmable menu*:

1. Store a string of up to seven characters in *register K*. This is the text that will appear in the *menu space* for the *softkey* specified.¹⁵⁰ **K** is not used 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. Specify which key you want to define:
 - a. Press one of the eighteen *softkeys* available, **[▲]**, **[▼]**, or **[EXIT]**.
 - b. Alternatively, enter the respective key number, 1 through 21.
4. Specify a program label using one of these three methods:
 - a. Select an existing global label by pressing the corresponding *softkey*.
 - b. Explicitly key in a global label using AIM.
 - c. Key in a two-digit local label.

Repeat this procedure for each *softkey* in the programmable *menu* you want to define. The new definition replaces any previous definition that may exist for that *softkey*.

To display the *programmable menu*:

Execute the MENU function, e.g. by entering **P.FN** **P.FN2** **MENU**.

To clear all **softkey** definitions in the *programmable menu*:

Call CLMENU (clear the programmable *menu*), e.g. by entering **CLR** **CLMENU**.

¹⁴⁹ See *HP-42S Programming Examples and Techniques*, pp. 29 - 39, 92 - 99, 158 - 160, and 184 - 192, for sample programs using the *programmable menu*.

¹⁵⁰ If space does not suffice, only the first characters will be displayed.

Basic Kinds of Program Steps

You have seen various program steps so far. Each step takes a single place in program memory, and each step is numbered automatically. Basically, the contents of these steps fall into four categories – one program step may contain...

- a global or local **label** (like `LBL 'Join'` or `LBL 07` above) or
- a complete **command** (like `-` or `yx` or `STOx →Prd2`) or
- an entire **alpha string** (a.k.a. an *alphanumeric constant*, like `"This is a text."`; such a constant will be automatically stored in **K**) or
- an entire **number** (a.k.a. a *numeric constant*, like `-1.902×10-16` or `1234.5` or `#g`; such a constant will be automatically stored in **X**).

Since each constant takes one step, there is no need for separating them by **ENTER↑** in a routine.

Example:

Think of calculating $12.3 + 45.67$ in a routine.

Then pressing `12.3 [ENTER↑] [→] 45.67 [+]` will result in a program snippet

12.3
45.67
+

which will do for returning 57.97 as it should. The missing **ENTER↑** saves one byte of program space (woohoo!) and makes the routine a tiny bit faster. It may not be really important here but you should know.

Constant vectors and matrices cannot be entered directly in a program step; though you can store them in *registers* or variables and manipulate these stored *items* (as described in *Section 2*) in routines as well.

Program steps may require two or more bytes of memory. We think you will hardly ever run out of program space (but you may, of course: if you do while trying to enter a new program step, you will read an error message `RAM is full` ; see *App. B* of the *ReM* for ways to escape from such a situation).

Deleting Programs

To delete some steps of a program, proceed as explained on pp. 197f.
Repeat as often as necessary.

To delete an entire program, move the program pointer into this program (e.g. by entering **GTO** . and picking the label of this program) and then press **CLR CLP [ENTER]**.

Note CLP will remove the entire program from memory, not only the routine the program pointer is in. And CLP cannot be undone! The space freed by CLP will be added to the pool of free space your WP 43S features.

To delete all programs stored in RAM, press **CLR CLPall** and confirm. Thereafter, program memory will be completely clear. Note that also CLPALL cannot be undone!

Serial Input and Output of Data and Programs

xxx

Local Data

After some time with your WP 43S you will have a number of routines stored, so keeping track of their resource requirements may become a challenge. 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 routine* only; when the routine is finished, the respective memory is released. On your WP 43S, mainly *registers* are used for data storage – so we offer you *local registers* allocated to your routines exclusively.

Example:

Let's assume you write a routine labeled **P1** and need five *registers* for your computations therein. Then all you have to do is just enter **PEM**, go

into the routine **P1**, and enter

P.FN LocR 5 ENTER↑

specifying that you want five *local registers*. Thereafter, you can access these *registers* by using local addresses **.00** ... **.04** throughout **P1**.

Now, if you call another routine **P2** from **P1**, also **P2** may contain a step **LOCR**, requesting *local registers* again. These will also carry *local register* addresses **.00** etc., but the *local register .00* of **P2** will be physically different from the *local register .00* of **P1**, so no interference will occur. As soon as the return step is executed, the *local registers* of the corresponding routine are released and the space they took is returned to the pool of free memory.

In addition, you get sixteen local *flags* as soon as you request at least one *local register*.

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 *flags* being different from the ones it got before.

See the commands **LOCR**, **LOCR?**, **MEM?**, and **POPLR** in the *IOI*; and look up *App. B* of the *ReM* for more information, also about the limitations applying to local data.

Flash Memory (FM)

In addition to the *RAM* provided, your *WP 43S* allows you to access *FM* for voltage-fail-safe storage of user programs and data. The first section of *FM* is a *backup region*, holding the image of the entire *RAM* (i.e. user program memory, *registers*, and *WP 43S* states) as soon as you have executed a **SAVE**. The remaining part of *FM* is for programs only.

Global labels in *FM* can be called using **XEQ** like in *RAM*. This allows creating program libraries in *FM*. Use **CATALOG'PROGS'FLASH** to see the global labels already defined in *FM*.

FM is ideal for backups or other relatively long-living data, but shall not be used for repeated temporary storage like in programmed loops. Conversely, *registers* and standard user program memory residing in

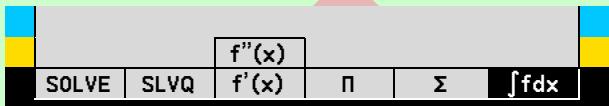
RAM are designed for data changing frequently but will not hold data with the batteries removed for longer than some minutes.¹⁵¹ So both *RAM* and *FM* have their specific advantages and disadvantages you shall take into account for optimum benefit and longevity of your *WP 43S*.

DRAFT

¹⁵¹ *FM* may not survive more than some 100 000 flashes. Thus, we made commands writing to FM (SAVE or PSTO) non-programmable.

SECTION 4: ADVANCED PROBLEM SOLVING

There are some powerful commands provided for computing programmable sums and products, for solving equations, for computing definite integrals as well as first and second derivatives. All are contained in ADV or EQN. Pressing **ADV** in *run mode* results in



The commands Σ , Π , $SLVQ$, $SOLVE$, \int , $f'(x)$, and $f''(x)$ are explained below in this order. All these commands may also be programmed. Integrating, deriving, and solving equations interactively may be reached through **EQN**. See below for details.

Programmable Sums

The command Σ is called with a loop control number in **X** and a label trailing the command. Said loop control number follows the format `cccccc.ffffii` (as it does in DSE etc. mentioned above).

In its heart, Σ then works like this:

1. It sets the sum to **0** initially.
2. It fills all *stack registers* with `cccccc` and calls the routine specified by the label. That routine returns a summand in **X**.
3. It adds this summand to said sum.
4. It decrements `cccccc` by `ii`; if `cccccc ≥ ffff` then Σ goes back to step 2, else it returns the final sum in **X**.

If `ii = 0`, `cccccc` will be decremented by **1** in each loop.

Example:

Compute $\sum_{k=0}^{100} \sqrt{k}$

Solution:

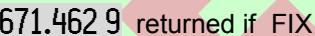
1. Write a little program for the internal calculation of the summands:

```
LBL 'ΣSQRT'  
  √X  
  RTN
```

2. Enter

100

 (or pick ΣSQRT from PROGS, cf. p. 197)

and get  returned if FIX 4 is set.

Σ deliberately sums from the last term to the first, on the assumption that summations will often be of convergent series and this summing order should generally increase accuracy.

Programmable Products

The command Π is called with a loop control number in X and a label trailing the command (like for the command Σ).

In its heart, Π works almost as Σ :

1. It sets the product to 1 initially.
2. It fills all stack registers with ccccccc and calls the routine specified by the label. That routine returns a factor in X .
3. It multiplies this factor with said product.
4. It decrements ccccccc by ii; if ccccccc ≥ ffff then Π goes back to step 2, else it returns the final product in X .
If $ii = 0$, ccccccc will be decremented by 1 in each loop.

Example:

Compute $\prod_{k=1}^{50} \frac{1}{\sqrt{k}}$

Solution:

1. Write a little program for the internal calculation of the factors:

```
LBL 'PROD'  
  √X  
  1/X  
  RTN
```

2. Enter

50.001

ADV **Π** **Q** **P R O D** **ENTER↑**

(or pick PROD from PROGS, cf. p. 197)

and get 5.734×10^{-33} returned if SCI 3 is set.

Solving Quadratic Equations

The command SLVQ finds the real and complex roots of a quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input stack [c, b, a, \dots] :

- If $r := b^2 - 4ac \geq 0$, SLVQ returns the two real roots $-\frac{b \pm \sqrt{r}}{2a}$ in Y and X. If called in a routine, the step after SLVQ will be executed then.
- Else, SLVQ returns the first complex root in X and the second in Y (the *complex conjugate* of the first). If called in a routine, the step after SLVQ will be skipped then.

So actually, SLVQ tests for real roots at its very end. In either case, SLVQ returns r in Z. Higher stack registers



are kept unchanged. **L** will contain equation parameter **c**.

Example:

Find the roots of $4x^2 - 3x - 2 = 0$.

Solution:

4 [ENTER] 3 [+/-] [ENTER] 2 [+/-]

[ADV] SLVQ returns (with FIX 4 chosen) $x = 1.1754$, $y = -0.4254$, $z = 41.0000$. Since z is positive, x and y are the two real roots of this equation here.

Check:

Store x in **R00** and y in **R01**. Then enter

RCL 0 0 [FILL] 4 [x] 3 [-] [x] 2 [-] returning $-1.000\ 0 \times 10^{-15}$ and
RCL 0 1 [FILL] 4 [x] 3 [-] [x] 2 [-] returning $0.000\ 0$.

Remember your **WP 43S** calculates with 16 digits precision, so any result within $\pm 3 \cdot 10^{-15}$ is equal to zero in this matter.

General Equations

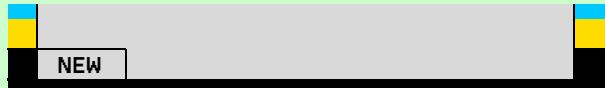
The *menu EQN* lets you store, select, and edit arbitrary equations; you may use each such equation for

- solving it interactively for any variable it contains, for
- integrating and
- deriving.

The number of equations you can store and the number of variables used in each equation are limited only by the amount of free memory available.

Example:

Press **[EQN]**. If there are no other equations in memory yet, your **WP 43S** will return:



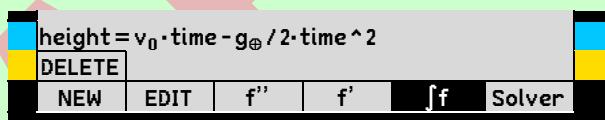
Press **NEW** to enter a new equation. You will get immediately:



with *alpha input mode* turned on. Enter your equation now, e.g.

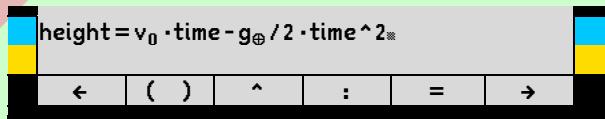
height **=** v **R** **D** **0** **X** time **-** g_⊕ **/** 2 **X** time **^** **2** ¹⁵²

for the height of e.g. a ball thrown vertically upwards with velocity v_0 . Press **ENTER↑** for closing the *Equation Editor* and see: ¹⁵³



You will get such a display whenever one or more equations are stored. The equation shown is called the *current equation*. To select another equation as the current equation, press **▲** or **▼** until the requested equation appears.

Pressing **EDIT** opens the *Equation Editor* for the current equation:



You may modify this equation at any position by moving the edit cursor to the location after the (last) character to be corrected and pressing **←** followed by the new character(s) to be inserted here.

¹⁵² The index of the earth acceleration constant is found in the punctuation menu a•, reached via **g** **•** in AIM.

¹⁵³ Note MULT· is set here for sake of better readability of equations.

For labeling this equation, move the cursor left to its very begin using **⬅**, and key in:

F **▼** **R** **E** **█** **A** **F** **▼** **A** **L** **:**

FreeFall:height = v₀ · time - g_⊕ / 2 · time²

ENTER↑

FreeFall: height = v₀ · time - g_⊕ / 2 · time²

DELETE

NEW

EDIT

f''

f'

ʃf

Solver

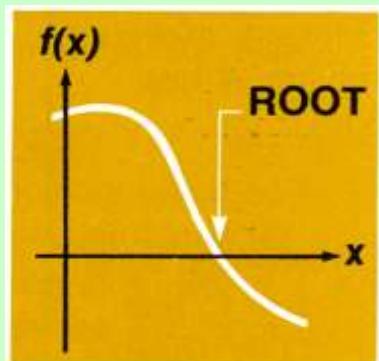
ENTER↑ closes the *Equation Editor* storing the modifications you made. Editing an equation clears all its variables.

If an equation should become longer than the display width ellipses will be displayed at its end(s); then use **→** and **←** in the *Equation Editor* for scrolling.

The Interactive Solver for Arbitrary Equations

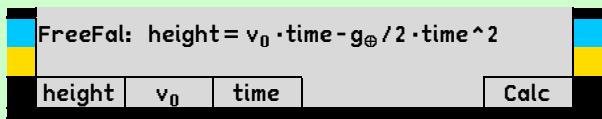
The built-in *Solver* application of your *WP 43S* is a special root finder that enables you to solve an equation for any of its variables. It allows for solving for an arbitrary unknown as well as for finding the root(s) of an arbitrary equation.¹⁵⁴

Press **EQN**, make the equation you want to solve the *current equation* (see previous chapter), and press **Solver**. Your *WP 43S* will check this equation for syntax errors (missing operators,



¹⁵⁴ Translator's note for German readers: Der eingebaute Gleichungslöser („Solver“) Ihres *WP 43S* erlaubt Ihnen das Auflösen nach einer beliebigen Unbekannten bzw. das Finden der Nullstellen einer beliebigen Gleichung.

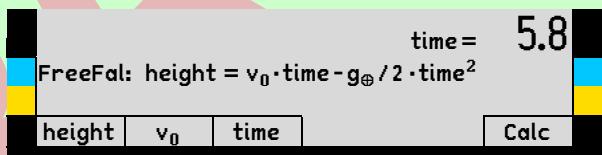
misspelled functions, illegal variable names, etc.). It will then return a menu of all applicable variables. In our **example**, it looks like this:



Note your *WP 43S* knows g_0 is a constant contained in CNST. Now you can enter values for any variables you know by pressing the respective softkeys, e.g. **-50 height 20 v₀** (corresponding to 50 m below start height and a velocity of 20 m/s upwards at time zero), until only one variable remains unknown. Optionally, enter one or two initial guesses for the unknown like **5 time 10 time**. Set the display format and precision unless done before:

DISP FIX 1.

Now, press the softkey for the unknown **time** once more (but now without any numeric input heading), and your *WP 43S* will solve the equation for this variable and return its value in **X**:



corresponding to 5.8 s until your ball passes said point. Note entering the known values and guesses disabled *automatic stack lifting*.

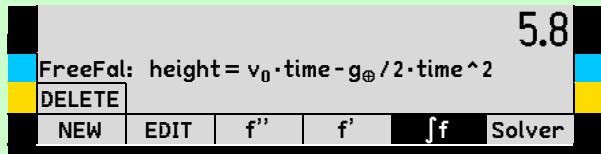
Another **example** (from the *HP-27S Owner's Manual* of 1988):

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 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 previous example):

Enter the equation for radioactive decay:

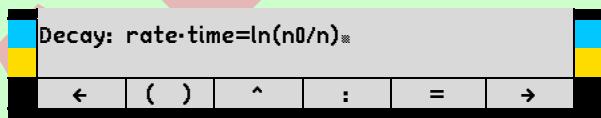
EXIT



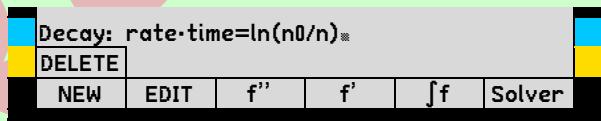
NEW



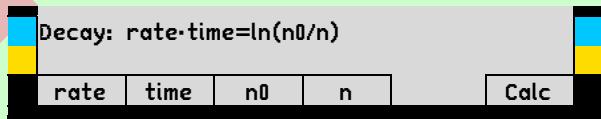
D ▾ E C A Y : RATE X TIME = LN (N 0) / N)



ENTER↑



Solver



1.2 E +/- 4 rate 15.3 n0 10.9 n time

time = 2 825.8

This is the computed age of this tree in years. – Now, calculate the half-life of ¹⁴C, that is, the time required for half the material present to decay:

2 n0 1 n time

time = 5 776.2

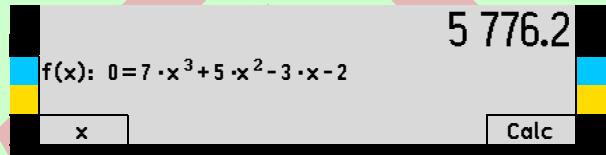
Good guess! Meanwhile, the half-life of ^{14}C is known to be 5 730 years.

One more **example:**

Find the roots of $7x^3 + 5x^2 - 3x - 2 = 0$.

Solution:

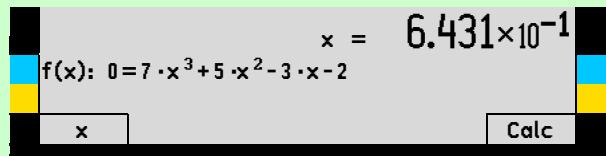
1. Enter the equation as demonstrated above.
2. Make this equation the *current equation* and press **Solver**. You will see:



3. Optionally enter one or two initial guesses for the unknown like **0 [x] 1 [x]**.
4. Set the display format and precision

DISP SCI [3]

5. Press the *softkey* **[x]** for the unknown once more (but now without any numeric input heading), and your WP 43S will solve the equation for this variable and return its value in **X**:



6. If you want to crosscheck you can enter

RCL [x] Calc returning

0.000

confirming the result of the *Solver*. Slightly greater x -values, e.g.

.65 **x** **Calc**, return positive values for $f(x)$, slightly smaller values, e.g.

.64 **x** **Calc**, return negative values for $f(x)$.

7. There may be one or two more roots:

- Enter two new initial guesses for the unknown like **-2 x 0 x**. Press the *softkey* for the unknown once more, and you will get:

$$x = -8.064 \times 10^{-1}$$

Slightly greater x -values, e.g. **-0.8**, return positive values for $f(x)$, slightly smaller values, e.g. **-0.81**, return negative values for $f(x)$. Thus, there must be one more root between the two roots already found.

- Enter two new initial guesses for the unknown like **-7 x .5 x**. Press the *softkey* for the unknown once more and you will get:

$$x = -5.510 \times 10^{-1}$$

Note that even a polynomial of same grade deviating just a bit (e.g. $7x^3 + 4.5x^2 - 3x - 2 = 0$) may feature one real root only.

Look into *Section 5* of the *HP-27S Owner's Manual* for more about interactive solving of equations.

The Interactive Solver for Expressions Stored in Programs

Instead of operating on an equation as described in previous chapters, your *WP 43S* can also solve an expression f stored in a program. Then, the procedure is as follows:

1. Write a program for f .
2. Press **ADV SOLVE**.
3. Enter values for all known variables of f .

4. Let your WP 43S compute the unknown variable.
5. Leave the Solver.

We will go through this step by step:

1. Write a program for f .
 - It must 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, proceeding as follows is recommended:
 From the second step of this program on, menu variables shall be declared using MVAR instructions (cf. p. 209) covering all variables of f . The subsequent body of the routine shall evaluate f recalling these variables. For a Solver routine, the original expression shall be rewritten that $f = 0$ is fulfilled.

Example:

Let's return to the equation we dealt with in the last two chapters:

$\text{height} = v_0 \cdot \text{time} - g_{\oplus} / 2 \cdot \text{time}^2$
 It is easily rewritten:

$$v_0 \cdot \text{time} - g_{\oplus} / 2 \cdot \text{time}^2 - \text{height} = 0$$

So the required program might look like this:

```
LBL 'FreeF'
  MVAR 'height'
  MVAR 'v0'
  MVAR 'time'
  # g⊕
  -2
  /
  RCLx 'time'
  RCL+ 'v0'
  RCLx 'time'
  RCL- 'height'
  RTN
```

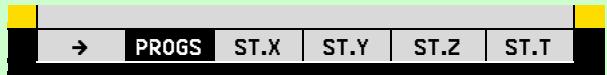
take this out of CNST.

now we have got f .

2. Press **ADV**. You will see:



Choose **SOLVE**. You will get (as expected from p. 197):



Press **PROGS** 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) one or two guesses for the unknown.

In our **example**, we may just take the values we know from above:

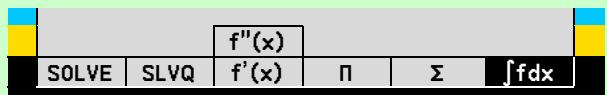
-50 **height**
20 **v₀**
5 **time** **10** **time**

4. Let your *WP 43S* compute the unknown variable.

Press **time** once more but without a heading numeric entry. Your *WP 43S* will return **time = 5.8** as you may have expected (cf. p. 227).

5. Leave the *Solver*.

Pressing **EXIT** will return to the top view of ADV.



Using the Solver in a Program

For using the *Solver* in programs, it has to be told what you did tell it interactively in the examples of previous chapters. Thus, when you press **ADV** in *PEM*, you will see a slightly different menu than the one you have seen overleaf:

PGMSLV		f''(x)	PGMINT		
SOLVE	SLVQ	f'(x)	Π	Σ	$\int f dx$

PGMSLV is for specifying the program calculating f . It must be found in your program before SOLVE is called.

Furthermore, define the necessary variables in advance and load them with the known values using STO. Eventually, the unknown variable must be specified calling SOLVE.

Example:

Let's return to the equation we dealt with in the last two chapters. So the required program for f might look like this (like the previous program but without the MVAR steps):

```
LBL 'FreeFp'  
# g⊕  
-2  
/  
RCLx 'time'  
RCL+ 'v0'  
RCLx 'time'  
RCL- 'height'  
RTN
```

take this out of CNST.

now we have got f .

The program one level above could contain a section looking 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 **SP**), fill the variables of the equation to be solved, e.g. with the start values known from above:

-50 [STO] height

20 [STO] v₀

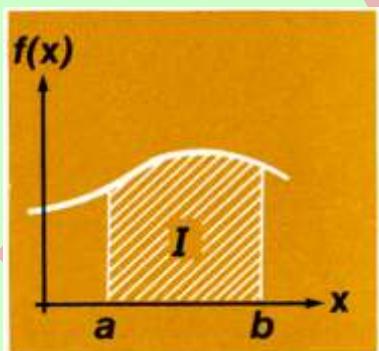
Option: Fill the unknown with a first guess, e.g. with 5 as we specified above (a second guess will be taken from X):

5 [STO] time

Call SP via [XEQ] PROGS SP and you will see time = 5.8 (as expected from p. 227).

Eventually turn to Part 3, Section 12 of the *HP-42S Owner's Manual*. Refer to the *HP-34C OH and Programming Guide* (Section 8 and Appendix A) or the *HP-15C OH* (Section 13 and Appendix D) for more information about automatic root finding and some caveats.

Numeric Integration of Equations



The command \int lets your WP 43S compute definite integrals numerically.

Example:

Let's compute the *Bessel function* of first kind and order 0. This function can be written as

$$J_0(x)$$

This is calculated in *radians*, thus enter [MODE] RAD and press [EQN] :

FreeFa: height = v ₀ · time - g ₀ / 2 · time ²	5.8				
DELETE					
NEW	EDIT	f''	f'	$\int f$	Solver

This function is not in the equation list yet.¹⁵⁵ So, press **NEW** and start entering the integrand:

I B E S S :

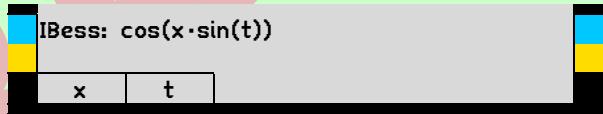


Continue with **C O S** () **X** **S I N** () **T**



Close and store this function by pressing **ENTER↑**. The *menu* will return to the previous one.

Then press **Jf**. Your *WP 43S* will check the current equation (see pp. 224ff) for syntax errors (missing operators, misspelled functions, illegal variable names, etc.).¹⁵⁶ It will then return a *menu* of all applicable variables:



You can enter values for any variables (i.e. integration constants) you already know by pressing the respective *softkeys* now, e.g.

2 **x**

(For recalling such an integration constant, just press **RCL VARS** before the respective *softkey*.)

Then select the variable of integration by simply pressing **t** here (there must not be any numeric input heading **t**). The *menu* will change:

¹⁵⁵ Actually, a function $J_y(x)$ is supplied with your *WP 43S* directly returning values of the Bessel function of first kind and order y . You can compare the results if you like.

¹⁵⁶ You will have noticed already that **IBess** is not an equation but just one side of it. To keep the system lean, such functions are listed under **EQN** nevertheless, but cannot be evaluated by the Solver, of course.



Even your *WP 43S* cannot 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 specified.

ACC is a *real number* that defines the relative error of the integration. With **ACC** = 0.001, for example, 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 $\downarrow\text{Lim}$ and $\uparrow\text{Lim}$.

We want to see the result with three decimals. Thus we enter

DSP 0 3

.001 ACC for the accuracy of computation,

0 **↓Lim** for the lower integration limit,

π **↑Lim** for the upper integration limit,

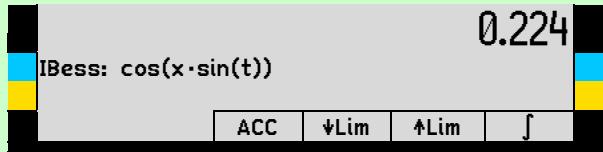
and start integrating by pressing \int . Your *WP 43S* will return:



Do not forget to divide this result by π to get the correct value for $J_0(2)$:

π **/** ¹⁵⁷

¹⁵⁷ Note that $\int \approx$ vanishes with **π** like every *temporary message* disappears with the next keystroke.



By entering other values for x and integrating again, you can get $J_0(x)$ at other locations easily.

Interactively Integrating Expressions Stored in Programs

Instead of operating on an 'equation' as described in previous chapter, your *WP 43S* can also integrate an expression f stored in a program. Then, the procedure is as follows:

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 *WP 43S* compute the definite integral specified.¹⁵⁸

We will go through this step by step:

1. Write a program for f :
 - It must 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.

It is recommended proceeding as follows: From the second step of this program on, menu variables shall be declared with MVAR

We could have included that division by π in our function **IBess**. We did not, however, since **IBess** is evaluated many times during the integration process; thus the fewer steps the integrand contains the faster the result can be returned.

¹⁵⁸ Note this follows closely the procedure as described for the Solver above.

instructions (cf. p. 209) covering all variables of f . The subsequent body of the routine shall evaluate f recalling these variables.

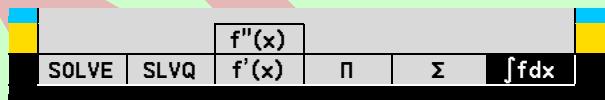
Example:

Let's return to the integrand we dealt with in the last chapter. So the required program for f might look like this:

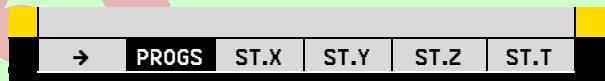
```
LBL 'IBessI'  
MVAR 'x'  
MVAR 't'  
RCL 't'  
sin  
RCLx 'x'  
cos  
RTN
```

now we have got f .

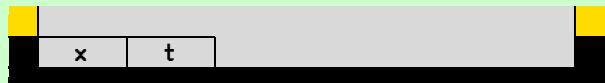
2. Press **ADV**. You will see:



Choose $\int f dx$. You will get (as expected from p. 197):

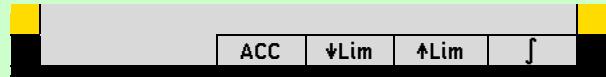


Press **PROGS** and pick the proper program for f (here **IBessI**). You will get the corresponding menu of variables, i.e. here:



3. Enter values for all known variables (integration constants) of f and select the variable of integration.

In our example, we may just take the values we know from above: **2** **x** **t**. So t will be the variable of integration. The menu will change:



We enter (just as we did in previous chapter) **.001 ACC 0 ↓Lim** **TU ↑Lim**.

4. Let your *WP 43S* compute the definite integral specified.

Press **∫** to integrate with all the parameters as chosen, and your *WP 43S* will return **∫ ≈ 0.704** as you have expected (cf. 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 did tell it interactively in the examples of the two previous chapters. Thus, when you press **ADV** in *PEM*, you will see a slightly different menu than the one you have seen above:



You shall define the necessary variables in advance and load them with the known values using **STO**. Then call the *menu* **∫fdx**. **PGMIN** is for specifying the program calculating **f**. It must be found in your program before the integration itself is called. And the integration limits as well as the requested accuracy shall be stored as well before integrating:



Eventually, the variable of integration must be specified calling **∫**.

Example:

Let's return to the integrand we dealt with in the last two chapters. So the required program for f might look like this:

```
LBL 'IBessP'  
RCL 't'  
sin  
RCLx 'x'  
cos  
RTN
```

now we have got f .

The program one level above could contain a section looking like this:

```
...  
PGMINT 'IBessP' specifying the function to be integrated.  
0  
STO 'vLim'  
# π  
STO '^Lim'  
0.001  
STO 'ACC'  
∫fd 't'  
VIEW ST.X integrate over time.  
...  
display the solution.
```

Before starting this program (let's call it **InP**), fill the variables staying constant under integration, e.g. with the start values known from above:

2 **STO** **x**.

Call **InP** via **XEQ** **InP** and you will see $\int \approx 0.704$ as you may have expected (cf. previous two chapters). Divide by π to get the value for $J_0(2)$ as above.

Eventually turn to Part 3, Section 13 of the *HP-42S Owner's Manual*. Refer to the *HP-34C OH and Programming Guide (Section 9 and Appendix B)* or the *HP-15C OH (Section 14 and Appendix E)* for more information about automatic integration and some caveats.

Differentiating Equations

There are two commands provided returning the values of the first two derivatives of the function $f(x)$ at position x . This function $f(x)$ can be specified in an equation.

$f'(x)$ returns the first derivative. For computing it, ...

1. $f'(x)$ will first look for a user routine labeled ' δx ' (or ' δX ', ' Δx ', or ' ΔX ', in this order), returning a fixed step size dx in X . If that routine is not defined, $dx = 0.1$ is set for default.
2. Then, $f'(x)$ fills all *stack registers* with x and calls $f(x)$. It will evaluate $f(x)$ at ten points equally spaced in the interval $x \pm 5 dx$ (if you expect any irregularities within this interval, change dx to exclude them).
3. On return, the first derivative will be in *stack register* X , while Y , Z , and T will be clear and the position x will be in L .

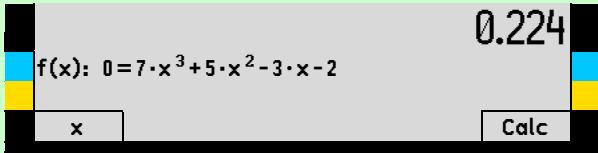
Example:

Take the equation $f(x) = 7x^3 + 5x^2 - 3x - 2$ again (you used it on pp. 227f for solving). Instead of checking two function values left and right of the root you could check the slope $f'(x)$ at the root just once.

Solution:

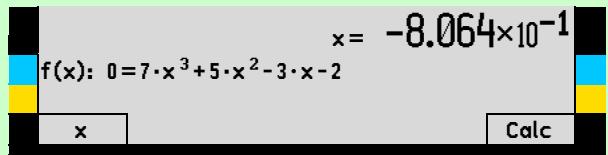
You have $f(x)$ in EQN already. For each of the three roots found, calculate the root first, then the first derivative of $f(x)$ at that point:

1. Press **EQN**, make $f(x)$ the *current equation*, and press **Solver**. You will see then:

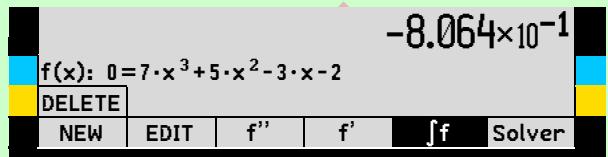


2. Find the first (leftmost) root as shown above:

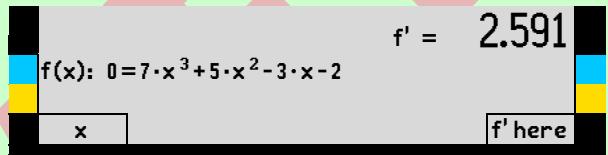
-2 **x** -1 **x** **x**



3. Pressing **EXIT** returns to the top view of EQN:



4. Press **f'** :



Note that **f'** returned the value of the first derivative at this very location immediately since **f(x)** features only one variable; else **f'** would have needed your input via the *softkeys* displayed and pressing **f' here** thereafter.

So the slope of **f(x)** at **x = -0.8064** is **2.591**. Get the slopes at the two other root positions the same way:

5. **EXIT** returns to the top view of EQN as in step 3.

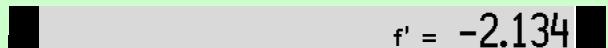
6. **Solver**

Find the second root of **f(x) = -1** **x** **0** **x** **x**



EXIT returns to the top view of EQN as in step 3.

f'



[EXIT] returns to the top view of EQN as in step 3.

7. Solver

Find the third (rightmost) root of $f(x)$: 0 1

$$x = 6.431 \times 10^{-1}$$

[EXIT] returns to the top view of EQN as in step 3.

f'

$$f' = 1.211 \times 10^1$$

So the slope of $f(x)$ at $x = -0.8064$ is 2.591, at $x = -0.5510$ it is -2.134, and at $x = 0.6431$ it is 12.11; the sequence of slopes is positive, negative, and positive as expected.

$f''(x)$ works in full analogy, computing the second derivative of $f(x)$.

Interactively Differentiating Expressions Stored in Programs

Instead of operating on an 'equation' as described in previous chapter, your *WP 43S* can also derive an expression $f(x)$ stored in a program. Then, the procedure is as follows:

1. Write a program for $f(x)$. It must begin with a global label. For interactive derivation, proceeding as follows is recommended: From the second step of this program on, menu variables shall be declared with MVAR instructions (cf. p. 209) covering all variables of $f(x)$. The subsequent body of the routine shall evaluate $f(x)$ recalling these variables.
2. Optionally, write another program labeled ' δx ' (see p. 241).
3. Enter values for all known variables (derivation constants) of $f(x)$. Put the requested location (where you want to know the derivative) into X .

4. Press **ADV**. You will get:

		f''(x)				
SOLVE	SLVQ	f'(x)	Π	Σ	$\int f dx$	

5. Press **f'(x)** or **f''(x)**. You will get as well:

	→	PROGS	ST.X	ST.Y	ST.Z	ST.T	
--	----------	--------------	------	------	------	------	--

6. Press **PROGS** to pick the label of the program containing the function **f(x)** (or enter its label directly as described on p. 196).
7. Let your *WP 43S* compute the first or second derivative at the location specified in *x*.¹⁵⁹

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.

When you press **ADV** in *PEM*, you will see a slightly different menu than the one you have seen above:

	PGMSLV	f''(x)				
SOLVE	SLVQ	f'(x)	Π	Σ	$\int f dx$	

Press **f'(x)** (or **f''(x)**) specifying the label of your program calculating **f(x)**. – or pick it from the list as explained in steps 5 and 6 above. Your *WP 43S* will compute the requested derivative for you in this program step.

¹⁵⁹ Note this follows loosely the procedures as described for the Solver and Integrator above.

Nesting Advanced Operations

You can nest SLV, \int , $f'(x)$, $f''(x)$, Σ , and Π in routines to any depth as far as memory allows and your patience and power last.

Example:

It is observed that light is diffracted when passing through circular holes. The effect is most obvious when using laser light. Its intensity behind the hole is

$$I(r) = I_0 \times \left(\frac{J_1(2\pi r)}{\pi r}\right)^2 \quad \text{with} \quad J_1(x) = \frac{1}{\pi} \int_0^{\pi} \cos[t - x \sin(t)] dt$$

being the *Bessel function of the first kind of order 1* (cf. p. 234).

Find the first three roots of the intensity (i.e. the radii where no light will be observed).

Solution:

1. Write a little program for the internal calculation of the integrand $f(t) = \cos[t - x \sin(t)]$:

```
LBL 'J1'  
    sin  
    RCLx 00  
    -  
    cos  
    RTN
```

$\sin(t)$

The entire stack is loaded with the integration variable t , so $x = 2\pi r$ (see below) must be recalled for calculating $x \sin(t)$

$t - x \sin(t)$

$\cos[t - x \sin(t)]$

2. Write a second little program for the internal calculation of the intensity $I(r)$. Note that just the parenthesis of the formula above must be evaluated since I_0 is a constant. And **ADV** displays the following when called in PEM:

PGMSLV	$f''(x)$				
SOLVE	SLVQ	$f'(x)$	Π	Σ	$\int f dx$

```

LBL 'I'
# π
x
2
x
STO 00
PGMINT 'J1'
0
STO '↓Lim'
# π
STO '↑Lim'
0.001
STO 'ACC'
∫fd ST.X
RCL 00
/
2
x
RTN

```

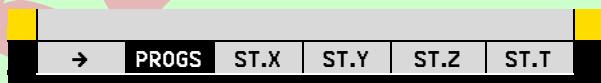
stores $2\pi r$ for later use, also in integration.
specifies the program of the integrand.¹⁶⁰

computes $\pi \times J_1(2\pi r)$.
The stack just contained the integration results before.

$J_1(2\pi r) / (2\pi r)$

$J_1(2\pi r) / (\pi r)$

- Enter **DISPL SCI** [3]
MODE RAD
0.1 [**ENTER↑**] 1
ADV SOLVE



- Pick **I** from **PROGS** or key in **α I** [**ENTER↑**].
 You will get 6.098×10^{-1} after some time.¹⁶¹

¹⁶⁰ Note you shall press the rightmost softkey to get the *menu* for accessing PGMINT etc.

¹⁶¹ i.e. after some 25s using the WP 43S emulator (cf. ReM, App. I) on a PC running Windows 7. It will take xxx minutes on your calculator. Nesting advanced operations may require very large amounts of calculations to be performed! We recommend connecting your WP 43S to an USB outlet for external power supply when dealing with such applications.

Note the Solver was not started at **0** since that would cause an error when dividing by $2\pi r$.

5. Enter **1** **[ENTER↑]** **1.5** **SOLVE** **α** **[ENTER↑]** and you will get
1.117.

6. Enter **1.5** **[ENTER↑]** **2** **SOLVE** **α** **[ENTER↑]** and you will get
1.619.

You will find further instructions and examples in *HP-42S RPN Scientific Programming Examples and Techniques*. Despite the title of this manual, it also contains significant material about the *Solver*, integration, matrices, and statistics.

DRAFT

SECTION 5: TWO BROWSERS, TWO APPLICATIONS, AND TWO SPECIAL MENUS

There are two helpful *browsers* featured for checking memory, *registers*, and *flags* (see below). And there are two very useful applications: a timer (see pp. 250f) and TVM (see pp. 253f). Furthermore, two special *menus* will ease your path in special fields and particular regions.

The Browsers RBR and STATUS

These two *browsers* may be called in all modes except *alpha input*. Some special keys and special rules apply within these *browsers* as explained on the two pages following. **EXIT** works as in *menus*, however, leaving the respective *browser*; and *browser* (like *menu*) calls cannot be programmed.

Keys to press	Contents and special remarks
g [FLAGS] STATUS	Displays the amount of free memory available and the status of all user accessible flags (inspired by STATUS on HP-16C and WP 34S) in two views:

and:

2015-08-05 23:02	RPL [®]	/max. 2:64	4
Global flag status (continued):			
80 81 82 83 84 85 86 87 88 89			
90 91 92 93 94 95 96 97 98 99			
100 101 102 103t 104 105o 106c 107d 108 109i			
110 111			
64 local registers are allocated.			
Local flag status:			
0 1	2 3	4 5	6 7
10 11	12 13	14 15	8 9

Flags set are displayed inverted. The last four rows of the second view will only be displayed if *local registers* are allocated at all.

▲ and ▼ toggle between these two views.

EXIT leaves STATUS.

g RBR

Browses all currently allocated *registers* showing their contents. RBR operates in TAM (see pp. 55ff). The first screen you see covers *registers X* through *I* (their contents will deviate on your screen – note numeric contents are shown explicitly in the display format currently set as long as they are individual numbers while strings and matrices are abbreviated. Within the range of lettered *registers*, every fourth *register* is displayed overlined to guide the eye:

2015-08-05 23:15	RPL [®]	/max. 2:64	4
I:	0.000 0		
L:	1.602 2×10 ⁻¹⁹		
D:	'This calculator is made in...		
C:	8AFE49 ₁₆		
B:	12 ⁴⁵ / ₃₇₈₉		
A:	0.000 0		
T:	0.000 0		
Z:	[6×2 C matrix]		
Y:	0.000 0		
X:	6.022 1×10 ²³		



goes up the *stack*, continuing with the remaining lettered *registers*, then with **R00**, **R01**, etc. as shown below. For **R00** ... **R99**, every fifth *register* is displayed overlined to guide the eye. After **R99**, **X** will be shown again:

2015-08-05 23:17 RLE [®] /max. 2:64	
R07:	0.000 0
R06:	'The train arrives at...
R05:	1010 1101 1000 0110 1011 ₂
R04:	0.000 0
R03:	0.000 0
R02:	[3x3 C matrix]
R01:	[3x3 Matrix]
R00:	[4x1 C matrix]
K:	57.000 0
J:	6.000 0



browses the *registers* going down from **R99** (if starting with the screen on p. 249) to **R00**; then continues with **K**, **J**, down to **X**. After **X**, **R99** will be shown again.



turns to *local registers* if allocated, starting with **R.00**. Then, and browse *local registers* up and down until another returns to the first screen as shown on p. 249. Else (i.e. if no local registers are allocated) directly returns to the first screen as shown on p. 249.

nn or a

Input of any legal letter or two-digit number jumps to the corresponding *register*.

RCL

recalls the *register* displayed in the lowest row. In *PEM*, these keys enter a corresponding step **RCL ...**

EXIT

leaves RBR.

No other keys will work within RBR and STATUS. And both browsers are not programmable.

The Timer Application

Your WP 43S provides a timer following the one of the HP-55.¹⁶² Start it by pressing **TIMER**. Then the top numeric row will be replaced by



0:00:00.0 [00]

or – depending on the radix mark setting –

0:00:00,0 [00]

unless the timer was running before already (then the accumulated run time will be indicated instead of zero here). In either case the *menu section* will change to this:



Within TIMER, the following keys will work:

Key	Remarks
R/S	starts or stops the timer without changing its value.
RESET	resets the timer to zero without changing its status (running or stopped). Deletes the total time if this is displayed. – Note this is <u>not</u> the global RESET command.
0.1s?	toggles displaying tenths of seconds (default is 'display').

¹⁶² This application works exactly as in the WP 34S but the display differs. With respect to the HP-55, there are two deviations:

1. Your WP 43S will not take the content of X at the time calling TIMER as start time of the timer; start times are supported by RCL within TIMER here instead.
2. Your WP 43S will display tenths instead of hundredth of seconds. Reaction times of the hardware do not allow for more precision anyway.

Key	Remarks
[ADD]	adds the present timer value to the statistics <i>registers</i> . This allows for computing e.g. the <i>arithmetic mean</i> and <i>standard deviation</i> of lap times after leaving TIMER.
[n][n]	sets the <i>current register address</i> (CRA, startup default is 0). The CRA will be displayed between rectangular brackets as shown here: ¹⁶³ 27:31:55.6 [01]
[ENTER↑]	stores the present timer value in the <i>current register</i> at execution time without changing the timer status or value. Then increments the CRA by one.
[RCL] nn	recalls <i>rnn</i> without changing the status of the timer. The value recalled may be used e.g. as start time for further incrementing.
[▲] or [▼]	increments or decrements the CRA by one, respectively.
[]	combines [ENTER↑] + [] in one keystroke, but the <u>total</u> time since the last explicit press of [] or [RESET] is shown and updated like: 21:04:15^T 0:02:29 [06] or 10:02:31.7^T 0:00:49.6 [11] [] allows for recording lap times, for example. Note the total time is volatile – it will disappear without a trace when [] or [RESET] is pressed alone.
[+]	Combines all the functionality of [ADD] , [ENTER↑] , and [] in one keystroke. This allows for recording lap times and total time for later offline analysis.
[EXIT]	leaves the application. The time indicated in the top numeric row will vanish from the screen. Unless already stopped, however, the timer continues incrementing in the background (indicated by [S] in the <i>status bar</i>) until

¹⁶³ On the HP-55, input of a single digit was sufficient for storing, since only 10 *registers* were featured for this purpose there. Furthermore, there was no automatic address increment. – Attempts to specify a CRA <0 or >99 will be blocked.

Key	Remarks
	<p>a) stopped explicitly by R/S within TIMER or ...</p> <p>b) your <i>WP 43S</i> is turned off by you. Note it will <u>not turn off automatically</u> with the timer running. A reliable power supply is recommended in such cases.</p>

☞ No other keys will work within TIMER – so e.g. for subtracting split times you have to leave this application.

☞ TIMER is not programmable.

The Time Value of Money (TVM)

TVM is a financial application (thus included in FIN) 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 (**i%/a**), the required payment per period (**PMT**), number of periods per annum (**per/a**), and the total number of payment periods (**n_{PER}**). This kind of financial problems will often occur to technical people, too, so we included TVM on your scientific calculator.



For your information, the general formula for such problems reads

$$FV = PV \times (1 + i)^{n_{PER}} + (1 + i \times p) \frac{PMT}{i} \times [1 - (1 + i)^{-n_{PER}}]$$

with the deduced parameter $i = \frac{i\%/\text{period}}{100} = \frac{i\%/\text{year}}{100} / \frac{\text{years}}{\text{year}}$

and the binary switch value p :

- If payments occur at the end of each period then $p = 0$.
- If payments occur at the beginning of each period then $p = 1$.

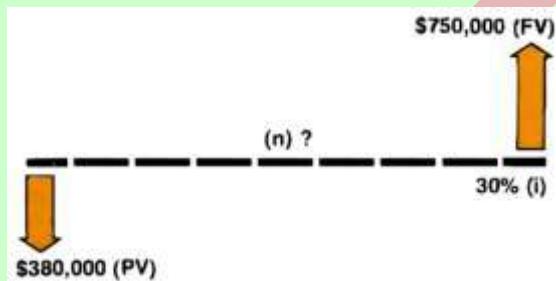
The application TVM uses two *menu* entries for p – choose **End** for payments at the end of each period, **Begin** for payments at the beginning of each period.

This application uses the convention that cash outlays are input as negative, and cash incomes are input as positive.

The present value **PV** always occurs at the beginning of the first 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 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 (from the HP-27 OH, like all following examples in this chapter; enjoy the amounts and interest rates of a time long ago):



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?

Solution:

DISP **FIX** **2** This will suffice. Then all you have to do is keying in the known parameters and boundary conditions:

FIN **TVM** **Begin**

Begin					End	
n_{PER}	$i\%/\alpha$	per/α	PV	PMT	FV	

380000 **PV**

380 000.00 present value,

30 **i%/a**

30.00 % interest rate per year,

0 **PMT**

0.00 no payments,

1 **per/a**

1.00 default.

750000 **FV**

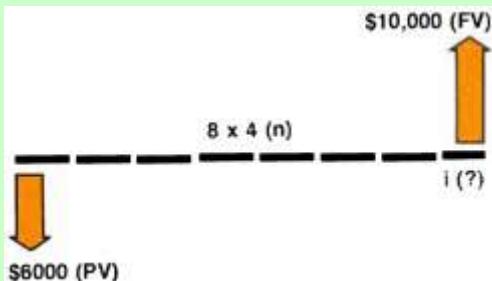
750 000.00 Now, how long does it take to

n_{PER}

2.59 reach this future value?

years.

Example for finding the necessary interest rate for compounded amounts:



6000 PV
4 per/a
10000 FV
8 ENTER↑ 4 × nPER
i%/a

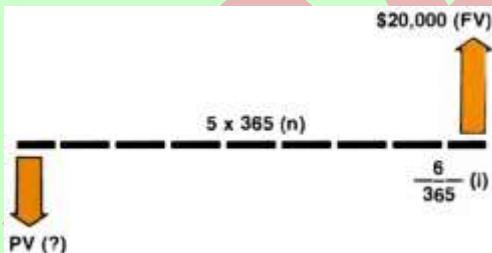
What annual interest rate must be obtained to amass \$10 000 in 8 years on an investment of \$6 000, with quarterly compounding?

(Continue keeping the settings of previous example.)

Solution:

6 000.00	present value,
4.00	quarters,
10 000.00	future value,
32.00	periods. Now, we need...
6.44	% interest rate per year to reach this.

Example for finding the present value of a compounded amount:



20000 FV
6 i%/a
365 per/a
5 ENTER↑ 365 × nPER
PV

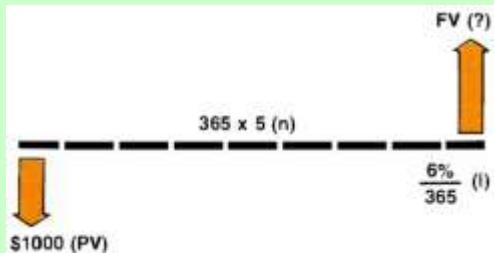
In 5 years when your son starts college, you will need \$20 000. You deposit a lump sum in a certificate account that earns 6% compounded daily. How much do you need to deposit today to reach that goal?

(Dream on with the settings of previous example.)

Solution:

20 000.00	future value,
6.00	% interest rate per year,
365.00	days per year,
1 825.00	periods. Now, we need...
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 \$1 000 at that time, and now wants to know his present balance and the total accrued interest after all this time.

(Continue dreaming ...)

Solution:

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 ENTER↑ 365 x nPER

1 825.00 periods. Now, Sam has...

FV

1 349.83 present balance meaning...

RCL VARS PV -

349.83 accrued interest.

Nominal interest rate converted to effective rate:

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 keeping the settings of previous example.)

Solution:

100 PV

100.00 base value,

12 i%/a

12.00 % nominal rate per year,

4 per/a

4.00 quarters per year,

4 nPER

4.00 periods;

FV

112.55

RCL VARS PV -

12.55 % effective interest rate.

Turn to App. 3 for more applications of TVM (annuities, savings, etc.), starting on p. 296.

Constants

Your WP 43S contains a *catalog* of 80 physical, astronomical, and mathematical constants, sorted alphabetically in CNST. Press **CNST** and the *menu section* will change to:



G	G_0	G_C	g_e	GM_{\oplus}	g_{\oplus}	
c_1	c_2	e	e_E	F_{α}	F_{δ}	
1/2	a	a_0	a_M	a_{\oplus}	c	

Besides by browsing with \blacktriangle and \blacktriangledown , you can access the contents of CNST most easily using the alphabetical method demonstrated in the *ReM, Section 2*.

Names of astronomical and mathematical constants are printed on colored background in the table starting overleaf. The unit of each physical constant is printed on light background if the numeric value of said constant is exactly defined or almost exactly known – the darker the background, the less precisely we know it.¹⁶⁴ Some ratios were added where their uncertainty is significantly better than the one of their components.

Name	Unit ¹⁶⁵	Remarks
1/2	1	Trivial but helpful constant for some iterations.
a	d	<i>Gregorian year</i>
a_0	m	<i>Bohr radius</i>
a_M	m	Semi-major axis of the Moon's orbit around the earth ≈ 1.3 <i>light seconds</i> .
a_{\oplus}	m	Semi-major axis of the Earth's orbit around the sun. Within its uncertainty, a_{\oplus} equals 1 <i>AU</i> (<i>astronomic unit</i>) ≈ 500 <i>light seconds</i> .

¹⁶⁴ For all these constants, their numeric values and uncertainties, underlying formulas and relations are printed in the respective chapter in *Section 2* of the *ReM*.

¹⁶⁵ Find all symbols explained in the chapter about unit conversions in the *ReM, Sect. 2*.

Name	Unit ¹⁶⁵	Remarks
c	m/s	Speed of light in vacuum
c_1	m^2W	First radiation constant
c_2	m K	Second radiation constant
e	C	Elementary charge
e_E	1	<i>Euler's e.</i>
e/m_e	C/kg	Electron charge to mass ratio
F	C/mol	<i>Faraday constant</i>
F_α	1	<i>Feigenbaum's α and δ</i>
F_δ		
G	$\text{m}^3/\text{kg s}^2$	<i>Newtonian constant of gravitation</i> ; also known as γ from other authors. See GM_{\oplus} below for a more precise value.
G_0	$1/\Omega$	Conductance quantum
G_c	1	<i>Catalan's constant</i>
g_e	1	<i>Landé's electron g-factor</i>
GM_{\oplus}	m^3/s^2	<i>Newtonian constant of gravitation times the Earth's mass with its atmosphere included (according to the Earth model WGS84 – see the ReM for more information)</i>
g_{\oplus}	m/s^2	Standard earth acceleration
h	J s	<i>Planck constant</i>
\hbar		So-called ' <i>Dirac constant</i> '
k	J/K	<i>Boltzmann constant</i>
K_J	Hz/V	<i>Josephson constant</i>
l_p	m	<i>Planck length</i>
m_e	kg	Electron mass

Name	Unit ¹⁶⁵	Remarks
M_M	kg	Mass of the Moon
m_n	kg	Neutron mass
m_n/m_p	1	Neutron to proton mass ratio
m_p	kg	Proton mass
M_p	kg	<i>Planck</i> 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	Muon mass
M_\odot	kg	Mass of the Sun
M_\oplus	kg	Mass of the Earth. See GM_\oplus above for a preciser value.
N_A	$1/\text{mol}$	Avogadro's number
NaN		"Not a Number", i.e. $\pm\infty \times 0$ or $\ln(x)$ for $x \leq 0$ or $\tan(90^\circ)$ unless in complex domain, for example. So NaN covers poles as well as regions where a function result is not defined at all. Note that infinities, however, are considered numeric in your WP 43S (see the end of this table). Non-numeric results will lead to an error message thrown – unless flag D is set.
p_0	Pa	Standard atmospheric pressure
R	$J/\text{mol K}$	Molar gas constant
r_e	m	Classical electron radius
R_K	Ω	<i>Von Klitzing</i> constant
R_M	m	Mean radius of the Moon
R_∞	$1/m$	<i>Rydberg</i> constant
R_\odot	m	Mean radius of the sun
R_\oplus	m	Mean radius of the Earth

Name	Unit ¹⁶⁵	Remarks
S_a	m	Semi-major axis
S_b	m	Semi-minor axis
Se^2	1	First eccentricity squared
Se'^2	1	Second eccentricity squared
Sf^{-1}	1	Flattening parameter
T_0	K	= 0°C, standard temperature
t_p	s	<i>Planck time</i>
T_p	K	<i>Planck temperature</i>
V_m	m^3/mol	Molar volume of an ideal gas at standard conditions $\approx 22.4 \text{ l/mol}$
Z_0	Ω	Characteristic impedance of vacuum
α	1	Fine-structure constant $\approx 1/137$
γ	$m^3/kg \text{ s}^2$	Newtonian constant of gravitation; also known as G from other authors. See GM_{\oplus} below for a more precise value.
γ_{EM}	1	<i>Euler-Mascheroni constant</i>
γ_p	$1/\text{s T}$	Proton gyromagnetic ratio
ϵ_0	F/m	Electric constant or vacuum permittivity
λ_c	m	<i>Compton wavelengths of the electron, neutron, and proton, respectively</i>
λ_{Cn}		
λ_{Cp}		
μ_0	H/m	Magnetic constant or vacuum permeability
μ_B	J/T	<i>Bohr magneton</i>
μ_e		Electron magnetic moment
μ_e/μ_B	1	Ratio of electron magnetic moment to <i>Bohr's magneton</i>

Name	Unit ¹⁶⁵	Remarks
μ_n		Neutron magnetic moment
μ_p	J/T	Proton magnetic moment
μ_u		Nuclear magneton
μ_μ		Muon magnetic moment
σ_B	$W/m^2 K^4$	Stefan-Boltzmann constant
ϕ	1	Golden ratio
Φ_0	Wb	Magnetic flux quantum
ω	rad/s	Angular velocity of the Earth according to WGS84 (see the ReM)
$-\infty$	1	May the Lord of Mathematics forgive us calling these two constants! Note both are counted as numeric values in your WP 43S, however.
∞		
#	1	Allows for inserting an integer $0 \leq n \leq 255$ in a single program step

Employ the constants stored here for further useful equivalences, like calculating the wavelength from the frequency of electromagnetic radiation via $\lambda = c/f$, expressing *joules* in *electron-volts* ($1 J = 1 As V = 1 eV/e \approx 6.24 \times 10^{18} eV = 6.24 \times 10^9 GeV$), etc.

Another **example**:

If you want to see the energy equivalent (in *electron-volts*) of one of the small masses given below, multiply its mass (as given in kg) by $c^2/e \approx 5.610 \times 10^{35} \frac{m^2}{As^3}$ and you are done: m_e corresponds to 511.0 *keV*, m_p to 938.3 *MeV*, etc.

One more **final example**:

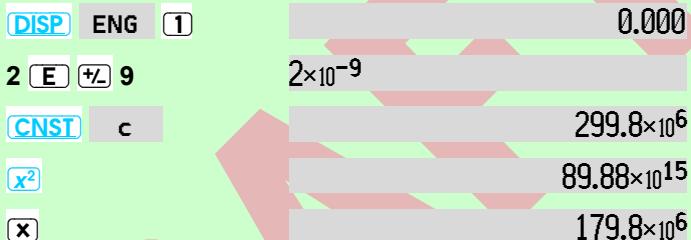
Assume American scientists will succeed in producing a tiny bit of anti-matter in a high-tech laboratory one day – e.g. 1 μg of anti-hydrogen,

carefully stored isolated in ultra-high vacuum. Most probably American power transmission lines will still look like they do today, however, since this is well-tried American (first) standard.

In the unlikely event of slightly extreme weather conditions (fostered by non-existing climatic change), an accidental blackout may happen for some days; and a subsequent vacuum breakdown will let atmospheric gas leak into the vacuum vessel where it will interact with the anti-matter and annihilate immediately. How much energy will be released then?

Solution:

You only need the same tiny amount of (usual) matter, so $2 \mu\text{g}$ will annihilate in total within the vessel. $1 \mu\text{g}$ equals 10^{-9} kg . Enter:



... resulting in 180 MJ set free.¹⁶⁶ The odds are high this lab will need no cleaning anymore.

Though $1 \mu\text{g}$ of anti-matter requires e.g. $N_A/10^6$ atoms of anti-hydrogen (with N_A being Avogadro's number), i.e. 6×10^{17} atoms or 3×10^{17} molecules of this gas. Luckily, this amount is far from being produced in any lab for the time being. And proper UHV vessels show very low leak rates as well.

¹⁶⁶ For comparison, 1 kg of TNT releases 4.6 MJ. The official definition is some 10% less than this value for historical reasons. Anyway, 180 MJ are equivalent to some 40 kg TNT.

Unit Conversions

Your WP 43S features fourteen angular conversions stored in $\angle \rightarrow$ (as shown on p. 118) and 82 unit conversions in $\underline{U} \rightarrow$. The latter *menu* mainly provides means to convert local to common units and vice versa.¹⁶⁷



Also the constant T_o may be useful for converting centigrade temperatures to *kelvin*. It is found in CNST and is not repeated in U because it is only added or subtracted.

In an attempt to bring some order in that heap of units, U is structured like a tree. Press $\underline{U} \rightarrow$ and the *menu section* will change to:

	$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	$\text{s} \rightarrow \text{year}$		V:	A:	
E:	P:	year \rightarrow s	F&p:	m:	x:		

containing the labels of *submenus* for conversions of energy, power, force & pressure, mass, length, area, and volume units. The entire structure of U is shown overleaf (with the *menu* rows printed top down instead of bottom up following common reading habits). Some softkeys require more than six characters due to long unit names – then extra high *menu* rows will be displayed:

¹⁶⁷ The *SI* system of coherent units of measurement is agreed on internationally and adopted by almost all countries on this planet for long, as was mentioned above already. Thus, most of the material appearing in U will look quirky or obsolete for the overwhelming majority of mankind. Those units die hard, however, in some corners of this world (English is spoken in all of those).

Thus, U may also help you when you get caught in a time loop and happen to be thrown back into such an obstinate environment. For symmetry reasons, we think about including some traditional Indian and Chinese units in U, too.

U may also give you a slight idea of the mess we had in the world of measuring before going metric over 200 years ago. In the *ReM*, you find comprehensive explanations of all conversions provided. Without *Imperial* and *US-American* units, U would need twelve entries only.

							Remarks
<u>E:</u>	E:	P:	year \rightarrow s	F&p:	m:	x:	time, temperature and ratio conversions – the latter in an extra-high menu row
	$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	s \rightarrow year		V:	A:	
	power ratio \rightarrow dB	dB \rightarrow power ratio			field ratio \rightarrow dB	dB \rightarrow field ratio	
<u>E:</u>	cal \rightarrow J	J \rightarrow cal	Btu \rightarrow J	J \rightarrow Btu	Wh \rightarrow J	J \rightarrow Wh	units of energy
<u>P:</u>	hp _E \rightarrow W	W \rightarrow hp _E	hp _{UK} \rightarrow W	W \rightarrow hp _{UK}	hp _M \rightarrow W	W \rightarrow hp _M	units of power
<u>F&p:</u>	lbf \rightarrow N	N \rightarrow lbf	bar \rightarrow Pa	Pa \rightarrow bar	psi \rightarrow Pa	Pa \rightarrow psi	units of force and pressure
	in.Hg \rightarrow Pa	Pa \rightarrow in.Hg	torr \rightarrow Pa	Pa \rightarrow torr	atm \rightarrow Pa	Pa \rightarrow atm	
<u>m:</u>	lbs \rightarrow kg	kg \rightarrow lbs	cwt \rightarrow kg	kg \rightarrow cwt	oz \rightarrow kg	kg \rightarrow oz	units of mass
	stones \rightarrow kg	kg \rightarrow stones	short cwt \rightarrow kg	kg \rightarrow sh.cwt	tr.oz \rightarrow kg	kg \rightarrow tr.oz	
	tons \rightarrow kg	kg \rightarrow tons	short tons \rightarrow kg	kg \rightarrow short tons			
<u>X:</u>	au \rightarrow m	m \rightarrow au	ly \rightarrow m	m \rightarrow ly	pc \rightarrow m	m \rightarrow pc	units of length
	mi. \rightarrow m	m \rightarrow mi.	nmi. \rightarrow m	m \rightarrow nmi.	ft. \rightarrow m	m \rightarrow ft.	
	in. \rightarrow m	m \rightarrow in.	pt. \rightarrow m	m \rightarrow pt.	yd. \rightarrow m	m \rightarrow yd.	
					s:feet _{US} \rightarrow m	m \rightarrow s:feet _{US}	
<u>A:</u>	acres \rightarrow m ²	m ² \rightarrow acres			acres _{US} \rightarrow m ²	m ² \rightarrow acres _{US}	units of area
<u>V:</u>	gl _{UK} \rightarrow m ³	m ³ \rightarrow gl _{UK}	qt. \rightarrow m ³	m ³ \rightarrow qt.	gl _{US} \rightarrow m ³	m ³ \rightarrow gl _{US}	units of volume
	floz _{UK} \rightarrow m ³	m ³ \rightarrow floz _{UK}			floz _{US} \rightarrow m ³	m ³ \rightarrow floz _{US}	

Find out more about the various units mentioned in these conversions in Section 2 of the ReM.

You may, of course, combine conversions as you like:

Example 1:

For filling your tires with a maximum pressure of 30 psi the following will help you at gas stations in Europe and beyond:

30	U→F&p:	psi→Pa	returns	206.8 Pa.
		Pa→bar		2.1 bar.

Now you can set the filler and will not blow your tires.

Example 2:

Your friend tells you she has got 10 *cubic feet* of debris on her veranda after flooding. What does this mean in real units?

1	U→x:	ft. \rightarrow m	returns	0.305
3	y^x			0.028
10	x			0.283 m ³ .

OK, some work – but manageable.

Example 3:

A network switch is specified for 3 320 Btu/h. What?

3320	U→E:	Btu→J	returns	3 502 786. J/h.
------	-------------	-------	---------	-----------------

Since $1J = c_c Wh \Leftrightarrow 1J/h = c_c W$ applies, you can use

J→Wh for converting and get **973. W.**

This is almost 1 kW. Now you know what will be going on there.

Example 4:

In Section 2, there was an example ending with a box featuring a volume of $19\frac{11}{16}$ *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 *Imperial* units nowadays still?

1 x: in. \rightarrow m

returns

2.54×10^{-2}

3

16.387×10^{-6}

19 11 16

$322.620 \times 10^{-6} \text{ m}^3$.

Since $1 \text{ m}^3 = 1000 \text{ liter}$, this volume is almost $1/3 \text{ liter}$.

And for those enduring life on *British Imperial* islands or territories, you must ask for their location first. Then choose either v: $\text{m}^3 \rightarrow \text{floz}_{\text{UK}}$ or $\text{m}^3 \rightarrow \text{floz}_{\text{US}}$ and give them the respective result, i.e. 11.355 or 10.909, for what it is worth.

Example 5:

A celestial object moves with a velocity of 0.1 *parsec* per year. What does this mean in standard units? What is this in relation to the velocity of light? And how does it translate for air pilots?

.1 x: pc \rightarrow m

returns

$3.09 \times 10^{15} \text{ m}$.

returns to the top *view* of .

1 year \rightarrow s

returns

$9.78 \times 10^7 \text{ m/s}$.

pushes the result on the *stack*.

c

recalls $c =$

$3.00 \times 10^8 \text{ m/s}$.

returns

3.26×10^{-1} , i.e. 32.6% of c .

Additionally, since $1 \text{ h} = 3600 \text{ s}$ and $1 \text{ km} = 1000 \text{ m}$, you can see directly that

$$3.6 \frac{\text{km}}{\text{h}} = \frac{3600 \text{ m}}{3600 \text{ s}} = 1 \frac{\text{m}}{\text{s}}$$

Thus, 3.6

returns

$3.52 \times 10^8 \text{ km/h}$.

Furthermore and for what it is worth, this corresponds to

1000 x: $\text{m} \rightarrow \text{nmi}$.

i.e.

$1.90 \times 10^8 \text{ nmi/h}$ or *knots*.

Supported by your *WP 43S*, you will find further easy ways to produce whatever conversions you may need in addition personally.

In cases of emergencies of a particular kind, it may be helpful knowing *becquerel* (Bq) equals *hertz* in your Geiger-Müller counter, *gray* (Gy) is the unit for deposited or absorbed energy, and *sievert* (Sv) is *gray* times a radiation dependent dose conversion factor (≥ 1) for the damage caused in biological material including human bodies.¹⁶⁸ Remember also the example on pp. 85ff.

In this field, some outdated units may be found in older literature as well: Pour les fidèles amis de Madame *Marie Skłodowska Curie*, $1Ci = 3.7 \cdot 10^{10} \text{ Bq} = 3.7 \cdot 10^{10} \text{ decays/s}$. And for those admiring the very first Nobel laureate in physics, Mr. *Wilhelm Conrad Röntgen*, for discovering the X-rays (ruining his hands in those experiments since he could not know better yet), the charge generated by radiation in matter was measured by the unit *roentgen* ($1R = 2.58 \cdot 10^{-4} \text{ As/kg}$). A few decades ago, *rem* (i.e. *roentgen equivalent in men*) measured what *sievert* does today (1 rem = 10 mSv).

¹⁶⁸ Our warmest regards go to Belarus, Bikini, Canada, France, Japan, Russia, the Ukraine, the United Kingdom, and the USA (in alphabetical order) so far.

SECTION 6: CREATING YOUR VERY PERSONAL WP 43S

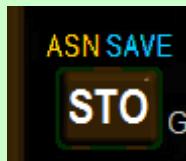
Your WP 43S is the first calculator worldwide allowing for full customizing of the user interface; i.e. you may assign an arbitrary function to almost any location, unshifted or shifted, on the keyboard or in a *menu*. *User mode* will then bring your personal assignments to the front, so you can interact via a user interface you designed yourself.

Even before doing such soft assignments, there are keyboard variants supported taking care of the demands of people living in different large ‘mathematical regions’. The keys for multiplication, division, and the radix mark may be labelled differently according to your preferences:

	Default	Alternative A	Alternative B
Division	/	:	÷
Multiplication	×	•	
Radix mark	.	,	

Note this manual generally uses the default key labels throughout its text unless there are readability reasons or ambiguities in notation making another choice necessary.

Use ASSIGN (**ASN**) for storing your soft assignments.
It allows for reassigning the full keyboard except **UM**.



In the explanations starting overleaf,

- stands for the *softkey* applicable (optionally headed by a *prefix*),
- **[key]** represents an arbitrary key of your WP 43S (optionally headed by a *prefix*),

- **menu** is the name of an arbitrary *menu* defined, either
 - entered directly by keying in the necessary characters terminated by **ENTER↑** or
 - picked from CATALOG'MENUS or
 - called from the keyboard, and
- **name** is the name of an arbitrary *item* defined. Remember an *item* may be an operation, function, digit, character, routine, variable, or a (*sub-*) *menu* defined. **The name of an item consists of up to seven characters and has to be unique.** Note upper and lower case letters are checked, so the system will regard **Menu01** and **MENU01** as being different names. Where a name is required, it may be either
 - entered directly by keying in the necessary characters terminated by **ENTER↑** or
 - picked from CATALOG or
 - called from the keyboard.

Just pressing **ENTER↑** where the operating system expects a name of an *item* is interpreted as input of an *empty name* and will delete the user assignment of the respective location.

Assigning Your Favourite Functions

Now here is how you can tailor the surface of your *WP 43S* according to your individual preferences:

[ASN] name [key]

will assign that named *item* to **[key]** in *user mode*. It will throw an error if said **name** does not exist.

Note that **[ASN] name ↗** will assign that named *item* to the respective position in the bottom *menu* row displayed at the time you press **↗**. In analogy, **[ASN] name f ↗** and **[ASN] name g ↗** assign said *item* to the corresponding position in a shifted *menu* row.

Each user assignment will hold until it is overwritten or **ENTER↑** is entered for **name** (see above).

Note all user assignments will be accessible in *user mode* only (see pp. 279f) – except the *items* assigned to top row of keys (see MyMenu and Mya below) which will be always displayed as long as no other *menu* is called.

Example 1:

Let's assign the statistical sample standard error to $\text{g} + \text{+/-}$ (this location is assigned to % in *startup default*). There are different ways to do this (specified here with all keystrokes necessary):

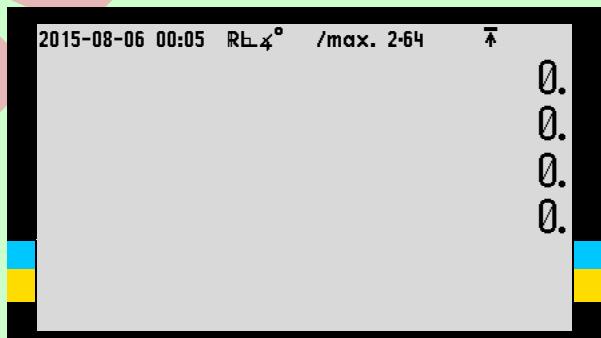
- 1) $f \text{ ASN } f \text{ } \alpha \text{ S } \downarrow f \text{ R } \downarrow \text{M } \text{ENTER} \uparrow \text{ g } \text{+/-}$
- 2) $f \text{ ASN } f \text{ CATALOG } \text{FCNS } \text{S } \text{M } s_m \text{ g } \text{+/-}$
This way will be shown step by step below.
- 3) $f \text{ ASN } \text{g } \text{STAT } s_m \text{ g } \text{+/-}$

On the other hand,

$f \text{ ASN } \text{ENTER} \uparrow \text{ g } \text{+/-}$

will reset g -shifted +/- to factory default as explained at the very end of last chapter.

We will demonstrate solution 2 step by step, starting with a clear *stack* for sake of clarity:



Only the *menu section* and the echo row will be shown in the following since all the action will take place there:

ASN

ASSIGN

0.

CATALOG

ASSIGN

0.

FCNS | PROGS | DIGITS | CHARS | VARS | MENUS

FCNS

ASSIGN

0.

ALL	AND	\arccos	arcosh	\arcsin	\arctan
2COMPL	$\sqrt[3]{x}$	ABS	$\text{ac} \rightarrow \text{ha}$	$\text{ac}_{\text{us}} \rightarrow \text{ha}$	AGM
$^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	$^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	10^x	1COMPL	$1/x$	2^x

This is the top view of the FCNS submenu in CATALOG. Now enter the first letter of the requested command:

S

ASSIGN

0.

SETEUR	SETIND	SETJPN	SETTIM	SETUK	SETUS
SDL	SDR	SEED	SEND	SETCHN	SETDAT
S	SAVE	SB	SCI	scw\kg	SCI\VR

Quickly entering the second letter helps significantly:

M

(if you find you waited too long before pressing **M**, just enter **SM** here instead)

ASSIGN

2

STOEL	STOIJ	STOP	STOS	sto _k g	STRI?
SSIZE4	SSIZE8	SSIZE?	STATUS	STO	STOCFG
s _m	SMODE?	s _{mw}	SOLV	SPEC?	SR

Now, press the bottom left *softkey* for the function to be assigned:

s_m

	ASSIGN s _m _						0.
	STOEL	STOIJ	STOP	STOS	sto _{kg}	STRI?	
	SSIZE4	SSIZE8	SSIZE?	STATUS	STO	STOCFG	
	s _m	SMODE?	s _{mw}	SOLV	SPEC?	SR	

g

	ASSIGN s _m g _m						0.
	STOEL	STOIJ	STOP	STOS	sto _{kg}	STRI?	
	SSIZE4	SSIZE8	SSIZE?	STATUS	STO	STOCFG	
	s _m	SMODE?	s _{mw}	SOLV	SPEC?	SR	

+/-

							0.
	STOEL	STOIJ	STOP	STOS	sto _{kg}	STRI?	
	SSIZE4	SSIZE8	SSIZE?	STATUS	STO	STOCFG	
	s _m	SMODE?	s _{mw}	SOLV	SPEC?	SR	

Note this last *menu view* will stay on screen until another *view* or *menu* is called or this *menu* is EXITed explicitly. And the function % will stay accessible in *user mode* via **CATALOG FCNS** ... – or when leaving *user mode* shortly.

Example 2:

Assign the weighted arithmetic mean to the first key in MyMenu (assume *startup default* state):

ASN

	ASSIGN _ _						0.

STAT

ASSIGN ...						0.
CLΣ	\bar{x}_g	σ	σ_p		σ_m	
Σ^-	\bar{x}_w	s_w	s_w		s_{mw}	
Σ^+	\bar{x}	s	s	SUM	s_m	

\bar{x}_w

ASSIGN \bar{x}_w ...						0.
CLΣ	\bar{x}_g	σ	σ_p		σ_m	
Σ^-	\bar{x}_w	s_w	s_w		s_{mw}	
Σ^+	\bar{x}	s	s	SUM	s_m	

[UM]

ASSIGN \bar{x}_w ...						0.
CLΣ	\bar{x}_g	σ	σ_p		σ_m	

Note that pressing **[UM]** will exit all *menu* being open at that time so MyMenu (which is empty still) can slip on the screen.

 (press the leftmost softkey)

... \bar{x}_w						0.
CLΣ	\bar{x}_g	σ	σ_p		σ_m	

Summarizing,

ASN **STAT** **\bar{x}_w** **[UM]** 

did this assignment.

Note that MyMenu will show up whenever any other *menu* is exited completely. It will remain on screen as long as no other *menu* is called unless your WP 43S is in *alpha input mode* (in AIM, Myq will appear when no other *menu* is called and will stay on screen until these

conditions will change). This applies regardless whether your *WP 43S* is in *user mode* or not (see below). Thus, filling MyMenu may well be the first step of customizing your *WP 43S*.

Creating Your Own Menus

[ASN] [UM] new_name [ENTER↑] will define a new user *menu*.

[ASN] [UM] turns on *alpha input mode* so you can immediately enter the new *menu* name (up to seven characters, no blanks, and the name must be unique).

Example:

To create a *menu FavFun* for your favourite functions, enter:

[ASN] [UM] F ▼ A V ▲ F ▼ U N [ENTER↑]

ASSIGN will throw an error if the new *menu* name turns out being defined already. – The new name will be inserted in CATALOG'MENUS. The new *menu* itself will be created with 18 blank entries – its size is fixed. You may fill it now.

Example:

Assign the y-forecasting function to the fourth key in that new user *menu* (assuming you did not define any other *menu* starting with 'FA' before).

Also the solution of this example 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 one function to it just above.

[ASN]



[STAT]

ASSIGN \hat{y}

CLΣ	\bar{x}_g	Σ	Σ_p		Σ_m
Σ^-	\bar{x}_w	S_w	G_w		S_{mw}
Σ^+	\bar{x}	S	G	SUM	S_m

ASSIGN \hat{y}

OrthoF					
LinF	ExpF	LogF	PowerF		BestF
L.R.	r	S_{xy}	cov	\hat{y}	\hat{x}

ASSIGN \hat{y}

OrthoF					
LinF	ExpF	LogF	PowerF		BestF
L.R.	r	S_{xy}	cov	\hat{y}	\hat{x}

CATALOG

ASSIGN \hat{y}

FCNS	PROGS	DIGITS	CHARS	VARS	MENUS
------	-------	--------	-------	------	-------

MENUS

ASSIGN \hat{y}

DATES	DIGITS	DISP	EQN	EXP	Expon:
CHARS	CLK	CLR	CNST	CPX	CPXS
A...F	A...Z	A:	Binom:	BITS	Cauch:

Here you see the first view on all the *menus* defined. Now, enter **F** and the view jumps to the corresponding position in this *submenu*:

	ASSIGN	\hat{y}		0.	
INTS	I/O	LgNrm:	Logis:	LOOP	MATRS
FRACS	F:	F&p	Geom:	Hyper:	IINTS

FavFun	FCNS	FIN	FINTS	FLAGS	FLASH
--------	------	-----	-------	-------	-------

Press the leftmost softkey to open FavFun

	ASSIGN	\hat{y}		0.	

Since FavFun was just created above there is nothing to be seen in the *menu section* of the display yet. Pressing the 4th softkey, however, you will get

	ASSIGN	\hat{y}		0.	
			\hat{y}		

Summarizing,

ASN **STAT** **▲** \hat{y} **CATALOG** **MENUS** **F** FavFun **▲**

did the job here. Note that FavFun will remain on screen until another *menu* is called or it is EXITed explicitly.

Browsing and Purging Menus, Variables, and Programs

As seen in last paragraph,

CATALOG'MENUS contains all the *menus* currently defined. Thus, **CATALOG** **MENUS** ... **▲** **◀** allows for deleting the *menu* selected. Predefined *menus* cannot be deleted.

Variables and programs are handled in full analogy:

CATALOG'VARS contains all variables currently defined. Thus,

CATALOG VARS ... allows for deleting the variable selected. Predefined variables cannot be deleted.

New programs must start with a global label. Such labels may be seven characters long and must be unique (see p. 269).

CATALOG'PROGS contains all programs currently defined. Thus,

CATALOG PROGS ... allows for deleting the program selected (cf. CLP and CLPALL).

Assigning Special Characters

As mentioned above, you must be in *alpha input mode* (AIM) to do the following. Then,

[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 above (see p. 182).

[key] may be any legal label location, shifted or unshifted, except **[UM]**, **ENTER↑**, or **EXIT**. The assignment will become valid when AIM is called in *user mode* or when *user mode* is called in AIM.

Example 1:

Let's assign the parentheses to **f** + **1/x** and **f** + **y^x** (these locations are not assigned yet in AIM). Remember **g** - calls αMATH in AIM – see the *ReM* for its contents.

α ASN **g** - **f** (**f** **1/x**
ASN **f**) **f** **y^x**

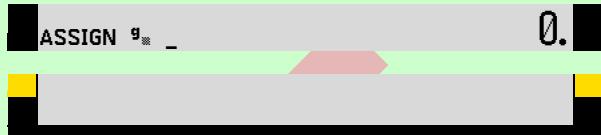
Example 2:

Assign the *Yuan* symbol ¥ (contained in α• at a **g**-shifted position) to the first key in Myα (assume *startup default* state once again):

α **(ASN)**



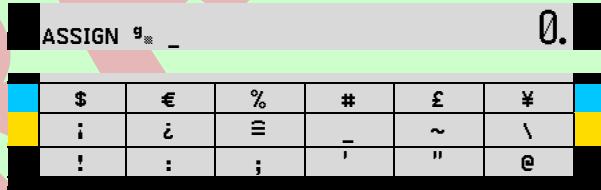
g



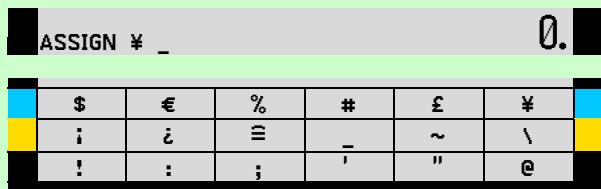
□



g



¥ (press the rightmost softkey)



UM



Note that **[UM]** will exit all *menus* being open at that time so Myα can slip on the screen (being empty still).

(press the leftmost softkey)



Summarizing,

did this assignment.

User Mode

[UM] toggles *user mode*.

Therein, your (user) assignments become valid wherever they apply. **[UM]** is the only label you cannot reassign; everything else is wide open for your ideas.

For obvious reasons, we recommend you leave **[f]**, **[g]**, and **[ENTER↑]** untouched as well; but this is just a recommendation; feel free to decide differently.

Remember that pressing any key (or a *prefix* plus a key)



displays the operation currently assigned to it in the left part of the top numeric row – if you should realize you have picked the wrong key, simply keep it pressed until the display returns to **NOP** after 1 second.

User mode gives you unexcelled freedom for creating your very personal layout. Enjoy – and play with the world of opportunities you have!¹⁶⁹

Once you have reached a stable user layout, however, we recommend you store it using STOCFG in a *register* or variable, together with the settings mentioned above (see p. 75); this applies especially if you plan having further alternative layouts – you can load any of them using RCLCFG. On the other hand, if you want to get rid of an outdated user layout and free the memory allocated for it, simply clear the respective *register* or delete the variable as described on p. 276.

It may pay well to print keyboard overlays for your favorite layouts, especially if you modify just the functions printed on the key plate. Note

you can fix such overlays in the three slots provided on either side of the keyboard – see App. G of the *ReM* for specifications.

If, however, you should get lost in your various assignments – look for **U** top right in the *status bar* and remember that pressing **UM** will bring you back from *user mode* to the default keyboard as you know it from the very beginning.

We sign off wishing you long lasting joy and benefit working with your very own, personalized *WP 43S*!

¹⁶⁹ These are far more opportunities than the *HP-41C “Blanknut”* gave you (one of the few vintage customizable calculators which may be still found for sale as a used product sometimes).



APPENDIX 1: OPERATOR PRECEDENCE

Your *WP 43S* does not have to care for operator precedence since it executes just one operation at a time (cf. p. 45). Hence it is your job to control the sequence of operations you present to your *WP 43S*. There are common rules and conventions in mathematics dealing with that – you have learned them in school. Here is just one **example** for affirmation and/or reminding:

$$1 - 2 \cdot 3^4 : 5 + \sin(6 - \sqrt[3]{7^2}) \cdot 8! + \ln \left[\left(-9^{2^3} \cdot 45^{(6/7)} \right)^2 \right]$$

(or, written for another part of this world needing more space:

$$1 - 2 \times 3^4 \div 5 + \sin(6 - \sqrt[3]{7^2}) \times 8! + \ln \left[\left(-9^{2^3} \times 45^{(6/7)} \right)^2 \right])$$

This may be solved the following way, for instance, using your *WP 43S* in *startup default* settings:

9 [ENTER↑] 2 [ENTER↑] 3 [y^x] [y^x] [+/-]

calculates -9^{2^3} ; note the arguments automatically fill in correctly.

6 [ENTER↑] 7 [/] 45 [x \gtrless y] [y^x]

calculates $45^{(6/7)}$.

[x] [x²] [ln x]

solves the rightmost term.

7 [x²] 3 [1/x] [y^x] 6 [x \gtrless y] [-] [sin]

solves the sine.

8 [x!] [x] [+]

solves the third term and adds it to the fourth.

3 [ENTER↑] 4 [y^x] 2 [x] 5 [/] [-]

solves the second term and subtracts it from said sum.

1 [+]

returns the overall solution

1 657.008 948 091 604

The colors indicate the three *stack registers* employed for this solution (cf. pp. 40ff). Note **[x \gtrless y]** is used twice herein to swap arguments.

APPENDIX 2: KEY RESPONSE TABLE

Here you find all direct keystroke inputs explained, top left to bottom right of the keyboard. For each key, its unshifted function is mentioned first, then its **f**-shifted and its **g**-shifted function, if applicable. Most keys will change functionality in *alpha input mode* (*AIM*), hence the “alpha” meanings are listed thereafter. See the pages mentioned explicitly or the *ReM* for details of all the functions mentioned below.

R	Keystrokes	Meaning
1		Calls the function displayed at the corresponding position in the bottom row of the <i>LCD</i> .
		Call the function displayed at the corresponding position in the golden or blue row, respectively.
		Does nothing if there is no function displayed at this position. Cf. pp. 27f.
2		Inverts the number x or all elements of the matrix x .
		Calls PROFRC allowing only <i>proper fractions</i> or mixed numbers on screen. Displays <i>real numbers</i> as <i>proper fractions</i> , if applicable.
		Calls IMPFRC allowing only <i>improper fractions</i> on screen. Displays <i>real numbers</i> as <i>improper fractions</i> , if applicable.
		Enters the letter A or a
		Enters the Greek letter A or α in <i>AIM</i> (see pp. 181ff)..
2		Computes y to the power of x .
		If pressed trailing integer input, defines its base. Else converts x into an integer of the base specified. See pp. 111ff.
		Calls a <i>menu</i> containing 2^x , logarithms, hyperbolic and some more exponential functions (see p. 27).
		Enters the letter B or b
		Enters the Greek letter B or β in <i>AIM</i> .

R	Keystrokes	Meaning
2	TRI	Calls a <i>menu</i> containing trigonometric functions.
	CC	Complex closing, composing, cutting, and converting, s. p. 293.
	CPX	Calls a <i>menu</i> containing commands operating on <i>complex numbers</i> like CONJ, CROSS, DOT, and Re>Im. S. pp. 148ff.
	C g C	Enters the letter C or c _____ in AIM (see pp. 181ff). Enters the Greek letter Γ or γ
2	In	Returns the natural logarithm of the number x or of all elements of the matrix x .
	d.ms	If pressed trailing numeric input, enters an <i>angle</i> in <i>degrees, minutes, and seconds</i> (i.e. sexagesimal notation). Else sets <i>angular display mode</i> to sexagesimal angles. See pp. 117ff.
	Ig	Returns the (common) decadic logarithm of the number x or of all elements of the matrix x .
	D g D	Enters the letter D or d _____ in AIM. Enters the Greek letter Δ or δ
2	e^x	Returns the inverse of the natural logarithm of the number x or of all elements of the matrix x .
	.d	If pressed trailing numeric input, enters a date (see p. 146). Else converts <ul style="list-style-type: none"> • a sexagesimal <i>angle</i> to a decimal number (s. p. 118), • a sexagesimal <i>time</i> to a <i>real number</i> (s. pp. 144f), • an integer or fraction to a decimal number (see p. 111), • a <i>DP</i> to a <i>single precision real number</i> (s. pp. 65ff).
	10^x	Returns the inverse of the decadic logarithm of the number x or of all elements of the matrix x .
	E g E	Enters the letter E or e _____ in AIM. Enters the Greek letter \mathbb{E} or \mathfrak{e}

R	Keystrokes	Meaning
2	[<i>x</i>]	Returns the square root of the number <i>x</i> or of all elements of the matrix <i>x</i> .
	[<i>h.ms</i>]	If pressed trailing numeric input, enters a sexagesimal <i>time</i> . Else converts <i>x</i> into such a <i>time</i> . See pp. 141f.
	[<i>x</i>²]	Squares the number <i>x</i> or all elements of the matrix <i>x</i> .
	[F]	Enters the letter F or f
3	[<i>g F</i>]	Enters the Greek letter Φ or φ
	[STO]	Stores <i>x</i> at the destination specified (see pp. 52ff).
	[ASN]	Assigns an <i>item</i> to a key, allowing you to create your very personal user keyboard layout (see pp. 268ff).
	[SAVE]	Saves all your data in the backup region (see p. 219) from where they may be recovered by LOAD.
3	[G]	Enters the letter G or g
	[<i>g G</i>]	Enters the Greek letter Γ or γ
	[RCL]	Recalls a stored object into X (see pp. 52ff).
	[VIEW]	Views the destination, i.e. displays its address and contents directly below the <i>status bar</i> until next keystroke (see p. 57).
3	[CNST]	Calls a catalog of fundamental physical, mathematical, and astronomical constants. See pp. 257ff for the full list.
	[H]	Enters the letter H or h
	[<i>g H</i>]	Enters the Greek letter Χ or χ
		in AIM.

R	Keystrokes	Meaning
3	R↓	Rolls the <i>stack</i> contents one level down (see p. 38).
	R↑	Rolls the <i>stack</i> contents one level up (see p. 38).
	RBR	Calls the <i>register browser</i> (see pp. 249f).
	I	Enters the letter I or i
	g I	Enters the Greek letter I or ι in AIM (see pp. 181ff).
	f R↓	Makes next character a subscript (if applicable)
3	UM	Toggles <i>user mode</i> (see pp. 279f).
	α	Sets AIM for entering characters (see pp. 181ff).
	a.FN	Calls the <i>menu</i> containing operations for alpha string manipulation. See pp. 186f.
3	f	<i>Prefix</i> to reach the golden labels.
3	g	<i>Prefix</i> to reach the blue labels.
4	ENTER↑	Context sensitive key, see p. 293.
	CATALOG	Opens the <i>catalog</i> of everything (functions, variables, menus, programs, etc.). See the <i>ReM</i> for its structure and contents.
	FILL	Fills all <i>stack registers</i> with <i>x</i> (see p. 38).
4	x↔y	Swaps the contents of X and Y (see p. 38).
	x↔	Swaps the contents of X and the destination specified.
	STK	Calls a <i>menu</i> containing <i>stack</i> related operations (drop, swap, and shuffle commands, and <i>stack size setting</i>). See pp. 37ff.
	J	Enters the letter J or j
	g J	Enters the Greek letter H or η in AIM.
	f x↔y	Enters the character ☒

R	Keystrokes	Meaning
4		If pressed during input of mantissa or exponent, changes its respective sign (see p. 25). Else multiplies the number x or all elements of the matrix x .times -1.
		Calculates the difference between y and x in percent of y , leaving y unchanged. The difference is positive for $x > y$.
		Calculates x percent of y , leaving y unchanged.
	 	Enters the letter K or k in AIM (see pp. 181ff).
4		Allows entering an exponent of ten for convenient entry of very large or very small numbers (see p. 25).
		Changes the number of decimals displayed for <i>real</i> and <i>complex numbers</i> keeping their basic numeric display format.
		Calls a <i>menu</i> containing FIX, SCI, ENG, and many more commands for numeric display formatting. See pp. 76ff.
	 	Enters the letter L or I in AIM. Enters the Greek letter Λ or λ in AIM. Makes next char a superscript (if applicable)
4		Context sensitive key, see p. 295.
		Undoes the last command executed (see p. 50).
		Calls a <i>menu</i> containing most commands for clearing; s. p. 29.

R	Keystrokes	Meaning
5	[/]	Divides y by x . For matrices, multiplies y times x^{-1} .
	[RMDR]	Returns the remainder of the integer division y / x .
	[Π]	Recalls the number π into X.
	[M]	Enters the letter M or m
	[g] [M]	Enters the Greek letter Μ or μ in AIM (see pp. 181ff).
5	[f] [/]	Enters the character /
	[7]	If there is an open question like Are you sure?, enters N for 'no'. Else enters the digit 7.
	[N]	Enters the letter N or n
	[g] [N]	Enters the Greek letter Ν or ν in AIM.
	[f] [7]	Enters the character 7
5	[8]	Enters the digit 8.
	[O]	Enters the letter O or o
	[g] [O]	Enters the Greek letter Ω or ω in AIM.
	[f] [8]	Enters the character 8
	[9]	Enters the digit 9.
5	[P]	Enters the letter P or p
	[g] [P]	Enters the Greek letter Π or π in AIM.
	[f] [9]	Enters the character 9

R	Keystrokes	Meaning
5	[XEQ]	If there is an open question like Are you sure? , confirms it; else (if in <i>PEM</i>) inserts a call to the subroutine with the label specified; else (i.e. in <i>run mode</i>) calls the routine with the label specified and starts executing it.
	[GTO]	Goes to a specified location in program memory.
	[LBL]	Enters a label for a particular location in program memory.
	[Q]	Enters the letter Q or q in <i>AIM</i> (see pp. 181ff).
6	[x]	Multiplies <i>y</i> times <i>x</i> .
	[MATX]	Calls a <i>menu</i> of matrix operations including e.g. $[M]^{-1}$, $ M $, $[M]^T$, CROSS, DOT, and the <i>Matrix Editor</i> (see pp. 157ff).
	[x!]	Calculates the factorial (see p. 89).
	[R]	Enters the letter R or r
	[g] [R]	Enters the Greek letter P or p in <i>AIM</i> .
6	[f] [x]	Enters the character x or ·
	[4]	Enters the digit 4 .
	[PROB]	Calls a <i>menu</i> containing combinations, permutations, the gamma function, a random number generator, and all the probability distributions supported. See pp. 89ff.
	[STAT]	Calls a <i>menu</i> containing the accumulated statistical sums as well as functions for sample statistics: $\Sigma+$, $\Sigma-$, $CL\Sigma$, various means and measures for scattering, as well as curve fitting functions. See pp. 93ff.
	[S]	Enters the letter S or s
	[g] [S]	Enters the Greek letter Σ or σ in <i>AIM</i> .
	[f] [4]	Enters the character 4

R	Keystrokes	Meaning
6	(5)	Enters the digit 5.
	R↔	Calls →REC, converting 2D polar coordinates (magnitude r and angle θ) to rectangular (<i>Cartesian</i>) coordinates x and y (see p. 120).
	→P	Calls →POL, converting 2D rectangular (<i>Cartesian</i>) coordinates (x and y) to polar coordinates magnitude r and angle θ (cf. pp. 18f).
	T	Enters the letter T or t
	g T	Enters the Greek letter Τ or τ in AIM (see pp. 181ff).
	f 5	Enters the character 5
6	(6)	Enters the digit 6.
	L↔	Call the <i>menus</i> of angular or unit conversions, respectively. See pp. 118 and 263ff for them.
	U↔	Enters the letter U or u
	g U	Enters the Greek letter Θ or θ in AIM.
	f 6	Enters the character 6
	▲	Context sensitive key, see p. 295.
6	≡▲	Moves the program pointer one step back. See pp. 189ff.
	MODE	Calls a <i>menu</i> containing operations for setting modes like angular display format, complex notation, fractions denominator, etc. See pp. 117ff and 141ff.

R	Keystrokes	Meaning
7		Subtracts x from y .
		Calls the <i>menu</i> containing advanced operations for solving arbitrary equations, finding roots, integrating, deriving, computing sums and products (see pp. 221ff).
		Calls the <i>menu</i> containing all equations currently defined (see pp. 224ff).
		Enters the letter V or v
		Calls a <i>menu</i> containing math symbols in AIM (s. pp. 181ff).
		Enters the character -
7		Enters the digit 1 .
		Calls a <i>menu</i> containing Boole's operations (AND, OR, NOT, etc.) as well as bit manipulating commands.
		Calls a <i>menu</i> containing operations for integers as well as sign mode settings.
		Enters the letter W or w
		Enters the Greek letter Ψ or ψ in AIM.
		Enters the character 1
		Enters the digit 2 .
7		Calls the <i>menu</i> containing financial functions (i.e. the TVM – see pp. 253f – and % functions).
		Calls a <i>menu</i> containing advanced mathematical (extra) functions. See the ReM.
		Enters the letter X or x
		Enters the Greek letter Ξ or ξ in AIM.
		Enters the character 2

R	Keystrokes	Meaning
7	[3]	If there is an open question like Are you sure? , enters Y for 'yes'. Else enters the digit 3 .
	TIMER	Calls the timer application (see pp. 250ff).
	CLK	Calls the menu containing all (clock based) time and date commands. See pp. 141f.
	[Y]	Enters the letter Y or y
	g [Y]	Enters the Greek letter Υ or υ in AIM (see pp. 181ff).
7	f [3]	Enters the character 3
	[▼]	Context sensitive key, see p. 295.
	≡▼	Moves the program pointer one step forward (see pp. 189ff).
	FLAGS	Calls a <i>menu</i> containing commands for <i>flag</i> handling. These operations are of most use in <i>PEM</i> . See pp. 189ff.
8	[+]	Adds <i>x</i> to <i>y</i> .
	[P X]	Sends the contents of X to the printer.
	I/O	Calls the <i>menu</i> containing all I/O-related operations (incl. printing). See pp. 218f.
	[?]	Enters the character ?
	g [+]	Calls a <i>menu</i> containing additional Latin letters in AIM.
8	[f +]	Enters the character +
	[0]	Enters the digit 0 .
	LOOP	Calls a <i>menu</i> containing INC and DEC and the related loop control commands ISG, DSE, etc. See pp. 204f.
	TEST	Calls a <i>menu</i> containing comparisons, conditionals, and other binary tests. See pp. 201ff.
	[Z]	Enters the letter Z or z
8	g [Z]	Enters the Greek letter Ζ or ζ in AIM.
	f [0]	Enters the character 0

R	Keystrokes	Meaning
8		Usually enters a decimal radix mark in numeric input. If pressed twice in input, allows for entering a fraction (see pp. 65ff and 141ff). – In register or flag addressing, heads a local address (see pp. 52ff).
	PARTS	Calls a <i>menu</i> containing ABS, FP, IP, SIGN, DECOMP, etc.
	INFO	Calls a <i>menu</i> containing commands to return system information. See pp. 201ff.
		Enters a comma
		Calls a <i>menu</i> containing punctuation marks etc.
		Enters a point
	R/S	Context sensitive key, see p. 294.
8	P/R	Toggles <i>program-entry</i> and <i>run mode</i> .
	P.FN	Calls a <i>menu</i> containing dedicated programming functions. These operations are of most use in <i>PEM</i> . See pp. 189ff.
		Enters a blank space
	R/S	Enters the printer character ☰
	EXIT / ON	Context sensitive key, see p. 294.
8	RTN	Returns to the caller. See pp. 189ff.
	OFF	Turns your <i>WP 43S</i> off unless in <i>PEM</i> , where it inserts OFF behind the <i>current step</i> (see p. 191).

Seven context sensitive keys need longer explanations. You find them in the table starting overleaf, sorted alphabetically. If any of these keys is pressed, your *WP 43S* will run top down through a sequence of key-specific tests – whichever test becomes true first, your *WP 43S* will execute the corresponding operation and return, waiting for your next input.

Key	Condition(s)	Meaning
CC	X contains an <u>open</u> (input) number, cf. p. 25	If RECTangular mode is set, CC closes input, checks, and saves it as real part of a forthcoming <i>complex number</i> , then waiting for your input of its imaginary part. Else POLAR mode must be set, so CC closes, checks, and saves the input as magnitude and waits for your input of the angular part. See pp. 148ff for more.
	X contains a closed <i>complex number</i> , vector, or matrix	If RECT is set, CC splits ('cuts') <i>x</i> into its real and imaginary part, returning the real part in X and the imaginary part in Y . Else, CC splits <i>x</i> into its magnitude <i>r</i> and angle <i>θ</i> , returning <i>r</i> in X and <i>θ</i> in Y .
	X and Y contain two closed <i>real numbers</i>	Interprets <i>x</i> and <i>y</i> either (with POLAR set) as magnitude and angle, or (for RECT set) as real and imaginary parts. CC combines <i>x</i> and <i>y</i> to compose one <i>complex number</i> <i>x</i> , then drops <i>y</i> .
	X and Y contain two closed real vectors (or matrices) of identical dimension	Returns one complex vector (or matrix) <i>x</i> , working in analogy to previous row.
	Else	Throws an error.
ENTER↑	Waiting for parameter input	Closes pending command input and executes said command (see p. 60 for more).
	Asking f. confirmation	Confirms the question.
	In TIMER	Is honored as described on pp. 250f.
	In RBR, STATUS	Does nothing.
	Else	Closes alphanumeric input and enters data in the stack (see pp. 32f and 38 for details).

Key	Condition(s)	Meaning
EXIT / ON	WP 43S turned off	Works as ON turning your WP 43S on.
	Pressed together with + , - , etc.	Executes the respective ON -key combination – see the ReM for more.
	Waiting for parameter input	Cancels the pending command.
	Waiting for alphanum. input	Closes input.
	<i>Temporary information displayed</i>	Clears this information (e.g. an error message) returning to the calculator state as was before it was thrown. See p. 65.
	Asking for confirmation	Denies the question.
	In RBR, STATUS, TIMER	Leaves the application (s. pp. 249ff).
	In a (<i>sub-</i>) menu or browser	Leaves the current (<i>sub-</i>) menu or browser without executing anything, returning to the status of your WP 43S as it was before.
	 flashing	Stops the running program like R/S .
	In PEM	Leaves program-entry mode like P/R .
	A or α	Closes x and leaves alpha input mode.
	Else	Does nothing.
R/S	In TIMER	Starts or stops the timer without changing its value (see p. 250).
	 flashing	Stops executing the running program immediately.  will be displayed until next keystroke.
	In PEM	Enters the command STOP.
	Else	Runs the <i>current routine</i> (s. pp. 189ff) or resumes its execution starting with the step after the <i>current step</i> .

Key	Condition(s)	Meaning
	Open numeric or alpha input	Deletes the last character entered. If none is left, cancels pending command like EXIT .
	<i>Temporary information displayed</i>	Clears the information returning to the calculator state as was before this (e.g. an error message) was thrown. See p. 65.
	Asking f. confirmation	Denies the question.
	In TIMER	Resets the timer (see p. 250).
	In PEM	Deletes the current program step.
	Else	Calls the command CLX.
	In STO or RCL	Honored as described on pp. 56ff.
	In RBR, STATUS, TIMER	Honored as described in Sect. 5 (pp. 248ff).
		Sets upper case if applicable (does nothing and passes control to next condition check if <u>αMATH</u> or <u>α•</u> is open).
	In a <i>multi-view menu</i>	Goes to next view in the current <i>menu</i> .
	In PEM	Goes to previous program step. Will repeat with 2Hz when held down longer than 0.5s.
	In <i>run mode</i>	Browses the <i>current routine</i> , going to previous program step.
	In STO or RCL	Honored as described on pp. 56ff.
	In RBR, STATUS, TIMER	Honored as described in Sect. 5 (pp. 248ff).
		Sets lower case if applicable (does nothing and passes control to next condition check if <u>αMATH</u> or <u>α•</u> is open).
	In a <i>multi-view menu</i>	Goes to previous <i>view</i> in the current <i>menu</i> .
	In PEM	Goes to next program step. Will repeat with 2Hz when held down longer than 0.5s.
	In <i>run mode</i>	Browses the <i>current routine</i> , executing the current program step and going to next step.

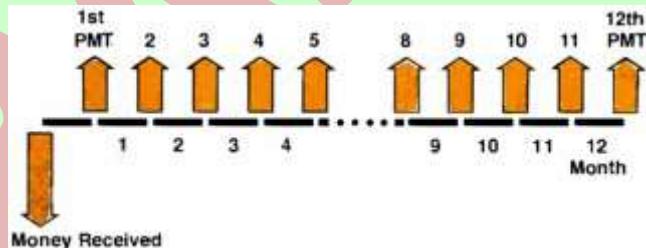
APPENDIX 3: FURTHER APPLICATIONS OF TVM

The examples as well as all the other text printed blue in this appendix are quoted from the *HP-27 OH*. Enjoy the boundary conditions of that time.

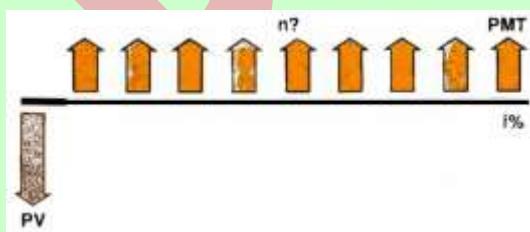
Ordinary Annuities (a.k.a. Payments in Arrears)

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:



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 semi-annually?

Solution:

[FIN] **TVM** **End**

withdrawals are due at the end of each period,

5 [ENTER↑] 2 [/] i%/a
 50000 PV
 3000 PMT nPER

2.50
 50 000.00
 21.83

% semiannual interest rate,
 principal (capital),
 semiannual withdrawals,
 so your savings will last for
 almost eleven years.

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?

Solution:

12 per/a
 12 [ENTER↑] 2 × nPER
 1775 PV
 83.65 PMT
 i%/a

12.00
 24.00
 1 775.00
 83.65
 12.11

months per year,
 periods in total,
 principal (capital),
 payment;
 % 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.)

Solution:

First, compute the payment amount which is based on \$50 000

9 i%/a
 12 per/a
 12 [ENTER↑] 30 × nPER
 50000 PV
 PMT

9.00
 12.00
 360.00
 50 000.00
 83.65

annual interest rate,
 months per year,
 periods in total,
 principal (capital);
 payment.

PMT	83.65	reuse payment,
RCL nPER nPER	360.00	recall and reuse periods,
RCL PV 2 % - PV	49 000.00	effective amount received,
i%/a	12.11	% 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.

Solution:

12 per/a	12.00	months per year,
12 ENTER↑ 30 × nPER	360.00	payment periods in total,
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 – ... 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 <i>balloon</i> ,
PV	1 823.37	present value of <i>balloon</i> ;
PV	1 823.37	input of PV of balloon;
RCL i%/a	10.00	recall & reuse interest rate,
RCL n _{PER}	360.00	recall and reuse periods,
8000	8 000.00	gross value of loan amount,
RCL PV	6 176.63	net present value of loan amount less <i>balloon</i> ;
PMT	131.24	monthly payment.

Example for finding the present value of an ordinary annuity:

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?

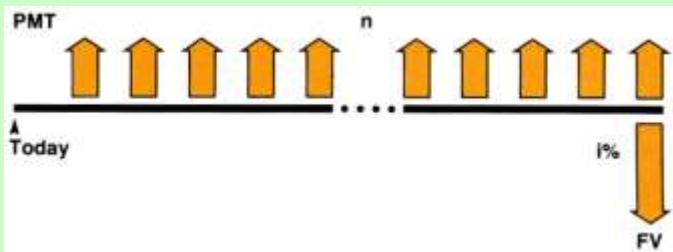
Solution:

12 per/a	12.00	months per year,
12 [ENTER↑] 3 × n _{PER}	36.00	payment periods in total,
10 i%/a	10.00	% annual interest rate;
80 PMT	9.00	monthly payment,
PV	2 479.30	price he can pay for the snowmobile.

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 first period, like so:



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	ENTER↑	20.00	payment periods in total,
5	i%/a	5.00	% annual interest rate;
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*.

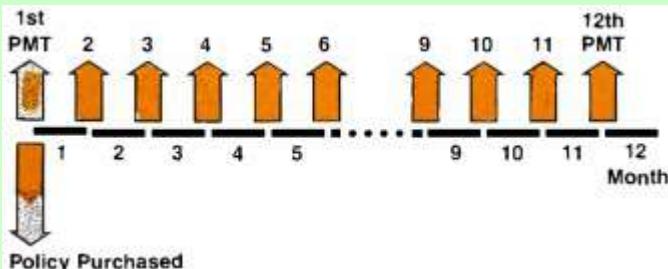
An annuity due with monthly payments for a year – say, a car insurance

policy¹⁷⁰ – looks like this →

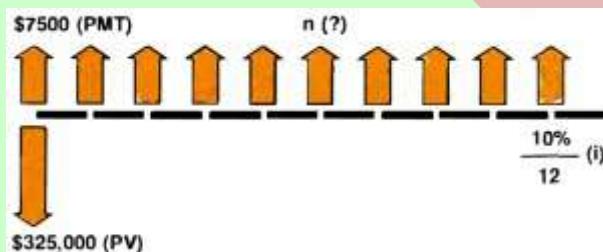
Notice that with an annuity due, you have a payment right away at the beginning of the first interval

(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.



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 \$7 500 per month, how long must the investment be held to yield 10% per annum?¹⁷¹

Solution:

[FIN] **TVM** **Begin**
12 **per/a**
10 **i%/a**
325000 **PV**
7500 **PMT** **n_{PER}**

payments are due at the begin of each period,
12.00 months per year,
10.00 % annual interest rate,
325 000.00 investment;
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?

¹⁷⁰ Translator's note for German readers: "Policy" entspricht hier einer *Police*.

¹⁷¹ I frankly admit I understand neither this problem nor its solution.

Solution:

12	per/a	12.00	months per year,
6	i%/a	6.00	% annual interest rate,
0	PV	0.00	start balance,
1000	FV	1 000.00	future value;
50	PMT	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	ENTER↑	12	x nPER	104.00	payment periods in total,
12000	PV	12 000.00	start value of equipment,		
0	FV	0.00	final value of equipment,		
200	PMT	200.00	payments;		
i%/a		13.07	% annual yield.		

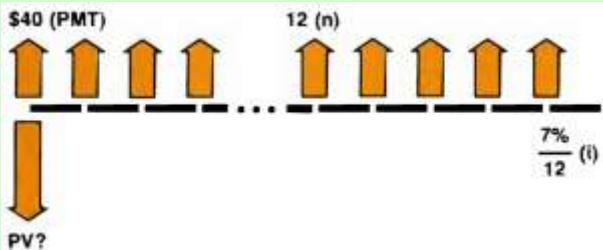
Example for finding the payment amount for an annuity due:

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). 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	ENTER↑	4	x nPER	240.00	payment periods in total,
10	i%/a	10.00	% annual target yield,		
70000	PV	70 000.00	PV of the building;		
0	FV	0.00	FV of the building;		
	PMT	1982.27	quarterly payments.		

Example for finding the present value for an annuity due:



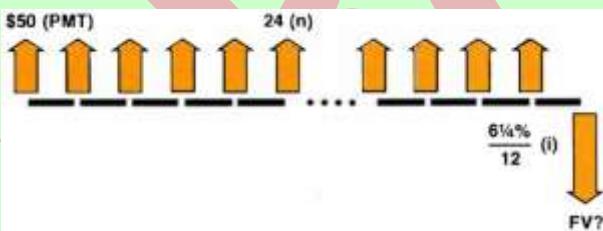
The owner of a downtown parking lot has achieved full occupancy and a 7% annual yield by renting parking spaces for \$40 per month payable in advance. Several regular customers want to rent their spaces on an annual basis. What annual rent, also payable in advance, will maintain a 7% annual yield rate?

Solution:

12 per/a
12 n_{PER}
7 i%/a
40 PMT
PV

12.00	months per year,
12.00	payment periods in total,
7.00	% annual target yield,
40.00	monthly payments;
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 1/4 % interest compounded monthly, how much will you have 2 years from now?

Solution:

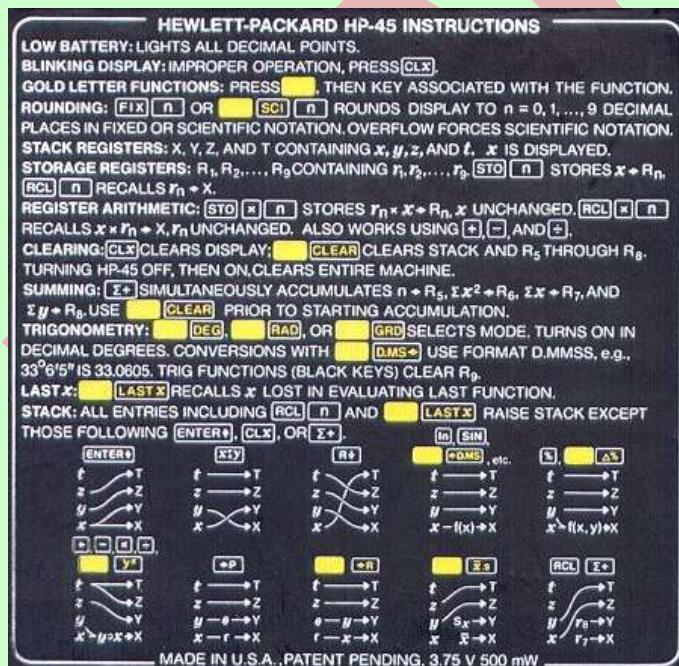
12 per/a
2 ENTER↑ 12 × n_{PER}
6.14 i%/a
50 PMT
0 PV
FV

12.00	months per year,
24.00	payment periods in total,
6.25	% annual target yield,
40.00	monthly payments;
0.00	start balance;
1 281.34	balance after two years.

APPENDIX 4: POWER SUPPLY

Your *WP 43S* is powered by a single CR2032 coin cell (3 V). Alternatively, it may be powered through its USB port – running with even higher speed then. See *App. A* of the *ReM* for more.

See here what sufficed for explaining the basic functionality of the *HP-45* on its back label in 1973:



Though the *HP-45* featured neither advanced operations (beyond statistical summation, means, and standard deviations), programming, menus, catalogs, data types, browsers, applications, named variables, nor customizing – but your *WP 43S* does.

APPENDIX 5: RELEASE NOTES

	Date	Release notes
0	29.11.12	Official project start with first publication on the MoHPC forum. There are found, however, far older traces of a '43S' denoting a 'Super HP-42S', though in various more or less fictional cases – pure vapourware™.
0.1	2.2.14	Manual setup based on the one of <i>WP 34S</i> .
	23.5.15	Passed to J. Schwartz, E. Smith, and R. Ottosen for first information.
0.2	3.10.15	Update based on Jake's feedback and further thoughts, distributed to Eric, Jake, Marcus, and Pauli.
0.3	21.3.16	Split the manual in three; moved LBL onto the keyboard, renamed STOM to STOCFG, RCLM to RCLCFG, SERR to s_m , and SERR _w to s_{mw} ; refined the <i>Key Response Table</i> . Passed to Michael for information.
0.4	28.3.16	Renamed LOGS to EXP and EEX to E . Added hardware information from 2 nd manufacturer.
0.5	29.10.16	Returned EEX . Changed the keyboard layout.
0.6	22.8.17	Merged the Applications and Owner's Manual. Changed the input order of complex number parts on Pauli's request. Changed the keyboard layout introducing D.MS, SST, BST, and % while removing ÿ, RAN#, 'FRC, and 'CFIT. Put 'CFIT into 'STAT and 'FRC into 'MODE. Placed OFF below EXIT for easier customizing. Renamed cc to C5, EEX to E , STOPW to TIMER, SHOW to REGS, 'SOLVE to 'ADV, DLINES to DSTACK, 12h to CLK12, and 24h to CLK24. Replaced IND by →. Deleted %MG since covered by Δ%, added EIGVAL and EIGVEC. Swapped CNST and CONST. Defined the echo rows for alphanumeric and command input. Expanded and modified the character sets for better use of display space. Added the QRG.
0.7	2.4.18	Changed keyboard layout. Replaced the labels BST by ≡Δ , SST by ≡▽ , and UNDO by ↶ ; added some <i>alpha input mode</i> reminders on the keyboard. Added AGRAPH, CLLCD, EQ.xxx, HYP, J/G, M.GOTO, ORTHOF, PIXEL, POINT, TDISP, and ≡USER . Moved the background considerations out of <i>ReM App. D</i> . Introduced K as <i>alpha register</i> for alphanumeric constants in programs. Removed <i>fraction data type</i> . Extended <i>items</i> from 6 to 7 characters to match

	Date	Release notes
		<p><i>HP-42S.</i> Specified <i>data types</i> more precisely in <i>ReM App. D.</i> Reduced the maximum number of <i>local registers</i> from 888 to 100. Deleted JG1582 and JG1752. Renamed two commands for TVM. Replaced the heading apostrophe for <i>menu names</i>. Put SUMS in STAT. Renamed the trigonometric and hyperbolic functions according to mathematical standards, and CHR to CHAR. Redistributed the chapter about constants. Modified STATUS display. Refined the unit conversions to ensure <i>SI</i> on one side. Specified 0 SEED. Expanded <i>ReM App. A</i>. Added formula output for L.R. Modified CPX?, DBL?, and REAL?. Changed output of binary tests for compatibility with <i>HP-42S</i>.</p>
0.8	7.5.18	Changed keyboard layout: introduced TRG containing trigonometric functions, removed HYP into EXP and π to g-shifted / , swapped some shifted labels. Refined the chapters about register arithmetic, <i>Command Parameter Input</i> , <i>Alphanumeric Input</i> , <i>Matrix Calculations</i> , and <i>Orthogonal Polynomials</i> . Introduced CLCVAR and more vintage examples. Rearranged <i>temporary information</i> on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE.
	20.9.18	Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9	3.1.19	Removed <i>angle data type</i> . Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded <i>App. B</i> . Added SHOW for displaying full precision of <i>DP</i> numbers and FBR for browsing our two fonts. Split a chapter. Expanded some titles. Added the overlay drawing. Modified functionality of EXIT and 1/x to match <i>HP-42S</i> . Added a chapter about curve fitting. Modified functionalities of ENTER↑ and G . Moved the printer character to f R/S . Expanded <i>App. K</i> . Renamed DOUBLE to →DP. Added →SP and conversions of quarts. Rearranged X.FN. Replaced USR by UM . Changed keyboard moving UM , ix , and TRI . Added XIN and XOUT. Added a chapter in <i>App. E</i> and information about infinite integers. Extended the domain of GCD and LCM. Refined and corrected.
0.10	16.1.19	Returned <i>angle data type</i> . Added IDIVR. Refined the explanation of ALL and the summary of integer functions.

INDEX

This index lists special terms and keywords used in this manual. Furthermore, it points to the most prominent of the over 130 examples, marked '(ex.)'.

Items are listed below only if they are extensively treated in this manual (remember you find each and every *item* provided explained in the *IOI* printed in the *ReM*; and the *IOI* will also point you to further explanations if applicable). Looking at the *Table of Contents* above is recommended as well – titles are not repeated below.

about to die (ex.) 183
account balancing (ex.) 31
address space 52
adjusting display contrast 16
ADM 117
advertising pictures (ex.) 90
AIM 181
air pressure and altitude (ex.) 81
aircraft navigation (ex.) 122
altimeter (ex.) 81
archery statistics (ex.) 93
ASSIGN 268
automatic stack lift 35
balloon trip (ex.) 79
bicycle gearing (ex.) 80
black or white cats (ex.) 184
bubble sort (ex.) 206
carry 132
CDF 91
chain calculation (ex.) 34
chi-square statistic (ex.) 100
closing numeric input 25
complex aircraft navigaion (ex.) 152
compound interest (ex.) 47
confidence limits (ex.) 102
connecting peaks (ex.) 88

continuous distribution 91
cross product (ex.) 154, 170
cubic inches (ex.) 265
current step 191
data type 65
dating (ex.) 145
debugging 200
dice from Las Vegas (ex.) 101
diffraction pattern (ex.) 245
digit group separation 75
discrete distribution 91
DP 70
dyadic functions 31
earthquakes (ex.) 84
electron-volts (ex.) 261
enter exponent 25
ENTER▲ 33
error probability 92
extrapolation (ex.) 99
falling around the earth (ex.) 213
falling with drag (ex.) 210
fencing land (ex.) 18
FILL 38
filling tires (ex.) 265
finite integers 128
forecasting (ex.) 99
free fall (ex.) 97

Fukushima accident (ex.) 86
general purpose registers 53
Golden Bow (ex.) 93
great circle distance (ex.) 119
Horner scheme (ex.) 48
improper fraction 142
indirect addressing 59
infinite integers 127
interpolation (ex.) 98
item 27
Mach number (ex.) 44
markup and margin (ex.) 114
measuring capability (ex.) 108
measuring system analysis
(ex.) 106
menu 27
menu section 28
menu view 27
monadic functions 30
Mother's Kitchen (ex.) 192
Mt. Everest (ex.) 82
MyMenu 272
My • 277
navigating in space (ex.) 124
overflow 129
parachutist (ex.) 210
PDF 91
PMF 91
prefix 17
primary function 16
process capability 95
program pointer 191
proper fraction 142
proton in magnetic field (ex.)
170

QF 92
R↓, R↑ 38
radix mark setting 75
Rigel Centaurus (ex.) 49
RPN 32
satellite orbits (ex.) 213
scrap rate (ex.) 102
secondary function 16
significant change (ex.) 108,
110
significant improvement (ex.)
95
skydiving (ex.) 211
softkey 28
softkeys 23
sorting numbers (ex.) 206
special registers 52
squaring circles (ex.) 78
stack 32
stack overflow 44
status bar 70
submenu 28
surfaces of Jupiter's moons
(ex.) 20
TAM 55
tax deduction (ex.) 115
temporary information 65
Tower of Pisa (ex.) 97
triadic functions 36
Upper Lagunia (ex.) 88
user flags 53
vector operations in 2D 172
VIEW 57
virtual keyboard 55
Willie's Widget Works (ex.) 89

Introducing the first
pocket programmable
that remembers your
programs and data even
when it's turned off.
The new HP-25C. \$200.00*

Meet the first Scientific Programmable calculator that lets you take a bathroom reading through your windows without having to turn it on again when you get back. The moon. In memory. And storage registers stay ON even when you shut it OFF—during a plane call, swimming or a weekend.

Find out the HP-25C (technically, the "C" stands for "Continuous Memory CMOS technology"). *Continuous Memory*

Here's what Continuous Memory does for you:

1. Let's you key in your favorite programs—define it—and retain it in memory for repeated use. Be sure your accuracy as well as time.
2. You can add additional functions not on the keyboard: derivatives, conditional instructions, etc., simply by writing programs and storing them permanently in memory.

3. Enables you to gather data one place (such as at a survey point) and retain it over months until you get back to work with these important finds of the time well into the future and the accuracy will gain because it's now stored in memory.

4. You don't lose data when you lose battery power. Think why—where's the thermal power display?—including a low battery—your data and programs will not be lost provided the calculator is turned OFF and the recharge is promptly completed. The HP-25C will store more word data and programs than any hand-held scientific calculator—without any battery at all—while you're replacing a discharged battery with a charged one.

But that's not all. Add from Continuous Memory the HP-25C is identical to our popular HP-25. Both give you a total of 72 pre-programmed functions and operations; correlation, log/antilog, programmability, branching and conditional test capability and fixed decimal, scientific and engineering programming as well.

*Subject to change. Order number 100-100000-0000. Hewlett-Packard Company, 1501 Page Mill Road, Palo Alto, California 94303. © 1976 Hewlett-Packard Company. All rights reserved. Hewlett-Packard is a registered trademark of Hewlett-Packard Company. The HP logo is a registered trademark of Hewlett-Packard Company. Other products mentioned may be trademarks or registered trademarks of their respective companies.



multiples of 100 minutes

You also get RPN logic entries

with 4 registers each and 8 storage registers

And every program retains its value
without modification for the HP-25C.

The HP-25C has been a Scientific Programmable Calculator like it before.

Best of all, the HP-25C gets HP's name on it. And you know what that means. Design, performance and a back-up support system you can't get anywhere else.

Why not "test touch" one for yourself? Or if Continuous Memory is not important to your particular application, try our standard HP-25. \$149.00. For complete HP-25C specs and the name of your nearest dealer, call toll-free 800-536-2922 (in Calif. 800-462-9982).



THE HP-25C AT A GLANCE

- Program memory and storage registers stay ON at all times
- 72 functions and operations
- Keyboard programmability
- Full editing capability
- Branching and conditional test capability
- 8 addressable memory
- Fixed decimal and scientific notation—plus engineering notation
- RPN logic entries with 4 registers each
- \$200.00

HEWLETT hp PACKARD

Sales and service offices 125 cities in all continents.

Distr. 30000, 1111 Page Mill Road, Cupertino, CA 95014

In 1976, *Continuous Memory* was an innovation; and 72 built-in functions, 8 general purpose storage registers, and 49 merged program steps were sufficient for professional engineers and scientists doing their work as well as for science and engineering students striving for their Ph.D. Linear regression, correlation, and forecasting had to be programmed if you needed them — the respective routine took 44 steps.