

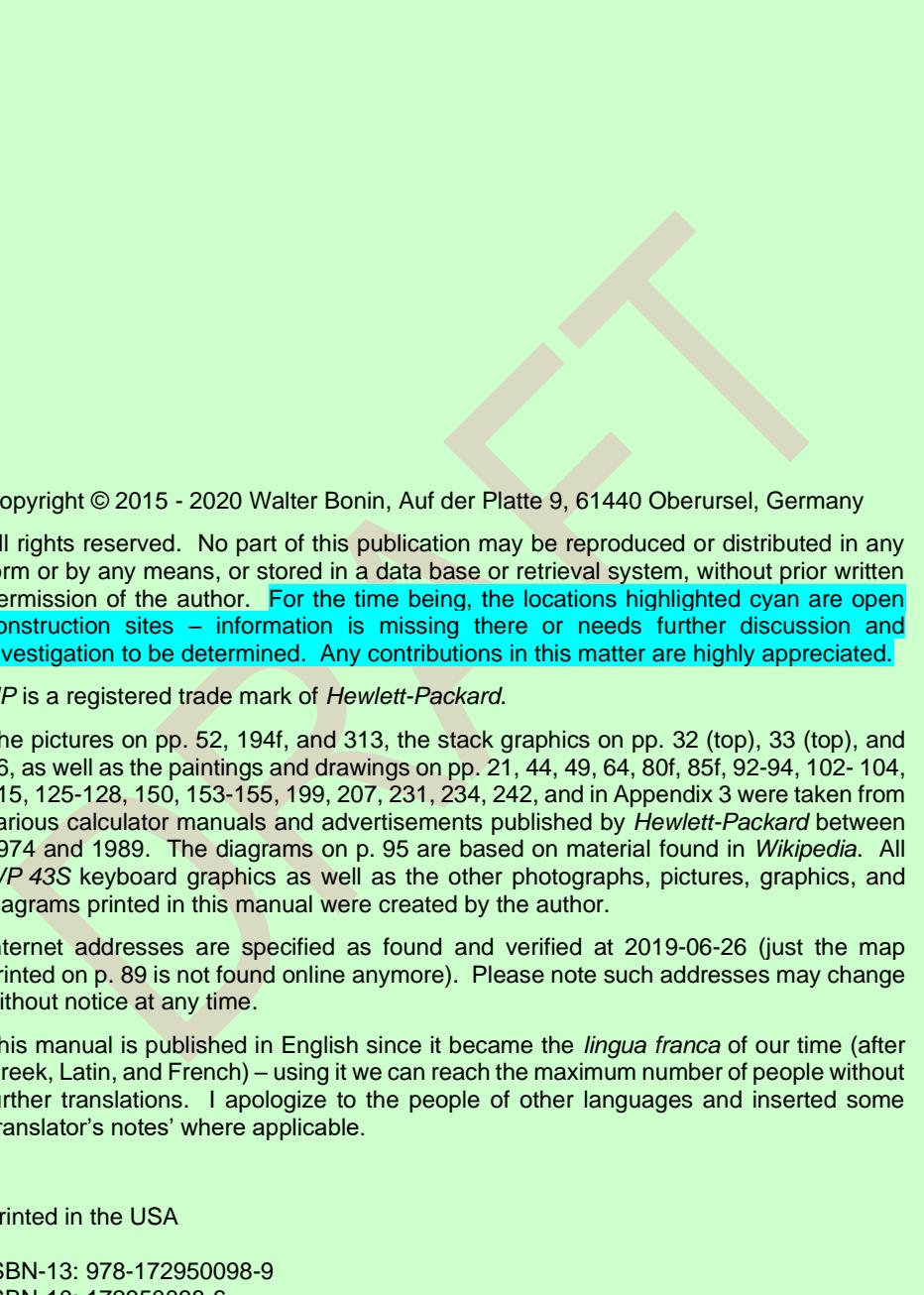


WP43S OWNER'S MANUAL

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 https://gitlab.com/Over_score/wp43s.



Copyright © 2015 - 2020 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. 52, 194f, and 313, the stack graphics on pp. 32 (top), 33 (top), and 36, as well as the paintings and drawings on pp. 21, 44, 49, 64, 80f, 85f, 92-94, 102-104, 115, 125-128, 150, 153-155, 199, 207, 231, 234, 242, and in Appendix 3 were taken from various calculator manuals and advertisements published by *Hewlett-Packard* between 1974 and 1989. The diagrams on p. 95 are based on material found in *Wikipedia*. All WP 43S keyboard graphics as well as the other photographs, pictures, graphics, and diagrams printed in this manual were created by the author.

Internet addresses are specified as found and verified at 2019-06-26 (just the map printed on p. 89 is not found 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	17
How to Enter Common Numbers (and How to Edit Them)	24
How to Enter and Execute Commands	25
Menus – Items à la carte	26
How the Keyboard is Organized	28
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
Clearing and Resetting Your WP 43S	51
Addressing and Manipulating Objects in RAM	52
Addressing Tables	59
Indirect Addressing – Working with Pointers	63
Store and Recall Arithmetic	63
Section 2: Dealing with Various Objects and Data Types	66
Some Display Basics	66
Supported Data Types	67
Recognizing Calculator Settings and Status	72
Getting Special Information: RBR, STATUS, VERS, etc.	75
Localising Numeric Output	76
Real Numbers: Changing the Display Format	78
Real Numbers: Squares and Cubes and their Roots	80
Real Numbers: Percent Change	82
Real Numbers: Logarithms and Powers (a.k.a. Antilogs)	84

Real Numbers: Hyperbolic Functions	91
Real Numbers: Probabilities – Factorials, Combinations, Permutations, and Distributions	93
Real Numbers: From Probability to Statistics – Accumulating Data, Calculating Means, Standard Deviations, and Confidence Limits; Curve Fitting, Forecasting, and Checking Dices	96
Real Numbers: Some Industrial Problems Solved	105
Real Numbers: Summary of Functions	114
Angles and Trigonometric Functions	121
Mixed Calculations: Coordinate Transformations in 2D, Flight Directions, Courses over Ground, etc.	124
Angles: Summary of Functions	130
Integers: Input and Displaying	131
Integers: Bitwise Operations on Short Integers	136
Integers: Arithmetic Operations	139
Integers: Overflow and Carry with Short Integers	141
Integers: Summary of Functions	144
Rational Numbers (Fractions)	146
Complex Numbers: Introduction	149
Complex Numbers Used for 2D Vector Algebra	153
Complex Numbers: Summary of Functions	156
Vectors and Matrices: Introduction and Input	158
Vectors and Matrices: Displaying and Editing Larger Objects	163
Vectors and Matrices: Complex Stuff	167
Vectors and Matrices: Calculating	168
Vectors and Matrices: Solving Systems of Linear Equations	173
Vectors and Matrices: Eigenvalues and Eigenvectors	174
Vectors and Matrices: Dealing with Statistical Data	179
Vectors and Matrices: Summary of Functions	181
Times	183
Dates	185

Alpha Input Mode: Introduction and Virtual Keyboard	187
Alpha Input Mode: Entering Simple Text and More	189
Combining Alpha Strings and Numeric Data	191
Working with Alpha Strings	192
Section 3: Programming	196
Recording a New Routine	198
Labels	202
Editing a Routine	204
Running a Routine from the Keyboard (also for Debugging)	206
Subroutines: Running a Routine from another Routine	207
Automatic Testing and Conditional Branching	208
Loops and Counters	211
Programmed User Interaction and Dialogues	214
Solving Differential Equations	218
The Programmable Menu (MENU)	223
Basic Kinds of Program Steps	225
Deleting Programs	226
Serial Input and Output of Data and Programs	226
Local Data	226
Flash Memory (FM)	227
Section 4: Advanced Problem Solving	229
Programmable Sums	229
Programmable Products	230
Solving Quadratic Equations	231
General Equations	232
The Interactive Solver for Arbitrary Equations	234
The Interactive Solver for Expressions Stored in Programs	238
Using the Solver in a Program	240
Numeric Integration of Equations	242
Interactively Integrating Expressions Stored in Programs	244

Using the Integrator in a Program	246
Differentiating Equations	248
Interactively Differentiating Expressions Stored in Programs	250
Computing Derivatives in a Program	251
Nesting Advanced Operations	252
Section 5: Two Browsers, two Applications, and two Special Menus	254
The Browsers RBR and STATUS	254
The Timer Application	257
The Time Value of Money (TVM)	259
Constants	263
Unit Conversions	269
Section 6: Creating Your Very Personal WP 43S	274
Assigning Your Favourite Functions	276
Creating Your Own Menus	280
Browsing and Purging Menus, Variables, and Programs	283
Assigning Special Characters	283
User Mode	285
Appendix 1: Operator Precedence	288
Appendix 2: Key Response Table	289
Appendix 3: Further Applications of TVM	303
Ordinary Annuities (a.k.a. Payments in Arrears)	304
Annuities Due (a.k.a. Payments in Advance)	308
Appendix 4: Power Supply	312
Appendix 5: Release Notes	314
Index	318

WELCOME!

Congratulations, you are holding your very own *WP 43S* in your hands now. 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, incorporating a state-of-the-art LCD, customizable by you, 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 of arbitrary functions.
- Numeric derivation, programmable sums and products.
- A wealth of functions, supporting real and complex numbers and matrices, integers, fractions, dates, times, and text strings.
- Matrix operations including a comfortable *Matrix Editor*, a solver for systems of linear equations, eigenvalues, eigenvectors, and many more matrix and vector functions in real and complex domain.
- A full set of statistical operations including probability distributions, data analysis functions, 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, see <https://www.youtube.com/watch?v=qRrAj-GCTQM>). He placed operators before numbers or variables instead between them as in conventional mathematical notation. Thus, *RPN* places operator behind numbers. See *Section 1* below for more.

Our hardware platform, the *DM42* of *SwissMicros*, was developed in parallel and launched in 2017. This is due to the fact that the layout of the *DM42* is very closely linked to the *HP-42S* of *Hewlett-Packard* produced until 1995 and its firmware uses *Thomas Okken's Free42*.

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

- + Base conversions and integer arithmetic in fifteen 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 using the bottom part of the display to assign up to eighteen operations to the top six keys according to your actual needs.
- + Keystroke programming including branching, looping, tests, *flags*, subroutines, and program-specific local data.
- + Easy system control via named *system flags* and variables provided.
- + 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 your various custom layouts on-board and recall them one by one as you need them. Keyboard overlays are supported.
- + Battery-fail-safe on-board backup memory for all your data (in *registers*, variables, *menus*, programs, layouts, and mode settings).
- + A micro *USB* socket allowing for external auxiliary power supply as well as for exchanging your programs with a computer, so you can edit, debug, and test them there and return them thereafter.
- + An infrared port for immediate recording of 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).
- + 14 probability distributions: *Normal*, *Student's t*, *chi-square*, *Fisher's F*, *Poisson*, *binomial*, *geometric*, *hypergeometric*, *Cauchy-Lorentz*, *exponential*, *logistic*, *Weibull*, and more.
- + 10 models for curve fitting (two kinds of linear, exponential, logarithmic, power, root, hyperbolic, parabolic, Cauchy and Gauss peak).

- + Over 50 fundamental physical constants as accurate as used today by national standards institutes such as *NIST* or *PTB* (following CODATA 2018), plus a selection of important constants from mathematics, astronomy, and surveying.
- + 102 unit conversions, mainly from old *British Imperial* to universal *SI* units and vice versa.
- + Plus a complete set of financial functions and applications for the inevitable business matters.

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

- + Your choice of 4 or 8 *stack registers* and up to 107 global general purpose *registers*, each taking any object of arbitrary *data type* (be it a matrix, a string, or any number of arbitrary kind).
- + Named variables – as many as memory can hold. Also each such variable can take any object of arbitrary *data type*.
- + 112 global *user flags*.
- + 16 local *user flags* and up to 100 local *registers* per program, allowing e.g. for recursive programming.
- + Up to 10 000 program steps in RAM, up to 20 000 program steps in *flash memory*.
- + A high-resolution low-power dot-matrix *LCD* (240 × 400 pixels), showing crisp results and *menus*, allowing for natural matrix display as well as a wide variety of mathematical symbols, Greek and extended Latin letters.

WP 43S is the result of an international collaboration of two teams: SwissMicros (<https://www.swissmicros.com/>) – i.e. Swiss *Michael Steinmann* and Czech *David Jedelsky* – created the hardware, French *Martin Lorang*, German *Gert Menke*, Italian *Gianluca Puggelli*, Dutch *Harald Overbeek*, Australian *Paul Dale*, and me 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* on the market since 2011. We started the *WP 34S* project in 2008. You find specific information about the *WP 34S* 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.

As our *WP 34S* and *WP 31S* before, also *WP 43S* is a hobbyist's project. It started in 2012 and was presented and discussed on <https://www.hpmuseum.org/forum/forum-8.html> until June 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. *Martin* and me presented the first version of the *WP 43S* simulator on the *Allschwil Meeting 2018*. We thank the participants of said meetings and all members of the international community who contributed their ideas, put their votes, and lent their support at various phases 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 these activities.⁵ May it be a worthy and valiant successor of the *HP-42S* – though we would have preferred *Hewlett-Packard* making it (the company 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 it 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. As we did since 2011, we will continue maintaining short response times.

Enjoy!

Walter Bonin

⁴ If you are interested in the long and winding road how your *WP 43S* got the features, shape, and layout you're facing now, see the *Release Notes* in App. 5 at the end of this manual and the two websites mentioned.

⁵ For us, a pocket calculator per definition is a device fitting comfortably in your shirt pocket. Marketing people are observed to see this term more elastic – our shirt pockets are not elastic enough for that. Being at it, we generally recommend not to put such a calculator in the back pocket of your jeans.

Print Conventions and Common Abbreviations

- Throughout this manual, standard text font is Arial. Emphasis is added by underlining or **bold** printing. Calculator COMMANDS, **MENUS**, PREDEFINED VARIABLES and SYSTEM FLAGS are generally called by their *names*, printed capitalized in running text (*menus* underlined). **Quoted text is printed blue** (as well as **translator's notes**).

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 employed for variables; bold regular letters for constant **sample values** (e.g. specific labels, numbers, or characters).

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

Times New Roman bold capitals are used for **REGISTER ADDRESSES**, lower case bold italics for *register contents*. So e.g. the value *y* lives in *register Y* and *r45* in **R45**. Overall stack contents are generally quoted in the order [*x, y, z, ...*]. We keep the term *register* for the space where an individual object is stored, although the actual size of such a *register* may vary widely following the size of the object stored therein.

- This **KEY** font (created by *Luiz Vieira* of Brasil) is taken for references to calculator keys, including **SOFTKEYS** in general. For shifted operations like **GTO** or **LBL**, the respective color is used. **Alphanumeric** and numeric calculator outputs (like 1.234×10^{-56} or $7,089 \cdot 10^{-12}$) are printed as you see them on the calculator screen.
- Courier is used for **file names** and describing numeric formats.
- Regarding mathematical symbols and notation, we generally follow ISO 80000-2. We use decimal points and multiplication crosses in most parts of this manual (but you may set your WP 43S to decimal commas and/or multiplication dots as well, of course). Although that point is less visible than a comma, 'comma people' seem to be more

tolerant against decimal points than ‘point people’ against decimal commas (based on the number of complaints read so far).

All this holds unless stated otherwise locally.

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

<i>AIM</i>	= alpha input mode (see p. 183).
<i>App.</i>	= Appendix .
<i>HP</i>	= Hewlett-Packard .
<i>IOI</i>	= Index of all Items provided in the <i>WP 43</i> (see <i>ReM</i> below).
<i>LCD</i>	= liquid crystal display .
<i>OH</i>	= Owner’s Handbook .
<i>OHPG</i>	= Owner’s Handbook and Programming Guide .
<i>OM</i>	= Owner’s Manual .
<i>PEM</i>	= program-entry mode (see p. 196).
<i>RAM</i>	= random access memory , allowing read and write operations.
<i>ReM</i>	= WP 43S Reference Manual , containing also the <i>IOI</i> .
<i>RPN</i>	= Reverse Polish Notation (see footnote 1 on p. 8 and p. 28).
<i>SI</i>	= système international d’unités , a coherent system of units of measurement agreed on internationally and adopted by almost all countries on this planet. ⁶

Further abbreviations are listed in the *Index* on pp. 318f. A few more may be used and explained locally.

Finally: **WARNING** indicates the risk of severe errors. There are only three warnings printed in this manual although *WP 43S* is distributed in the USA as well. Resetting your calculator will help in almost all cases – but it will erase your data in *RAM*.

⁶ 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 need basic information about *SI* and its foundations, please turn to https://en.wikipedia.org/wiki/International_System_of_Units. See also the chapters about *Constants* and *Unit Conversions* in Section 5 below.

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 and computational 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 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 putting your *WP 43S* ahead of previous scientific pocket calculators.

On the other hand, if this is your first *RPN* calculator, we recommend going through 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 lifelong lasting, reliable basis of your work.

Most commands on your *WP 43S* work 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 the new features of your *WP 43S* and providing information about them. It includes also some formulas and technical explanations; but it is not intended to replace textbooks on mathematics, statistics, physics, engineering, economy, computer science, 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.

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*.⁷

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*.

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 the *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). There you also find instructions for keeping your *WP 43S* up-to-date whenever new firmware revisions will be released. Continue using both 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 analytical 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. Gather information, think before and while keying in and calculating, and check your results: these tasks will remain your responsibilities always. We will not be liable for any of your results.

⁷ Getting an *HP-42S OM* 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 between 1968 and 1990 – on media distributed by the *Museum of HP Calculators* (see <http://www.hpmuseum.org/cd/cddesc.htm>).

SAFETY WARNING: Your *WP 43S* is not designed to be used by children under 3 years. This is not a toy. Do not use it before you can read. Your *WP 43S* contains small parts which if swallowed are hazardous for health. Swallowing the battery can be fatal within 2 hours – seek medical advice immediately!⁸

Do not use your *WP 43S* for any other purposes than specified above (e.g. not as a hammer, a lever, a door stop, or a missile); else you may destroy your *WP 43S* and/or other objects easily, or even hurt animals or human beings including yourself.

Do not drop your *WP 43S* on solid ground – it may break.

Your *WP 43S* shall be operated in a clean and dry environment (relative humidity less than 35%) at ambient temperatures between 5 and 40 °C (41 to 104 °F).

Do not leave your *WP 43S* laying in the sun; its dark surface may become hot, and hot surfaces may cause burns. Do not leave your *WP 43S* laying in the cold; humidity may condense on its surface when a cold *WP 43S* is put in a warmer environment then.

Should your *WP 43S* become wet, turn it off, remove the battery (see *Disassembly* below), and let it dry for sufficient time before turning it on again. Do not try to accelerate drying by blowing hot air (exceeding 60 °C or 140 °F) into your *WP 43S* or by putting it into a microwave oven or the like.

Disassembly: Do not disassemble your *WP 43S* unless you are qualified for such work and have the proper tools handy. You will need a small Phillips screwdriver for opening it.

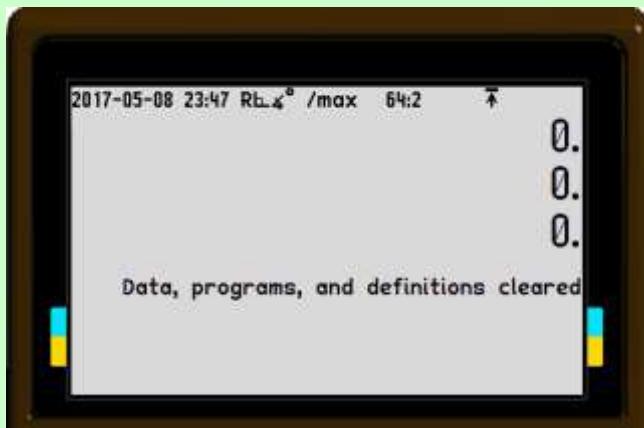
Inserting / replacing the battery: Your *WP 43S* contains a battery. When it runs out of power, open your *WP 43S* (cf. *Disassembly* above) and replace its battery by a fresh one. Dispose of old batteries responsibly in the appropriate in-store containers or at your local disposal center; do neither take them apart nor throw them in a bin nor in fire nor in your environment (cf. top of this page).

Disposal of the calculator: Your *WP 43S* must not be disposed of along with household waste. Dispose of your *WP 43S* according to applicable laws and regulations at your electronics collection point.

⁸ No, we do not think you are stupid, irresponsible, and lack any experience. Blame the respective people of the USA who made such a waste of print space inevitable.

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 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). For conserving battery power, it 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 a *WP 34S* or an *HP-42S*). Your new *WP 43S*, however, looks cleaner than a *WP 34S* while more colorful than an *HP-42S*. This is due to your *WP 43S* featuring two prefix keys **f** and **g** – offering you up to four functions per calculator key – and *menus*.

Looking at an arbitrary one of its 35 black keys, white print is for its *primary function*. For additional (*secondary*) functions, golden and blue labels are printed on the *key plate* above 37 keys, and grey characters are printed bottom right of 30 keys.

For accessing a primary function, simply press the corresponding key. For a golden or blue (a.k.a. **f**- or **g**-shifted) function, first press the respective *prefix* **f** or **g**, then the corresponding key.

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

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

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

- **f** + **x** will call **x!** calculating the *factorial* – e.g. press **9 x!** and you will get

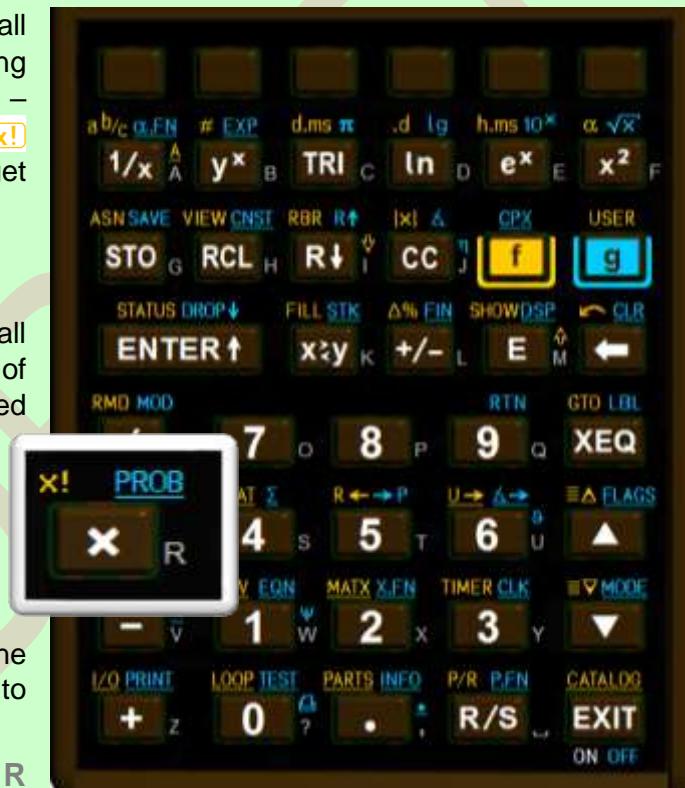
$$9 \times 8 \times 7 \times 6 \times$$

$$5 \times 4 \times 3 \times 2 \times$$

$$1 = 362\,880.$$

- **g** + **x** will call **PROB**, a *menu* of functions related to probability (**EXIT** will let you leave the *menu* again). Each label printed underlined on the keyplate refers to a *menu*.

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

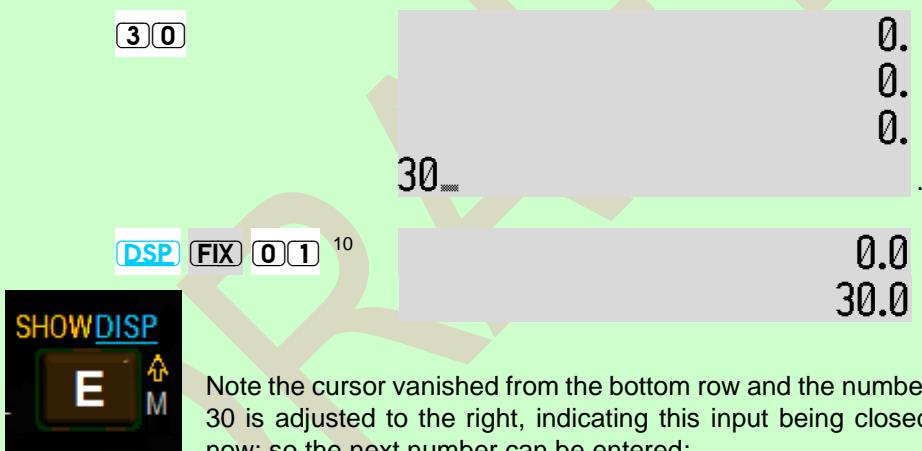


Note all the 143 labels printed on the keyboard of your *WP 43S* are explained top left to bottom right in *App. 2* from p. 289 on.

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 then.

Now, let's assume you want to fence a rectangular patch of land, 40 *yards* long and 30 *yards* wide.⁹ You have already set the 1st corner post (A), and also the 2nd (B) in a distance of 40 *yards* from A. Where do you set the 3rd and 4th 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 this input being closed now; so the next number can be entered:



⁹ We use old *British Imperial* units here so our US-American readers can follow. But rest assured this example will work with *meters* instead of *yards* as well.

¹⁰ Press + to access ; the menu will open in the lower third of the screen. Press the leftmost key for . Then enter telling your *WP 43S* it shall display only 1 decimal digit (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.

→P

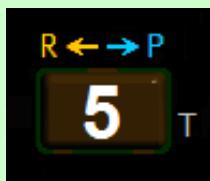
s =

r =

0.0

36.9°

50.0



All you need is the number in the bottom row,¹¹ 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 dis-

tances may apply (you will need a tapeline in the general case). As soon as you press →P, 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 – only 12 moons of Jupiter were known in 1975):

¹¹ 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

x^2 27 688 644

π 3.1

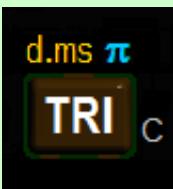
\times 86 986 440.6

diameter of Ganymede

square of the diameter

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

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 (PEM)*.

GTO . .

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

LBL **α** **A** **ENTER↑** This is the opening step of your program named **A** for obvious reasons – for **A** just press **1/x** here as you see a grey **A** printed next to it.

x^2

T

X

RTN

EXIT

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

This is the closing step of the program. Finally, exits *PEM* and returns to *run mode*.

So a straight program on your *WP 43S* consists of an opening **LBL** step and a closing **RTN** 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] PROG 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 **PROG** (it is the 2nd from left) and the *menu* will change. Then press the **□** key directly under **A** and you are done.)

When you press **[XEQ] PROG 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] PROG 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] PROG A**. For example, to compute the surface area of *Jupiter’s* moon *Io* with a diameter of 3643 km:

3 6 4 3

XEQ PROG A

3 643

Io's diameter

3 1 2 2

XEQ PROG A

3 122

Europa's diameter

4 8 2 1

XEQ PROG A

4 821

Callisto's diameter

30 620 739.2

its surface area;

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** ...¹²

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 individual 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.

First, let's demonstrate how to generally enter common decimal numbers in your *WP 43S*. Therefore, please return to *startup default* display format via **DSP ALL 0 0**.¹³

¹² Program **A** as recorded above is a very short one: its center part contains four keystrokes only (**x²** **g** **T** **x**). You may store far, far more keystrokes in program memory – the overall procedure of storing and running programs, however, will remain unchanged. Only the center part of the program will grow.

Programming is comprehensively covered in Section 3 of this manual.

¹³ **ALL** may well be on your screen still since you called **DSP** on p. 19.

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

Numeric 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 ..

You may key in up to 44 digits at once; they are echoed immediately in the bottom numeric row of the screen (note the gap inserted automatically after each group of three digits for easier reading). Any digit mistyped may be erased by **⬅** and can be overwritten then.

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 keyed 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 E 9** which is echoed

13.8×10⁹ ..

in ‘mantissa plus exponent’ format. The key **E** stands for ‘enter exponent’. Note your *WP 43S* allows for a naturally readable display instead of showing you cryptic machine formats like **138E9**.

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:

13 800 000 000 ..

Note the number moved to the right (cf. p. 19). Closed numbers in the bottom row may be cleared at once by pressing **⬅**. This puts **0.** in said row, and subsequent input will overwrite this **0.** then.

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

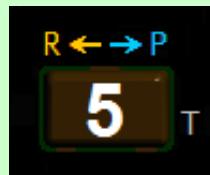
Note you did not have to enter **1 E +/− 1 0** here: If there is no numeric input heading **E**, 1 is assumed for the mantissa per default. And pressing **+/-** after **E** will change the sign of the exponent – if you want to change the sign of the mantissa, press **+/-** before entering **E** or after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

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

How to Enter and Execute Commands

This is easy as well: Just enter the keystrokes required to access the label calling the command you wish to be executed (cf. p. 18). 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 echoed by **f** or **g**, 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 really see during command entry since the next keystroke may well terminate command input already (as observed with **g →P** above), 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. 59f and the *ReM*).

Menus – Items à la carte

Your *WP 43S* features more than 700 operations. The keyboard offers far too little space for showing all of them. Hence most operations live in *menus*. In addition to commands, also arbitrary characters, constants, conversions, digits, programs, *submenus*, system *flags*, or variables defined may be stored in *menus*: we collectively call them *menu items* or simply *items*. By using *menus* we can keep the keyboard relatively tidy.

Your *WP 43S* features 30 *menus* on its keyboard, printed underlined for easy recognition there (except *TRI*).¹⁴ In alphabetic order, these are ADV, BITS, CATALOG, CLK, CLR, CNST, CPX, DSP, EQN, EXP, FIN, FLAGS, INFO, INTS, I/O, LOOP, MATX, MODE, PARTS, PROB, P.FN, STAT, STK, TEST, TRI, U \rightarrow , X.FN, α .FN, Σ , and $\angle\rightarrow$.

Call any *menu* by simply accessing its label (cf. p. 18). This will open the *menu* and cause the lower part of the calculator screen (called the *menu section* from here on) displaying the respective *menu view*.



Example: Press **EXP**

– EXP will open with a *menu view* as pictured here.

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

- the 2nd **[F4]** for **x \bar{y}** ,
- **[F1]** and the 1st **[F2]** for **$\sqrt[3]{x}$** ,
- **[F2]** and the 3rd **[F3]** for **cosh**, the hyperbolic cosine.

¹⁴ We print them underlined 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 would press **f** and the 2nd **□**, nothing will happen since no label is displayed there – no operation is linked to this **f**-shifted **□**.

We may as well print **3fx** if we want to indicate the access path to this **f**-shifted **□** compactly. In analogy, blue background may be printed for a **g**-shifted **□** (like **cosh** here), and grey background for an unshifted **□** function (like **xy** here).

Generally, whenever a *menu* is called, its top *view* will be displayed in the *menu section*. Any such 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 top row of keys (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 *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 stored *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 the up to 18 functions displayed – **until you leave it** (e.g. via **▼** or **▲**, if applicable, or via **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 in the example at top of this page from here on. Note that *submenus* contained in a *menu* are displayed **inverted** without the ‘*menu underline*’. Pressing **EXIT** in a *submenu* will bring you back to its parent *menu* (containing the label of said *submenu*).

How the Keyboard is Organized

You might have already recognized that labels on your WP 43S are printed grouped according to their purposes. Beyond digits and the four arithmetic operations $+$, $-$, \times , and $/$, five larger groups are provided:



Functions for programming and calling programs:
E.g. **XEQ** for executing a program,
R/S for running or stopping it

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

Before showing the operation of your *WP 43S* in detail, let's return to our 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. 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.¹⁵ If you should need to convert special inputs into units being part of such a set or require results expressed in particular units, however, and will help (see pp. 269ff).
3. Although you entered just integers for both edges of your little patch of land in the example on p. 19f, 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 this is beneficial for you. Your *WP 43S* features also more *data types* – we will introduce them to you in *Section 2*.
4. In four decades of scientific pocket computing, a wealth of sample applications has been created and published 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 printed for vintage *HP* calculators in two heroic decades beginning with their very first desktop calculator, the *HP-9100A* of 1968. Be assured that all computations described therein 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 160 new and vintage examples in this manual to support you in learning your new tool.

¹⁵ A quick and simple unit analysis is strongly recommended before starting a calculation of a formula you may have derived yourself. The big advantage of *SI* is that this is the largest coherent set of units available on this planet. Unit prefixes in *SI* simply represent powers of 1000 (see the *ReM* if necessary).

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 0.12 or 3.141 592 653 59 or -5.67×10^{-8} . Note that integers like 3 or 12 345 678 or -121, as well as fractions like $\frac{2}{5}$ or $\frac{137}{7}$ are mere subsets of *reals*.

Depending on the particular command you choose, it may operate on one, two, or three such numbers to generate a result. In spite of the over 700 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* will execute the operation assigned to it immediately (if it requires parameters it will execute with parameter input completed).

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

Ten *monadic* functions are found on the keyboard, starting top left: the reciprocal $\frac{1}{x}$, the logarithms \ln and \lg , the exponentials e^x and 10^x , square x^2 and square root \sqrt{x} , $|x|$ (making negative numbers positive), \pm (multiplying closed numbers by -1), and the factorial $x!$.

Examples:

8	x^2	returns	64	,
$\frac{1}{x}$			$0.015\,625$,
\pm			$-0.015\,625$,
$ x $			$0.015\,625$,
\sqrt{x}			0.125	,
$\frac{1}{x}$			$8.$,
10^x			$100\,000\,000.$,
\lg			$8.$,
e^x			$2\,980.957\,987\,041\,728$,
\ln		returns	$8.$, and
$x!$			$40\,320.$.

Generally, *monadic* functions replace the value (called x) displayed in the lowest numeric row on the screen before calling the function by the respective function result $f(x)$ (e.g. $f(x) = x!$ in the last example). Everything else on the screen stays as it was.¹⁶

Check the *I/O* for the many *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 1234 US\$ and taking 56.7 US\$ away from it. What will remain? An easy way for solving such a problem works as follows:

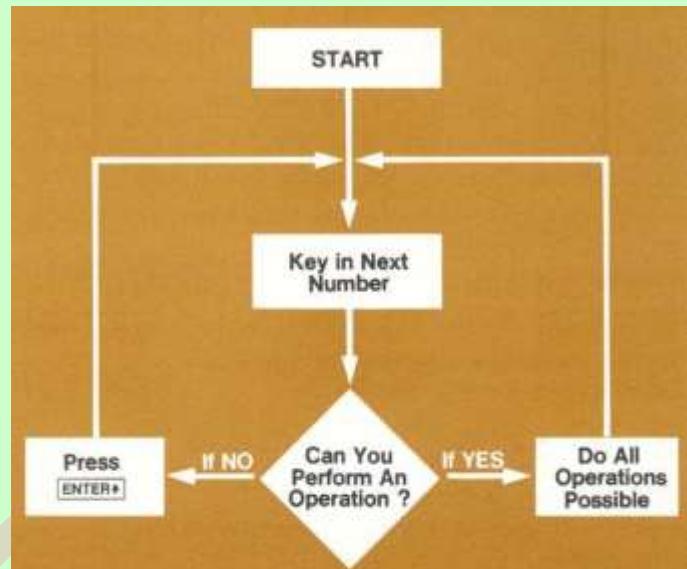
On a piece of paper	→	On your WP 43S
Write down the 1 st number: 1234		Key in the 1 st number: 1 2 3 4
Start a new line.		Close 1 st input: ENTER↑
Write down the 2 nd number: 56.7		Key in the 2 nd number: 5 6 . 7
Draw a line below.		
Subtract: 1177.3		Subtract: -

This is the essence of *RPN*:

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

¹⁶ Your WP 43S features also *monadic* functions operating on other data than *reals* and/or returning different output. These functions work the same way: x will be replaced by $f(x)$, everything else remains untouched.

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



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

Display

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

Bottom up, these *registers* are traditionally called **X**, **Y**, **Z**, and **T**,

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 after 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 *WP 31S*.

optionally followed by **A**, **B**, **C**, and **D** on your *WP 43S*.¹⁹ New input is always entered in **X**, and at least x is always displayed in *run mode* – y , z , and t may be (so you may see the contents of up to four *stack registers* on the screen at the same time if you want).

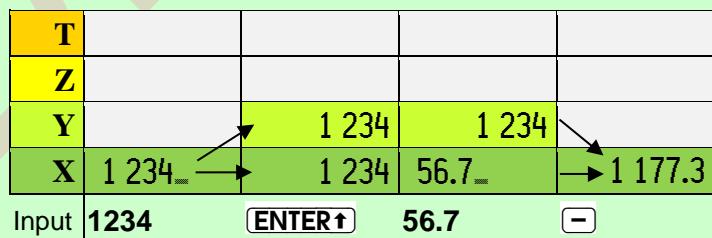
ENTER↑ separates two input numbers by closing the first number x and copying it into **Y**, so **X** can take a new number then without loss

Press	Contents	Location
ENTER↑	$t \xrightarrow{\text{(lost)}}$ $z \rightarrow T$ $y \rightarrow Z$ $x \rightarrow Y$	

of 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 this picture).

Let's look at our account example again, putting it in a *stack diagram*:²⁰



¹⁹ This optional 8-register stack was invented by *Pauli* and me and launched with *WP 34S* in 2011. *WP 31S* features it as well. See the further text for its advantages.

²⁰ 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 beginning, some data may be present in the upper *stack registers* **Y**, **Z**, and **T**, remaining from earlier operations. These data are irrelevant for this calculation, so we left them aside here; in further *stack diagrams* we will omit entirely all *stack registers* not containing any data relevant for the particular calculation, for sake of clarity and print space.

After having entered the 2nd number (**56.7**, the new x), pressing **[-]** subtracts this from the 1st number (**1234** in Y) and puts $f(x, y) = y - x = \mathbf{1177.3}$ in X. This procedure applies to the overwhelming majority of functions featured on your WP 43S:

**Put the operands on the stack,
then execute the operation $f(x, \dots)$,
and the result will be displayed.²¹**

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, one 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 while solving:

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 1st parenthesis in the numerator was solved exactly as demonstrated in our little account example above. Now proceed to the 2nd parenthesis:

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]

²¹ This completely eliminates the need for an **[=]** on the keyboard.

It is solved like the first. Though in the 1st step of this sequence, the prior result (of 1st 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). *Automatic stack lifting* is standard on *RPN* calculators, reducing the number of keystrokes necessary, and will not be indicated from here on anymore.²² Remember you generally need pressing **ENTER↑** just for separating two consecutive numbers in input – cf. the flow diagram on p. 32.

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

After having solved the 2nd parenthesis of the chain calculation 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 1st step:

	-2 667.33	-2 667.33		-2 667.33			
-2 667.33	3.4	3.4	-2 667.33	-2.2	-2 667.33		
3.4	3.4	5.6		-2.2	7	-249.43...	10.693...
3.4	ENTER↑	5.6		-	7	y^x	/

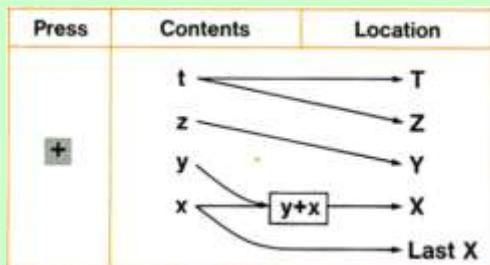
²² Also an *automatic stack lift* affects all *stack registers*, and the previous content of the top *register* gets lost again. Of all commands provided on your *WP 43S* (more than 700), there are only 4 disabling automatic stack lift: **ENTER**, **CLX**, **Σ+**, and **Σ-**. Some reasoning:

- 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 a useless extra zero on the stack).
- Regarding **Σ+** and **Σ-**, please see the chapters about statistical functions in *Section 2* for the reasons.

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.

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* as pictured here:



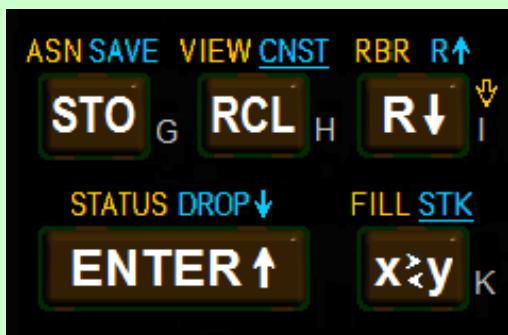
x and *y* are combined resulting in $f(x, y) = y + x$ loaded into **X**; then *z* drops to **Y**, and *t* to **Z**; since nothing is available above *t* on a 4-register stack for dropping, the top *register* content is repeated (see also p. 39 – ‘Last X’ will be covered on p. 49).

There are also a few **three-number (*triadic*) functions** featured (e.g. $\rightarrow\text{DATE}$). 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 (see p. 39 for an example). All *triadic* functions provided on your *WP 43S* are found in *menus*.

And some functions operate on one number but return two (like *DECOMP*) or three (e.g. *DATE* \rightarrow). Other operations do not consume any *stack* input at all but just return one, two, or three objects (like *RCL*, *SUM*, or *L.R.*). 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 will 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 memory control operations **STO**, **RCL**, **R↓**, **ENTER↑**, **x↔y**, **VIEW**, and **R↑** are known from previous *RPN* calculators for decades already. They are all found within this small area of the keyboard, together with **RBR**, **FILL**, **DROP↓**, and **STK**.

Your *WP 43S* contains even more *stack* and *register* commands, e.g. CLSTK, CLREGS, DROPy, STOCFG and RCLCFG, STOS and RCLS, $x\ddot{z}$, $y\ddot{z}$, $z\ddot{z}$, $t\ddot{z}$, and \ddot{z} . Most of them are found in STK.

And your *WP 43S* allows for expanding the traditional *4-register stack* to eight *registers*: just enter **FLAGS SF SYS.FL SSIZE**.

In consequence, the fate of *stack* contents will depend on the particular operation executed as well as on the *stack size* set at execution time. Operations on the *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* (see p. 43 for an example).

Please find below what **ENTER↑**, **FILL**, **DROP↓**, **DROPy**, **x↔y**, **R↓**, **R↑**, and further representative functions do in detail on *stacks* of either size. Then you will also know why **ENTER↑** and the *stack* rotation command **R↑** show arrows pointing up while **R↓** and **DROP↓** point down.

		Stack register	Assumed initial contents	Stack contents after executing ...							
				[ENTER↑]	[FILL]	[DROP↓]	[DROP↑]	[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.	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.	2.	8.	

Clearing the entire *stack* can be done by [0] [FILL] most easily. Nevertheless, a dedicated command CLSTK is provided in CLR for backward compatibility and program space economy (see p. 51).

[x↔y] takes the initial *stack* contents (as listed in the third column left) and swaps the contents of *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 [yx].

[R↓] and [R↑] may come handy for reviewing *stack registers* else unseen (or use the register browser RBR – see Section 5).

Stack register	Assumed initial contents										
		[RCL] ²³	[1] ²⁴	[s] ²⁴	[x] ²⁵	[+] ²⁶	[DATE] ²⁷	[DATE] ²⁸			
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	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.	4.	4.	10.			
X	$x = 1.1$	last x	s_x	1.21	3.1	1-02-03		1			

[**RCL**] ^L represents the vintage command LAST x (see p. 49 for more about it). Note that the previous contents of the top stack register are lost when [**ENTER**↑] or [**RCL**] are executed. Functions like [**s**] or [**DATE**→]

²³ This represents an arbitrary function pushing one object on the stack.

²⁴ This represents an arbitrary function pushing two objects on the stack.

²⁵ This represents an arbitrary *monadic* function.

²⁶ This represents an arbitrary *dyadic* function.

²⁷ Assume →DATE is called in *startup default mode* (i.e. YMD). It represents an arbitrary *triadic* function here.

²⁸ Assume **1.102** or **1-10-20** in X initially here and *startup default mode* set, cf. Sect. 2.

will even cost the contents of two *stack registers*. We recommend mitigating the effects of such losses by setting the *stack* to eight *registers*. – Please see the *I/OI* for further information about the commands mentioned above.

Problem Solving, Part 3: The Stack in Advanced Calculations

Using the *stack* as described, *RPN* eliminates the need for an **=** key as well as for any parentheses **(** **)** keys! See the following **example**, showing a more elaborate formula than above. Find below a way for solving it step by step, and the corresponding *stack diagrams*. Enter **MODE RAD** and start at 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...	0.91...
	7	[ENT↑]	4	/	5	[ENT↑]	6	/	-

								0.25...	0.25...
				0.25...	0.25...		0.25...	12.4	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** **+** **x²** **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...	4_

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^y - 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...

|x|²⁹ .3 y^x 1 + x^y / x^y 2 +

Even solving this formula requires only four *stack registers*.³⁰ Note there are no pending operations – each operation is executed individually, one

²⁹ You would not execute this step manually since you will see immediately that x is positive. In an automatic evaluation of such a formula, however, this step is important unless you know in advance that a negative intermediate result will not occur.

³⁰ We admit we were cautious seeing this formula and set SSIZE before starting the calculation here.

Additional remark: In the fifth step of last diagram, we have got the complete result for the numerator in \mathbf{X} . And the result for the denominator is in \mathbf{Y} whereto it silently traveled during all the other calculations (see above). In the subsequent step, we swapped x and y to put the operands in proper order for division of the numerator y by the denominator x .

at a time, allowing **perfect control of each and every intermediate result.**³¹

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.

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 tried solving 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 entire solution instead of only four (the colors in the record below represent the top *stack register* involved in each step):

The sequence of key presses is as follows:

- 1 [ENTER↑]
- 30 [ENTER↑]
- 7 [/]
- 7.6 [ENTER↑]
- .8 [x]
-
- 4 [y^x]
- 5.1 [√x]
- 6 [ENTER↑]
- 5 [/]
-
- x²
-
- |x|
- .3 [y^x]
- +
- RAD [RAD]
- π [PI]
- [ENTER↑]
- 7 [ENTER↑]
- 4 [/]
- 5 [ENTER↑]
- 6 [/]
-
- x
- sin [sin]
- 1.7 [ENTER↑]
- 6.5 [ENTER↑]
- 5.9 [+]
- 3 [ENTER↑]
- 7 [/]
- y^x [y^x]
- x [x]
- +
- x² [x²]
- 3.5 [-]
- / [√x]
- 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:

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

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]	[x]	3

T					51	51					
Z	51	51		51	7	7	51				
Y	3	3	51	7	11	11	7	51			170
X	3	4_-	7	11_-	11	6_-	17	119	170	5_-	
	[ENT↑]	4	[+]	11	[ENT↑]	6	[+]	[x]	[x]	[+]	5

T					170	170					170
Z	170	170		170	-2	-2	170			170	-44
Y	5	5	170	-2	10	10	-2	170	-44	13	
X	5	7_-	-2	10_-	10	12_-	22	-44	13_-	13	
	[ENT↑]	7	-	10	[ENT↑]	12	[+]	[x]	[x]	13	[ENT↑]

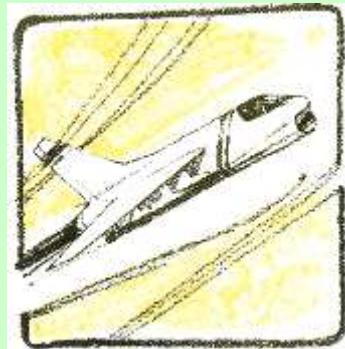
A				170	170						
T	170		170	-44	-44	170					
Z	-44	170	-44	27	27	-44	170				
Y	13	-44	27	15	15	27	-44	170			
X	14	27	15_-	15	16_-	31	837	-881	-0.19296...		
	14	[+]	15	[ENT↑]	16	[+]	[x]	[x]	[x]	[x]	[/]

If you had set your *WP 43S* to four stack registers, however, (as all of HP's pocket calculators featured 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	-44	
Z	-44	170	-44	27	27	-44	-44	-44	-44	-44	
Y	13	-44	27	15	15	27	-44	-44	-44	-44	
X	14	27	15_-	15	16_-	31	837	-881	0.04994...		
	14	[+]	15	[ENT↑]	16	[+]	[x]	[x]	[x]	[x]	[/]

Then it would return a wrong result due to *stack overflow* in step 4 above

and subsequent repetition of wrong top *register* contents. Note this is possible – and there is (and will be) no warning since your *WP 43S* cannot know what you still need or what may be discarded without a problem.³² Thus, we recommend setting *SSIZE* 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*³³ (here: 350) and *pressure altitude (PA)* in *feet* (here: 25 500) was used for demonstrating the simplicity and coherence of *RPN*:

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

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 [-]
```

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

³² 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 stay nonzero further on. See previous page and trick #1 three pages below.

³³ The *knot* is an ancient *British Imperial* unit of velocity surviving in aviation business and navigation: $1 \text{ knot} = 1 \text{ nautical mile/hour} = 463/900 \text{ m/s} \approx 0.5144 \text{ m/s}$ $\approx 1.85 \text{ km/h}$. The *foot* is another one from that heap of pre-modern units made obsolete by *SI* for long (see [U→](#) in Section 5). We quote this US-American *Mach-number* formula without having verified it.

Translator's note for Europeans: *CAS* does not mean C·A·S, and *PA* does not mean P·A !

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 8-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. Promised!³⁴

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

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.

³⁴ Of course, constructing an example leading to *stack overflow* even for eight *registers* is trivial. But first of all it 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.

- You can calculate in the same order you do with pencil and paper.
You don't have to think the problem through ahead of time.

RPN takes a few minutes to learn. But you'll be amply rewarded by the ease with which the *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:

**Set sSIZE and
let your *WP 43S* care for the arithmetic
while you care for the mathematics!**³⁵

³⁵ You might ask: With the opportunity of an *8-register stack*, why are there only up to four *stack registers* displayed, 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. Thus, watching the entire *stack* mechanics reliably working all the time does not carry any valuable information 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 always. Users were doing all sorts of tricks on that *stack* – just tracking *y*, *z*, and *t* in their minds. Even *HP*'s *RPL* calculators (although they feature a so-called 'infinite' *stack*) did not display more than four *registers*.

Assuming people's mental abilities did not deteriorate generally in 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 recommendations above. For the same reason, we omit heading indicators *X*, *Y*, *Z*, and *T* in display. Since you chose this calculator for yourself, you are obviously able to remember these four letters naming the bottom four *stack registers*.

On the other hand, if you feel distracted or even annoyed by the screen showing more than necessary, you may reduce the number of *stack registers* displayed to three, two, or even just one (using *DSTACK*), letting your brains compete with the ones of your fellow users since 1972. Free space will flow in the display top down – *x* will always be displayed directly above the menu section. And multi-line output will be shown entirely always, regardless of current *DSTACK* setting.

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 pp. 36 and 39. You may employ this *top stack register repetition* for some nice tricks.

See the following compound interest calculation:³⁶

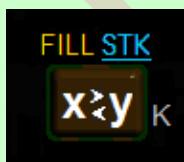
Example:

Assume your bank pays you 3.25% p.a.³⁷ on an amount of 15 000 US\$; what would be your status after 2, 3, 5, and 8 years?

Solution:

Here, you are interested in currency values only, so set the display format by **DSP FIX 2**. This causes the output being rounded to cents (internally, numbers are kept and calculated 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			
	1.0325	FILL	15000	x x	x	x x	x x	x x x			
			after	2 years	3 years	5 years	8 years				



Each multiplication consumes *x* and *y* for the new product *x × y* put in **X**, followed by *z* dropping into **Y**, and *t* copied into **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.

This is demonstrated here for a *4-register stack*. It works for an *8-register stack* as well – with the contents of **D** repeated.

³⁶ Translator's note for German readers: *Compound interest* = Zinseszins.

³⁷ Those were the times, my friend... ! Inflation was balancing those interest rates but saving was definitely more fun then, nevertheless.

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

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

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \\ = (\dots(a_nx + a_{n-1})x + \cdots + a_1)x + a_0$$

Example:

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

Solution:

This problem can be rewritten to

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

and is easily solved this way (with the display set to **DSP FIX 0 1**):

	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	0.9	0.9	0.9	0.9	0.9	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

FILL loads the entire *stack* always – be it 4 or 8 *registers* deep – it spares you the necessity 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?



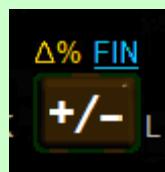
Example (from the *HP-15C OH* of 1987):

Two close stellar neighbors of Earth are *Rigel Centaurus*³⁸ (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 $9.460\ 54 \times 10^{15}$ meters/year), to figure the distances to these stars in meters.

Solution (with SCI 1 set):

4.3	4.3	4.3
[ENTER]↑		
9.46073 [E] 15	$9.460\ 54 \times 10^{15}$	
[x]		4.1×10^{16}
8.7	8.7	
[RCL] L		9.5×10^{15}
[x]		8.2×10^{16}

[RCL] L is reached by pressing [RCL], then [+/-]; note the grey L printed bottom right of [+/-].



Result: *Rigel Centaurus* has a distance of 4.1×10^{16} m (or 4.1×10^{13} km) to our planet, *Sirius* 8.2×10^{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.³⁹

³⁸This is identical with *Alpha Centauri*. *Rigel* usually means a star in constellation *Orion*.

³⁹There are only very few commands changing x but not loading L. Those are mentioned explicitly in the IOI. – Allocating a dedicated label to LASTx on the keyboard (like on the *HP-42S*) would not pay here since no keystrokes will be saved.

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 that error occurred.

1. If you receive an **error message** in response to your function call, press or **EXIT**; this will erase that message and return to the state before that error happened (see pp. 67 and 301f). Then do it right! 
2. If you have erroneously executed a **wrong function**, just press to undo it immediately. recalls the entire calculator state as it was before that wrong operation was executed. Then resume calculating where you were interrupted.⁴⁰

Example:

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

T	yzx	...	yzx
Z	xyz	yzx	xyz	yzx	yzx
Y	numerator	xyz	num	xyz	xyz
X	denominator	<i>num × den</i>	<i>den</i>	<i>num / den</i>	<i>correct result</i>

Oops! Undo
Fine so far. Undo Resume

So don't worry – be happy!

⁴⁰ operates on everything but program memory. Note, however, will not revert any operations you have confirmed explicitly (like **RESET**). And will undo the very last operation before only, nothing more – i.e. = .

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

Clearing and Resetting Your WP 43S

There are several ways you can remove obsolete information from your WP 43S. The most basic one is  – you have learned about it on p. 24. Almost all other clearing commands are contained in CLR:



CLX	Clears stack register X (i.e. sets it to zero)	CLSTK	Clears all stack registers
CLΣ	Clears all statistical registers	CLREGS	Clears all global and local g.p. registers ⁴¹
CF	Clears the flag specified	CLFALL	Clears all user flags
CLP	Clears current program	CLPALL	Clears all programs
CLMENU	Clears the programmable menu	CLCVAR	Clears all variables of the current program
CLALL	Clears all programs and data (variables, user flags, and all registers including the stack) ⁴²	RESET	Resets your WP 43S to startup default (just flash memory contents will stay untouched) ⁴³

For your reference, *startup default* settings are:

2COMPL, ALL 0, DEG, DENMAX 9999, DSTACK 4, GAP 3, J/G 1752-09-14, LinF, LocR 0, RM 0, TDISP -1, WSIZE 64, and Y.MD.

The system flags ALLSCI, AUTOFF, DECIM., DENANY, MULTx, RECTN, TDM, and aCAP are set, all others are clear.

Red commands ask you for confirmation. Turn to the ReM for more information about the commands and *system flags* mentioned above.

⁴¹ G.p. = general purpose. Find more about g.p. registers in next chapter. Note stack and statistical registers as well as variables are not touched by CLREGS.

⁴² Note display formats as well as other user settings and assignments will remain unchanged. Only RESET clears everything except flash memory (see Sect. 3 and 6).

⁴³ If you cannot reach RESET for any reason whatsoever, a hard reset will do the same. Use the RESET hole on the calculator back side.



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.



That's almost all you have to know about number crunching for the time being – calling commands and calculations with *reals* on the *stack*. Such capabilities did suffice for high flying applications already – see the picture above. 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

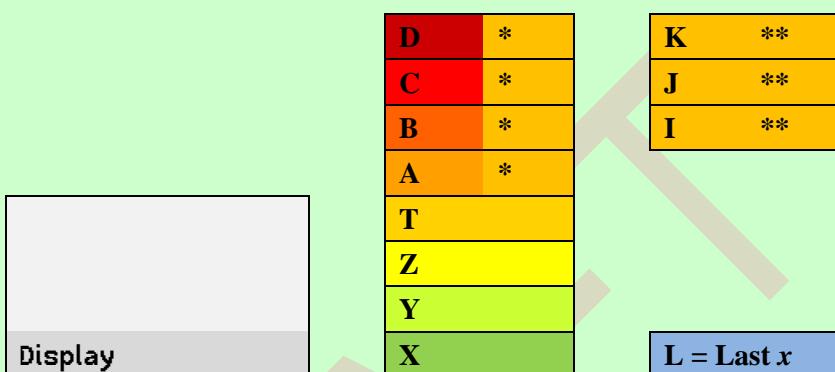
You have learned about the *stack* providing work space and temporary storage during your calculations. For long term storage, feel free to use other *registers*, variables, *flags*, and program memory. The remaining chapters of this section will tell you how to use the first three.

The pictures on the next two pages 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 applicable. **I**, **J**, and **K** may carry parameters of statistical distributions (see pp. 94ff);

I and **J** will also be used in matrix editing (see pp. 158ff), and **K** is the also the default *alpha register* for some special operations (see pp. 223f). Unless required for the purposes mentioned, **A**, **B**, **C**, **D**, **I**, **J**, and **K** may be employed as additional global general purpose *registers*.

Special registers and stack



Turn overleaf to see all *registers* available as well as all *flags*. Generally, *registers* or *flags* can be addressed as shown in the tables on pp. 59ff. Addresses ≥ 112 are used for local data (see pp. 226f).

Flags are elementary *items* having only two states, *set* and *clear*. You may think of them as switches being either on or off. *User flags* you can employ for signaling whatever you want. There are also *system flags* reflecting specific system states. Since *flags* are most useful in programming, they will be dealt with in Section 3.

Statistical data are accumulated in a set of dedicated summation *registers* not interfering with your other data (like in 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).

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

General purpose registers

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

Local registers

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

Global registers

Program steps

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

User flags

.15	= 127
.14	= 126
...	
...	
.00	= 112
K	= 111
J	= 110
I	= 109
L	= 108
D	= 107
C	= 106
B	= 105
A	= 104
T	= 103
Z	= 102
Y	= 101
X	= 100
99	
...	
...	
02	
01	
00	

Example (with *startup default* settings):



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 1st factor of the expression,

[RCL] K [\ln]

solves the numerator,

[\times]

multiples,

[RCL] K [TRI] [\cos]

solves the 2nd part of the denominator,

[/]

divides by it,

4 [/]

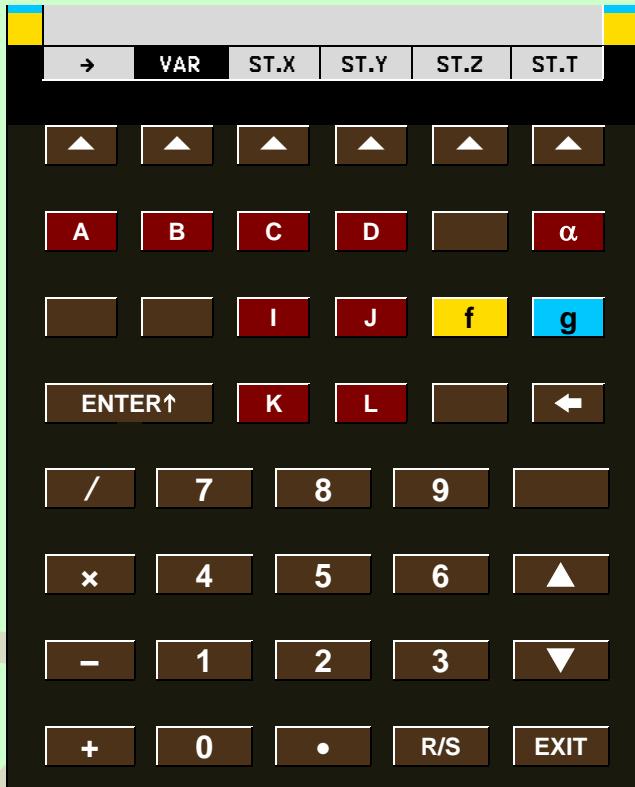
divides by four finally, returning 0.559 63...

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

Variables are named storage locations. As well as each *register*, also each variable can hold any type of data (see *Section 2*).

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 labels plus the *prefixes* will do instead. The calculator mode supporting these 29 exclusively is called *temporary alpha mode (TAM)*. As shown in examples on the next pages, it may be automatically set in memory addressing.

Entering *TAM*, the operational keyboard is temporarily reassigned as pictured overleaf.



This kind of picture is called a ***virtual keyboard*** since it may deviate from the physical keyboard of your *WP 43S*. In such a picture, **dark red** background is used to highlight changed key functionalities. White print denotes *primary* functions also on *virtual keyboards*, such as the left key in row two directly accessing (*stack*) register **A** in *TAM*.⁴⁴

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 indirect addressing (see p. 59), ⓧ for local memory addresses here (see p. 226).

Variables already defined at execution time will show up in the *submenu VAR* in alphabetical order – so you can select the variable of your choice by pressing the respective *softkey*. You can also access them via @ (or create new variables this way – see pp. 59f 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,⁴⁵ for

⁴⁴ What is printed white on your physical *WP 43S*, on the other hand, is called a *default primary function*.

⁴⁵ Comparisons are most useful in programming – see *Section 3*.

instance, the **f**-shifted row will look like this:

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

This allows for directly comparing x with the numbers 0 or 1 (see p. 59).

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

	STO	__	
	...EL	...IJ	
Config	Stack		
→	VAR	ST.X	ST.Y
		ST.Z	ST.T
		Max	Min

This allows for storing and recalling all your specific settings easily via **STO Config** and **RCL Config**, respectively (see p. 78). **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 the contents of the source automatically (see the *IOI*). You may press **▲** as shortcut for **f Max** and **▼** for **f Min** here. **...EL** and **...IJ** may be helpful when dealing with matrices (see *Section 2*).

For commands operating on *flags*, **SYS.FL** will be displayed instead of **VAR**, granting access to the system *flags* provided.

For almost all other operations requiring trailing parameters, the *menu* will stay with a single row of *softkeys* as pictured on p. 56.

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**; this will leave *TAM* immediately, returning to the mode set and the *menu* displayed before, if applicable.

If you just want to look up the current contents of a storage location without disturbing the stack, use **VIEW**.

Example:

DSP FIX 5

VIEW K

returns

$\kappa = 3.374\ 98$

$0.000\ 00$

$0.000\ 00$

$0.559\ 63$

... as expected from previous example.

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

For inspecting a row of various *registers*, take **RBR** instead; press **STATUS** (or **FLAGS STATUS**) for checking the status of all *flags* (RBR and STATUS are explained in *Section 5* from p. 254 on).

- ☞ You are granted unlimited access to all the global *registers* and *user 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.⁴⁶
- ☞ You will not get 10 000 program steps and 212 *registers* and 128 *user flags* all together at the same time – see the *ReM, App. B*, for the reasons and for resource management.

⁴⁶ In *Section 3*, you will learn about a method preventing your programmed routines from interfering with data of other programs.

Addressing Tables

Parameterized Comparisons:

1 User input		[TEST] $x < ?$, $x \leq ?$, $x = ?$, $x \approx ?$, $x \neq ?$, $x \geq ?$, $x > ?$			
Echo		OP _ ? (with TAM set), e.g. $x < _ ?$			
2 User input	0. or 1.	<i>Stack or lettered register</i> ST.Y , ..., D , L , I , J , K , or variable defined ⁴⁷	<i>Register number</i> (range as specified on p. 62)	opens indirect addressing	[α] ⁴⁸ turns on alpha input mode (see pp. 187ff) for a (new) variable name
Echo	OP n ? e.g. $x = 0. ?$	OP? x e.g. $x \geq ?$ ST.Y	OP? nn e.g. $x \neq ?$ r23	OP? \rightarrow _	OP? ' _
3 User input	Compares x with the number 0 .	Compares x with the content of stack register Y .	Compares x with the content of R23 .	See overleaf and p. 63 for more about indirect addressing.	Variable name (see overleaf for more) OP? 'xx' e.g. $x > ?$ 'ST1'

Compares x with the content of the variable called 'ST1'.

Press **[g] [TEST] $x > ?$ [α] [S] [T] [**f**] [**1**] [ENTER↑]** for this.

⁴⁷ It is recommended calling variables being already defined via **VAR** instead of keying them in using **[α]**.

⁴⁸ Note you can skip pressing **[f]** here (cf. the *virtual keyboard* above).

Register operations (requiring just registers or variables trailing):

1 User input	RCL , STO , VIEW , (x) , (y) , (z) , (t) , DEC , DSE , DSL , DSZ , INC , ISE , ISG , ISZ , etc.			
Echo	OP _ (with TAM set), e.g. RCL _ ⁴⁹			
2 User input	Stack or lettered register ST.X , ..., K , or variable defined ⁴⁷	Register number (range as specified on p. 62) OP nn e.g. VIEW 10	OP → _ opens indirect addressing where applicable (see p. 63 and the I/O)	OP α ⁴⁸ turns on alpha input mode (see pp. 187ff) for a (new) variable name
Echo	OP x e.g. DEC K		OP → _	OP ‘_’
3 User input	Decrement <i>k</i> . Register number (range as specified on p. 62) OP → nn e.g. STO →45			
Echo		OP → x Stack or lettered register or variable defined ⁴⁷ e.g. x → L		Variable name (up to 7 characters incl. one letter at least) ⁵⁰ OP ‘xx’ e.g. INC ‘Zähler1’

Stores *x* in the location where *r45* is pointing to (see p. 63).

Swaps *x* and the content of the register where *I* is pointing to.

Increments the variable called **Zähler1**.

⁴⁹ For **RCL** and **STO** only, any of the keys **+**, **-**, **×**, **/**, **▲**, or **▼** may precede step 2 here. Entering such a key twice will cancel it (e.g. **RCL ▲ ▲** equals **RCL**). See the chapter after next chapter for more about this *store and recall arithmetic*.

Such operators are not allowed in **RCL Config** (calling RCLCFG), **RCL Stack** (calling RCLS), RCLEL, RCLIJ, and the corresponding store operations, however.

⁵⁰ This name must be unique. If a variable with this name is not defined at execution time yet, it will be created automatically, containing zero initially.

Clearing an individual *register* or *variable* is most easily done by storing zero in it. Deleting a variable from memory is demonstrated on pp. 283f.

Other operations requiring trailing parameters:

1 User input	ALL , FIX , SCI , ENG , ASR , CB , FB , SB , BestF , CF , FF , SF , DSTACK , ERR , LocR , PAUSE , RM , SIM EQ , TDISP , TONE , WSIZE , →INT , bit and <i>flag</i> tests, etc. (see the <i>I/O</i> for a complete list)		
Echo	OP _ (with <i>TAM</i> set), e.g. FIX _		
2 User input	<i>Lettered flag</i> A , ..., K , or system <i>flag</i> or <i>variable defined</i> , ⁵¹ if applicable	<i>Flag number</i> or <i>number of bit(s)</i> or <i>decimals</i> (see p. 62 for valid ranges) or any other number applicable	→ opens indirect addressing where applicable (see p. 63 and the <i>I/O</i>)
Echo	OP x e.g. SF J	OP nn e.g. SCI 12	OP → _
3 User input	Sets <i>flag</i> 110.		Register number (range as specified on p. 62)
Echo	OP → nn e.g. DSTACK → 12		Stack or <i>lettered register</i> <i>or variable defined</i> ⁴⁷
	Shows as many stack levels as specified in R12 (see p. 63).		OP → x e.g. FIX → A
			Sets fix point format with # of decimals stored in A .

⁵¹ Where applicable, it is recommended calling *system flags* via **SYS.FL** and variables already defined via **VAR** instead of keying them in using **0A**.

	Valid number range ⁵²
<i>Registers</i>	<p>0 ... 99 for direct addressing of <u>global numbered registers</u></p> <p>.099 for direct addressing of <u>local registers</u></p> <p>0 ... 211 for <u>indirect addressing</u> (≤ 111 without local <i>registers</i>)</p>
<i>Flags</i>	<p>0 ... 99 for direct addressing of <u>global numbered user flags</u></p> <p>.015 for direct addressing of <u>local user flags</u> if allocated</p> <p>0 ... 127 for indirect addressing (≤ 111 without local <i>user flags</i>)</p>
Decimals	0 ... 15 (entering any digit except 0 or 1 will terminate waiting for a further digit and close input)
Integer bases	2 ... 16
Bit numbers	1 ... 64
Word size	1 ... 64 bits

Please see the *ReM* for all other parameters and their valid ranges, as well as for a list of all *system flags*.

⁵² Specifying low numbers (and numeric addresses), you may key in e.g. **5** **ENTER↑** instead of **0** **5**. Remember some *registers* and *user flags* may also be addressed by letters. Variables and *system flags* are generally called by their *names*.

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 may supply the *register* or variable pointing to the actual parameter.

Example:

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

STO **J**

RCL **→** **J**

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

FLAGS **SF** **→** **X** will set flag 8, while...

DSP **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.⁵³ And each and every register can be accessed this way (also the *stack*). Indirect addressing is most beneficial in programs when the parameter for a function is calculated (see Section 3, also for examples).

Store and Recall Arithmetic

As mentioned in footnote 49 on p. 60, arithmetic (and 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.

⁵³ Several vintage calculators, on the opposite, featured just a single dedicated *register* for indirect addressing if at all. See the *HP-34C* or *HP-15C*, for instance.

Example for store arithmetic:

123.4

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

The same result could be achieved by the keystroke sequence

123.4

RCL **K**

x \gtrless y

-

STO **K**

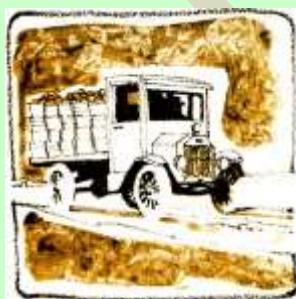
RCL **L**

but that is far clumsier (replacing one step by five) and would cost one *stack register* in addition.

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

$$\text{new content of the register or variable specified} = \text{old content of this register or variable} \left\{ \begin{array}{l} + \\ - \\ \times \\ / \\ \max \\ \min \end{array} \right\} x$$

Example (from the HP-67 OHPG 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:

DSP **FIX** **2**

25 **ENTER** **↑** 27 **+**

Total of Monday's & Tuesday's tonnage

19 **+** 23 **+**

94.00

55 **×**

5 170.00

Gross amount for these days

STO **J**

5 170.00

Take **J** for accounting

2 [FIN] %	103.40	Deduction for these days
[STO] - [J]	103.40	Subtracted from the total in J
26 [ENTER] ↑ 28 [+]	54.00	Wednesday's tonnage
57.5 [X]	3 105.00	Gross amount for Wednesday
[STO] + [J]	3 105.00	Added to the total in J
3 [%]	93.15	Deduction for Wednesday
[STO] - [J]	103.40	Subtracted from the total in J
[RCL] [J]	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. **L** is loaded with **78.91**. The rest of the *stack* and **R23** stay unchanged.

Alternatively, the same result could be achieved by the sequence

78.91

[RCL] [2] [3]

/

but that would replace one step by two and also cost one additional *stack register*. And **L** is different here.

General rule for recall arithmetic:

$$\text{new } x = \text{old } x \left\{ \begin{array}{l} + \\ - \\ \times \\ / \\ \max \\ \min \end{array} \right\} \text{ content of the register or variable specified}$$

Stack-wise, both 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. 211ff for more examples and advantages and the *IOI* for further details.

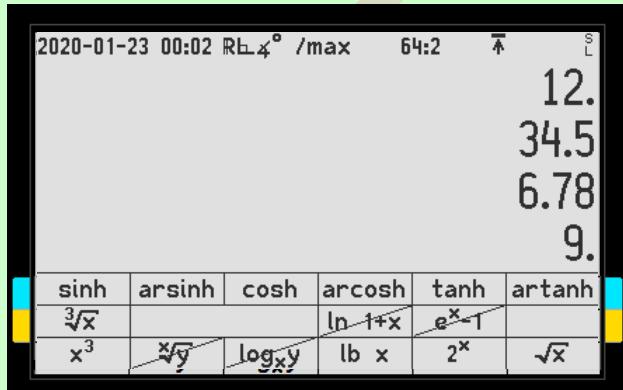
Although these techniques have been more important in times when program memory was very limited, they may be still beneficial today.

SECTION 2: DEALING WITH VARIOUS OBJECTS AND DATA TYPES

Some Display Basics

The screen 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 more – see points 1 to 4 below), and
- the *menu section* displaying up to three rows of soft-keys (cf. pp. 26f).



The numeric rows deserve some additional explanations first – the *status bar* will be covered further below:

1. The **left side of the top (T) numeric row** is also used for output of *VIEW* (cf. p. 58) and *SHOW* (see the *I/O*) and for echoing command input until completed, i.e. until all the required command parameters are entered and the command can be executed. *Prefixes* (like **f** and **g**) will be displayed (using **f** and **g**) until they are resolved (if, however, you pressed **f** or **g** erroneously, recovery is as easy as **f f = g g = NOP**). And you may edit any pending operation character by character using **⬅** or cancel it by **EXIT** (cf. p. 57).
2. The **left side of the Z numeric row** is used for displaying any error message or the output of a binary test, if applicable. Then, pending command input will stay in the top numeric row.

3. The **left** side of the **Y numeric row** is used for displaying additional (temporary) information heading *y*, if applicable.
4. The **left** side of the **bottom (X) numeric row** is used for...
 - a. echoing numeric or alphanumeric input (see pp. 24 or 187ff). Note it can take up to 41 digits, a sign, and a radix mark in *startup default* numeric format or some 40 alphanumeric characters. You may edit pending input character by character using . Numeric input will be checked and interpreted as soon as it is completed and closed, according to the calculator settings at closure time.
 - b. showing additional (temporary) 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**.⁵⁴ It will vanish with the next keystroke you enter: pressing or **EXIT** will just clear messages, returning (for $DSTACK > 2$) to the pure display of *x, y*, and *z* – any other key will be executed in addition, if applicable.

Supported Data Types

You learned how your *WP 43S* calculates with *real numbers* in Sect. 1. It can do more for you: it can deal with *integers*, *fractions*, and *complex numbers* as well as *angles*, *times*, and *dates* in various formats.⁵⁵

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

⁵⁴ If you choose less than three *stack registers* to be displayed (see *DSTACK*), *temporary information* will nevertheless show up at the positions mentioned above, whenever applicable. And operations resulting in multiple output rows will display their entire output independent of the *DSTACK* setting always.

⁵⁵ Furthermore, it can also deal with *real* and *complex vectors* and *matrices* as well as with *alphanumeric character strings*, – these *data types* are covered comprehensively in dedicated chapters further below in this section.

All *data types* provided are listed in the *ReM*.

Input	Display	Meaning
12345.678901 EXIT	12 345.678 901	<i>Real numbers</i> , see pp. 78ff
12 E 345 ENTER↑	12×10^{345}	
123.45678901 d.ms	$123^{\circ}45'67.89''$	Sexagesimal angle; see pp. 121ff also for other angular formats
1234567890 ENTER↑	1 234 567 890	
1234567890 #H	12 34 56 78 90 ₁₆	<i>Integers</i> of various bases, see pp. 131ff
10100110111 #2	101 0011 0111 ₂	
901.23.4567 ENTER↑	901 $\frac{23}{4}$ 567 >	<i>Fraction</i> , see pp. 146ff
12.3 CC -4.56 EXIT	12.3 - i \times 4.56	<i>Complex numbers</i> in rectangular or polar notation; mantissa plus exponent format is settable as well; see pp. 149ff
	12.3 \angle -4.56°	
1.2345678901 h.ms	1:23:45.678 901	Sexagesimal time, see pp. 183f
1.0203045 .d	0001-02-03	<i>Date</i> , see pp. 185f

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

	DECIM. set	DECIM. clear
GAP 4	1 2345.6789 01	1 2345,6789 01
GAP 3	12 345.678 901	12 345,678 901
GAP 2		
GAP 1		
GAP 0	12345.678901	12345,678901
MULTx set	12. $\times 10^{345}$	12, $\times 10^{345}$
clear	12. $\cdot 10^{345}$	12. $\cdot 10^{345}$
	123°45' 67.89"	123°45' 67,89"
MULTx, \neg CPXj	12.3-i×4.56	12,3-i×4,56
\neg MULTx, \neg CPXj	12.3-i·4.56	12,3-i·4,56
MULTx, CPXj	12.3-j×4.56	12,3-j×4,56
\neg MULTx, CPXj	12.3-j·4.56	12,3-j·4,56
	12.3 ₄ -4.56°	12,3 ₄ -4,56°
	1:23:45.678 901	1:23:45,678 901
	1:23:45.678 901 a.m.	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 and combine data of various types in calculations? The matrix below lists in its 1st column ten *data types* your

WP 43S supports; and 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 return an object of the *data type* at the intersection (thus, wherever a *data type* number is printed at the intersection, the corresponding combination is legal for addition or subtraction).

Data type and meaning		<i>y</i>									
<i>x</i>		1	2	3	4	5	6	7	8	9	10
1 \mathbb{Z} Long integer		1	2	3	4	5	6	7	-	-	1
2 \mathbb{R} Real number		2	2	3	4	5	6	7	-	-	2
3 \mathbb{C} Complex number		3	3	3	-	-	-	7	-	-	3
4 Angle (in various formats) ⁵⁶		4	4	-	4	-	-	7	-	-	4
5 Time interval (in H.MS)		5	5	-	-	5	-	7	-	-	-
6 Date (in various formats)		6	6	-	-	-	1 ⁵⁷	7	-	-	-
7 Alpha string ⁵⁸		-	-	-	-	-	-	7	-	-	-
8 Real matrix or vector		-	-	-	-	-	-	7	8	9	-
9 Complex matrix or vector		-	-	-	-	-	-	7	9	9	-
10 Short integer		1	2	3	4	-	-	7	-	-	10 ⁵⁹

Example:

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

⁵⁶ Angular output is tagged according to the current *angular display mode* chosen.

⁵⁷ A *date* minus a *date* returns an integer number of days (there are no other arithmetic operations on two *dates*). And a *date* plus a *real number* takes the integer part of that number and adds the respective number of days to said *date*.

⁵⁸ In additive operations on *alpha strings*, such a string must be present in **Y** at the beginning. Adding corresponds to appending *x* (converted to a string according to the display format set at execution time, if applicable) to string *y*. Adding a matrix appends its abbreviation (e.g. **[3x4 C matrix]**, see the chapters about vectors and matrices below). Subtractions from strings are not allowed.

⁵⁹ If *short integers* of different bases are combined by an arithmetic operation, output will be a *short integer* of the base given in **Y**.

The following matrix shows the resulting *data types* of products and ratios in the same way (note that *dates* and *alpha strings* cannot be multiplied or divided):

	An object <i>y</i> of <i>data type</i> ...									
	1	2	3	4	5	8	9	10		
... times an object <i>x</i> of the <i>data type</i> below returns a product of the <i>data type</i> printed at the intersection. ⁶⁰										
1 <i>Z Long integer</i>	1									
2 <i>R Real number</i>	2	2								
3 <i>C Complex number</i>	3	3	3							
4 <i>Angle</i>	4	4	-	-						
5 <i>Time interval</i>	5	5	-	-	-					
8 <i>Real matrix or vector</i>	8	8	9	-	-	8				
9 <i>Complex matrix or vector</i>	9	9	9	-	-	9	9			
10 <i>Short integer</i>	1	2	3	4	5	8	9	10 ⁵⁹		
... divided by an object <i>x</i> of the <i>data type</i> below returns a ratio of the <i>data type</i> printed at the intersection.										
1 <i>Z Long integer</i> ⁶¹	1/2	2	3	4	5	8	9	10		
2 <i>R Real number</i>	2	2	3	4	5	8	9	2		
3 <i>C Complex number</i>	3	3	3	-	-	9	9	3		
4 <i>Angle</i>	-	-	-	2	-	-	-	-		
5 <i>Time interval</i>	-	-	-	-	2	-	-	-		
8 <i>Real matrix</i> ⁶²	8	8	9	-	-	8	9	8		
9 <i>Complex matrix</i> ⁶²	9	9	9	-	-	9	9	9		
10 <i>Short integer</i>	1	2	3	4	5	8	9	10 ⁵⁹		

⁶⁰ This table applies to exponentiation in analogy.

⁶¹ For example, $15 / 3$ returns 5 while $14 / 5$ returns 2.8.

⁶² The matrix *x* must be invertible. Dividing by *x* is equivalent to multiplying times x^{-1} . (see the chapter *Vectors and Matrices: Calculating* below).

Furthermore, for integer divisions and remainders:

An object y of data type ...	1	2	10
... IDIVR-divided by an object x of the data type below returns an integer ratio in X and a remainder in Y of the data types printed at the intersection.			
1 \mathbb{Z} Long integer	1; 1	1; 2	1; 10
2 \mathbb{R} Real number	1; 2	1; 2	1; 2
10 Short integer	1; 1	1; 2	10; 10⁵⁹

Additionally, explicit type conversions are available where necessary:

An object x of data type ...						
1	2	4	5	6	10	... will be converted in an object x of the data type below by the command printed at the intersection.
-	IP	-	-	-	-	1 \mathbb{Z} Long integer
\rightarrow REAL (press .d)						2 \mathbb{R} Real number
L ...						4 Angle
\rightarrow INT (press #)		-	-	-	\rightarrow INT	Short integer 10 (optionally of another base)

Recognizing Calculator Settings and Status

As seen 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 top left within the *status bar*. Also *program-entry mode (PEM)* is easily recognized (see pp. 196ff).

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 date and time string there, listed from left to right in various groups – indicators shown in *startup default* are printed in a light blue field again:⁶³

⁶³ The symbol \neg means “not”, i.e. the trailing *system flag* cleared, while “ $\&$ ” denotes a logical “and” and a comma a logical “or”.

Indicator	Set by	Deleted by	Explanation, remarks
64:1	1COMPL	setting any other <i>integer sign mode</i> (<i>ISM</i>)	Settings for <i>short integers</i> . First two digits tell the <i>word size</i> , the character after the colon the <i>ISM</i> . <i>Startup default</i> is <i>64 bits</i> (the maximum) and 2's complement. <i>Carry</i> and <i>Overflow</i> may trail the <i>ISM</i> but are only lit if set.
64:2	2COMPL		
64:0	UNSIGN		
64:s	SIGNMT		
A	ALPHA; if is set	pressing EXIT in AIM unless in a menu,	<i>Alpha input mode</i> (AIM) is set. Upper (A) or lower () case letters can be entered now.
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 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. 256f.
	serial I/O in progress	idle communication line	See Serial Input and Output of Data and Programs on pp. 226f.
	data are being sent to printer		
	USER	USER	Toggles <i>user mode</i> (see pp. 285f).
	low battery	battery voltage > 2.5 V	A 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 a *status bar* like this:

2017-05-08 23:49 RL4° /max 64:2 ↑

On the other hand, choosing 12h time format (or M.DY), setting CPXRES, FRACT, DENFIX and a four-digit DENMAX, selecting unsigned *short integers*, setting CARRY and OVERFL, having a program waiting for input with AIM set, timer and printer running in background, *user mode* set, and a low battery would be reflected in the following *status bar*:

05/08/17 11:49pm RL4° /3546f 64:u° A ⓘ ⌚ ☰

Note also ☰ and ☱ might show up at right end of the *status bar*.

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

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

1. The *Matrix Editor* is described comprehensively on pp. 158ff.
2. **RBR** allows for browsing the contents of all *registers* currently allocated (see pp. 254ff).
3. **STATUS** (or **FLAGS STATUS**) returns free space available, memory currently used, *user* and *system flags* set (see pp. 256f).
4. **TIMER** calls the timer or stopwatch application (see pp. 257ff).
5. FBR browses all the characters defined in the fonts provided.

Further commands throw *temporary information* as defined on p. 67:

1. ERR and MSG display the corresponding error message. See the *IOI* and *App. C* of the *ReM* for more.
2. **r**, **VIEW**, **x**, and **y** return results headed by text.
3. Commands returning two or three values at once (like **-P**, **R-**, **DATE→**, **DECOMP**, **×**, **s**, **L.R.**, **SUM**, **M.DIM?**, **RCLIJ**, **Σ+** and **Σ-**) tag their output (see e.g. pp. 19 and 106f).

4. VERS generates a string showing version and build of the firmware running on your *WP 43S* (WHO works in a similar way):

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

A few far-reaching commands (like **CLALL**, **CLPALL**, or **RESET**) ask you for confirmation before executing. Answer either **Y**es by pressing **3** (or **ENTER↑** or **XEQ**) or **N**o by pressing **7** (or **EXIT** or **◀**); any other input will be ignored. Note that such an action explicitly confirmed cannot be undone by **UNDO**.

Localising Numeric Output

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



- **radix mark** denotes the decimal separator;
- **GAP** states the digit group interval – after *n* digits a narrow blank is displayed (cf. examples on p. 69); this follows *ISO 80000-1*.⁶⁴
- **JG** states the year the *Gregorian calendar* was introduced in the particular region, typically replacing the *Julian calendar* (or national calendars in East Asia);⁶⁵
- background colors are chosen as on pp. 68f.

Most people using radix commas employ multiplication dots while those using radix points need a cross for multiplication to avoid misunderstandings.

⁶⁴ As far as we know, the *WP 43S* is the first pocket calculator displaying numbers the way 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 .

⁶⁵ Officially, the *Gregorian calendar* became effective at 1582-10-15 in the catholic world. Many states and territories switched later for various reasons (check the dates in *Wikipedia*). You can enter the local date applicable using J/G (see the *I/O* for this command). Note there are still other calendars widespread, e.g. in the Muslim world. See also the chapter *Dates* below.

standings. This latter convention causes further ambiguities in vector multiplication (see pp. 168ff).

Com- mand	GAP	Radix mark ⁶⁶	Time	Date ⁶⁷	JG	Remarks
SETCHN	4 ⁶⁸	point	24h	Y.MD	1949	
SETEUR	3	comma	24h	D.MY	1582	Also applies to South America (and – with other JGs – to Indonesia, South Africa, the area of the former Soviet Union, and Vietnam).
SETIND	3 ⁶⁹	point	24h	D.MY	1752	Also applies to India, Pakistan, Nepal, Bhutan, Myanmar, Bangladesh, and Sri Lanka.
SETJPN	3	point	24h	Y.MD	1873	
SETUK	3	point	12h	D.MY	1752	Also applies to Australia and New Zealand. ⁷⁰
SETUSA	3	point	12h	M.DY	1752	

⁶⁶ See https://en.wikipedia.org/wiki/Decimal_separator for a world map of radix mark use. Looks like an even score in this matter. Thus, the international standard ISO 80000-1 allows either a decimal point or a comma as radix mark and requires a narrow blank as unambiguous separator of digit groups (it explicitly states that points or commas shall not be used as group separators to avoid ambiguity).

⁶⁷ See https://en.wikipedia.org/wiki/Date_format_by_country also for a world map of date formats used. The international standard ISO 8601 states Y.MD for dates and 24h for times. This combination is common in East Asia (see SETCHN and SETJPN).

⁶⁸ 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 1000. Thus, Chinese count using intervals — (= 1), + (= 10), 百 (= 100), 千 (= 1000), 万 (= 10 000), 十万 (= $10 \times 10 000$), 百万 (= $100 \times 10 000$), 千万 (= $1000 \times 10 000$), 亿 (= 10^8), 十亿 (= 10^9), 百亿 (= 10^{10}), 千亿 (= 10^{11}), etc. The command GAP **4** takes care of this notation while GAP **3** formats the European way.

⁶⁹ Proper South Asian (a.k.a. Indian) formatting would require separators every two digits for numbers over thousand. Think of *lakh* = 10^5 and *crore* = 10^7 . Actually, an amount of 50 cr. Rupees (= 5×10^8) reads 50,00,00,000 Rs. in Indian newspapers.

⁷⁰ 24h is taking over in the UK, so SETIND will work there then as well.

Note the following settings and formats can be stored collectively at one location: *stack size*, entire decimal display format (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 all *system flags*. STO CFG puts this configuration information in the *register* or variable you specify.⁷¹ RCL CFG recalls such information and will set your *WP 43S* accordingly.

Real Numbers: Changing the Display Format

As mentioned in Section 1, the numbers you calculate with (decimal numbers or measured values) are *reals* frequently. 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. 67f). The majority of functions provided by your *WP 43S* operate on *reals*.

As soon as input of a *real number* is closed, its mantissa will be displayed right adjusted as far as possible (cf. p. 24). *Startup default* format (ALL 0) shows all digits of the number if less than 16 are needed to do so. Your *WP 43S* will automatically turn to mantissa plus exponent format (cf. pp. 24f) if more than 15 digits are needed.⁷²

There are two flavors of the latter format: SCI and ENG. SCI is called scientific notation. ENG looks almost like SCI but the exponent will always be a multiple of three, corresponding to the S/ unit prefixes – thus it is called the ENGineer's notation (see examples below).

You can choose whether ALL shall turn either to SCI or to ENG. And you can define the switch point from ALL to SCI or ENG by specifying a positive parameter for ALL (telling up to how many decimal zeros you allow before the output shall be switched):

⁷¹ Actually, it stores even more – see Section 6.

⁷² 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 34 digits for *real numbers*) internally always; this can be displayed by SHOW until next keystroke.

Example (beginning with *startup default* settings):

Input:	Display:
-700	-700
1/x	-1.428 571 428 571 429 $\times 10^{-3}$
DSP ALL 3	-0.001 428 571 428 571
10 /	-1.428 571 428 571 429 $\times 10^{-4}$
FLAGS CF SYS.FL ALLSCI	-142.857 142 857 142 9 $\times 10^{-6}$
▼ ALL 4	-0.000 142 857 142 857
10 /	-14.285 714 285 714 29 $\times 10^{-6}$
SF SYS.FL ALLSCI	-1.428 571 428 571 429 $\times 10^{-5}$

There is one more format provided: FIX. With FIX, the radix mark is set at a fixed position on the screen and stays there; it floats in the other formats – see the examples below.⁷³

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 shown while the parameter of ENG specifies the total number of digits displayed within the mantissa minus one):

Format Input	Startup default format (ALL 00, SCIOVR)	FIX 5	SCI 5
107.12345678 ENTER↑	107.123 456 78	107.123 46	1.071 23 $\times 10^2$
1/x 2 x	1.867 004 725 311 852 $\times 10^{-2}$	0.018 67	1.867 00 $\times 10^{-2}$

See more examples of displays varying according to popular choices for GAP, decimal radix mark, and multiplication symbol (cf. the examples shown on p. 69):

⁷³ Deviating from previous calculators, output of $\times 10^0$ is suppressed on your WP 43S.

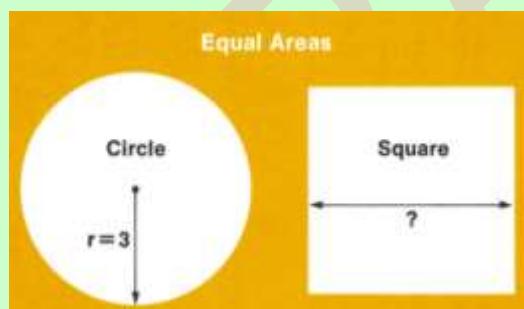
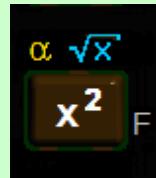
Input	Format FIX 3	Format ENG 6
89012345678.9 [+/-] [ENTER]	-89 012 345 678.900	-89.012 35×10 ⁹
	-89 012 345 678,900	-89,012 35·10 ⁹
	-890 1234 5678.900	-89.0123 5×10 ⁹

Nearly all functions for real number display format control are found in DSP: FIX, SCI, ENG, ALL, GAP, rounding, and more. Please see the ReM.



Real Numbers: Squares and Cubes and their Roots

You find x^2 and \sqrt{x} on the keyboard of your WP 43S, while x^3 and $\sqrt[3]{x}$ are in EXP (cf. p. 26). The following example using these four 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,

DSP FIX 3

3 [x²] [π] [x] returns 28.274 for the area of the circle. Then
[\sqrt{x}] returns 5.317 for the edge length of the square.

Furthermore,

3 EXP x^3 π x

4 x 3 / returns 113.097, the volume of the sphere. Then

$\sqrt[3]{x}$ returns 4.836 for the edge length of the cube with same volume. Thus,

x^2 6 x returns 140.320 for the surface of the cube.

Finally,

3 x^2 π x 4 x returns 113.097 for the surface of the sphere.

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

Here a little winter sports problem of our time:

Example:

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

Solution:

The centrifugal force is $F_c = r\omega^2 m = 2\pi \frac{v^2}{r} m$, thus the corresponding acceleration is $a_c = 2\pi \frac{v^2}{r}$. In consequence, the total acceleration acting along Chuck's body axis is $a_T = \sqrt{g^2 + a_c^2}$. Measured in multiples of g, this means $a_T/g = \sqrt{1 + (a_c/g)^2}$.

DSP FIX 0 1

30 E 3 ENTER↑

30 000.0

3600 / x^2

69.4

12 / 2 x π x

36.4

CNST g⊕ /

3.7

x^2 1 + \sqrt{x}

3.8

meaning 3.8 g.

Although this might be possible to stand for a young sportsman like *Chuck*, the snow on the run can hardly bear the corresponding forces – *Chuck's* skis will inevitably slide leading to a greater radius and less acceleration.

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.⁷⁴ *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.

8.25 [EXP] \times^3
7.5 \times^3 [−]
[π] [×]
4 [×] 3 [/] returns

561.5
139.6
438.7
584.9 m³ for the volume increase.

Real Numbers: Percent Change

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

Example (continued from above):

This is a volume increase of how many percent?

⁷⁴ 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

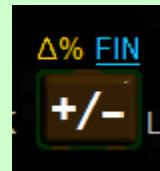
8.25 \times^3

561.5

 $\Delta\%$

returns

33.1 % increase.

**Another example:**

How about designing an almost optimum bicycle gearing for a hilly area? 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 taking 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/r_c in meter) for a 26" MTB:

Gear	1	2	3	4	5	6	7	8	9	10	11	12
Front	24				36			48				
Rear	30	26	23	20	26	23	20	23	20	17	15	13
d/r_c	1.66	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

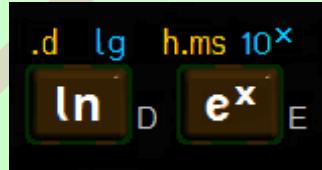
Assuming you pedal with 60 rpm constantly, such a bicycle will cover velocities between 6 and 28 km/h (or up to 37 km/h for 80 rpm). Using also some statistical functions provided on your WP 43S (i.e. $\Sigma+$, \bar{x} , and s explained on pp. 96ff), you will determine a mean speed increase per gear step of $(15.0 \pm 1.4)\%$, being quite uniform and convenient for town and country.⁷⁵ Feel free to try other configurations.

⁷⁵ Note that you will get just 12 out of 3×7 theoretically possible gears this way. This is due to gear overlaps; and you will want to avoid extreme chain skew for sake of chain life. On the other hand, if you are planning for a recumbent bike, the latter restriction might not apply anymore. Then you may think about a combination of three sprockets

Real Numbers: Logarithms and Powers (a.k.a. Antilogs)

Your WP 43S features two logarithmic functions on its keyboard and two more in EXP (cf. p. 26):

- [**In**] calculates the *natural logarithm* of x , i.e. the logarithm of x to the base e (being Euler's constant, see CNST).
Thus, [**In**] inverts [**e^x**].



- [**lg**] returns the (*common*) *decadic logarithm*, i.e. the logarithm of x to the base 10. [**lg**] inverts [**10^x**.⁷⁶]
[**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 pocket-able 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:⁷⁷

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 will be 9% per gear step; this gearing will cover velocities from 5 to over 40 km/h.

For detailed specifications as well as pictures, graphics, diagrams, tables, and further information about gearing bicycles yourself, please order "*Die Fahrradschaltung*" (144 pages written in German) written by the same author – just contact me.

⁷⁶ You may be used to the calculator label LOG for the decadic logarithm; though this is a mathematically ambiguous notation or worse, so we avoided it (cf. ISO 80000, 2-12.5 and -12.6).

⁷⁷ Emphases in these quoted examples were added by me.

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

Solution:

DSP FIX 00 should suffice here.

30 ENTER↑ 9.4 /

In

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

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

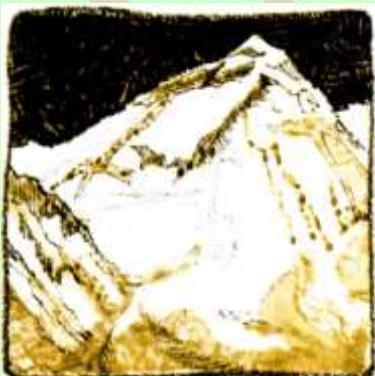
Note the concise and factual style of this text. The *HP-35* was a calculator made by engineers for engineers, and the manual was alike. This example was reprinted in the *HP-45 OH*. Thereafter, it underwent slight modifications:

Example (from the HP-21 OH of 1975):

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. A picture of the scenery was added in 1976, and not every snowstorm was worth mentioning anymore. Then, however, a switch of units reached the

Himalayas – and also the weather and the methods changed:



Example (in Solving Problems with Your Hewlett-Packard Calculator of 1978):

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:

$$\frac{\text{altitude}}{[\text{m}]} = 7620 \times \ln\left(\frac{760}{\text{pressure}/[\text{mm of Hg}]}\right)$$

Where is Wallace Quagmire?

Solution:

760 **ENTER↑** 238 **/**
In 7620 **X** returns 8 847..

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



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

78

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

$$\frac{\text{altitude}}{[\text{m}]} = 7620 \times \ln\left(\frac{1013}{\text{pressure}/[\text{mbar}]}\right) = 7620 \times \ln\left(\frac{101300}{\text{pressure}/[\text{Pa}]}\right)$$

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

⁷⁸ 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?

- the *decibel* (dB) in acoustics and electronics (see U, pp. 269f) and
- the so-called *upwardly unlimited Richter scale* for magnitudes of earthquakes.⁷⁹

Example:

One of the strongest earthquakes observed recently was the one causing the devastating tsunami in the Indian Ocean (near Indonesia) in December 2004. It had a magnitude of 9.1. Another one near Japan in March 2011 – with a magnitude of 9.0 – led to another tsunami and the ‘*Fukushima nuclear accident*’. Compare with the ‘*great San Francisco earthquake*’ of 1906 with 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 – we can continue with the display settings as they are:

9.1 **ENTER↑** 7.9 **-**

returns 63. and

9 **ENTER↑** 7.9 **-**

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 in such cases.

⁷⁹ 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:

What difference in magnitude will 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 FIX 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 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 tri-state logic, however,

RCL L 3 log₂y 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. Compare 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.

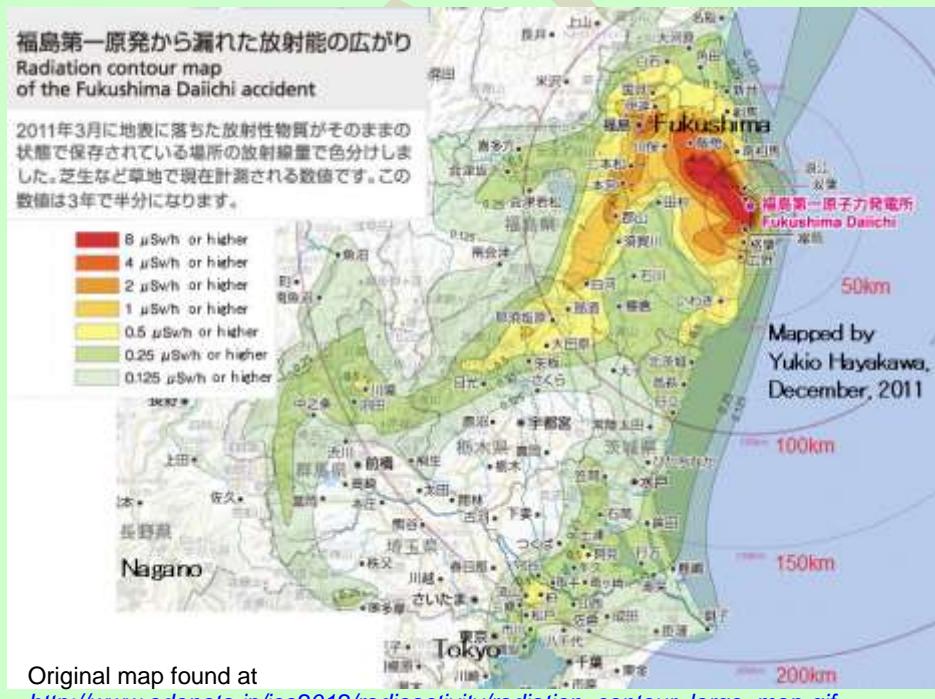
This solution path may be faster accessed and executed than the alternative 17 ENTER↑ 5 g EXP √y. Both keystroke sequences, however, will return 1.762 340 347 832 317.

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...3 MBq/m² corresponding to an annual radiation dose of 4 mSv in 2013 (see the map). Assume this was mainly caused by ¹³⁷Cs then; this radioactive Caesium isotope has a half-life of 30.2 years.⁸⁰

To the best of our knowledge today, an unborn child must not receive a dose of 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?



⁸⁰ With a probability of 94%, ¹³⁷Cs decays emitting an electron with a kinetic energy of up to 512 keV plus a γ -ray of 662 keV. These facts are just for your information – they do not affect the calculation here.

Solution:

Assuming there will be no further nuclear accident there, the isotopes set free will stubbornly decay following the inevitable laws of physics. Having had a radioactivity a_0 at time zero, the activity a at an arbitrary later 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, the 2013 annual dose of 4 divided by $\frac{4}{3}$ equals 3, and

DSP 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. Senior inhabitants may return far sooner.⁸¹

⁸¹ 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, a large fraction of worldwide knowledge about damage caused by radiation in human bodies in the long range is still based on extrapolation of experiences collected since 1945 following two large-scale events in Japan (and 67 more near Bikini until 1958). Another experiment well known was started in the USSR in 1986 – 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!

Note there are further risks linked to agriculture in the area around Fukushima – they are beyond the scope of this simple sample calculation though.

Also note this example covers a worst case scenario. Actually, radioactivity is 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 contaminated soil were removed, storing them in big black plastic bags ‘elsewhere’; 2.3 million m³ of soil are deposited there already, 12 million more are expected by the authorities – an area of 1.6 km² is provided for ‘interim storage’. Cost of disposal is going to be 1.9×10^{12} ¥ (estimated by the administration in 2019). Furthermore, 1.1 million m³ of contaminated water are stored separately – no idea yet where they shall go (see [Frankfurter Allgemeine Zeitung of 2019-03-10](#)). – Success of all these mitigation 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, however, can ever reduce the given natural half-lives of the radioactive isotopes set free and spread in this nuclear accident.

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 have to ‘put them away’ safely for really long times – a task kept under wraps for decades but not solved by waiting so far.⁸²

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

Real Numbers: Hyperbolic Functions

Hyperbolic functions tell us something about free hanging ropes, cables, chains, and the like. Your WP 43S provides three hyperbolic functions and their inverses in the **g**-shifted row of EXP (see p. 26) and TRI:

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

⁸² Surprise! Mankind has absolutely no experience with locking something away for several thousand years. Note the material must also be tagged properly in a way staying readable and comprehensible for all that time – no experience either.

Sad example: A huge concrete coffin holds almost 90 million *liters* of US nuclear waste on the Marshall Islands (remember Bikini). Now sea-level rise (caused by *anthropogenic global warming*) is eating away at the dome, and the USA is not interested in helping the tiny Pacific Ocean republic to do anything about it (see the [Los Angeles Times of 2019-11-10](#)).

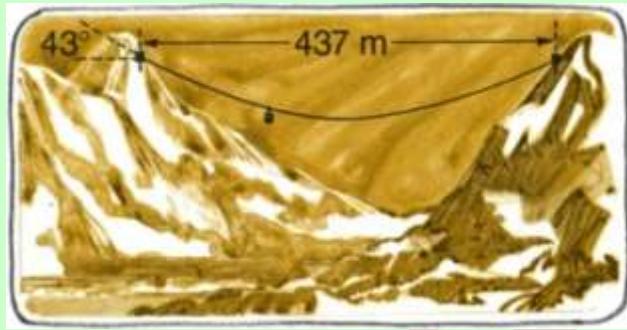
Sometimes you might meet people talking about ‘transmuting’ that entire long-living radioactive waste by converting it to isotopes with significantly shorter half-lives by some nuclear reactions (never met anybody being more specific in this matter so far). If that would be physically possible for all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

As far as mankind knows today, prospective fusion plants will not produce any long-lived isotopes in operation.

We found the following **example** in the *HP-32 OH*⁸³ though we modified it a bit:

In *Upper Lagunia*, a tram⁸⁴ carries tourists between two peaks in the *Baruvian Alps* that are the same height and 437 meters apart. How long does it take the tram to travel from one peak to the other if it moves along its cable at 135 meters per minute? Before the tram latches onto the

cable, the angle from the horizontal to the cable at its point of attachment is found to be 43°.



Solution:
The travel time is given by the formula

$$t = \frac{d}{v} \times \frac{\tan \alpha}{\operatorname{arsinh}(\tan \alpha)}.$$

Let's set **DSP FIX 2** since we do not need more decimals displayed.

43 [TRI] tan

[ENTER]

duplicates this intermediate result on stack for numerator and denominator.

arsinh [/]

437 [x]

489.30 m is the length of the cable.

135 [/]

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

⁸³ 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.

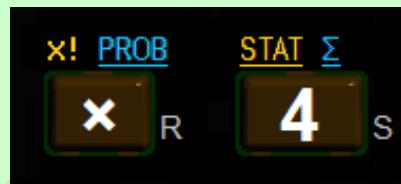
⁸⁴ Translator's note: British readers might frown here at least.

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

Besides the keyboard commands $\Delta\%$ and $x!$, 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 also the functions for *combinations* and *permutations*.



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 possible 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 this number.

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 here):

8 **ENTER↑** 5

PROB

C_{y,x}

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 $5!$ according to the statement above. Thus,

5 returns **120** for that number.

And...

returns **6 720** for the number of significantly different pictures.

This is the number of possible *permutations* of 5 items out of 8 (i.e. the number of possible different arrangements of y different objects

taken in quantities of x objects at a time; no object appears more than once in an arrangement, and different orders of the same x objects are counted separately here).⁸⁵ 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.⁸⁶ These functions share a few features:

⁸⁵ These challenging tasks changed the photographer significantly within one year!

⁸⁶ 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 1000 kids of this age.

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

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

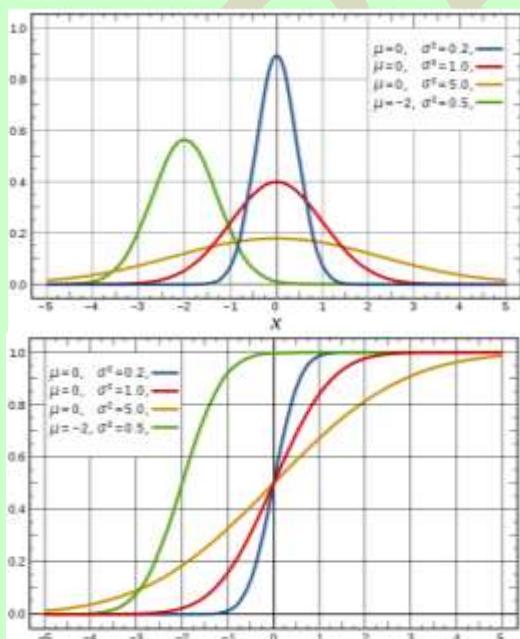
- Discrete distributions (like *Poisson*, *binomial*, *negative binomial*, *geometric*, and *hypergeometric*) are confined to integers. Whenever your *WP 43S* sums up a *probability mass function (PMF)* $p(n)$ to get a *cumulated distribution function (CDF)* $P(m)$, it starts at $n = 0$. Thus,

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

- Continuous distributions (like *Cauchy*, *exponential*, *logistic*, *log-normal*, two kinds of *normal*, *Fisher's F*, *Student's t*, *Weibull*, and *chi-square*) operate on *reals*. 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

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

- Many frequently used continuous *PDFs* look more or less like the ones plotted in the upper diagram overleaf. The lower diagram shows their corresponding *CDFs*, using the same scale and colors. Typically, any *CDF* starts at 0 with a slope of almost zero, becomes steeper then, and runs out at 1 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* $Q = 1 - P$ will be more precise. Thus, also Q is computed in your *WP 43S* for each distribution, independently of P . Definitions are:

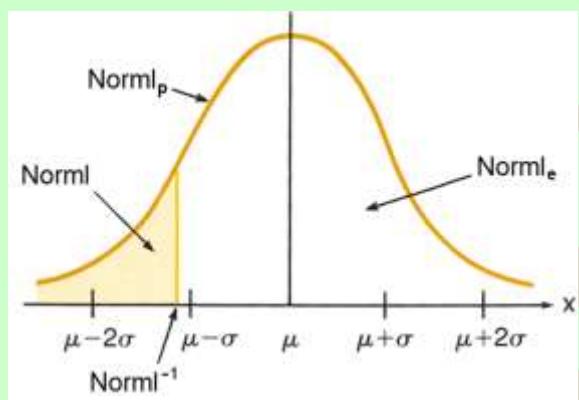
- for discrete distributions:

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

- for continuous distributions:

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

- With an arbitrary CDF, e.g. NORML (returning P), you will find the name NORML_e used for the function returning the error probability Q ,



NORML^{-1} for the inverse of the CDF (the so-called *quantile function*), and NORML_P for its PDF on your WP 43S. This naming scheme applies also to the **binomial**, **Cauchy** (a.k.a. Lorentz or Breit-Wigner), **exponential**, **Fisher's F**, **geometric**, **hypergeometric**, **log-normal**, **logistic**, **negative binomial**, **Poisson**, **Student's t**, and **Weibull** distributions. The **Chi-square** distribution is denoted differently following mathematical tradition. See PROB on p. 107 or the ReM.

Find application examples of distributions in the next two chapters.

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); weighted data require the weight in **Y**, pairs of data or coordinates of data points shall be entered in **X** and **Y**. **Σ-** is provided for easy data correction.

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 (a.k.a. *standard deviation of the mean*).⁸⁷

Example:

Archibald is champion of the *Golden Bow*, his archers club. In his standard exercise, aiming at a target disk of 1.5 m diameter at a distance of 50 m, his arrows scatter symmetrically around the center of 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? ⁸⁷

Solution:

DSP FIX 3
0 STO J
1 U→ x: feet→m
STO J STO 0 1
1.5 ENTER↑ 2 /
PROB Norml: Normle
2 X 1/x

Archibald's mean = center of disk.

0.305 , 1 *foot* in *meters*, Archibald's *SD*.

store this *SD* for later re-use.

0.750 , the radius of the target disk.

0.007 , the error probability.

72.102 so *Archibald* has to collect an arrow in the green only once in 72 shots on long term average.

Example (continued):

One of his buddies and competitors, *Bill*, also sends his arrows to the same target disk with his hits scattering symmetrically around the center of said disk, too. 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:

1 STO J
15 1/x

to get the *standard normal* distribution.

0.067 , i.e. about 7% of *Bill*'s arrows miss the target disk.

⁸⁷ 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. Foreigners travelling through that country, watch out! Please note that we refrained from using firearms here, though our resistance was strained almost to the limit.

2 **/**

Norml⁻¹

+/- .75 x²y /

STO 0 0

RCL 0 1

Δ%

0.033 ~3% misses on either side.

-1.834, the corresponding lower limit of the distribution.

0.409 $m = Bill's\ SD$. Just store it since we will need this again soon:

Note that the SD of *Archibald's* arrows is just...

-25.47 % narrower than *Bill's*, but his rate of misses is more than 10 times less.

There are also applications of this methodology in industry, where the scattering or variation of a production process 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 /

2 /

Norml⁻¹

+/- .75 x²y /

0.030

0.015 = 1.5% misses on either side.

-2.170, the corresponding lower limit of the standardized normal distribution.

0.346 $m = Bill's$ new SD .

Now, is this *significantly* better than his previous SD ? Statisticians have found 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_0) 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 95% probability and 59 degrees of freedom. Calculate inside out as usual:

59 [STO] J
.
.95 $\chi^2:$ $(\chi^2)^{-1}$
[/]
[χ^2]
[RCL] \times 0 0

the degrees of freedom must be stored in J.
calls the inverse chi-square, returning
77.931.
0.757
0.870
0.356 m for σ_L .

Looks like Bill's training made a difference!

Well ... within 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.⁸⁸

STAT contains also operations for curve fitting, featuring ten different regression models (linear, exponential, logarithmic, power, root, hyperbolic, and more 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).

⁸⁸ Applying statistics may cause that you might have more doubts than without – but such is life: doubts increase with knowledge. Only very dumb people have no doubts and may easily feel great therefore.

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.

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*):

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 rock⁸⁹ from various levels of the *Tower of Pisa* (which was leaning even then) and timed its descent by counting his pulse. The following data are measurements Galileo might have made:

<i>t (pulses)</i>	2	2.5	3.5	4	4.5 ⁹⁰
<i>h (Pisan feet)</i>	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:

DSP FIX 4

STAT

CLΣ	\bar{x}_G	σ	σ_p	σ_m	PLOT
Σ^-	\bar{x}_w	s_w	s_{mw}		
Σ^+	\bar{x}	s	σ	s_m	SUM

CLS

30 [ENTER] 2

Σ^+ returns

0.355 9
30.000 0
2.000 0

⁸⁹ I hope not! A pebble would have done as well if not better.

⁹⁰ These 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 customers would believe in that story in 1976.

Note that **$\Sigma+$** takes x and y , adds them to the statistical sums, increments the count of data points, and gives you feedback (note this output contains *temporary information* as explained on p. 67). Your next input after **$\Sigma+$** will overwrite x :⁹¹

50 [ENTER↑] 2.5 **$\Sigma+$**
 90 [ENTER↑] 3.5 **$\Sigma+$**
 130 [ENTER↑] 4 **$\Sigma+$**
 150 [ENTER↑] 4.5 **$\Sigma+$**

Data point 005
 150.000 0
 4.500 0



OrthoF	GaussF	CauchF	ParabF	HypF	RootF
LinF	ExpF	LogF	PowerF		BestF

BestF 0 instructs your *WP 43S* to pick the curve fit model matching these experimental data best (as explained above).



		\bar{x}_{RMS}			
		\bar{x}_H			
L.R.	r		s_{xy}	cov	\hat{x}
					\hat{y}

L.R.

4.500 0
 Power* $a_1 =$ 1.994 0
 $y = a_0 x^a_1$ $a_0 =$ 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

⁹¹ Remember $\Sigma+$ disables *stack lift*. Though note that accumulation of 2D data will slowly overwrite the *stack*.

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 frequency is left for the reader.

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



Big Lyle Hephaestus, owner-operator of the Hephaestus Oil Company, wishes to know the slope and y-intercept of a least squares line for the consumption of motor fuel in the United States (of America⁹²) against time since 1945 (in 1978!). He knows the data given in the table:

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

Solution:

Hephaestus⁹³ 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.**.⁹⁴

DSP FIX 2

STAT CLΣ

696 **ENTER↑** 1945 **$\Sigma+$**
1330 **ENTER↑** 1955 **$\Sigma+$**

994 **ENTER↑** 1950 **$\Sigma+$**
1512 **ENTER↑** 1960 **$\Sigma+$**

⁹² Differentiating from *Los Estados Unidos Mexicanos*, for example.

⁹³ Maybe his ancestors emigrated from Greece: Hephaistos is the ancient Greek god of fire and forging (and maybe of underground natural resources as well?).

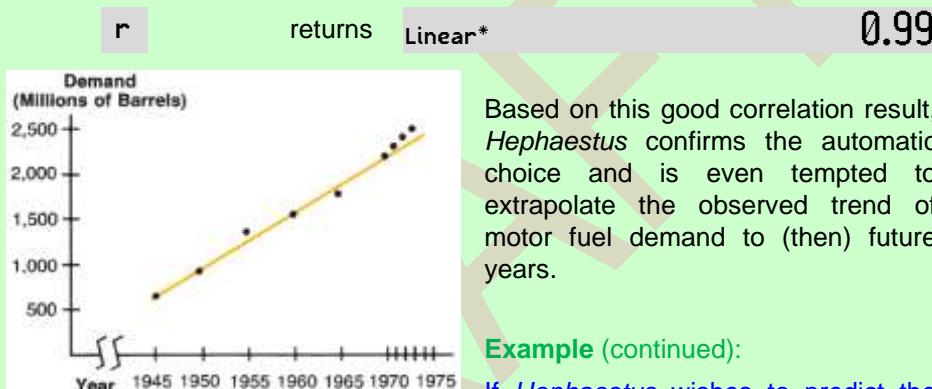
⁹⁴ The boss computes himself! And he seems being even able to do it properly! Looks like that was a time before general managers, CEO's, and large staffs became fashionable. But see also next footnote.

1750 **ENTER↑** 1965 **Σ+**
 2243 **ENTER↑** 1971 **Σ+**
 2484 **ENTER↑** 1973 **Σ+**

▲ L.R.

$$\text{Linear*} \quad a_1 = 61.16 \\ y = a_0 + a_1x \quad a_0 = -118\ 290.63$$

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



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 **ŷ**. 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 **✧**.

1980 ŷ	returns	Linear*	2 808.63
2000 ŷ	returns	Linear*	4 031.85

These were forecasts (i.e. extrapolations based on the fit model employed) of the demands in 1980 and 2000 at that time.

3500 ✧	returns	Linear*	1 991.30
----------------------	---------	----------------	-----------------

– the demand was expected to pass 3.5 billion *barrels* in 1992.⁹⁵

⁹⁵ Hephaestus should have been warned looking at his last three data points. Often, plots carry extra information which may be lost easily when dealing with numbers only. Actually, HP's example is not a good choice for extrapolating without plotting.

Another example from the HP-27 OH:

The *chi-square statistic* measures the goodness of fit between two sets of frequencies.⁹⁶ It's used to test whether a set of observed frequencies differs from a set of expected ones sufficiently to reject the hypothesis under which the expected frequencies were obtained.

In other words, you are testing whether discrepancies between the observed frequencies (O_i) and the expected frequencies (E_i) are significant, or whether they may reasonably be attributed to chance. The formula generally used is

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

If there is a close agreement between the observed and expected frequencies, χ^2 will be small. If the agreement is poor, χ^2 will be large.

Let's demonstrate the application of such a *chi-square statistic*⁹⁷ 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:

DSP FIX 0 0
25 ENTER 20 - x² 25
17 ENTER 20 - x² + 34

⁹⁶ 'Goodness of fit' tells us how good both sets match.

⁹⁷ Do not confuse this χ^2 defined here with the χ^2 distribution mentioned in previous chapter (and employed below very soon). They are different! Unfortunately, however, both chi-squares are called and spelled equally. It looks like the naming commission was inattentive here at the crucial time, perhaps distracted by discussing the outcome of a recent casino visit.

15	ENTER	20	–	x^2	+	59
23	ENTER	20	–	x^2	+	68
24	ENTER	20	–	x^2	+	84
16	ENTER	20	–	x^2	+	100
20	/					5

Now, is this χ^2 large or small? Statisticians have found it is to be considered 'small' if χ^2 is less than the value of the inverse χ^2 CDF for the degrees of freedom (here $n - 1 = 5$) and the significance level applicable (here 5%). As seen above already, also this χ^2 function is provided in your WP 43S. Note that a significance level of 5% equals 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** $\chi^2:$ $(\chi^2)^{-1}$ 11.

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

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?

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):

DSP FIX 3

STAT CLΣ

0 ENTER↑ 12.356 Σ+

Data point 001	0.000
	12.356

Continue accumulating 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

Knowing these pins are drawn from a *Gaussian* process, you get the best estimates for mean and standard deviation of your batch by pressing

\bar{x}

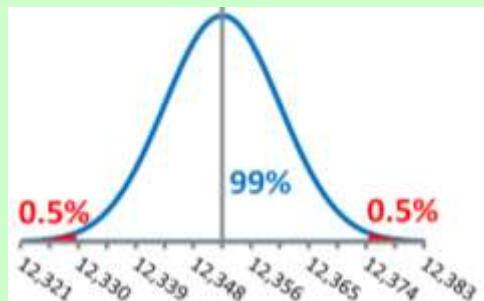
$\bar{y} =$	0.000
$\bar{x} =$	12.352

STO I s

$s_y =$	12.352
$s_x =$	0.000
	0.009

STO J 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**

		NBin:	Geom:	Hyper:	Binom:	Pois:	
LgNrm:	Cauch:			Expon:	Logis:	Weibl:	
Norml:	t:	C _{yx}	P _{yx}	F:	χ^2 :		

Norml:

	Norml _p	Norml	Norml _e	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% of your pins 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 as long as your process continues running the way you found it). 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↑**

12.350

.02 **–**

12.330 = lower limit.

Norml

0.006 = lower scrap = 0.6%.

x \gtrless y

.02 +

Norml_e

[+]

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 now (i.e. more than 1 out of 40 pins) will be the inevitable necessity to establish a very precise sorting tool or machine to ensure only good pins will pass to your customer. Thus, stay firm (if you can afford it) and refuse that customer request to constrict your tolerance limits – it may well be you cannot afford becoming weak here.

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

Example (continued):

Since we have got a sample drawn 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

1 [−]

STO J

[▲] s_m

10.000 recall the number of points.

9.000 store the degrees of freedom.

0.003 is the standard error.

Having 95% inside means having 2.5% outside at either end (cf. previous diagram). ⁹⁸ Thus, one must generally take 0.025 and 0.975 as

⁹⁸ The value 95% is called the *confidence level* of this calculation. In this example, you calculate the 95% *confidence limits* for the mean value. Instead of 95%, also 99% are frequently applied. We recommend checking the applicable valid standards before blindly copying any 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* und *confidence level* dem *Vertrauensniveau*.

arguments in two subsequent calculations using the *quantile* function of *t* to get both 95% limits below and above the sample result:

.025	PROB	t:	$t^{-1}(p)$	-2.262
x				0.006
STAT		\bar{x}		12.352
$x \bar{y}$	R↓			12.352
$x \bar{y}$				0.006
-				12.346 = lower limit.
RCL	L			0.006 = last <i>x</i> .
2	x	+	¹⁰⁰	12.358 = upper limit.

Now you know what to expect for the future average 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*.¹⁰¹ 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.

⁹⁹ **x** returns \bar{x} and \bar{y} (as was shown above). Only \bar{x} is interesting in this example, however, so pressing **$x \bar{y}$** **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 (see, Sect. 3).

¹⁰⁰ The upper *confidence limit* can be calculated this easy way since $t^{-1}(p)$ is symmetric around the mean value. Else it would have been necessary to repeat the above calculation (except the last two steps) with an input value of 0.975.

¹⁰¹ 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.

Application 2 (quick and easy measuring system analysis):

Your colleagues in R&D have specified that 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 the laboratory 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. This range could extend e.g. from 0 to about 0.015 here.¹⁰² Mark each sample unambiguously (e.g. by a number).
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 2nd time, but following another sample sequence (just shuffle the samples). Don't allow for looking at the data measured previously (hiding these data will be very helpful if you acquire the values manually)! Write each 2nd measured value behind the 1st one in the row carrying the respective sample number.
4. Get your WP 43S. Clear its statistical registers via **CLR CLΣ**. Then enter all 30 pairs of values using **STAT Σ+**. The 1st measured value shall be y , the 2nd be x – thus, input will be

mv1 [ENTER↑] mv2 $\Sigma+$

for each sample (alternatively, you can enter your statistical data into a matrix, then accumulate all points at once – see pp. 179f).

5. It is recommended to plot these 30 points. The plot shall look like an ant trail following the center line $y = x$ (see a typical plot overleaf).¹⁰³

¹⁰² 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, cover it fairly homogeneously (cf. the plot overleaf), and the samples must be resilient enough to stay constant in your measurements. No need for any investment in expensive gauges here – real life has proven scrap may well do.

¹⁰³ 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 the ReM.

6. Let your WP 43S fit a straight line through the 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:

Select the orthogonal linear fit model:

OrthoF

r x^2 **STO** **K**

get the coefficient of correlation and store its square.

s x^2

get s_x^2 .

R **D**

roll it out of the way.

x^2 **X**

get $r^2 s_y^2$.

R **UP** **+**

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

1 **RCL** **-** **K**

calculate the denominator.

/ **✓X**

this is the 2nd factor now.

s **X** **/** **30** **/**

divide by $30 s_x s_y$.

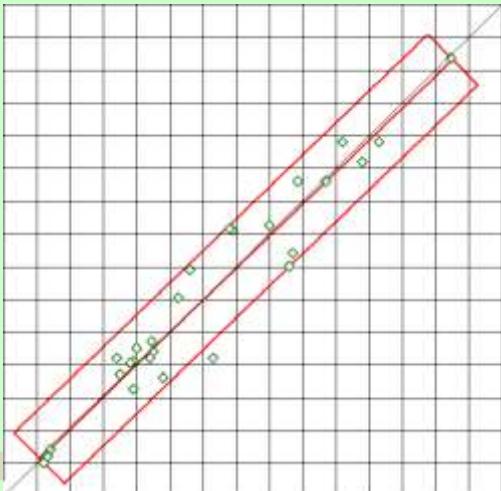
.01 **X**

this returns c_0 for our exemplary T now.

If you get $c_0 \geq 1$ then this measuring device may be used for controlling series production with 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.

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

In four decades professional experience, I found such correlation diagrams being the most powerful though easy tools for assessing the quality of real-life measuring processes. Even manually drawn clean correlation diagrams will support your decisions far better than staring at numbers only. With capable measuring systems, the resolution required for showing all measured points clearly separated from each other might, however, exceed the capability of a pocket calculator screen by far. The calculations will be correct nevertheless.



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 these two samples show any *significant difference*?

The following simple three-step test is well established.¹⁰⁴ 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:

The screen shows:
STAT S_m
 x^2 $x \geq y$ x^2 + fx
 \bar{x}
 $-$ $|x|$
 $x \geq y$ / 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 .

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:

The screen shows:
 Σ n
1 - STO I
.975 PROB t: t⁻¹(p)

recall the number of samples measured.

calculate the *degrees of freedom f* and store them for *Student's t*.

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

If $d < t_{cr}$, the test indicates the difference between both samples is

¹⁰⁴ This test assumes your samples are both drawn from a *Gaussian* process which is frequently the case in real life (but shall be verified).

due to random deviations only. Congratulations – you have got a robust process regarding the parameters you changed!

Else continue.

- 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 seriously 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” at the utmost.

Application 4 (operating characteristics):

Assume you draw a sample of 20 parts out of a production batch of 100 parts and check the sample thoroughly. What is the probability P to find at least one random defect in such a sample if the overall probability for a defect in such a 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:

DSP	FIX	3	
100	STO	I	store batch size
20	STO	K	store sample size
0.05	STO	J	store 5% overall defect probability
0	PROB	9	Hyper: Hyper
1	x \gtrless y	-	returns 0.319
0.02	STO	J	store 2% overall defect probability
0	Hyper		returns 0.638
1	x \gtrless y	-	returns 0.362

0.01	STO J	store 1% overall defect probability
0 Hyper		returns 0.800
1 x>y	-	returns 0.200

Even with 5% defects in the batch the odds are about 1 out of 3 that no defect at all is detected in such a relatively large sample! And note that such sample tests are certainly not adequate for controlling industrial processes with overall defect probabilities less than 1%.

STAT encompasses many more statistical functions (e.g. *covariances*, means and standard deviations for weighted data, *geometric means* and *scattering factors*) – just look them up there and check the respective entries in the **I/OI**.

You will find all accumulated sums of your data in Σ . Summon these sums individually by calling their names (no need to memorize any register numbers in this matter).

More examples of statistical applications can be found in the manuals of various vintage *HP* calculators, especially the *HP-27* and *HP-21S*.

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, before applying them.

Real Numbers: Summary of Functions

The majority of the functions your *WP 43S* features are for calculations operating on *reals*. It provides many more than the numeric functions shown on pp. 19ff, 28ff, and 80ff in various applications and examples. See all *real* functions listed below:

- General mathematics:

- *Monadic* functions:

- $\sqrt{-}$** , **$1/x$** , **$x!$** , **\sqrt{x}** and **x^2** , **$\sqrt[3]{x}$** and **x^3** , **2^x** and **10^x** , **e^x** and **\ln** , **\sin** , **\cos** , **\tan** , and their inverses work as

demonstrated above and you learned in school (see also pp. 121ff for more information about angular I/O),

e^x-1 and **ln(1+x)** return more accurate results for $x \approx 0$,

ceil returns the smallest integer $\geq x$, while **floor** returns the greatest integer $\leq x$,

SDL n shifts digits left by n decimal positions, equivalent to multiplying x times 10^n ,

SDR n shifts digits right by n decimal positions, equivalent to dividing x by 10^n ,

for **sinh**, **cosh**, **tanh**, and their inverses cf. pp. 91f,

(-1)^x returns $\cos(\pi x)$ for non-integer x .

- *Dyadic functions:*

[+], **[-]**, **[x]**, **[/]**, **[y^x]**, and **[sqrt]** 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)

(and **IDIVR** if you want also the remainder returned in **Y**),

log_xy for the logarithm of y for the base x

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

RMD for the remainder of y/x (see p. 139 for examples),

MOD for $y \bmod x$ (see p. 140 for examples),

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

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

0 else, being handy in electrical engineering in particular.



- *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: Use...

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 $\times 10^{-95}$),

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. 19f), while **R→** converts vice versa.

Angular, time, and date conversions are covered on pp. 121ff and 183ff.

For *unit conversions* see pp. 269ff.

- Boole's algebra:

AND, **NAND**, **OR**, **NOR**, **XOR**, **XNOR**, and **NOT** operate on *reals* 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).

Example: 13.5 **ENTER↑** -7.2 **[BITS]** **AND** returns 1.

- Probability and statistics (unless introduced and explained on pp. 93ff already):

$\Gamma(x)$ calculates the *Gamma function*,

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

Example: What is $5432!$?

Remember $\Gamma(x + 1) = x!$ So, entering **5433 X.FN lnΓ** **10 ln /** returns 17 931.480 374 010 87 as decadic logarithm of the result. Calling **PARTS FP 10^x** will return some 3.023 553 598 420 006 for its mantissa. Thus, $5432! \approx 3.024 \times 10^{17931}$.

RAN# returns a (pseudo) random *real* number between 0 and 1,

SEED stores a seed (i.e. a start value) for RAN#,

RANI# returns a (pseudo) random integer number between x and y , and the other contents of

PROB cover combinations, permutations, and the 14 distributions introduced on pp. 94ff.

Σ contains all accumulated sums of your data, callable by their names.

In **STAT**, you find the summation commands **$\Sigma+$** and **$\Sigma-$** , various mean values (\bar{x} , \bar{x}_w , \bar{x}_G , \bar{x}_H , \bar{x}_{RMS}), sample standard deviations (s , s_w) and standard errors (s_m , s_{mw}), population standard deviations (σ , σ_w), various scattering factors (ε , ε_m , ε_p), as well as all commands related to curve fitting.

Turn to the *ReM* for comprehensive information about all the statistical and probability functions provided on your *WP 43S*.

- Percentages:

% calculates $\frac{xy}{100}$, leaving y unchanged (so you can easily calculate another percentage of the same base after CLX).¹⁰⁵

Example (from the HP27 OH):

If you buy a new car, you have to figure the sales tax percentage, then add that to the purchase price to find the total cost of the car.
... For example, if the sales tax on a \$6200 car is 5%, what is the amount of the tax and total cost of the car?

6200 **ENTER↑** 5 **FIN %** returns 310. US\$ for the sales tax;
+ returns 6 510. US\$ for the total cost.

If the dealer 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 (for same reason as with %).

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,

¹⁰⁵ Actually, that's the (almost only) real benefit of the function **%**.

%T calculates $100 \frac{x}{y}$ (called “% of total”),¹⁰⁶

%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 186,40.¹⁰⁷

- Advanced mathematics (see the *ReM, App. H* for comprehensive information about the functions following):

- Monadic functions:

B_n and **B_n*** return the *Bernoulli numbers*,

erf and **erfc** the *error function* and its complement,

FIB the extended *Fibonacci number*,

g_d and **g_d⁻¹** the *Gudermann function* and its inverse, and

NEXTP the next *prime number* greater than x ;

sinc returns $\sin(x) / x$ for $x \neq 0$ and 1 for $x = 0$ (*input* is converted to *radians* before calculating – see pp. 121ff),

¹⁰⁶ I still wait for somebody convincing me of the use of this financial function. Please see also **%+MG**.

¹⁰⁷ Every engineer or scientist will be able to produce the very same result significantly faster via 221.82 **ENTER↑** 1.19 **7**.

Seeing functions like **%+MG** and **%T** in particular provided on financial calculators, however, you might get the impression that average financial people might be mathematically slightly challenged and need some extra support. On the other hand, there is a saying in technical quarters (already before 2008): ‘Looking at the results financial people produce with plus and minus alone, their access to more advanced operations should be strictly limited’ (originally: “Wenn man sieht, was Kaufleute mit plus und minus alles anstellen, sollte man sie an höhere Rechenarten gar nicht ranlassen”).

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

$\zeta(x)$ *Riemann's Zeta function*.

Call H_n for the *Hermite polynomials* for probability and

H_{np} for the *Hermite polynomials* for physics,

L_n for *Laguerre's polynomials* and

$L_{n\alpha}$ for *Laguerre's generalized polynomials*,

P_n for the *Legendre polynomials*,

T_n for the *Chebyshev polynomials of 1st kind* and

U_n for the *Chebyshev polynomials of 2nd kind*.

- *Dyadic functions:*

AGM returns the *arithmetic-geometric mean*,

$J_y(x)$ the *Bessel function of 1st 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*, and

IG_p and IG_q return the *regularized gamma function* (1 of 2 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, MUL π , and D.MS.¹⁰⁸ Angles are entered as *reals*. They are interpreted according to the current *ADM* as indicated in the *status bar* by 4° , 4^r , 4^g , 4π , or $4''$ (cf. p. 73) as soon as a function expecting angular input is called.

Exception: Sexagesimal angles must be entered in the format dddd.d.msspp – 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°34'54.32"**.

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

In ADM5 TRI arccos will return ...
4^r	1.047^r
4π	0.333π
4°	60.000[°]
$4''$	60° 0' 0.00" ¹⁰⁹
4^g	66.667^g

¹⁰⁸ All ADM setting commands except D.MS are found in MODE.

Translator's note: The traditional calculator notations DEG and GRAD are misleading in German at least: DEGrees on your *WP 43S* mean "Grad", while calculator GRADs are generally called "Gon" in Continental Europe.

¹⁰⁹ Note there are no leading zeroes in the angular *minutes* and *seconds* sections. And this *ADM* 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.



14 angular conversions are provided, all found in **L→**:

From ... to ...	sexagesimal degrees	decimal degrees	radians	grads/ gon	multiples of π	current ADM or tagging
sex. degrees	—	D→D.MS	—	—	—	→D.MS
dec. degrees	D.MS→D	—	R→D	—	—	→DEG
radians	—	D→R	—	—	—	→RAD
grads/gon	—	—	—	—	—	→GRAD
multipl. of π	—	—	—	—	—	→MULπ
current ADM	D.MS→	DEG→	RAD→	GRAD→	MULπ→	—

Example:

DSP FIX 5

MODE MULπ

Choose *multiples of π* as *ADM* and $\frac{\pi}{3}$ will appear in the status bar and stay there for the time being.

300 $\frac{1}{\sqrt{x}}$

L→ →RAD

→DEG

→D.MS

→MULπ

0.003 33 So $\pi/300 \dots$

0.010 47^r are 0.010 47 radians

0.600 00[°] or exactly 0.6°

0°36' 0.00" or 36 angular minutes

0.003 33 π equivalent to $\pi/300$ still.

Note →RAD ‘knew’ it had to convert from *multiples of π* since this function expects angular input and took the current *ADM* setting into account. Angular output of operations is tagged and will stay so. Thus, →DEG above converted from *radians*, →D.MS from *decimal* and →MULπ from *sexagesimal degrees* since the respective inputs were tagged.

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

Example (found in the HP-25 OH):

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.¹¹⁰

Solution:

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

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

with B_s 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*).¹¹¹ 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*.

d.ms

Since we will use *sexagesimal degrees* throughout this calculation, we set *ADM* accordingly.

DSP FIX 2

We will not need more decimals displayed.

5.43 d.ms

ENTER↑

$5^{\circ}43' 0.00''$

12.18 d.ms

$12^{\circ}18' 0.00''$

[−]

$-7^{\circ}15' 0.00''$

TRI cos

0.99

¹¹⁰ This example was reprinted thereafter in each and every *HP* scientific pocket calculator manual until the *HP-41C/41CV OH and Programming Guide*.

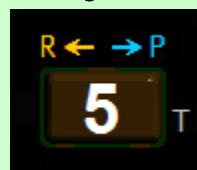
¹¹¹ This formula means that 1 nmi corresponds to 1 angular minute on a great circle. This doesn't hold exactly but precisely enough for practical sailing. See U→ for nmi. Translator's note: *Latitude* means "geographische Breite" in German. Hence *B* is used in the formula above.

15.55	STO	J	cos	0.96
	[x]			0.96
37.03	STO	K	cos	0.80
	[x]			0.76
	RCL	K	sin	0.60
	RCL	J	sin	0.27
	[x]			0.17
	+			0.93
	arccos			21°55' 24.66"
	.d			21.92 convert to a real number.
60	[x]		returning	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.

→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.

Solution:

DSP FIX 2
4.5 **ENTER↑** 6 **→P** returns

θ =	36.87°
r =	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← does the reverse, it 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 \leftrightarrow y

7.50

36.87°

L \rightarrow RAD

7.50

0.64^r

x \leftrightarrow y

4.50

R \leftrightarrow

6.00

returns

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, i.e. $-\pi$ to $+\pi$ in *radians*, $-200g$ to $+200g$ in *gradians*, and -1 to $+1$ in *multiples of π* .

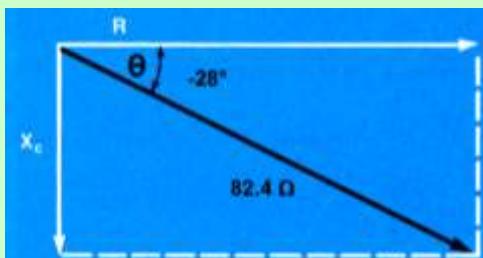


Example (triggered by the HP-67 OHPG):

In an electronic circuit designed for alternating current, an overall impedance of 82.4Ω 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 FIX 0 1

-28 ENTER 82.4

R \leftarrow returns

y = -38.7
x = 72.8

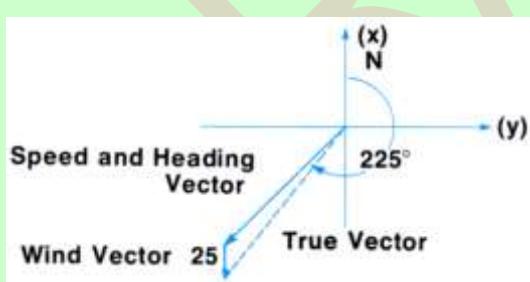
i.e. a resistance of 72.8Ω and a reactance of 38.7Ω .

By the way, you can use $\rightarrow P$ and $R\leftarrow$ also to convert 3D cylinder coordinates to Cartesian and vice versa, since z is kept unchanged.

Having learned about $\rightarrow P$ and $R\leftarrow$ as well as about $\Sigma +$ and $\Sigma -$, 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.)

CIR CL Σ

clears the summation registers.

DSP FIX 2

225 ENTER 125 indicated air speed and heading.

R-	returns	y =	-88.39
		x =	-88.39
STAT	$\Sigma+$	adds x and y to the summation registers.	
180	ENTER↑	25	north wind.
R-	returns	y =	0.00
		x =	-25.00
$\Sigma+$		adds x and y to the summation registers.	
SUM		recalls the summation registers Σx and Σy	
→P	returns	s =	-142.06°
		r =	143.77
x\geqy	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. 153.

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:

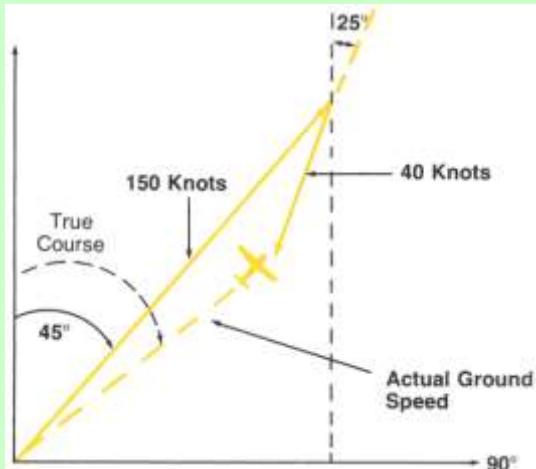


Example:

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:

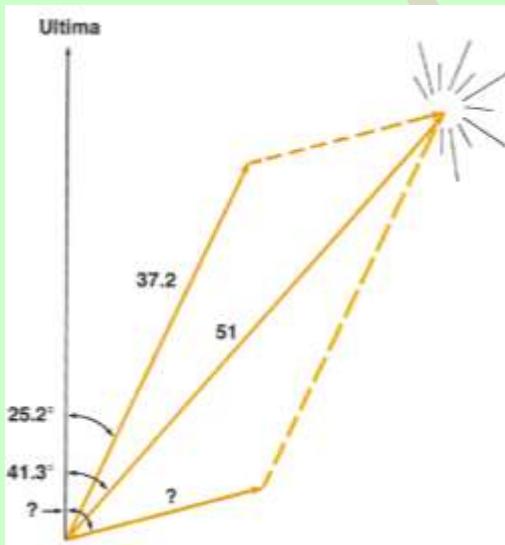
Method 1: The course and ground speed are equal to the difference of the two vectors.



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 OHPG of 1978).

We leave it to you to solve this problem using $\Sigma+$ and $\Sigma-$ (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:¹¹²

Federation starship *Felicity* has emerged victorious from a furious battle with the starship Θανατος¹¹³ from the renegade planet *Maldek*. However, its automatic pilot is kaput,¹¹⁴ and its main thrust engine is locked on at 37.2 meganewtons (MN) directed along an angle of 25.2° from the star *Ultima* (= Latin for 'the last'). Consulting the ship's star map, the navigator reports a hyperspace entrance vector of 51 MN at an angle of 41.3° from *Ultima*. To what thrust and angle should the auxiliary engine be set, for *Felicity* to

achieve alignment with the hyperspace entrance vector?

¹¹² ... of 1978. Note the first episode of *Star Wars* was launched in 1977.

¹¹³ 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.

¹¹⁴ Oh, why can't the (American) English learn to speak ... ummh ... spell?

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_↔**, and their difference is calculated using **Σ+** and **Σ-**. 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 **→P**.

CLR CLΣ

clears the summation registers.

41.3 [ENTER] 51

hyperspace entrance vector

R_↔

returns

y =

33.66

x =

38.31

STAT Σ+

adds the x and y components of the hyperspace entrance vector to the summation registers.

25.2 [ENTER] 37.2

main engine thrust vector

R_↔

returns

y =

15.84

x =

33.66

Σ-

subtracts the x and y components of the main engine thrust vector from the summation registers.

SUM

recalls the summation registers:

Σy =

17.82

Σx =

4.65

→P

returns

θ =

75.36°

r =

18.42

meaning the auxiliary engine shall be set at 18.42 MN and an angle of 75.36° from *Ultima*.¹¹⁵

¹¹⁵ Those looking for an extra challenge can compute now how flat the crew of *Felicity* will become within *seconds* after the auxiliary engine is ignited.

By the way, the plane of action in 3D space seems to be defined sufficiently by *Felicity*, *Ultima*, and said hyperspace entrance (hopefully its center) here with all parameters specified to three digits maximum – a proper error calculation would have been appreciated. This problem was reprinted in the *HP-34C OH* one year later. It vanished in hyperspace thereafter, without a trace.

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.

See the operating manuals of vintage *HP* calculators (especially the *HP-27*) for further applications from the areas of mathematics (e.g. triangle solutions), navigation, and surveying.

Angles: Summary of Functions

The number of functions operating on and with *angles* is quite limited. They are important nevertheless. 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. 70f,
`max` (or `min`) return the maximum (or minimum) of *x* and *y*.
- Rounding:
`ROUND` rounds *x* using the current display format (cf. pp. 114f),
- Conversions:
`→P` converts rectangular *coordinates* to polar ones (cf. pp. 19f),
while `R→` converts vice versa. Cf. the examples on pp. 124ff.
Angular conversions are covered comprehensively on p. 122.

Integers: Input and Displaying

Any single number (e.g. a counted value) you enter without using **[.** or **[E]** is regarded as an integer by your *WP 43S* (cf. pp. 67f). It allows for integer computing in fifteen bases from binary to hexadecimal.

Any single number displayed without any punctuation on your *WP 43S* is an integer (see examples below). And it will stay integer as long as it is exclusively combined with other integers and only integer functions operate on it; else it will be converted to another *data type* (cf. the matrices on pp. 70f and *Section 3* of the *ReM*). Note that any closed integer *x* will be converted to a *real number* by **[.d]**, while even an open one will be converted to an *angle* by any angular conversion (cf. p. 72).

There are two kinds of integers provided by your *WP 43S*: integers of almost arbitrary length (called *long integers*) and integers of finite length (called *short integers*).

Long integers are useful e.g. for numeric tasks. If you enter a number of arbitrary length just without using **[.]** or **[E]**, it is taken as a *long integer* of base 10. For **example**,

111 111 111 **[x²]** returns 12 345 678 987 654 321

Note the number is adjusted to the right again when closed, though no point (or comma) is displayed. A 17-digit result is shown with ease. Large *long 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 their digits are kept internally, so *long integers* can be of very high precision.

Example (a mathematical problem solved in 2019):

It was proven for integers *n* from 1 up to 100 that they¹¹⁶ can be expressed as sum of three integer cubes $n = i^3 + j^3 + k^3$ – except for 42. In September 2019, two mathematicians of Bristol and Boston published that the numbers 12 602 123 297 335 631, 80 435 758 145 817 515, and -80 538 738 812 075 974 should solve this problem. Verify!

¹¹⁶ Unless $\text{mod}(n; 9) = 4$ or 5. See two chapters below for the function mod.

12 602 123 297 335 631 **EXP** x^3 **SHOW** returns
 2 001 387 454 481 788 542 313 426 390 100 466 780 457 779
 044 591

80 435 758 145 817 515 x^3 **+** **SHOW** returns
 522 413 599 036 979 150 280 966 144 853 653 247 149 764 3
 62 110 466

-80 538 738 812 075 974 x^3 **+** returns **42 !**

This is all you need to know about entering and displaying *long integers* – turn to pp. 139ff for further information about calculating with them.

Short integers feature a finite word size (up to 64 bits) and are especially useful for computer logic and system design tasks incl. debugging. Your WP 43S encompasses all the integer and bit manipulation operations of the dedicated *Computer Scientist's HP-16C* and even all the bases and the entire extended function set of the *WP 34S*.

Short integers are entered with trailing **# base** (=2 ... 16). For decimal short integers, you may use **#D** instead of **#10**, for hexadeciml **#H** instead of **#16**. Open INTS (see its top view displayed here) for the digits **A** ... **F** required for numeric input in bases >10.



From the 2nd integer input on, you can save keystrokes: If you enter a new number omitting **#** and base (as well as

(**.
E**, and **CC**), your *WP 43S* takes it as a *short integer* of the same base you keyed in before – as long as you did not enter any other *data type* in between.¹¹⁷

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* in *bits* 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. 74); these *ISM*'s control the handling of negative numbers (see examples below).

Carry and overflow – if set – will be shown as **c** or **o** or **e**, respectively, trailing *ISM* display in the *status bar*. Both behave like they did on *HP-16C* or *WP 34S*, corresponding to *flags* (cf. p. 54) – if you want to set, clear, or check them one by one, use the commands provided in FLAGS.



Example:

Enter **FLAGS SF SYS.FL LEAD.0**

[INTS] ▲ WSIZE 1 2 This allows seeing all bits at a glance easily.

147 [ENTER] Enters 147 (base 10)

2 Converts decimal 147 to binary.

1COMPL and you will see¹¹⁸

0000 1001 0011₂ and – after **[+/-]** – **1111 0110 1100₂**.

Obviously **[+/-]** in 1COMPL flips every bit, equivalent to NOT here.

Return to the original number via **[+/-]**, press **2COMPL**, and you will get

0000 1001 0011₂ and – after **[+/-]** – **1111 0110 1101₂**.

Note the negative number equals the inverse + 1 in 2COMPL.

¹¹⁷ This shortcut will be left as soon as you enter a **.**, **E**, **CC**, or **#** in input, even if deleted thereafter.

Illegal digits keyed in (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 may key in more than the current *word size* can take – also this will be checked when input is closed.

¹¹⁸ Note the gap automatically inserted every four bits here for easy reading this output.

Return via [+] again, press **SIGNMT** and you will see

$0000\ 1001\ 0011_2$ and – after [+] – $1000\ 1001\ 0011_2$.

Negating a number will just flip the top bit in SIGNMT (hence the name of this mode).

Return via [+] once more, press **UNSIGN** and you will get

$0000\ 1001\ 0011_2$ and – after [+] – $1111\ 0110\ 1101_2$.

Note the 2nd number looks like in 2COMPL, but in addition an *overflow* is set here – see the F in the *status bar* trailing the *ISM*.¹¹⁹ Thus, pressing [+] will not suffice anymore for returning to the original number here; you must also clear the *overflow flag* explicitly by **FLAGS CF SYS.FL OVERFL**.

As you have seen, positive numbers stay unchanged in all those four modes. Negative *short 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.

¹¹⁹ This needs explanation, since changing signs should have no meaning in unsigned mode per definition. Thus, [+] 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, [+] 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.

Example:

Compare the outputs for different bases in 12:2 :

	-147 ₁₀	corresponds to...
# 2	1111 0110 1101 ₂	,
# 3	-12 110 ₃	,
# 4	33 12 31 ₄	,
# 5	-1 042 ₅	,
# 6	-403 ₆	,
# 7	-300 ₇	,
# 8	75 55 ₈	, and
# 9	-173 ₉	.

You may have noticed that the displays for bases 2, 4, and 8 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 *short 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.¹²⁰

Let's look to bigger words now:

Example (continued):

Enter **FLAGS** CF **SYS.FL LEAD.0**
INTS ▲ **WSIZE 6 4 UNSIGN**
▼ **9 3 A 1 4 C 6 # H** (cf. the *menu* shown on p. 132).

Then your WP 43S will display

9 3A 14 C6₁₆

¹²⁰ During numeric input, however, a gap is inserted every 3 digits as for real numbers – your WP 43S cannot know in advance what you have in mind.

In binary representation, this number will need 28 digits and would look like

1001 0011 1010 0001 0100 1100 0110₂.

Obviously, your *WP 43S* cannot display a binary number of this size this way 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

This binary number is displayed using the small font provided. If leading zeros were turned on via **FLAGS SF SYS.FL LEAD.0**, all 64 bits would be displayed in one row making use of a minimal font:

... with the 36 most significant bits all containing 0.

Integers: Bitwise Operations on Short Integers

Your *WP 43S* carries all the bitwise operations you may know from the vintage *HP-16C Computer Scientist*, plus some more you may have learned with the *WP 34S*. You find them all in [BITS](#). Generally, bits in a word are counted from right to left, starting with number 1 for the least significant bit. This convention is important for specifying correct 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.



BITS ▲ **WSIZE** **8**

FLAGS SF SYS.FL LEAD.0

1011 0011 **#** **2** **STO** **K**

This is the common initial number for the operations presented in the table below. You find seven shift and rotate functions with schematic pictures herein like they were printed on the backplane of the *HP-16C*, wherein the boxed **C** represents the *carry bit* indicated in the *status bar* if set.

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
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

Operation	Schematic picture if applicable	E. g.	Output
Arithmetic Shift Right		ASR 3	in 1/2COMPL: $1111\ 0110_2$ in UNSIGNED: ¹²¹ $0001\ 0110_2$ 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**:¹²²

Common input		Y	0110 1011 ₂
Operation	Symbol	Output	
AND	\wedge	0010 1001 ₂	
NAND	$\bar{\wedge}$	1101 0110 ₂	
OR	\vee	1111 1011 ₂	
NOR	$\bar{\vee}$	0000 0100 ₂	
XOR	\circlearrowright	1101 0010 ₂	
XNOR		0010 1101 ₂	

See the *IOI* 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.

¹²¹ 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 ISM setting. Turn to the *IOI* for further details.

¹²² Remember: $(A \wedge \neg B) \vee (\neg C \wedge D) = (A \vee \neg C) \wedge (\neg B \vee D)$

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 *reals*; the only difference lies in precision: up to 64 digits precision for *short integers* in binary representation on your *WP 43S* or even (almost) 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. 142ff).

Divisions, however, must be handled differently in integer domain since the result cannot feature a fractional part here. Generally, the formula

$$\frac{a}{b} = (a \text{ div } b) + \frac{1}{b} \times \text{rmd}(a; b)$$

applies; therein, the horizontal bar denotes *real* division, div represents integer division, and rmd 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 *reals* – see the first row of the **examples** here):

$$\frac{25}{7} = 3 + \frac{1}{7} \times 4 \quad (\text{and for a real case: } \frac{25}{7.5} = 3 + \frac{1}{7.5} \times 2.5)$$

$$\frac{-25}{7} = -3 + \frac{1}{7} \times (-4) \Rightarrow \text{rmd}(-25; 7) = -4$$

$$\frac{25}{-7} = -3 + \frac{1}{-7} \times 4 \Rightarrow \text{rmd}(25; -7) = 4$$

$$\frac{-25}{-7} = 3 + \frac{1}{-7} \times (-4) \Rightarrow \text{rmd}(-25; -7) = -4$$

In general, $\text{rmd}(a; b) := a - b \times (a \text{ div } b)$ 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 rmd only if both parameters have equal signs. The general formula for mod is a bit more sophisticated than the one above:

$$\text{mod}(a; b) := a - b \times \text{floor}\left(\frac{a}{b}\right) \quad \text{with e.g. } \text{floor}\left(\frac{25}{7}\right) = 3 \text{ and} \\ \text{floor}\left(-\frac{25}{7}\right) = -4.$$

By the way, this formula applies to *reals* 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, RMD, MOD, and FLOOR¹²³ on your WP 43S for obvious reasons. They are found in INTS (cf. p. 132), together with more integer operations like CEIL, xMOD, and ^MOD (see p. 145 for an example); RMD and MOD are also on the keyboard as shifted function of **/**.



Furthermore, many exponential and logarithmic operations, x^2 and \sqrt{x} , x^3 and $\sqrt[3]{x}$, $x!$, COMB and PERM, as well as SIN, COS, and TAN operate on integers, too. Note that some of these functions will stay in integer domain while others may or will return *real* or even *complex numbers*. See the summary on pp. 144f for further information.

¹²³ Note FLOOR and CEIL are functions operating on a *real number* and returning a *long integer*.

Integers: Overflow and Carry with Short Integers

There are conditions where *overflow* and/or *carry* will be touched in arithmetic operations on *short integers* on your WP 43S.

Note there is a maximum and a minimum integer displayable for each word size and *ISM* 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* and/or *carry* will be lit in the *status bar* in this course, we will clear them (using **FLAGS CF SYS.FL OVERFL** and/or **FLAGS CF SYS.FL CARRY**) before next increment:

4:u		4:2		4:1		4:s	
0001_2	1	0001_2	1	0001_2	1	0001_2	1
0010_2	2	0010_2	2	0010_2	2	0010_2	2
...
0111_2	7	0111_2	7	0111_2	7	0111_2	7
1000_2	8	1000_2 o	-8	1000_2 o	-7	1000_2 o	-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 c	0	0000_2 c	0	0001_2 c	1		
0001_2	1	0001_2	1				

For comparison, we start another turn from 1 following the same rules but decrementing by 1 instead:

4:0		4:2		4:1		4:5	
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 \oplus	15	1111_2 \oplus	-1	1110_2 \oplus	-1	1001_2 \oplus	-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 \oplus	7	1000_2 \oplus	-0
0111_2	7	0111_2 \oplus	7	0110_2	6	1001_2	-1
0110_2	6	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 presented in the table overleaf:

Operation	Effect on CARRY	Effect on OVERFL
Shift and rotate	As demonstrated on pp. 137f.	None.
Boole's, MIRROR	None (cf. pp. 137f).	None.
$ x $, ABS	None.	Clears OVERFL (but sets it for $x = I_{min}$ in 2COMPL).
$+$, RCL+, STO+, INC, etc.	Sets CARRY if there is a <u>carry out</u> of the most significant bit, else clears CARRY.	Sets OVERFL if the result exceeds $[I_{min}; I_{max}]$, else clears OVERFL.

Operation	Effect on CARRY	Effect on OVERFL
-, RCL-, STO-, DEC, etc.	<p>Sets CARRY 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>¹²⁴ <u>into</u> the most significant bit, • in UNSIGN if $m < s$, • in SIGNMT if $m < s \& m \cdot s > 0$ <p>Else clears CARRY.</p>	Sets OVERFL if the result exceeds $[I_{min}; I_{max}]$, else clears OVERFL.
x , RCLx, STOx, +/-, $(-1)^x$, x^2 , x^3 , LCM, $x!$, etc.	None.	Thus, in UNSIGN, $\boxed{\pm}$ always sets OVERFL and $(-1)^x$ does so for odd x .
2^x	<p>Clears CARRY.</p> <p>Sets CARRY 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 CARRY for $x < 0$ (as well as for 0^0), else clears CARRY.	Sets OVERFL if the result exceeds $[I_{min}; I_{max}]$, else clears OVERFL.
e^x	Sets CARRY for $x \neq 0$, else clears.	
DBLx	None.	Clears OVERFL.
/, RCL /, STO /, DBL /, LN, LOG ₁₀ , LOG ₂ , LOG ₁₀ y, \sqrt{x} , $\sqrt[3]{x}$, $\sqrt[x]{y}$	Sets CARRY if the remainder is $\neq 0$, else clears CARRY.	Clears OVERFL but sets it in 2COMPL for the division $I_{min}/(-1)$.

¹²⁴ See the examples on previous page.

Translator's note: The so-called *borrow* in subtraction seems to be 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 *reals* also work for integers. In addition, there are some specialties as shown in the preceding chapters, and beyond:

- General mathematics:

- *Monadic* functions:

`INT`, `3√X`, `LB X`, and `LG` return *long integers* if possible (else *real* or *complex numbers*) for *long integer* input¹²⁵ or just the *short integer* part of the solution for *short integer* input,

`INT`, `X²`, `X³`, `2^X`, and `10^X` return integers as you expect, and

`INT` works for *short integers* as demonstrated on p. 133.

- *Dyadic* functions:

`+`, `-`, `X`, and `Y/X` return integers as you expect,

`/` returns an integer or a *real* in analogy to `INT`,

`IDIV` returns just the integer part of the division always,

`RMD` returns the remainder of y/x (cf. pp. 139f for examples),

`IDIVR` combines IDIV and RMD,

`MOD` returns $y \bmod x$ (cf. p. 140 for examples),

`INTY` and `LOGXY` return integers or *reals* in analogy to `INT`
(e.g. `625 ENTER↑ 5 EXP LOGXY` 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?).

- *Triadic* functions:

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

¹²⁵ E.g. `INT` returns 8 for an input of 64, i.e. for a proper square, and 8.062... for an input of 65, for instance. For an input of -64, it may return 0.+i×8 (see the chapters about *Complex Numbers* below). In analogy, `3√X` returns 2 for an input of 8, `LB X` returns 10 for an input of 1024, `LG` returns 3 for an input of 1000, etc.

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

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

- *Boole's algebra:*

AND, **NAND**, **OR**, **NOR**, **XOR**, **XNOR**, and **NOT** operate bitwise on *short integers* as shown on p. 138. They operate on *long 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. 116.

- Bitwise operations are exclusively for *short integers*:

CB, **FB**, **SB**, **ASR**, **SL**, **SR**, **RL**, **RLC**, **RR**, **RRC**, and **MIRROR** work as demonstrated on pp. 136ff.

LJ (or **RJ**) justifies the bit pattern to the left (or right) 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 *I/OI* for more information about these commands.

- Probability (cf. pp. 93f):

x! returns the factorial,

C_{yx} calculates the number of *combinations* and

P_{yx} the number of *permutations*, while

RANI# returns a (pseudo) random integer number between x and y .

- Advanced mathematics (see the *ReM*, *App. H* for more information):

B_n and **B_n*** return the *Bernoulli numbers*,

FIB the *Fibonacci number*, and

NEXTP the next *prime* number greater than x .

Many more functions accept integer input but will return different, mostly *real* output. See the *I/OI* 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 (up to 9999, 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 2nd radix mark in numeric input (see the examples below). In this case the 1st radix mark is interpreted as a blank space, the 2nd as a fraction mark. This way of input is straightforward and logically coherent:

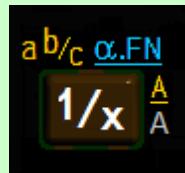
Examples:

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 $0\frac{0}{1}$ = (= 1 $\frac{0}{2}$) ¹²⁶

Each closed *real number* on the stack will be **displayed** as a fraction after **a b/c** is pressed, after a fraction is entered, or after that number is combined with a fraction by an arithmetic operation. If the fraction displayed is exactly equal, slightly less, or slightly greater than the underlying *real number*, **=**, **<**, or **>** will trail this fraction display, respectively (see examples overleaf).



¹²⁶ 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, each closed number x displayed as a fraction will be shown as a decimal *real number* after **.d** or after **DSP 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*.¹²⁷ **a b/c** toggles them. 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**

a b/c **a b/c**

and you will see:

$3.333\ 333\ 333\ 333\ 333 \times 10^{-1}$

$\frac{1}{3} >$

since $\frac{1}{3} > 0.333\ 333\ 333\ 333\ 333$.

1/x

$\frac{3}{1} =$

since this is exact.

78.40625 **-**

$\frac{-2\ 413}{32} =$

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*.¹²⁸
You will get

$11\ 372\ \frac{105}{512} =$

$1\ 033\ 4\ 713 / 5\ 632 <$

This fraction is less than the *real value*, deviating less than 0.5/5 632 from it.

¹²⁷ 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 hier der Zähler größer als der Nenner sein kann.

¹²⁸ This conversion was newly introduced on *RPN calculators* with the *WP 34S*.

Now, let's reduce the maximum denominator by

64 MODE DENMAX

64

R↓

1 033 41/49 <

FLAGS CF SYS.FL DENANY

CF SYS.FL DENFIX

1 033 27/32 >

since DENANY and DENFIX both cleared allow 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 shown 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 those isolated irrational islands in the vast sea of S/ where you may come across dimensions like in the following **example**:

A calculator stand is specified to measure $9'' \times 3\frac{1}{2}'' \times \frac{5}{8}''$. It goes without saying that your WP 43S will support you also in such harsh environments. Only absolute greenhorns, however, will expect that a tight thin-walled box around this stand will displace

9 ENTER 3 . 1 . 2 X

31 1/2 =

. 5 . 8 X

19 11/16 =

cubic inches of water.

Instead, a magic conversion factor from *cubic inches* to so called *fluid ounces* is required now.¹²⁹ 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.

¹²⁹ 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 might achieve top positions in administration – see recent experimental evidence).

Complex Numbers: Introduction

So far, we dealt with *reals* 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 *reals*; these are called *complex numbers*. If you do not know of them, leave them aside; you can profit from 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 p. 300). With *startup default* settings, **CC** separates and concatenates *real* and *imaginary* part in numeric input.



Examples (with *startup default* settings):

$3 + i \times 4$ is keyed in

3 CC 4 ENTER↑ while the display (set e.g. to FIX 5) shows in lowest numeric row:

3

$3 + i \times$

$3 + i \times 4$

3.000 00 +i×4.000 00

You enter the *real* part first – **CC** closes it – the *imaginary* part second as you write the number.¹³⁰

Input of negative *complex numbers* works in full analogy to *real number* input (cf. p. 24). Following our example above,

$3 - i \times 4$ is keyed in

3 CC 4 +/- ENTER↑,

¹³⁰ 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 is called **CPXj** for obvious reasons).

$-3 + i \times 4$ is keyed in **3 [+/-] CC 4 [ENTER]**,

$-31 - i \times 42$ is keyed in **3 1 [+/-] CC 4 2 [+/-] [ENTER]**.

Alternatively, use **3 1 CC 4 2 [ENTER] [+/-]** here.

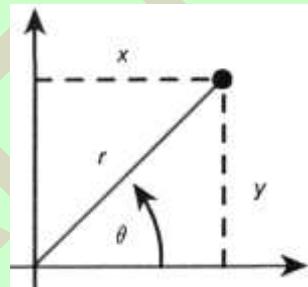
Choosing ‘scientific notation’, e.g. SCI 5, this last number will be displayed like

-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 notation, *complex numbers* may be written in polar notation as well. With polar notation set (by **FLAGS CF SYS.FL RECTN** causing \odot lit in the *status bar*), its *magnitude* (or *radius*) r shall be entered first for a new *complex number* and its *phase* (angle or *argument*) θ second. This θ may be entered in any angular notation; though often *radians* or *multiples of π* make most sense here (e.g. set **MODE RAD** causing r° lit in the *status bar*).



Example:

With *polar display mode*, *radians*, and FIX 2 set, the *complex number* ($5 ; 1.2^\circ$) is keyed in **5 CC 1.2 [ENTER]** with the display showing in lowest numeric row successively:

5

5 °

5 ° 1.2

5.00 ° 1.20

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. 125). Larger phase input is legal, but the output will be normalized always. If **0** is entered for the magnitude, θ will be set to **0** as well.

Composing and decomposing a *complex number*: Alternatively to entering a *complex number* directly, it may be generated from two closed *reals* provided in **X** and **Y**. If **L** is lit, **[CC]** will take **y** as *real part* and **x** as *imaginary part* composing the *complex number*. If **S** is lit on the other hand, it will take **y** as magnitude and **x** as phase of the new *complex number* (compare numeric input above).

Example:

```
MODE SF SYS.FL RECTN
CF SYS.FL CPXRES
DEG
4 [ENTER] -3 [EXIT]
```

These three entries return to *startup default settings* **R_L4°**.

4.
-3.

[EXIT] closes input without disturbing the stack.

[CC] composes a *complex number* out of **x** and **y** now, lights **C** and returns¹³¹

4.-i×3.

CF SYS.FL RECTN
and displays

turns **L** to **S**

5.4-36.869 897 645 844 02°

Vice versa, **[CC]** may also *cut* a *complex number* **x** into two *reals* in **X** and **Y** following the same rules.

Example (continued):

[CC] returns

r = 5.
s = -36.869 897 645 844 02°

C S remains lit in the *status bar*.

RAD **[CC]** returns

5.4-6.435 011 087 933×10⁻¹r

¹³¹ Whenever a *complex number* is returned, your WP 43S will set CPXRES and **C** will be lit in the status bar unless set before.

SF SYS.FL RECTN

turns Θ to $\underline{\Theta}$

and displays

4.-i \times 3.

[CC]

returns

Re =

4.

Im =

-3.

$\text{C}\underline{\Theta}\text{4}^r$ remains lit in the *status bar*.

Generally, *complex number* outputs follow *real number* formats (see pp. 78ff). The number of displayable decimals, however, may be limited by screen space. If you want to view both parts of a *complex number* in higher precision, press [CC], watch, and press [CC] again.¹³²

Complex results in calculations: As long as you work exclusively with *real* input, you will get only *real* results with CPXRES clear (*startup default*); you can, however, also set CPXRES to allow for *complex* results. Try **1** [+/-] [x] 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 **C** in the *status bar*).

With input closed for a *complex* x and RECTN set, for example,...

- [+/-] will change the signs of both the *real* and the *imaginary* part (as shown above),
- [CPX] conj will change the sign of the *imaginary* part only, and
- [CPX] Re \leftrightarrow Im will swap *real* and *imaginary* parts.

Press **ENTER↑** for separating *complex* input as you do in *real* domain.

¹³² Choosing RECTN and multiplication dots allows for displaying *real* and *imaginary* components using large font within $(10^{-999}, 10^{999})$ in SCI 4 together. 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 this 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 the ReM.

Example (with startup default settings):

$(1 + 2i) \times (3 + 4i)$ is entered and solved like this:

1 [CC] 2 [ENTER]

1. +i×2.

3 [CC] 4

3. +i×4

[X]

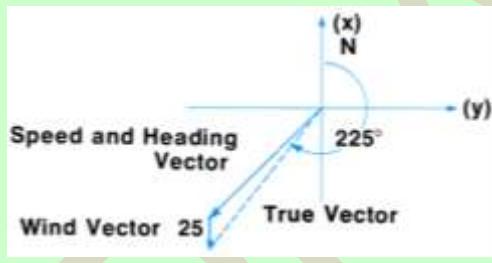
returning

-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. 156f.

Complex Numbers Used for 2D Vector Algebra

You can use *complex* domain for 2D vector algebra as demonstrated below. The functions $|x|$, $+$, $-$, CROSS, DOT, and UNITV wait for you – see the menu CPX and the IOI.



We can, for **example**, compute Mr. Sweeney's ground course (as explained on p. 126) according to the following alternative way:

[DSP] FIX [2]

[MODE] DEG

CF [SYS.FL] RECTN

125 [CC] 225 125 ↴ 225

indicated air speed and heading.

[ENTER]

125.00 ↴ 225.00°

25 [CC] 180

25 ↴ 180

for the north wind.

[+]

143.77 ↴ -142.06° for the resulting vector

[CC]

r =

143.77

θ =

-142.06°

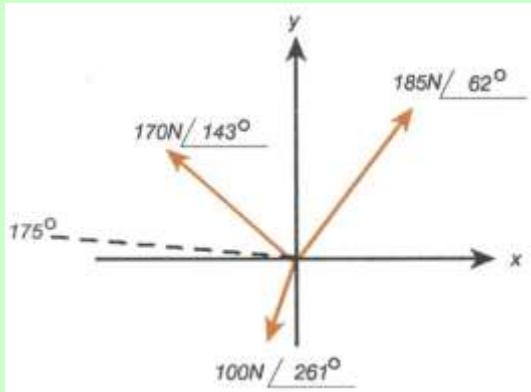
360 [+]

143.77

217.94°

(according to navigational convention, the angle must be positive. – Compare with pp. 126f.)

Two examples more (taken from the HP-42S OM):



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

CF SYS.FL RECTN

170 [CC] 143 [ENTER]

125.00 ↗ 225.00°

185 [CC] 62

185 ↗ 62

270.12 ↗ 100.43°

[+]

100 [CC] 261

178.94 ↗ 111.15°

[+]

Now, take the unit vector at 175° :

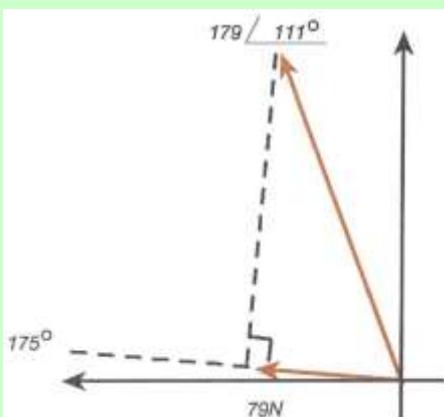
1 [CC] 175

1 ↗ 175

[MATX] dot

78.86

Thus, the resulting vector sum has a component of approximately 79 Newtons in the direction of 175° .

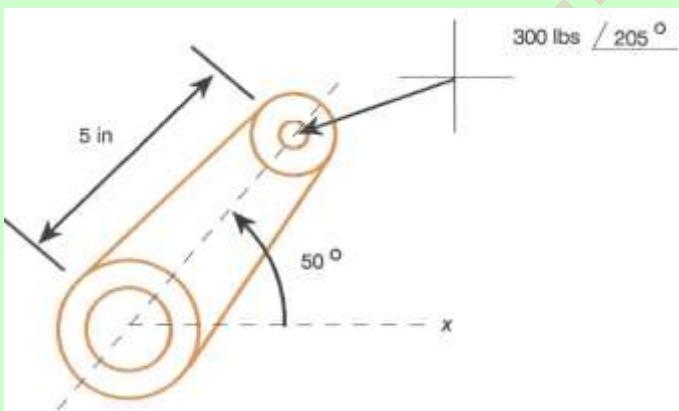


Computing Moments.

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 picture shows a two-dimensional situation. Lever and force are both acting in the drawing plane.

Solution:

MODE DEG

CF SYS.FL RECTN

ensure proper ADM and coordinates.

Key in the radius vector and the force vector:

5 [CC] 50 [ENTER]

5.00 ↘ 50.00°

300 [CC] 205

300.00 ↘ 205.

MATX 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.¹³³

¹³³ 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. See next chapters.

Complex Numbers: Summary of Functions

Many of the numeric functions operating on *reals* also work for *complex numbers*:

- General mathematics:

- *Monadic* functions:

\sqrt{x} and x^2 , $\sqrt[3]{x}$ and x^3 , 2^x and $\ln x$, 10^x and \lg , $1/x$, e^x and \ln , \sinh , \cosh , and \tanh as well as their inverses work as usual; the same applies to \sin , \cos , \tan , and their inverses (cf. also pp. 121ff for more information about angular I/O),

e^{x-1} and $\ln(1+x)$ return more accurate results with $x \approx 0$,

\pm returns $x \times (-1)$ (a.k.a. ‘unary minus’) for closed input and RECTN set while it turns x by 180° for RECTN clear, and

$(-1)^x$ returns $\cos(\pi x)$ for non-integer x .

- *Dyadic* functions:

$+$, $-$, \times , $/$, y^x and \sqrt{xy} 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

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

- Isolating and manipulating parts of *complex numbers*:

Use CC for composing and cutting,

RE for isolating the *real* part of x and IM for its *imaginary* part,

$\text{Re} \leftrightarrow \text{Im}$ for swapping its *real* and *imaginary* part,

$|\text{ }|$ for the *magnitude* of x and \angle for its *phase* (a.k.a. *argument*),

FP for the fractional part of x and IP for its integer part;

sign and UNITY 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 did on HP-42S), and

RSD *n* rounds *x* to *n* significant digits.

- Advanced mathematics (see the ReM, App. H for comprehensive information about the functions following):

- *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 – cf. pp. 121ff),

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

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

x! (= $\Gamma(x + 1)$) and **Γ(x)** calculate the *complex Gamma function*, and

lnΓ returns its natural logarithm.

- *Dyadic* functions:

AGM returns the *arithmetic-geometric mean*,

COMB and **PERM** calculate with *complex Gamma*,

β(x,y) returns *Euler's Beta function*, and

lnβ its natural logarithm.

Vectors and Matrices: Introduction and Input

So far, we dealt with just 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 in an array of 4, 6, 8, 9, 10, 12, 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*. Its function set is based on the one of *HP-42S* and extends it.

Generally, 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 **DSP 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*:

$[0.0 \quad 0.0 \quad 0.0]^T$						0.0
RNORM	ENORM	STOEL	RCLEL	PUTM	GETM	
dot	cross	UNITY	DIM	INDEX	EDITN	
NEW	$[M]^{-1}$	$ M $	$[M]^T$	SIM EQ	EDIT	

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*:

						0.0
$[0.0 \quad 0.0 \quad 0.0]^T$						
$1;1 = 0.0$						
INSR	DELR	WRAP	GROW			
←	↑	OLD	GOTO	↓	→	

Note the 1st 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**

						0.0
$[4.0 \quad 0.0 \quad 0.0]^T$						
$1;1 = 4.$						

Move the cursor to the next element: →

						0.0
$[4.0 \quad 0.0 \quad 0.0]^T$						
$2;1 = 0.0$						

Continue editing: **5** **+/-** → **6.7**

						0.0
$[4.0 \quad -5.0 \quad 6.7]^T$						
$3;1 = 6.7.$						

EXIT

						0.0
						[4.0 -5.0 6.7]^T
	RNORM	ENORM	STOEL	RCLEL	PUTM	GETM
	dot	cross	UNITY	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]
						[0.0 0.0 0.0]
						[0.0 0.0 0.0]
						1;1= 0.0

	INSR		DELR		WRAP	GROW
	←	↑	OLD	GOTO	↓	→

Three numeric rows are required for editing *x* now. The 3×1 matrix in **Y** above is the 3D vector we just entered before; note any matrix is displayed in this short form (with a **x** even for MULT• chosen) in any *stack register* but **X**.

Again, all elements of the new matrix start containing zero. Its 1st element is displayed inverted as the 1st 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 → 7 →

[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 1st element of row 2. So you can simply continue row-wise:

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	UNITV	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 2/3; and you want more than just one decimal displayed in the result:

DSP FIX 3

2 . 3

0.000

[3x1 Matrix]

[2x3 Matrix]

$$0 \ 2/3...$$

Press **X** and you will get immediately

$$\begin{bmatrix} 0.000 \\ [3 \times 1 \text{ Matrix}] \\ -0.667 & 8.000 & 2.667 \\ 16.333 & 0.000 & 1.000 \end{bmatrix}$$

which are all matrix elements multiplied by $2/3$ 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 quadratic matrix $[MA] = \begin{bmatrix} 4 & -3 \\ -2 & 1 \end{bmatrix}$ and fill it directly.

2 **ENTER↑** **DIM** **α M A** **ENTER↑** creates **MA** as a 2×2 matrix.
EDITN **VAR** **MA**

$$\begin{bmatrix} 0.000 & 0.000 \\ 0.000 & 0.000 \\ 1;1= 0.000 \end{bmatrix} [2 \times 3 \text{ Matrix}]$$

INSR	DELR	WRAP	GROW
←	↑	OLD	GOTO
→	↓		→

4 **→** 3 **+L** **→** 2 **+L** **→** 1

$$\begin{bmatrix} 4.000 & -3.000 \\ -2.000 & 1.000 \\ 2;2= 1 \end{bmatrix} [2 \times 3 \text{ Matrix}]$$

Now, press **EXIT** and you are done with **MA** – while the screen looks just as before again:

							0.000	
							[3×1 Matrix]	
							-0.667	8.000 2.667
							16.333	0.000 1.000
	RNORM	ENORM	STOEL	RCLEL	PUTM	GETM		
	dot	cross	UNITV	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 *reals*). 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* 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 automatically.

Example (continued):

DSP FIX 5

							0.000	
							[3×1 Matrix]	
							-0.666 67	8.000 00 2.666 67
							16.333 33	0.000 00 1.000 00

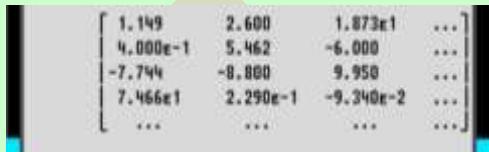
If font switching should not suffice, your *WP 43S* will furthermore automatically turn to abbreviated **SCI 3** for the elements of the respective matrix. This allows for showing arbitrary 5×4 *real* matrices entirely. If a *real* matrix exceeds five rows, its fifth row is displayed filled with ellipses (...); if it exceeds four columns, its fourth column is shown filled with ellipses.

Example:

Assume a 6x5 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 always.

For matrices larger than 5 rows and/or 4 columns, the display may vary depending on the cursor position: ellipses may appear on top and bottom, left and right side. A view of 3x3 matrix elements including the one selected by the cursor can be seen always at least – this selected element is also displayed below of the matrix in the format you have chosen for **reals**. Since the indices of this element are shown there as well you always know where you are.

Example (continued):

Press **MATX EDIT** and you will see:



The 1st 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 you want 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 1st, 2nd, etc. to the last row and then return to the 1st row in default WRAP mode. If GROW is set instead, another **→** from the very last (i.e. bottom right) matrix element 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:

[2×3 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 1st matrix element is not highlighted anymore since you left the *Matrix Editor*. Thus, just entering **4** will display (due to *automatic stack lift*) now:

[7×5 Matrix]

4...

So matrix editing is easy and straightforward. The *I/O* contains additional information, also about the further commands DELR, INSR, and R↔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. 163):

Create and store a *complex* matrix $\begin{bmatrix} 5 + 8i & \pi i \\ -2 & 4 - 3i \end{bmatrix}$.

Solution:

Remember we have created a 2×2 matrix just a few pages ago. So it is most easy to recall it for using it as a template:

RCL VAR MA

DSP FIX 2

MATX EDIT

since we will not need five decimals for the process following.

[6×5 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** **T** **→** **→** 4 **CC** 3 **+/-**
EXIT
STO 0 1

[6×5 Matrix]
[5.00+i×8.00 i×3.14]
[-2.00 4.00-i×3.00]

Since we edited on the stack and stored the resulting new *complex* matrix in a new location, the old *real* matrix **MA** is not affected at all.

Compare pp. 149f 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. 162, 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 \mathbf{X} , pressing...

- [x] will extract 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).
- $\text{[x}^2]$ will square each matrix element individually (use $\text{ENTER} \uparrow \text{x}$ for squaring the matrix instead);
- [|x|] will calculate the absolute value of each matrix element (instead, use $\text{[MATX]} \text{ENORM}$ for calculating the *Euclidean* norm of the matrix or take |M| for getting its determinant);
- [+/-] will change the sign of each matrix element.

You can also let the *dyadic* functions $\text{[+]} \text{, } \text{[-]} \text{, } \text{[x]} \text{, } \text{[/]}$ operate on two matrices or vectors alone (i.e. *data types* 8 and 9), provided the rules of *linear algebra* are obeyed:

	y	x	Op.	Resulting x
[+]	$[m \times n \text{ Matrix}]$	$[m \times n \text{ Matrix}]$	$[y] + [x]$	$[m \times n \text{ Matrix}]$
-			$[y] - [x]$	
\times	$[m \times n \text{ Matrix}]$	$[n \times p \text{ Matrix}]$	$[y] \cdot [x]$	$[m \times p \text{ Matrix}]$
/	$[m \times n \text{ Matrix}]$	$[n \times n \text{ Matrix}]$	$[y] \cdot [x]^{-1}$	$[m \times n \text{ Matrix}]$

The 1st 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.¹³⁴ If either matrix is *complex*, the result will be *complex* in most cases as well.

Example (continuation of p. 163):

Multiply the matrices in **R00** and **MA**. Output format shall be FIX 3.

Solution (we omit the *menu section* in the following pictures):

DSP FIX 3

RCL 0 0

[2×2 C matrix]

$$\begin{bmatrix} -0.667 & 8.000 & 2.667 \\ 16.333 & 0.000 & 1.000 \end{bmatrix}$$

Note the '2×2 C matrix' in **Y** is the *complex* matrix we entered in previous chapter – the *stack* handles matrices as it handles other objects. Now let's recall **MA**:

¹³⁴ Remember matrix multiplication behaves different than multiplication of numbers. Generally, for two arbitrary matrices A and B of matching sizes, $A \cdot B \neq B \cdot A$. Also note that only square matrices can 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 (i.e. invertible) matrix here – else you cannot divide by that matrix. Only square matrices can be inverted.

RCL **VAR** **MA**

(or **RCL** **α M A** **ENTER↑**, if you have defined many variables already – cf. p. 56)

$$\begin{bmatrix} 2 \times 3 & \text{Matrix} \\ [4.000 & -3.000] \\ [-2.000 & 1.000] \end{bmatrix}$$

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

$$\begin{bmatrix} 2 \times 2 & \text{c matrix} \\ [-79.000 & 48.000 & 19.000] \\ [27.000 & -24.000 & -11.000] \end{bmatrix}$$

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 **VAR** **M**

EDIT etc.

$$\begin{bmatrix} 3 \times 2 & \text{Matrix} \\ [1.000 & 2.000] \\ [3.000 & 4.000] \end{bmatrix}$$

[M]⁻¹

$$\begin{bmatrix} 3 \times 2 & \text{Matrix} \\ [-2.000 & 1.000] \\ [1.500 & -0.500] \end{bmatrix}$$

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 $\mathbf{m} = \mathbf{p} = 1$. 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

$$\begin{aligned} \vec{F} &= q \vec{v} \times \vec{B} \\ &= 1.6 \cdot 10^{-19} \cdot 10^7 \cdot (0.4 \quad 2.8 \quad -1.2) \times (1.3 \quad -0.3 \quad 0.7) \end{aligned}$$

Solution:

Just remember that in cross products, vectors must be entered in proper sequence as written from left to right:

DSP FIX 2

since this will suffice for that process.

3 ENTER↑ 1 MATX DIM α **V ENTER↑** creates **v** as a 3x1 matrix.

RCL VAR V

EDIT etc.

$$\begin{bmatrix} 2 \times 2 & \text{Matrix} \\ [0.40 & 2.80 & -1.20] \end{bmatrix}$$

STO α **B** **ENTER↑**

creates **B** as a 3x1 matrix, too.

RCL **VAR** **B**

EDIT etc.

[3x1 Matrix]

[1.30 -0.30 0.70]

cross

[2x2 Matrix]

[1.60 -1.84 -3.76]

E 7 **X**

1.6 **E** **+/-** 19 **X** resulting in

[2x2 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

[2x2 Matrix]

5.14×10^{-11}

Compare with the weight of a proton:

CNST **α** **M** **P** **^**

G **^**

recall the proton mass.

recall earth acceleration.

5.14×10^{-11}

1.64×10^{-26}

So this is a force ratio of

/

3.14×10^{15}

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. 149ff). In *complex* domain, 2D vector multiplication is possible since the commands DOT and CROSS are contained in CPX as well. Cf. pp. 153ff for 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.¹³⁵ To deal with such a system of linear equations, proceed as follows:

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:

	Mat A	Mat B	Mat X

¹³⁵ 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* at execution time.

2. Press **Mat A**. The *Matrix Editor* will open and you can enter the elements of the 4×4 *coefficient matrix* (see on pp. 158ff 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 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 **VAR** **M**
DSP **FIX** **0** **1**

MATX **EDIT** etc.

$$\begin{bmatrix} 2.0 & 1.0 \\ 6.0 & 1.0 \end{bmatrix}$$

STO **VAR** **M**

The eigenvalues are the solutions of the *characteristic polynomial* of this problem:

$$(2 - \lambda)(1 - \lambda) - 6 = 0$$

▲ EIGVAL returns

$$M = [2 \times 2 \text{ Matrix}]$$

$$\begin{bmatrix} 4.0 & 0.0 \\ 0.0 & -1.0 \end{bmatrix}$$

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 **VAR** **Q**

EDIT etc.

$$\begin{bmatrix} 2 \times 2 \text{ c Matrix} \\ [0.0 \ 0.0 \ -2.0] \\ [1.0 \ 2.0 \ 1.0] \\ [1.0 \ 0.0 \ 3.0] \end{bmatrix}$$

EIGVAL returns

$$Q = \begin{bmatrix} 3 \times 3 \text{ Matrix} \\ [2.0 \ 0.0 \ 0.0] \\ [0.0 \ 2.0 \ 0.0] \\ [0.0 \ 0.0 \ 1.0] \end{bmatrix}$$

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 \leftrightarrow y returns **Q** into **X**:

$$\begin{bmatrix} 3 \times 3 \text{ Matrix} \\ [0.0 \ 0.0 \ -2.0] \\ [1.0 \ 2.0 \ 1.0] \\ [1.0 \ 0.0 \ 3.0] \end{bmatrix}$$

EIGVEC pushes this matrix on the stack:

$Q = [3 \times 3 \text{ Matrix}]$

$$\begin{bmatrix} 1.0 & 0.0 & -2.0 \\ 0.0 & 1.0 & 1.0 \\ -1.0 & 0.0 & 1.0 \end{bmatrix}$$

STO **α** **V** **ENTER↑**

X returns

$[3 \times 3 \text{ Matrix}]$

$$\begin{bmatrix} 2.0 & 0.0 & -2.0 \\ 0.0 & 2.0 & 1.0 \\ -2.0 & 0.0 & 1.0 \end{bmatrix}$$

RCL **L** recalls V.

▲ **[M]⁻¹**

X returns

$[3 \times 3 \text{ Matrix}]$

$$\begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

This looks very much like what was returned for the eigenvalues of Q above. Let's check:

DSP **FIX** **4**

- returns

$[2 \times 2 \text{ C Matrix}]$

$$\begin{bmatrix} 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \\ 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \\ 0.000\ 0 & 0.000\ 0 & 0.000\ 0 \end{bmatrix}$$

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 $\begin{bmatrix} -38 & \frac{43}{7} & \frac{63}{2} & \frac{1149}{14} \\ -14 & \frac{19}{7} & 7 & \frac{181}{7} \\ -\frac{8}{7} & -\frac{122}{49} & \frac{24}{7} & \frac{177}{49} \\ -16 & \frac{26}{7} & 13 & \frac{244}{7} \end{bmatrix}$?

Solution:

4 [ENTER] [MATX] NEW

creates a 4x4 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]

-38.000 0	6.142 9	31.500 0	82.071 4
-14.000 0	2.714 3	7.000 0	25.857 1
-1.142 9	-2.489 8	3.428 6	3.612 2
-16.000 0	3.714 3	13.000 0	34.857 1

[DSP] FIX [0] [1]

shall suffice here:

-38.0	6.1	31.5	82.1
-14.0	2.7	7.0	25.9
-1.1	-2.5	3.4	3.6
-16.0	3.7	13.0	34.9

[MATX]



EIGVAL returns:

-5.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	3.0	0.0
0.0	0.0	0.0	5.0

Note the 2nd eigenvalue is zero here.

RCL [L]

EIGVEC

displays:

$$\begin{bmatrix} 4.0 & 5.0 & 4.0 & 5.0 \\ 3.0 & -2.0 & 2.0 & -4.0 \\ 1.0 & -4.0 & -3.0 & 5.0 \\ 1.0 & 4.0 & 3.0 & 1.0 \end{bmatrix}$$

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. 110 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:

4. Create a 1×2 named matrix and open it for editing:

1 [ENTER] **2** **DIM** **α M S A** [ENTER] creates **MSA** accordingly.
EDITN **VAR** **MSA**



GROW allows the matrix to grow with data entered.

Now key in all 30 pairs of measured values. The 1st value shall be x , the 2nd be 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 VAR MSA

Σ+

Calling $\Sigma+$ with a 30×2 matrix in **X** will display the data of the 30th data pair in **X** and **Y** (and save a copy of the data point matrix in **L**).

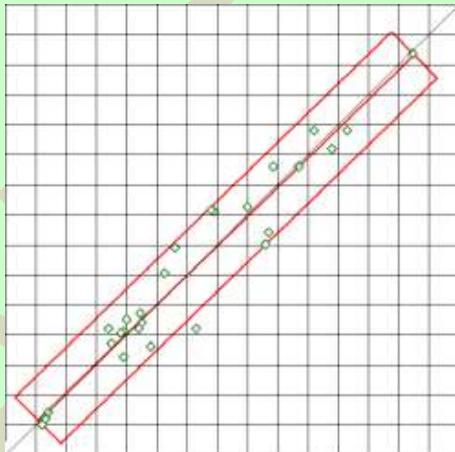
5. It is recommended to plot these 30 data points ¹³⁶ (see a typical diagram here but check p. 110 and its footnote as well).
6. Let your *WP 43S* fit a straight line through the points and compute

$$c_0 = \frac{T}{30s_x s_y} \sqrt{\frac{s_x^2 + r^2 s_y^2}{1 - r^2}} \quad \text{with } T$$

being the width of the tolerance zone you want to control:

▼ OrthoF
▼ r x² STO K
▼ s x² R↓
x² X
R↑
+
1 RCL - K
/ R
s X / 30 /
.01 X

select the orthogonal linear fit model.
get the *correlation coefficient* and store its square.
get s_x^2 and 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 2nd factor now.
divide by **30 s_x s_y**.
this returns c_0 for our **T** now (cf. pp. 110ff).



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.

¹³⁶ Steps 5 and 6 are actually the same as shown in the application above with input of separate real numbers (instead of one matrix) already. They are just repeated here for sake of completeness.

Vectors and Matrices: Summary of Functions

Assume \mathbf{X} contains a matrix. Then there are functions operating on x as a whole and others just operating on its elements individually. Let us list the first set first:

- General mathematics:

- *Monadic* functions operating on the entire matrix x :

ENORM computes the *Euclidean norm* of x (i.e. a *real number*),

RNORM computes the row norm of x (i.e. a *real number*),

RSUM computes the row sum of x (i.e. a vector),

|M| computes the determinant of x (i.e. a *real or complex number*),

[M]^T returns the transpose matrix of x ,

[M]⁻¹ returns the inverse matrix of x ,

EIGVAL returns the eigenvalues of x , and **EIGVEC** its eigenvectors (cf. pp. 174ff), while

UNITY returns the unit vector of x (see the ReM).

- *Monadic* functions operating on each element x_{ij} of x individually:

(+/-), **(1/x)**, **(|x|)**, **(\sqrt{x})**, **($\sqrt[3]{x}$)** and **(x^2)**, **($\sqrt[n]{x}$)** and **(x^3)**, **(2^x)** and **(lb x)**, **(e^x)** and **(ln)**, **(10^x)** and **(lg)**, **(sin)**, **(cos)**, **(tan)**, **(sinh)**, **(cosh)**, and **(tanh)** as well as their inverses work as explained for *real and complex numbers* above,

e^x-1 and **ln 1+x** return more accurate results with $x_{ij} \approx 0$;

sinc returns a matrix containing $\frac{\sin(x_{ij})}{x_{ij}}$ for $x_{ij} \neq 0$ and 1 for $x_{ij} = 0$ (input is converted to *radians* before calculating – cf. pp. 121ff),

(-1)^x returns $\cos(\pi x_{ij})$ for non-integer x_{ij} .

RDP n rounds x_{ij} to n decimal places in FIX format,

ROUND rounds x_{ij} using the current display format, and
RSD n rounds x_{ij} to n significant digits.

For *complex* matrices, **conj** returns a matrix with the *complex conjugates* of x_{ij} .

For *real* matrices,

ceil returns a matrix with the smallest integers $\geq x_{ij}$ and
floor with the greatest integers $\leq x_{ij}$,

FP returns a matrix with the fractional parts of x_{ij} and
IP with their integer parts, while

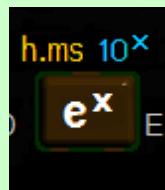
sign returns a matrix with each x_{ij} replaced by $\text{signum}(x_{ij})$.

- *Dyadic* functions operating on x and y :
 - **[+]**, **[-]**, **[X]**, and **[/]** work as explained on pp. 168ff,
 - **cross** operates on two *real* 2D or 3D vectors of identical size as shown on pp. 171f, and
 - **dot** operates on two matrices of identical size.
- *Dyadic* function operating on each element y_{ij} of y individually:
 - **log_xy** calculates the logarithms of y_{ij} for base number x .
- Isolating and manipulating bulk parts of a *complex* matrix x : Use ...
 - **CX→RE** for cutting x in its two parts,
 - **RE→CX** for composing x from its two parts,
 - **Re** for isolating its *real* part and **Im** for its *imaginary* part, and
 - **Re↔Im** for swapping its *real* and *imaginary* part.

The functionality of the Matrix Editor was demonstrated in the chapters above. Turn to the *ReM* for additional information about all matrix operations provided on your *WP 43S*. If you look for more general information about vectors and matrices, and further applications, please turn to a textbook covering *linear algebra*.

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.mmmffff` 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 or less than three digits).



Example (with *startup default* settings):

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 seen in digital clocks or watches. Note there is no rounding of *hours*, *minutes*, or *seconds* for *times*.

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 2nd colon. You can choose 12- or 24-hour display for *time of day*.

Example (continued):

Call TIME in the evening and you might get

21:47

FLAGS **CF** **SYS.FL** **TDM**

9:47 p.m.

- ☞ When *time of day* is returned by a function, it will be displayed according to your choice – internally, however, it is stored as standard 24-hour *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* – the result will stay a *time*. If you add an integer,

rational or *real number* to such a *time*, it will be automatically converted to a *time* before adding. This applies to subtractions in analogy. Compare the matrices on pp. 70f.

Example (with *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	
.12	[h.ms]		[+]	1:45	
.14			[+]	1:57	
5.25	[h.ms]		[x\Rightarrowy]	[−]	2:12
				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) sexagesimal *times* to decimal numbers using **[.d]**, and reconvert the decimal result to sexagesimal *times* by pressing **[h.ms]**.

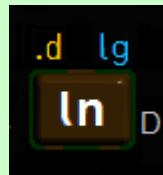
There is only one more dedicated *time* command – **SETTIM**, serving obvious purposes.¹³⁷ **GAP**, **ALL**, **ENG**, **FIX**, or **SCI** have no effect on *times*.

¹³⁷ Note the real time clock in your *WP 43S* may deviate from true time by up to one minute per month (i.e. ± 25 ppm approximately, caused by parts tolerances; you live with this wearing a quartz watch as well – mechanical watches are less accurate generally). This deviation does neither affect real-world time calculations nor the **TIMER** application described in *Section 5*. If you are accustomed to radio controlled timepieces, however, you might find regular adjustments necessary.

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](#)). /SO Y.MD is *startup default*.

Date input is decimal according to the *DDM* chosen and is terminated by [.d](#) (as shown on pp. 67f).



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 x→DATE](#), and any triple of *reals* or integers via [→DATE](#) (cf. p. 39). Input containing more than the necessary digits for a *date* in the *DDM* selected will be rounded.

Vice versa, [DATE→](#) splits a *date* in three integers and pushes them on the stack as demonstrated on p. 39. Note that both [DATE→](#) and [→DATE](#) observe the *DDM* chosen. If you want to extract particular information from a *date* independent of current *DDM*, we recommend using one of the operations DAY, MONTH, or YEAR.

Like in the *status bar*, a 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 effective *DDM* from looking at the date format in the *status bar*.

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 .¹³⁸

Expect similar returns after **CLK DATE**.¹³⁹

There is only one more dedicated *date* command – SETDAT, serving obvious purposes.

Note integers (or the integer parts of *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.

But that's it – these are all the legal operations. Compare the matrices on pp. 70f. GAP, ALL, ENG, FIX, or SCI have no effect on *dates*.

¹³⁸ Translator's note: Numeric output of WDAY corresponds to Chinese weekdays 1 to 6 directly. For Portuguese weekdays ('segunda-feira' etc.), add 1 to days 1 to 5.

¹³⁹ 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 1582-10-15 in the catholic world. Large parts of the world took their time and switched later (see the chapter *Localizing Calculator Output* above and check *Wikipedia* for the dates applicable). Note, however, there are still also other calendars in use on this planet today, e.g. in the Muslim world.

Dates before the year 8 A.D. may be indicated differently than they were experienced at the time due to the inconsistent application of the leap year rule before. We count on your understanding and hope this shortcoming will not affect too many calculations.

Note that 8 A.D. should be written A.D. 8 or even better A.D. VIII instead – 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 (actually, this notation was broadly introduced some XL – or even CD – years later). 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 daggered in DCCIX A.V.C.; calendars may be a sensitive topic.

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. 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);¹⁴⁰
3. prefix **g** leads to homophonic Greek letters.¹⁴¹

Upper and lower case are set by **▲** and **▼**, respectively, applying also to the letters in Myα and CATALOG'CHARS'αINTL (see pp. 189f).

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** **+L** calling \pm .

¹⁴⁰ For people writing German, for example, Myα might look like pictured overleaf. See Section 6 for learning how to populate Myα.

¹⁴¹ This will work wherever applicable, with “homophonic” following classic Greek pronunciation. Kudos to *Thales*, *Pythagoras*, *Heraclitus*, *Leucippus*, *Democritus*, *Aristotle*, *Archimedes*, *Euclid* (ό Θαλῆς ὁ Μιλήσιος, ο Πυθαγόρας ο Σάμιος, ο Ἡράκλειτος ο Ἐφέσιος, ο Λεύκιππος, ο Δημόκριτος, ο Ἀριστοτέλης, ο Ἀρχιμήδης ο Συρακούσιος και ο Εύκλειδης), and their colleagues for laying the foundations of logics, mathematics, and physics (i.e. ή λογική και ή μαθηματική τέχνη και ή φυσική ἐπιστήμη – note that the first two were called “practical arts” and the latter “theoretical science”) as we know them today – starting some 2600 years ago. And kudos to the unnamed Babylonian mathematicians who built the foundations for those Greeks, actually recording e.g. what we call “*Pythagoras’ theorem*” nowadays 1200 years before Pythagoras.

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** corresponding to **Tau**), and **Eta** via **g** **J**. These three letters are printed in blue on the keyboard as reminders. **Omicron** is not featured since looking exactly like the Latin letter **O** in either case.

Two extra 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 **I** and **M**.



And three alpha *menus* become accessible for more letters, punctuation marks as well as mathematical and other symbols (abbreviations are printed in gold and blue on the keyboard as reminders). Look up their contents in Section 2 of the ReM, and use FBR to browse the entire character set provided.

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
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 Ρ Σ Τ Υ Φ Χ Ψ Ω
α β γ δ ε ζ η θ ι Κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω
0 1 2 3 4 5 6 7 8 9 + - × or ÷ / . , : ? ξ ± # ☰

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**

a b c d e i j k l m n o p r s u w y z α δ ρ

and **0 1 2 3 4 5 6 7 8 9 + -**

as well as superscripts **a 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'αINTL (together with 73 more supporting international communication), the 24 basic (plus eleven accented) Greek letters in CATALOG'CHARS'A...Ω. αINTL can be called via **f A** in AIM; see the ReM for its content.

So you may, for **example**, easily store and display an actual modern Greek message like

Οι μελλοθάνατοι σε χάιρετουν.

Actually, we could have written the major part of this manual just using said small font. It covers at least 47 languages from Afrikaans to Zhōngwén (see the ReM), 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 store Deng Xiaoping's famous and successful slogan

Bùguǎn bái māo, hēi māo, dàizhù lǎoshǔ
jiù shì hǎo māo!

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 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* (αMATH , called via $\text{g } \boxed{-}$, and $\alpha\bullet$, called via $\text{g } \boxed{\square}$) contain further mathematical and non-mathematical symbols and marks (see the *ReM*):

! ; ¿ " ' \$ \$ £ ¥ € % & () * < = > @ [] \ ^ _ { } | ~
¢ « » ¬ ± · ÷ ← ↑ → ↓ ♀ ø © £ ¤ || ∞ √ ∫ ≈ ≡ ≈ ≠ ≤ ≥
⊗ ⊥ ⊕ ⊖ ⊗ ⊖ ⊕ ⊕ etc.

Pressing **USER** in *AIM* toggles *USER@*. Individual characters may be assigned to particular locations on the keyboard or within *menus* in *AIM* only (see pp. 283ff for how to do that). Such user assignments will become accessible when *USER@* is set (indicated by U and A or α being both lit in the *status bar*).

Alpha input can be edited character by character (like numeric input can be edited digit by digit) using **◀** as long as it is open still.

AIM is closed by **ENTER↑** (duplicating string x in y) or by **EXIT** unless pressed in a *menu*. Empty strings will not be pushed on the stack.

Example (continued):

Pressing **EXIT** with Dèng Xiǎopíng's slogan keyed in will display it as shown above. Hence, the contents of **Y** and **Z** will be shifted up on the screen and *t* is not displayed anymore for the time being.

Pressing **ENTER↑** instead will display the text twice (in **X** and **Y**). The screen allows for showing *z* in addition still.

Stack *register* contents pushed out of the screen by long *alpha strings* will show up again when said strings are not on display anymore.

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 here 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. 70). Let's enter a 2nd string now by pressing **α** and the necessary characters:

The train will arrive at 23:55
sharp at Victoria station..

Leave AIM and close *x* by pressing **EXIT**:

The train will arrive at 23:55
sharp at Victoria station.

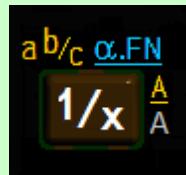
Now we have got *alpha strings* in **X** as well as in **Y**, so pressing **+** once more will append *x* to *y* returning

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 string in the source 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 starting (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 *alphanumeric string*, the entire string is appended to the destination.

If **X** contains a matrix, $x \rightarrow \alpha$ uses each element in the matrix as a character code or *alphanumeric string*. $x \rightarrow \alpha$ begins with the 1st element (1; 1) and continues row-wise (to the right) until reaching the end of the matrix.

aSL source shifts the source string by x characters to the left, deleting the first x characters from the string.

aSR source shifts the source string by x characters to the right, deleting the last x characters from the string.

You can also compare strings using commands in [\(TEST\)](#) to create something like a sorted list. A string (A) is called “smaller” than another one (B) if it precedes (B) in sorting.

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-holds at quality department stores or campus bookstores. Call 800-538-7922 (in California, 800-662-9862) 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 172 offices in 65 countries

Mail to: Hewlett-Packard, 1504 Page Mill Road, Palo Alto, CA 94304.
Please send me further information on:

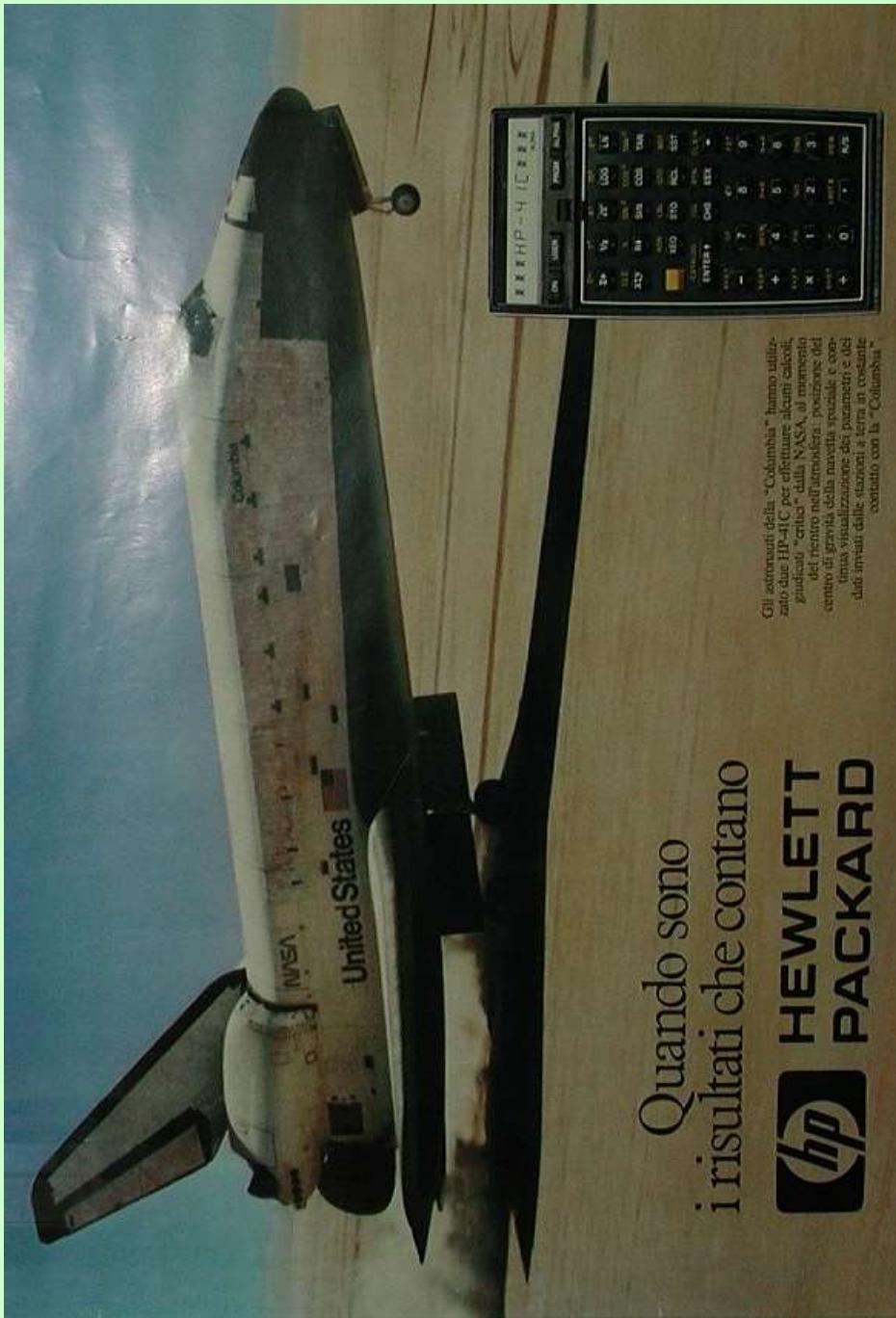
- HP 2000 Series A Logic/Stat Analysis
 HP 43S Hand-Held Programmable Calculator

Name _____
Company _____
Address _____
City _____ State _____ Zip _____

*Domestic USA price only.

HP43S

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**



**HEWLETT
PACKARD**



Gli sbarcati della "Columbia" hanno utilizzato due HP-41C per effettuare alcuni calcoli, guidati "on-line" dalla NASA, al momento del rientro nell'atmosfera: posizione del centro di gravità della navetta speciale e controllata visualizzazione dei parametri e dati inviati dalle stazioni a terra in costante contatto 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. 20ff). 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 one routine only. 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) – cf. p. 22. 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.¹⁴² 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 commands in your routine may also access each and every global *register*, variable, or *flag* provided –

¹⁴² You cannot see that first END but the last one is visible – as pictured e.g. overleaf.

there are (almost) no limits. You are the sole and undisputed master of the memory! ¹⁴³

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 differentiating these two kinds of routines further on.

Enough of theory – press **RTN** and switch to *PEM* via **P/R**. The display of your *WP 43S* will change to something like this:



2018-07-15 14:31 R _E 4° /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. 22f (the *status bar* on top will differ according to your time and settings).

¹⁴³ 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 *user flags* a particular routine uses, and to document their purposes and contents for later reference.

An alternative – 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.¹⁴⁴ 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****000**. 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. 20ff). Each new step will be inserted right after the *current step*. You find

- **LBL** for LaBeLing a routine or a program step following,

¹⁴⁴ There is no routine-specific step counting like in the *HP-42S* or *HP-35S*.

- **XEQ** for e~~XEC~~Uting or calling a specific routine,
- **RTN** for Re~~Tur~~Ning to the caller of the current routine,
- **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. 197, continued to the left by 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. 204).

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:

$$\text{base area} = \pi r^2$$

$$\text{volume} = \text{base area} \times \text{height} = \pi r^2 h$$

$$\text{surface area} = 2 \text{ base areas} + \text{side area} = 2 \pi r^2 + 2 \pi r h$$

<i>r</i>	<i>h</i>	Base Area	Volume	Surface Area
2.5	8.0	?	?	?
4.0	10.5	?	?	?
4.5	4.0	?	?	?
TOTALS		?	?	?

Method:

1. Enter an *r* value into the calculator and save it for other calculations. Calculate the base area (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 M K ENTER↑
STO α ▽ R ENTER↑
x²
π
x
STO α B A S E ENTER↑
STO + α 9 S B ENTER↑
VIEW VAR BASE
P.FN INPUT α ▽ H ENT↑
x
STO α V O L U M E ENT↑
STO + α 9 S V ENTER↑
VIEW VAR VOLUME
RCL VAR r
/

```

LBL 'MK'	Switch to PEM
STO 'r'	Store radius
x ²	
π	
x	Compute base
STO 'BASE'	
STO+ 'ΣB'	Sum of bases
VIEW 'BASE'	Show base for
INPUT 'h'	1 s
x	Enter height
STO 'VOLUME'	Compute volume
STO+ 'ΣV'	
VIEW	Sum of volumes
'VOLUME'	Show vol. for 1 s
RCL 'r'	

[2]	/	
[X]	2	
RCL VAR BASE	x	Compute side
[2]	RCL 'BASE'	
[X]	2	
[+]	x	
STO + [α] 9 [S] [S] [ENTER]	+	Compute surface
RTN	STO 'ΣS'	Sum of surfaces
P/R	RTN	End of routine
		Leave PEM

Now, let's run the program:

2.5	2.5	2.50	1 st can: radius
DSP FIX [2]			
XEQ PROG MK	BASE = 19.64		
8	8		Height
R/S	VOLUME = 157.08	164.93	Surface
4	4		2 nd can: radius
XEQ PROG MK	BASE = 50.27		
10.5	10.5		Height
R/S	VOLUME = 527.79	364.43	Surface
4.5	4.5		3 rd can: radius
XEQ PROG MK	BASE = 63.62		
4	4		Height
R/S	VOLUME = 254.47	240.33	Surface
RCL VAR ΣB		133.52	Sum of bases
RCL VAR ΣV		939.34	Sum of volumes
RCL VAR ΣS		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 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 preceding programmable pocket calculators. Your *WP 43S* allows for specifying a wide variety of alphanumeric labels as described overleaf.

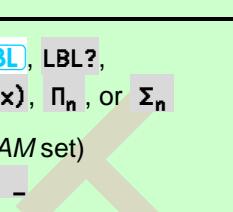
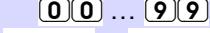
Whenever a step like e.g. GTO **labl** is encountered in *run mode* (with **labl** representing an arbitrary label), your *WP 43S* will **search this label** using one of the two following methods:

1. If **labl** 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 **labl** so far, the quest will continue right after previous END (so the search will stay in the *current program*). This is the procedure for *local labels*. So, *local labels* are valid in the *current program* only and may hence be reused in another program.
2. If, however, **labl** 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*.¹⁴⁵

So, *global labels* can be accessed from anywhere in memory, while *local labels* can only be accessed from within their own program.

¹⁴⁵ These search procedures for local and global labels are as known from the *HP-41C*.

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_n , or Σ_n OP _ (with TAM set) e.g. GTO _	
2	User input Echo	α ¹⁴⁶ sets AIM. OP ‘	 ¹⁴⁷ opens indirect addressing. OP → _
			2-digit numeric (local) label  OP nn e.g. LBL 07
3	User input Echo	Alphanumeric (global) label (up to 7 chars ¹⁴⁸) OP ‘label e.g. SLV ‘F1μ’	Stack or lettered register  OP → x e.g. $\int fd \rightarrow ST.T$
			Register number   OP → nn e.g. XEQ → 44

SLV ‘F1μ’ solves the function given in the routine labeled **F1μ** (see pp. 238ff).

$\int fd \rightarrow ST.T$ integrates the function given by PGMINT over the variable whose label is on *stack register T* (see pp. 246ff).

XEQ → 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.

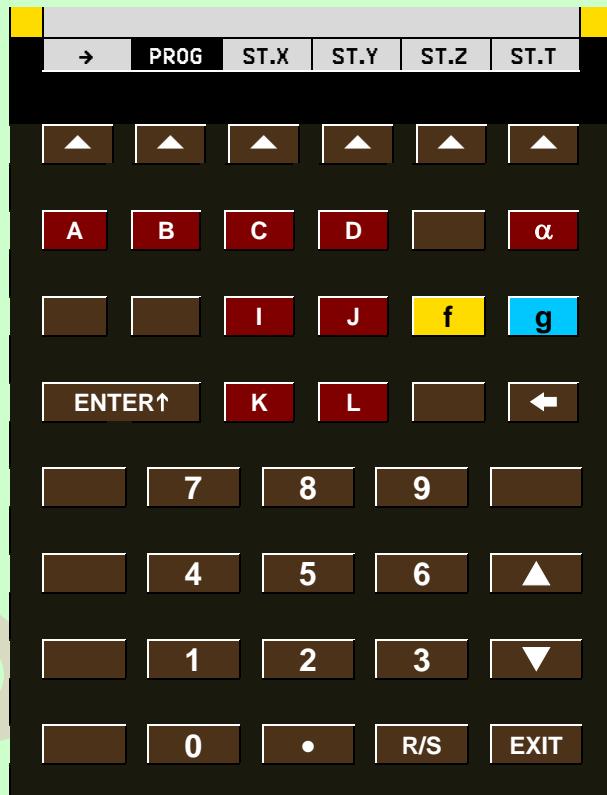
¹⁴⁶ Note you can skip pressing **f** here – see overleaf. See also an alternative there.

¹⁴⁷ Works with all these operations except **LBL**.

¹⁴⁸ Said label must contain at least one letter. The seventh character will terminate entry and close AIM – shorter labels need a closing **ENTER↑**.

¹⁴⁹... 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 2nd softkey compared to p. 56. So, instead of keying in a longer global label in *alpha input mode*, it may be easier to press **PROG** 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, start with ensuring you are in *run mode* – then enter **GTO** *label*. This will position the program pointer onto the corresponding LBL step (as explained on p. 202). Then switch to *PEM* using **P/R** and start browsing from this LBL step.

Let's browse the program steps in our example routine: press **▼** four times:

2015-07-15 14:52 RE 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
R-SWAP	TICKS
LocR	PAUSE
OFF	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** **. .** **α 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. 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. **For 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 encountered¹⁵⁰ where it will halt and display *x*.
2. **For normal execution of a selected routine:**¹⁵¹ Press **XEQ** and specify the label of the program you want to execute (or press **PROG** and pick the label from the *menu*). This will move the program pointer to the corresponding LBL step (cf. p. 202) and start automatically executing the following steps until a STOP, a final RTN, or an END will be found¹⁵⁰ where it will halt and display *x* (cf. pp. 20f).
3. **For stepwise execution of a selected routine:** Press **GTO** instead and specify the label of the program you want to execute (or press **PROG** and pick it from the *menu*). This will move the program pointer to the corresponding LBL step (cf. p. 202) 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.¹⁵²

Following this procedure, you will go through the routine as in normal execution but significantly slower – and you may perform additional checks after each program step. This procedure is especially useful for *debugging* (i.e. looking for errors in a routine).¹⁵³

¹⁵⁰ ... or you interrupt it manually by pressing **R/S** or **EXIT** – then it stops after the current step is completely executed. For resuming its execution, press **R/S** again.

¹⁵¹ 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*.

¹⁵² Pressing **▲** (or **▀▲** if a *multi-view menu* is displayed), on the other hand, moves the program pointer backwards in the current routine without executing anything.

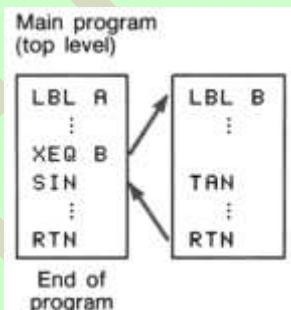
¹⁵³ Watch that your additional checks, if applicable, do not alter the status of your *WP 43S* in a way deviating from its status in automatic execution; else you shall compensate. Also take care when browsing backwards.

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**.

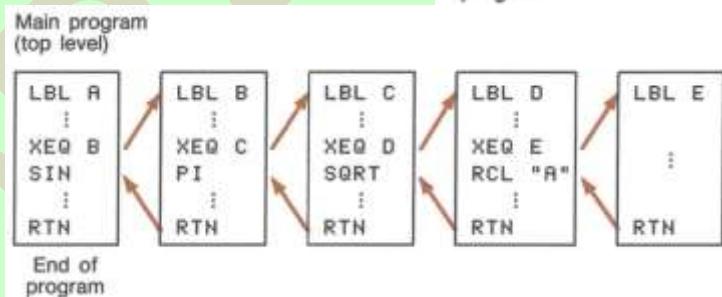
Subroutines: Running a Routine from another Routine

XEQ is programmable as well. Whenever a running routine encounters an XEQ, it will search for the associated label as described on p. 202, 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 WP 43S 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



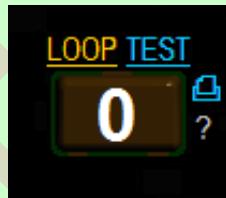
is stopped; pressing **R/S** or **≡V** or **▼**, however, will not cause a loss of return locations.



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 information* at left of the Z 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").¹⁵⁴ 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.

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.
- **CPX?** tests if **X** contains a *complex* object (*data type* 3 or 9) ...

¹⁵⁴ The one and only exception: KEY? skips if true.

- MATR? tests if \mathbf{X} contains a matrix (*data type* 8 or 9) ...
- STRI? tests if \mathbf{X} contains an *alpha string* (*data type* 7) ...
- SPEC? tests if x is special (i.e. $\pm\infty$ or ‘Not a Number’) ...
- NaN? tests if x is ‘Not a Number’ ...

They can **check its numeric content**:

- INT? tests if x is an integer number (i.e. has no fractional part) ...
- 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 and executes the next program step if true, else skips it.

They can **compare its numeric content** with 0, 1, or the content of another source specified (let’s call it s , cf. also pp. 57 and 59):

- $x < ?$ 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 and executes the next program step if true, else skips it.

They can **check its internal structure**:

- BC? (or BS?) tests if \mathbf{X} contains a *short integer*, then checks its bit specified and executes the next program step if said bit is clear (or set), else skips it.
- LEAP? tests if \mathbf{X} contains a *date*, then extracts the year and tests for a leap year ...
- M.SQR? tests if \mathbf{X} contains a matrix, then checks if it is square ...

General ***flag* tests** operate on the *flag* specified:

- FC? tests this *flag* and executes the next program step if said *flag* is clear, else skips it.

- 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 it 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 level routine (cf. the sketch on p. 207).
- 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 here and p. 217 for an application).
- 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).

11	12	13	14	15	16
<i>y^x</i>	<i>y^x</i>	TRI	In	<i>e^x</i>	<i>x²</i>
21	22	23	24	25	26
STO	RCL	R↓	CC	f	g
31	32	33	34	35	36
ENTER↑	<i>x>y</i>	<i>+/-</i>	E		
41	42	43	44	45	
/	7	8	9	XEQ	
51	52	53	54	55	
x	4	5	6		▲
61	62	63	64	65	
-	1	2	3		▼
71	72	73	74	75	
+	0	.	R/S	EXIT	
81	82	83	84	85	

ENTRY? is useful e.g. after PAUSE.

See the *IOI* for more information about all 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. Turn to the ReM for information about them.

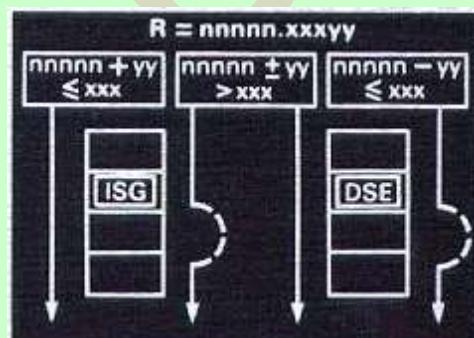
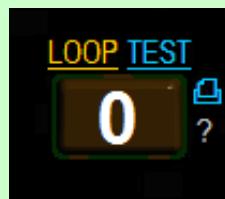
As mentioned further above, routines end with RTN (typically) and programs with END. Executing a program, both RTN and END work in a very similar way and show only subtle differences: a RTN immediately after a binary test returning **false** will be skipped – an END will not.

Loops and Counters

The commands DSE, DSL, DSZ, ISE, ISG, and ISZ 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 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 GTO placed in the skipped step pointing to a label upstream in the same routine you can create loops running until the specified condition is met.



Without such an exit condition you can create an infinite loop – such a routine will run until you interrupt it manually by **EXIT** or **R/S** or until battery voltage falls below the limit. Note such loops are allowed by the operating system of your *WP 43S*.

Example (also for indirect addressing, cf. Section 1):¹⁵⁵

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	LBL 'A'
PROB ▲ RAN#	RAN#
STO → 2 4	STO →24
LOOP ISG 2 4	ISG 24
GTO A	GTO 'A' if $r24 \leq 39$ return to A
RTN	RTN else return to run mode.
EXIT	

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.

Solution:

We will use the so-called ‘bubble sort’ algorithm. Re-initialize the loop counter via **25.039** **STO 2 4** ($r24$ was modified 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	LBL 'B'
P.FN LocR 2 ENTER↑	LocR 002 Allocate two local registers.
LBL 0 1	LBL 01
RCL 2 4	RCL 24 Put the start pointer $r24$
STO .0 0	STO .00 into local register 00.

¹⁵⁵ This example follows an idea of Gene Wright.

```


[LOOP] INC X
STO .0 1
[FLAGS] CF .0 0
LBL 0 3
RCL → .0 0
RCL → .0 1

[TEST] x< ? Y
GTO 0 2
LBL 0 4
[LOOP] ISG .0 0
ISG .0 1
GTO 0 3

[FLAGS] FS? .0 0
GTO 0 1
RTN
LBL 0 2
SF .0 0
STO → .0 0
x>y
STO → .0 1
x>y
GTO 0 4
EXIT


```

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 <i>r.00</i> and <i>r.01</i> are pointing to.
RCL →.01	
x< ? ST.Y	Is <i>x</i> < <i>y</i> ?
GTO 0 2	Then go to label 02
LBL 04	else...
ISG .00	increment <i>r.00</i> and...
ISG .01	if <i>r.00</i> ≤ 39 increment <i>r.01</i> and
GTO 03	if (<i>r.00</i> > 39 or <i>r.01</i> ≤ 39) return to label 03;
FS? .00	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 <i>r.00</i> and the greater where <i>r.01</i> is pointing to.
STO →.01	
x>y	Restore the stack and
GTO 04	return to label 04.

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**.

Note this program allocates two *local registers* for its exclusive use (**R.00** and **R.01**). Furthermore, it uses 1 local flag and 4 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 25.039 XEQ α C ENTER↑ .

Cf. HP-42S OM, 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 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. 226ff 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**.
store what the user entered.

☞ 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:

SF ALPHA
RCL xyz
STOP

sets AIM for upcoming input.
displays the *register* with the message string.
waits for your input. Whatever you key in now is appended to **x** when you continue by pressing **R/S**.

CF ALPHA returns to the numeric mode previously set.
STO xyz stores x to wherever you like.

Again, more elegant is using INPUT for this task:

SF ALPHA sets A/M for upcoming input.
INPUT xyz
CF ALPHA 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 *menu* for the variables defined immediately after 'Var.In' and shows it.
STOP stops for user interaction.
EXITall exits the *menu* when program continues.
RCL 'xy2' recalls what you need first (it may have been entered in any order).

The label called 'Var.In' here should be located close to the program top. It shall be followed by up to 18 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 in the value or calculate it, then press the corresponding *softkey*. The content of **X** will be stored.
- **recall** the present value of one of the variables displayed, enter **(RCL)**,¹⁵⁶ then press the corresponding *softkey*.

¹⁵⁶ The standard *menu* as shown on p. 56 will not appear after **(RCL)** in such a case.

- **view** the present value of one of the variables displayed, enter **VIEW**, then press the corresponding softkey.¹⁵⁷
 - **exit** this *menu*, press **EXIT**.
 - **continue** program execution, press **R/S**.
- Directly react on particular keys pressed: The key codes returned by KEY? (cf. p. 210) allow for ‘real time’ response to user input from the keyboard. KEY? takes a *register* argument (**X** is allowed but does not lift the stack) and stores the key most recently pressed during program execution or pause in the *register* specified.¹⁵⁸ Although the keyboard is active during program execution it is desirable to display a message and suspend the routine by PAUSE while waiting for user input. Since PAUSE will be terminated early by a key press, simply use PAUSE 99 in a loop to wait for input. Since KEY? acts as a test as well, a typical user input loop may well look like this:

```
LBL 'US.in'
RCL xyz
PAUSE 99
KEY? 00
GTO 'US.in'
LBL? →00
XEQ →00
GTO 'US.in'
```

displays the *register* with the message string.
waits 9.9 s for user input unless a key is pressed.
tests for user input and puts the key code in **R00**.
If there was no input then return to the beginning;
else: if a label corresponding to the key code exists...
... then call it, ...
... else return to the beginning.

Instead of the dumb waiting loop, the routine can do some computations and update the display before the next call to PAUSE and KEY?

To be even more versatile, you can use KTyp? to return the type of the key pressed if its row / column code is given (see the *IOP*).

¹⁵⁷ The standard *menu* as shown on p. 55 will not appear after **VIEW** in such a case. Note that the HP-42S allowed for just viewing the present value of one of the variables displayed by pressing **f** and the corresponding softkey; we cannot support this on your WP 43S since it features three *menu* rows for you.

¹⁵⁸ Note **R/S** and **EXIT** cannot be queried since they stop program execution immediately.

If you decide not to handle the key in your program you may feed it back to the main processing loop of the *WP 43S* with the command **PUTK nn**. It will cause the program to halt, and the key will be handled as if pressed after the stop. This is especially useful if you want to allow numeric input while waiting for some special keys like the arrows. After execution of the **PUTK** command you are responsible for letting the routine continue its work by pressing **R/S**.

See the *I/OI* 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 2nd order differential equations, a type frequently occurring in physics.¹⁵⁹

In a **first example**, we will solve the equation of motion for the fall of a parachutist $\frac{d^2f}{dt^2} = g - b \left(\frac{df}{dt}\right)^2$ with earth acceleration g and b taking care of drag.

Proceeding in small constant time steps Δt , the following set of equations controls the vertical motion of the parachutist (or skydiver):

$$\begin{aligned} \left(\frac{df}{dt}\right)_{1/2} &= \left(\frac{df}{dt}\right)_0 + \left[g - b \left(\frac{df}{dt}\right)_0^2 \right] \times \frac{\Delta t}{2} \\ f_1 &= f_0 + \left(\frac{df}{dt}\right)_{1/2} \quad \text{and} \quad \left(\frac{df}{dt}\right)_{3/2} = \left(\frac{df}{dt}\right)_{1/2} + \left[g - b \left(\frac{df}{dt}\right)_{1/2}^2 \right] \times \Delta t,^{160} \\ f_2 &= f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t \quad \text{etc.} \end{aligned}$$

¹⁵⁹ Turn to the *ReM, App. H*, for background information about the method applied here.

¹⁶⁰ Note the contents of the rectangular brackets must be ≥ 0 always. Thus, this routine will work for velocities $< \sqrt{g/b}$ only, not for abruptly decelerating fast initial movements (e.g. by opening a parachute).

Assume start height at time zero ($t = 0$) is 1000 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 , 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'           initialize all variables used
    .003
    STO 'b'             assumed realistic drag value for this case
    1000
    STO 'f'             start height
    0
    STO 't'             start time and velocity
    STO 'df/dt'         end of initialization

LBL 01
    # g
    RCL 'b'
    RCL 'df/dt'
    x2
    ×
    -
    RCL 't'             begin of time loop
    x>0 ?
    GTO 02              take g out of CNST

    b × (df/dt)2
    g - b × (df/dt)2

    RCL 't'             check time – it will be zero in 1st run
    x>0 ?               from 2nd run on go to local label 02
    GTO 02

    DROP
    2
    /
    GTO 03              1st run only: forget t
                        1st run only:
                        1st run only: [g - b × (df/dt)2] / 2
                        1st run only: go to common part

LBL 02
    DROP
    from 2nd run on:
    from 2nd run on: forget t

LBL 03
    RCL× 'Δt'
    STO+ 'df/dt'
    RCL 'Δt'
    STO+ 't'
    RCL× 'df/dt'
    STO- 'f'             common part of time loop resumes here again
                        [g - b × (df/dt)2] × Δt (or half of it in 1st run)
                        calculate the new df/dt

                        calculate the new time
                        df/dt × Δt
                        calculate the new f

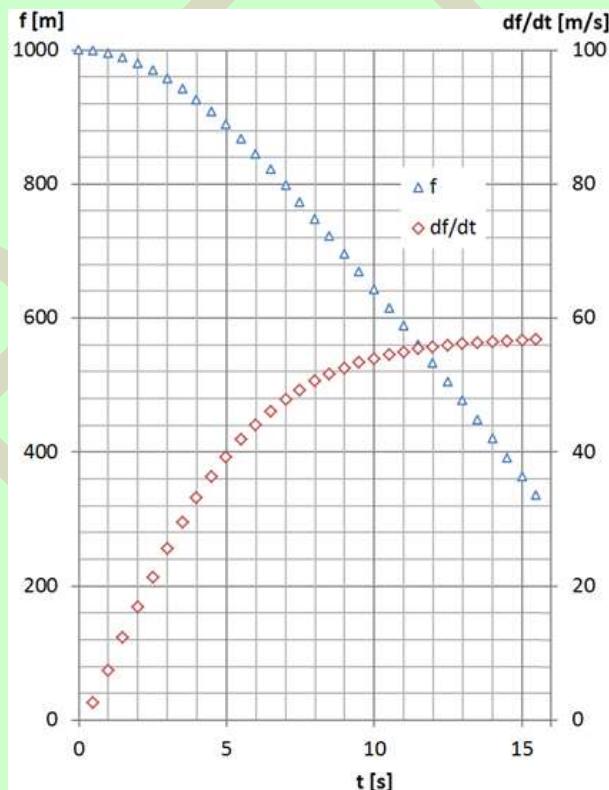
```

VIEW 't'	display new time	}	for plotting next data points $(t; f)$ and $(t; v)$.
STOP			
VIEW 'f'	display new height		
STOP			
VIEW 'df/dt'	display new velocity		
STOP			
GTO 01			
END			

end of time loop, return for next run through it

Now, leave *PEM* and start program execution via **[XEQ] PROG PFall** – plotting the points calculated will result in a diagram like this here. Height decreases following a parabola over time in the beginning but becomes linear later. Note the vertical velocity does not increase much anymore after some 12 s here approaching some 57 m/s while skydiving with parachute closed.

For comparison: the velocity limit with an open parachute ($b = 0.3$) would be < 6 m/s, 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:

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

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

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

with $-\frac{GM}{(x^2+y^2)^{3/2}} x = K_x$ 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. 216):

```

LBL 'Satell'
  INPUT 'x'          start of variable initialization
  INPUT 'y'
  INPUT 'dx/dt'
  INPUT 'dy/dt'
  .1
  STO 'Δt'          initialize the remaining ‘fixed’ start values
  1
  STO 'a'            (for earth satellites, take GM out of CNST instead)
  0
  STO 't'            start at time zero
                      end of initialization

LBL 01
  RCL 'y'
  RCL 'y'
  x2              y2
  RCL 'x'
  x2
  +
  -1.5              y2+x2

begin of time loop

```

y^x
 $RCLx 'a'$
 x
 $RCL L$
 $RCLx 'x'$
 $RCL 't'$
 $x>0 ?$
GTO 02

$(y^2+x^2)^{-1.5}$
 $a (y^2+x^2)^{-1.5}$
 $y a (y^2+x^2)^{-1.5} = -K_y$
 $a (y^2+x^2)^{-1.5}$
 $a (y^2+x^2)^{-1.5} = -K_x$. Stack is $[-K_x, -K_y, \dots]$ now.

check time – it will be zero in 1st run
from 2nd run on go to local label **02**

DROP
 2
 $/$
 $x\bar{y}$
 2
 $/$
 $x\bar{y}$
GTO 03

1st run only: forget t
 1st run only:
 1st run only: $-K_x / 2$
 1st run only: stack is $[-K_x, -K_x / 2, \dots]$ after $x\bar{y}$
 1st run only:
 1st run only: $-K_y / 2$
 1st run only: stack is $[-K_x / 2, -K_y / 2, \dots]$ after $x\bar{y}$
 1st run only: go to common part of time loop

LBL 02
DROP

from 2nd run on:
from 2nd run on: forget t

LBL 03
 $RCLx '\Delta t'$
 $STO- 'dx/dt'$
DROP
 $RCLx '\Delta t'$
 $STO- 'dy/dt'$
 $RCL '\Delta t'$
 $STO+ 't'$
 $RCLx 'dx/dt'$
 $STO+ 'x'$
VIEW 'x'
STOP

the common part of the time loop resumes 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

$RCL '\Delta t'$
 $RCLx 'dy/dt'$
 $STO+ 'y'$
VIEW 'y'
STOP

$dy/dt \times \Delta t$
 calculate the new y
 display also the new y for plotting the new point (x,y)

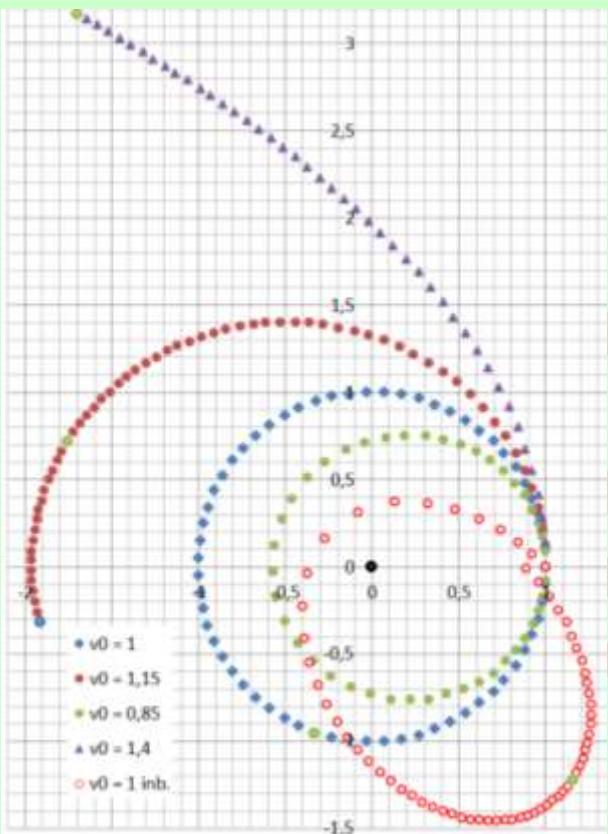
GTO 01

end of time loop, return for next run through it

END

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



decelerating force which may even depend on height over ground. Your imagination is the limit – and your ability in mathematical modeling.

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° inwards ('NW'). Note this curve does not close due to the perigee speed being too high for the time step Δt chosen. We recommend watching the limits of such computational models always.

For a descent to a planet or a moon, you can introduce a deceler-

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 re-define \blacktriangleleft , \triangleright , and **EXIT**.¹⁶¹

¹⁶¹ See HP-42S Programming Examples and Techniques, pp. 29 - 39, 92 - 99, 158 - 160, and 184 - 192, for sample programs using the *programmable menu*.

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 (If space does not suffice, only the first characters will be displayed). **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 (the *menu view* changes):
 - a. Press one of the 18 *softkeys* available, **▲**, **▼**, or **EXIT**.
 - b. Alternatively, enter the respective key number, 1 through 21 (unshifted leftmost *softkey* carries #1, **g**-shifted leftmost #13).
4. Specify a program label using one of the following three methods (the *menu view* changes to the one shown on p. 204):
 - a. Select an existing global label by pressing the corresponding *softkey* in **PROG**.
 - b. Key in a global label character by character 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**.

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 text will be automatically stored in **K**) or
- an entire **number** (a.k.a. a *numeric constant*, like `-1.902×10-16` or `123 4516` or `#λc`; 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 two bytes of program space and makes the routine a tiny bit faster. You will achieve the same by `12.3` **EXIT** `45.67` **+**. It may not be really important here but you should know.

Constant vectors and matrices cannot be entered directly in a program; 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. 204f.
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 wiped out. 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 *user 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 *user 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.¹⁶² Conversely, *registers* and standard user program memory residing in *RAM* are designed for data changing frequently but will not hold data with

¹⁶² *FM* may not survive more than some 100 000 flashes. Thus, we made commands writing to *FM* (**SAVE** or **PSTO**) non-programmable.

the batteries removed for longer than a few minutes. 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

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 1st and 2nd derivatives. All are contained in ADV or EQN. Pressing **ADV** in *run mode* results in



f''(x)					
SOLVE	SLVQ	f'(x)	Π_n	Σ_n	$\int f dx$

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 and examples.

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 `ccccccc.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 `ccccccc` 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 `ccccccc` by `ii`; if `ccccccc ≥ ffff` then Σ goes back to step 2, else it returns the final sum in **X**.

If `ii = 0`, `ccccccc` 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

ADY Σ_n **g** S S Q R T **ENTER↑**

(or pick ΣSQRT from PROG, cf. p. 204)

and get 671.462 9 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 ≥ fff 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_n] [α] [P] [R] [O] [D] [ENTER↑]

(or pick PROD from PROG, cf. p. 204)

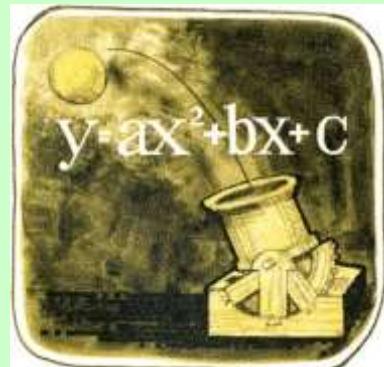
and get **5.734×10^{-33}** returned if SCI 3 is set.

Solving Quadratic Equations

The command SLVQ finds the *real* and *complex* roots of a quadratic equation $ax^2 + bx + c = 0$ with its *real* parameters on the input stack **[c, b, a, ...]**:

- 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 1st *complex* root in **X** and the 2nd in **Y** (the *complex conjugate* of the 1st). 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) $x = 1.175\ 4$, $y = -0.425\ 4$, $z = 41.000\ 0$. Since z is positive, x and y are the two *real* roots of this equation here.

Check:

Store x in J and y in K. Then enter

[RCL] [J] [FILL] 4 [x] 3 [-] [x] 2 [-] returning 2.000 0 $\times 10^{-33}$ and

[RCL] [K] [FILL] 4 [x] 3 [-] [x] 2 [-] returning 0.000 0.

Remember your WP 43S calculates with 34 digits precision, so any result within $\pm 3 \cdot 10^{-33}$ 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 equations in memory yet, your WP 43S will return:



Press **NEW** to enter a new equation. You will get immediately:

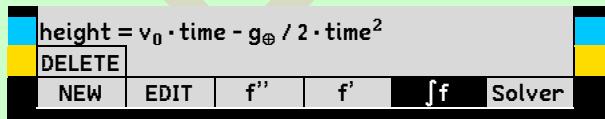


with *alpha input mode* turned on (cf. pp. 183ff). Enter your equation now, e.g.

▼ height = v [R] [O] [X] time - g_⊕ [I] [2] [X] time ^ [2] ¹⁶³

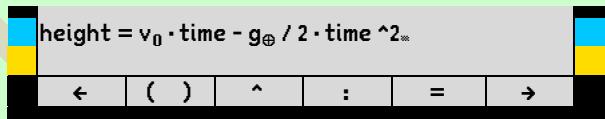
for the height of e.g. a ball thrown vertically upwards with velocity v_0 .

Press **ENTER↑** for closing the *Equation Editor* and see: ¹⁶⁴



You will get such a display whenever one or more equations are stored. The equation shown is called the *current equation*. (A dashed line will show up when there are more equations; to select another one 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 behind the character(s) to be corrected and pressing **◀** followed by the new character(s) to be inserted here.

For labeling this equation, move the cursor left to its very begin using **◀**, and key in:

F ▼ R E E ▲ F ▼ A L :

¹⁶³ The index of the earth acceleration constant is found in the punctuation menu **ao**, reached via **g [.]** in AIM.

¹⁶⁴ Note **MULT-** is set here for sake of better readability of equations. And some spaces are inserted automatically for the same reason.

	FreeFall: height = $v_0 \cdot time - g_{\oplus} / 2 \cdot time^2$						
ENTER↑	←	()	^	:	=	→	

FreeFall: height = $v_0 \cdot time - g_{\oplus} / 2 \cdot time^2$							
DELETE							
NEW	EDIT	f''	f'	f	Jf	Solver	

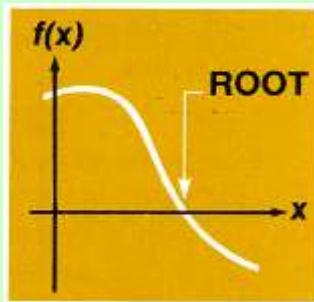
ENTER↑ closes the *Equation Editor* storing the modifications you made. Note editing an equation clears all its variables.

If an equation become wider than the display 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.¹⁶⁵

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, misspelled functions, illegal variable names, etc.). It will then return a *menu* of all applicable variables, like this in our **example**:



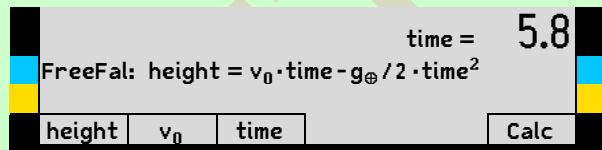
FreeFall: height = $v_0 \cdot time - g_{\oplus} / 2 \cdot time^2$						
height	v_0	time				Calc

¹⁶⁵ 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.

Note your WP 43S knows g_{\oplus} 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:

DSP FIX 0 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**:



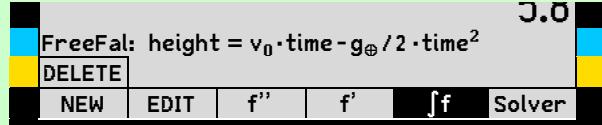
corresponding to 5.8 s until your ball passes said point. Note entering the known values and guesses disabled *automatic stack lifting*.

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, ¹⁴C, is 1.20×10^{-4} . How old is the tree? What is the half-life of ¹⁴C?

Solution (assuming you continue directly after previous example):
Exit the current equation and enter a new one for radioactive decay:

EXIT



NEW



D ▼ E C A Y : R A T E X T I M E = L N () ← N 0 / N
ENTER↑

Decay: rate · time = ln(n0 / n)
DELETE
NEW EDIT f'' f' ∫f Solver

Solver

Decay: rate · time = ln(n0 / n)
rate time n0 n Calc

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 ^{14}C , that is the time required for half the material present to decay:

2 [n0] 1 [n] [time]

[time] = 5 776.2

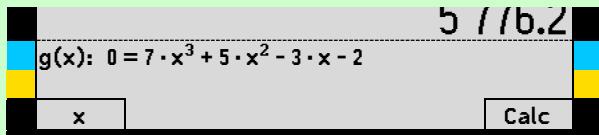
Good guess! Meanwhile, the half-life of ^{14}C is known to be 5 730 years.

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:



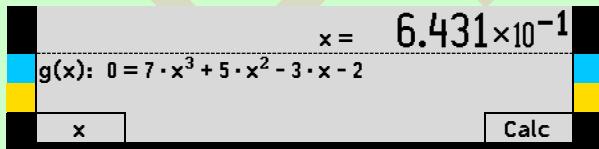
3. Optionally enter one or two initial guesses for the unknown like

0 [x] 1 [x].

4. Set the display format and precision

DSP 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 $g(x)$, while slightly smaller values, e.g.

.64 [x] [Calc], return negative values for $g(x)$.

7. There may be one or two more roots:

- a. 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 × 10⁻¹

Slightly greater x-values, e.g. **-0.8**, return positive values for $g(x)$, slightly smaller values, e.g. **-0.81**, return negative values for $g(x)$. Thus, there must be one more root between the two roots found.

- b. Enter two new initial guesses for the unknown like **-7** **x** **.5** **x**.
Press the softkey for **x** once more and you will get:

x = -5.510×10⁻¹

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 for this program: From its 2nd step on, menu variables shall be declared using MVAR instructions (cf. p. 216) 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 in a way that *f=0* is fulfilled.

Example:

Let's return to the equation we dealt with in the last two chapters:

$$\text{height} = v_0 \cdot \text{time} - g_0 / 2 \cdot \text{time}^2$$

This is easily rewritten:

$$v_0 \cdot \text{time} - g_0 / 2 \cdot \text{time}^2 - \text{height} = 0$$

So the required program might look like this:

```
LBL 'FreeF'  
MVAR 'height'  
MVAR 'v0'  
MVAR 'time'  
# g0  
-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:

		$f''(x)$				
SOLVE	SLVQ	$f'(x)$	Π_n	Σ_n	$\int f dx$	

Choose **SOLVE**. You will get (as expected from p. 204):

	→	PROG	ST.X	ST.Y	ST.Z	ST.T
--	---	------	------	------	------	------

Press **PROG** and pick the proper program for *f* (here **FreeF**). You will get the corresponding menu of variables, i.e. here:

	height	v ₀	time
--	--------	----------------	------

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 v_0

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. 235).

5. Leave the Solver.

Pressing **EXIT** will return to the top view of ADV.

SOLVE	SLVQ	$f''(x)$	Π_n	Σ_n	$\int f dx$
-------	------	----------	---------	------------	-------------

Using the Solver in a Program

For using the *Solver* in a 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 above:

PGMSLV	$f''(x)$	PGMINT
SOLVE	SLVQ	$f'(x)$

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 **P**), fill the variables of the equation to be solved, e.g. with the start values known from above:

-50 [STO] height
20 [STO] v0

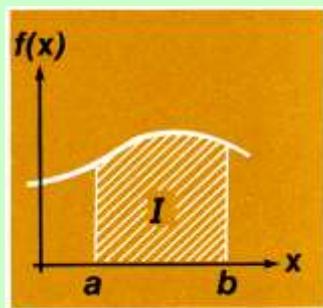
Option: Fill the unknown with a 1st guess, e.g. with **5** as we specified above (a 2nd guess will be taken from **X**):

5 [STO] time

Call **P** via [XEQ] PROG **P** and you will see time = **5.8** (as expected from p. 235).

Eventually turn to *Part 3, Section 12* of the *HP-42S OM*. Refer to the *HP-34C OHPG (Section 8 and App. A)* or the *HP-15C OH (Section 13 and App. 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 1st kind and order 0. This function can be written as

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin t) dt$$

Solution:

This is calculated in *radians*, thus enter **MODE RAD** and press **EQN**:

FreeFa: height = v₀ · time - g₀ / 2 · time²

DELETE

NEW EDIT f'' f' ∫f Solver

This function is not in the equation list yet.¹⁶⁶ So, press **NEW** and start entering the integrand:

I B E S S :

IBess:

← () ^ : = →

Continue with **C O S** **()** **←** **X** **X** **S I N** **()** **←** **T**

IBess: cos(x · sin(t))

← () ^ : = →

Close and store this function by pressing **ENTER↑**. The *menu* will return to the previous one.

¹⁶⁶ Actually, a function $J_y(x)$ is supplied with your WP 43S directly returning values of the Bessel function of 1st kind and order y . You can compare the results if you like.

Then press **Jf**. Your *WP 43S* will check the current equation (cf. pp. 232ff) for syntax errors (missing operators, misspelled functions, illegal variable names, etc.).¹⁶⁷ It will then return a *menu* of all applicable variables:

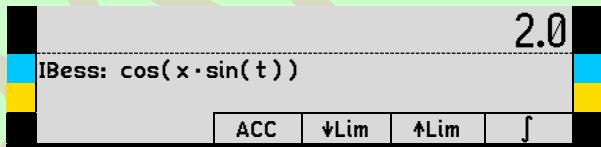


You can enter values for any variables (i.e. integration constants) you already know by pressing the respective *softkeys* now, e.g.

2 x

(For recalling such an integration constant, just press **RCL VAR** 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:



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 **vT** being the true value and **vc** the computed value of the integrand) at any point between **↓Lim** and **↑Lim**.

¹⁶⁷ 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.

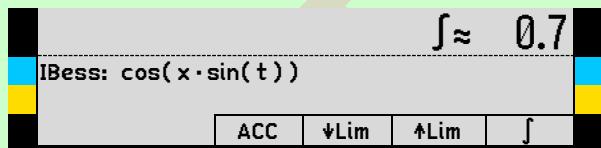
We want to see the result accurate to three decimals. Thus we enter

.001 ACC for the accuracy of computation,

0 ↓Lim for the lower integration limit,

π ↑Lim for the upper integration limit,

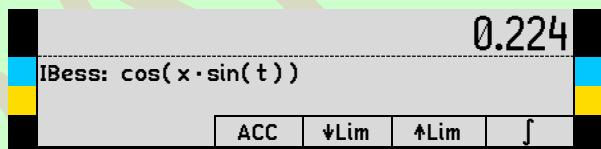
and start integrating by pressing \int . Your WP 43S will return:



Do not forget to divide this result by π to get the correct value for $J_0(2)$:

π / 168

DSP FIX 3



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 $\int f dx$.

¹⁶⁸ Note that \int vanishes with π like every *temporary information* disappears with the next keystroke.

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.

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.¹⁶⁹

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 2nd step of this program on, menu variables shall be declared with MVAR instructions (cf. p. 216) 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:

		$f''(x)$	
SOLVE	SLVQ	$f'(x)$	Π_n
		Σ_n	$\int f dx$

Choose **$\int f dx$** . You will get (as expected from p. 204):

¹⁶⁹ Note this follows closely the procedure as described for the Solver above.

	→	PROG	ST.X	ST.Y	ST.Z	ST.T	
--	---	------	------	------	------	------	--

Press **PROG** and pick the proper program for f (here **IBessI**). You will get the corresponding menu of variables, i.e. here:

	x	t	
--	---	---	--

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:

	ACC	↓Lim	↑Lim	∫	
--	-----	------	------	---	--

We enter (like in previous chapter) **.001 ACC 0 ↓Lim** **π ↑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 $\int \approx 0.704$ as you might 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:

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

You shall define the necessary variables in advance and load them with the known values using *STO*. Then call the *menu* **∫fdx**. *PGMINT* 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:

PGMINT	STO AC	STO ↓L	STO ↑L	∫
--------	--------	--------	--------	---

Eventually, the variable of integration must be specified calling \int .

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 '↓Lim'  
# π  
STO '↑Lim'  
0.001  
STO 'ACC'  
∫fd 't' integrate over time.  
VIEW ST.X 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] and you will see $\int \approx 0.704$ as you may have expected (cf. previous chapter). Divide by π to get $J_0(2)$ as above.

Eventually turn to Part 3, Section 13 of the *HP-42S OM*. Refer to the *HP-34C OHPG (Section 9 and App. B)* or the *HP-15C OH (Section 14 and App. 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 1st derivative. For computing it, ...

1. $f'(x)$ will first look for a user routine labeled ' δx ' (or ' δX ', ' Δx ', or ' ΔX ', in this order), 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 1st derivative will be in *stack register X*, while Y , Z , and T will be clear and the position x will be in L .

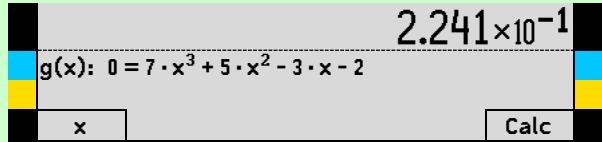
Example (with SCI 3 set):

Take the equation $g(x) = 7x^3 + 5x^2 - 3x - 2$ again (used on pp. 235f for solving). Instead of checking two function values left and right of the root you could check the slope $g'(x)$ at the root just once.

Solution:

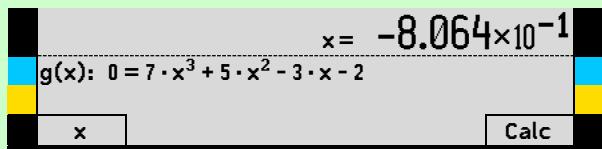
You have got $g(x)$ in EQN already. For each of the three roots found, calculate the root first, then the 1st derivative of $g(x)$ at that point:

1. Press EQN, make $g(x)$ the *current equation*, and press Solver. You will see then:

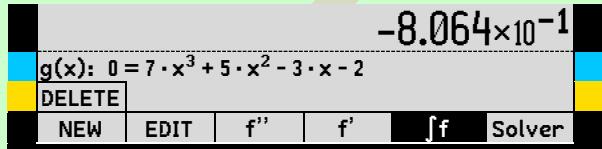


2. Find the 1st (leftmost) root as shown above:

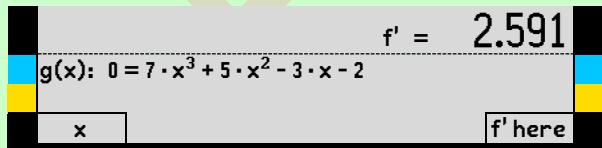
-2 [x] -1 [x] [x]



3. Pressing **EXIT** returns to the top view of EQN:



4. Press **f'** :



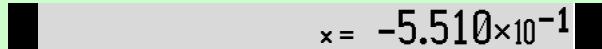
Note that **f'** returned the value of the 1st derivative at this very location immediately since **g(x)** features only one variable; else **f'** would have needed your input via the softkeys displayed and pressing **f' here** thereafter.

So the slope of **g(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 2nd root of **g(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 $g(x)$: 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 $g(x)$ 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 2nd derivative of the function specified.

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 2nd step of this program on, menu variables shall be declared with MVAR instructions (cf. p. 216) 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. 248).
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_n	Σ_n	$\int f dx$

5. Press $f'(x)$ or $f''(x)$. You will get as well:

	\rightarrow	PROG	ST.X	ST.Y	ST.Z	ST.T
--	---------------	------	------	------	------	------

6. Press **PROG** to pick the label of the program containing the function $f(x)$ (or enter its label directly as described on p. 203).
7. Let your *WP 43S* compute the 1st or 2nd derivative at the location specified in x .¹⁷⁰

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:

			$f''(x)$		
PGMSLV	SLVQ	$f'(x)$	Π_n	Σ_n	$\int f dx$

Press $f'(x)$ (or $f''(x)$) and specify 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.

¹⁷⁰ 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:

Light is observed to be diffracted when passing through small circular holes, an effect most obvious when using laser light. Its intensity is

$I(r) = I_0 \times \left(\frac{J_1(2\pi r)}{\pi r}\right)^2$ behind the hole; $J_1(x) = \frac{1}{\pi} \int_0^\pi \cos[t - x \sin(t)] dt$ is the *Bessel function of the 1st kind of order 1* (cf. p. 242). 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

$\sin(t)$

$t - x \sin(t)$

$\cos[t - x \sin(t)]$

The entire stack is loaded with the integration variable t , so $x = 2\pi r$ (see below) must be recalled from a global register for calculating $x \sin(t)$

2. Write a 2nd 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**, when called in PEM, displays:

PGMSLV	$f''(x)$	Π_n	Σ_n	$\int f dx$
SOLVE	SLVQ	$f'(x)$		

LBL 'I'

π

x

2

x

STO 00

stores $2\pi r$ for later use, also in integration.

```

PGMINT 'J1'      specifies the program of the integrand.171
0
STO '↓Lim'
# π
STO '↑Lim'
0.001
STO 'ACC'
∫fd ST.X
RCL 00
/
2
x
RTN

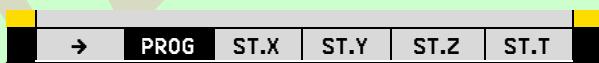
```

computes $\pi \times J_1(2\pi r)$.
 the stack just contained the integration results before.
 $J_1(2\pi r) / (2\pi r)$
 $J_1(2\pi r) / (\pi r)$

3. Enter **DSP SCI 3**

MODE RAD

0.1 **ENTER↑ 1 ADV SOLVE**



4. Pick **I** from **PROG** or key in **α I ENTER↑**. You will get **6.098×10^{-1}** after some time.¹⁷²

5. Enter **1 ENTER↑ 1.5 SOLVE α I ENTER↑** and you will get **1.117**.

6. Enter **1.5 ENTER↑ 2 SOLVE α I 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.

¹⁷¹ You shall press the rightmost softkey to get the *menu* for accessing PGMINT etc.

¹⁷² 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 $2\pi r$.

SECTION 5: TWO BROWSERS, TWO APPLICATIONS, AND TWO SPECIAL MENUS

There are two *browsers* featured for quick and easy checking memory, *registers*, and *flags* (RBR and STATUS, see below). And there are two very useful applications: a TIMER (or stopwatch, see pp. 257f) and “*time value of money*” (TVM, see pp. 259ff). Furthermore, two special *menus* will ease your path in special areas of application and particular regions of this planet (see pp. 263ff).

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
 RBR	Browses all currently allocated <i>registers</i> showing their contents. RBR operates in <i>TAM</i> (cf. pp. 55ff). The first screen you see covers <i>registers X</i> 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 may be abbreviated and matrices will be. Fractions are displayed with their decimal value. Within the range of lettered <i>registers</i> , every fourth <i>register</i> is displayed overlined to guide the eye:

2015-08-05 23:15	R _E 4°	/max. 2:64	↑
I:	0.000 0		
L:	1.602 2×10 ⁻¹⁹		
D:	'This calculator is made in...		
C:	8A FE 49 7C ₁₆		
B:	123 456 789 012 345 678		
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	R _E 4°	/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:	[3×3 C matrix]		
R01:	[3×3 Matrix]		
R00:	[4×1 C matrix]		
K:	57.000 0		
J:	6.000 0		



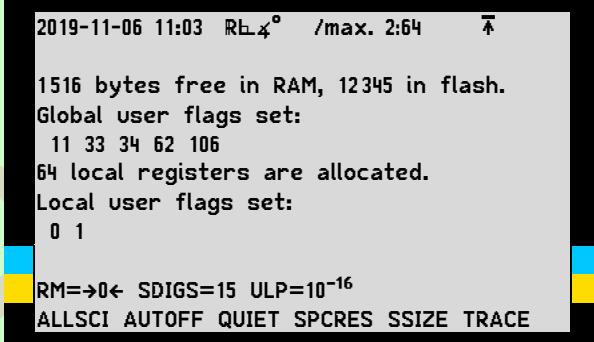
browses the *registers* going down from **R99** (if starting with the screen on previous page) to **R00**; then continues with **K**, **J**, down to **X**. After **X**, **R99** will be shown again.



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 of RBR as shown on p. 254. Else (i.e. if no *local registers* are allocated) directly returns to this screen.



browses immediately to the corresponding (global or local) *register*. If no such *local register* is allocated, it returns to the first *global register*.

	R/S	toggles display to show the <i>register</i> contents or the <i>space</i> allocated for them.
	RCL	in <i>run mode</i> , recalls the <i>register</i> displayed in the lowest row and leaves RBR; in <i>PEM</i> , enters a corresponding step RCL ... and leaves RBR.
	EXIT	leaves RBR.
	STATUS or g FLAGS STATUS	Displays the amount of free memory available and the user accessible <i>flags</i> set (inspired by STATUS on HP-16C and WP 34S). Local <i>user flags</i> will only be displayed if <i>local registers</i> are allocated at all. Some global settings and system <i>flags</i> set are shown in the bottom rows (covering only what is not shown in the <i>status bar</i>):  <pre> 2019-11-06 11:03 RLx⁰ /max. 2.64 ⌂ 1516 bytes free in RAM, 12345 in flash. Global user flags set: 11 33 34 62 106 64 local registers are allocated. Local user flags set: 0 1 RM=>0< SDIGS=15 ULP=10⁻¹⁶ ALLSCI AUTOFF QUIET SPCRES SSIZE TRACE </pre>
	▲ and ▼ EXIT	toggle between <i>views</i> if more <i>flags</i> are set. leaves STATUS.

☞ 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-55.¹⁷³ Start it by pressing **TIMER**. Then the top numeric row will be replaced by

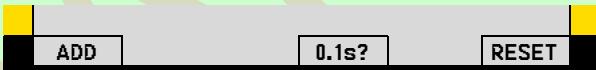


0:00:00.0 [00]

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). It deletes the total time if applicable. – Note this is not the global RESET command.
0.1s?	toggles displaying tenths of <i>seconds</i> (default is ‘display’).
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.

¹⁷³ 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.

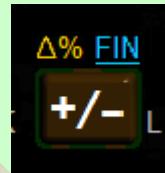
Key	Remarks
[n][n]	sets the <i>current register address</i> (<i>CRA</i> , <i>startup default</i> is 0). The <i>CRA</i> will be displayed between rectangular brackets as shown here: ¹⁷⁴ 27:31:55.6 [01]
[ENTER↑]	stores the present timer value in the <i>current register</i> at execution time without changing the timer status or value. Then increments the <i>CRA</i> 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 <i>CRA</i> by one, respectively.
[.]	combines [ENTER↑] + [◀] in one keystroke, but the <u>total time</u> 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 [⌚] in the <i>status bar</i>) until <ol style="list-style-type: none"> stopped explicitly by [R/S] within TIMER or ... 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.

¹⁷⁴ 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.

- ☞ 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 well proven financial application (thus found 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 (**nPER**). This kind of financial problems will often occur also to technical people, so we included TVM on your **WP 43S**.

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 $i = \frac{i\%/\text{period}}{100} = \frac{i\%/\text{year}}{100} / \frac{\text{periods}}{\text{year}}$

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 $p = 1$.

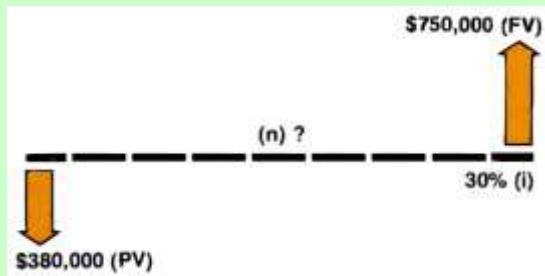
The application TVM uses two *menu* entries for p – choose **End** for payments at the end of each period, **Begin** for payments at the beginning of each period.

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 1st period. It can also be an initial cash flow or a discounted value of a series of future cash flows.

The future value **FV** is always meant to occur at the end of the 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:

[DSP] FIX [2] This will suffice. Then all you have to do is keying in the known parameters and boundary conditions:

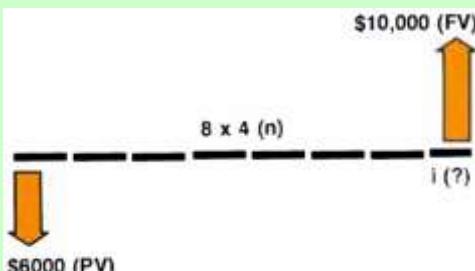
[FIN] TVM Begin

Begin					End	
n_{PER}	i%/a	per/a	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 reach this future value?
n_{PER}	2.59		years.

Example for finding the necessary interest rate for compounded amounts:

What annual interest rate must be obtained to amass \$10 000 in 8 years on an investment of \$6 000, with



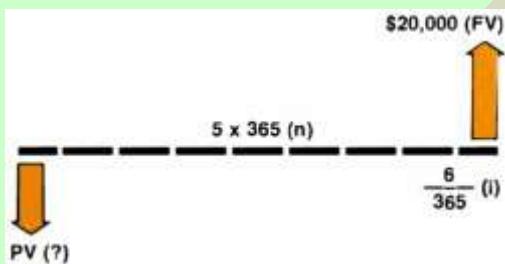
quarterly compounding? (Continue keeping the settings of previous example.)

Solution:

6000 PV
4 per/a
10000 FV
8 ENTER↑ 4 × nPER
i%/a

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:



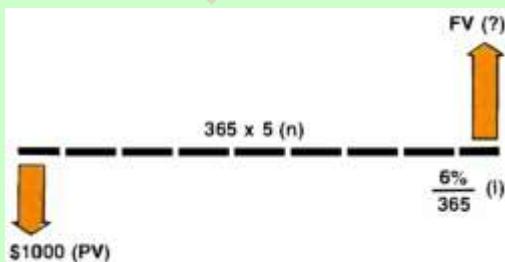
In 5 years when your son starts college, you will need \$20 000. You deposit a lump sum in a certificate account that earns 6% compounded daily. How much do you need to deposit today to reach that goal? (Dream on with the settings of previous example.)

Solution:

20000 FV
6 i%/a
365 per/a
5 ENTER↑ 365 × nPER
PV

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	1 825.00	periods. Now, Sam has...
FV		1 349.83	present balance meaning...
RCL	VAR 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	VAR PV	12.55	% effective interest rate.

Turn to App. 3 for more applications of TVM (annuities, savings, etc.), starting on p. 303.¹⁷⁵

¹⁷⁵ TVM was launched with HP's very first financial 'problem solver', the HP-80 of 1972, and implemented on each and every 'business calculator' of HP thereafter. An early HP advertising sheet promised 'you can improve and simplify your time-and-money management' applying TVM quickly. 'Random-entry financial keys let you key in problems in any order. And you can change any number at any time.' All this was true for certain – remember it was a time before spreadsheet software became available – and may even hold nowadays to some extent since hardly any modern financial tool solving such problems is more compact than a pocket calculator featuring TVM.

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_{Moon}	a_{\oplus}	c		

Besides by browsing with \blacktriangle and \blacktriangledown , you can access the contents of CNST most easily using the alphabetical access method demonstrated in the *ReM, Section 2*.

Names of astronomical and mathematical constants are printed on colored background in the table starting below. The unit of each physical and astronomical constant is listed here as well. Find the numeric values of the constants and their uncertainties in the *ReM*.

Name	Unit ¹⁷⁶	Remarks
1/2	1	Trivial but helpful constant for some iterations.
a	d	Gregorian year
a_0	m	Bohr radius
a_{Moon}	m	Semi-major axis of the Moon's orbit around the earth.
a_{\oplus}	m	Semi-major axis of the Earth's orbit around the sun. Within its uncertainty, a_{\oplus} equals 1 AU (<i>astronomic unit</i>).
c	m/s	Speed of light in vacuum
c_1	m^2W	First radiation constant
c_2	m K	Second radiation constant

¹⁷⁶ Find all unit symbols used here explained in the chapter about unit conversions in the *ReM, Section 2*.

Name	Unit ¹⁷⁶	Remarks
e	C	Elementary charge
e_E	1	Euler's e
F	C/mol	Faraday constant
F_α	1	<i>Feigenbaum's</i> α and δ
F_δ		
G	$m^3/kg\ s^2$	Newtonian constant of gravitation; also known as γ from other authors. See also GM_{\oplus} below.
G_0	$1/\Omega$	Conductance quantum
G_C	1	Catalan's constant
g_e	1	Landé's electron g-factor
GM_{\oplus}	m^3/s^2	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)
g_{\oplus}	m/s^2	Standard earth acceleration
h	J s	Planck constant
\hbar		So-called 'Dirac constant'
k	J/K	Boltzmann constant
K_J	Hz/V	Josephson constant
l_p	m	Planck length
m_e	kg	Electron mass
M_{Moon}	kg	Mass of the Earth's Moon
m_n	kg	Neutron mass
m_p	kg	Proton mass
M_p	kg	Planck mass
m_p/m_e	1	Proton to electron mass ratio

Name	Unit ¹⁷⁶	Remarks
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 also GM_\oplus above.
N_A	$1/\text{mol}$	Avogadro's number
NaN		<p>"Not a Number", i.e. e.g. $0/0$ or $\pm\infty \times 0$ or $\ln(x)$ for $x < 0$ or $\tan(90^\circ)$ unless in <i>complex</i> domain.</p> <p>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 SPCRES is set. NaN allows that functions written by you can return it.</p>
p_0	Pa	Standard atmospheric pressure
R	J/mol K	Molar gas constant
r_e	m	Classical electron radius
R_K	Ω	Von Klitzing constant
R_{Moon}	m	Mean radius of the Moon
R_∞	$1/\text{m}$	Rydberg constant
R_\odot	m	Mean radius of the sun
R_\oplus	m	Mean radius of the Earth
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

... according to WGS84
(see the ReM)

Name	Unit ¹⁷⁶	Remarks
T_0	K	= 0°C, standard temperature
t_p	s	<i>Planck</i> time
T_p	K	<i>Planck</i> temperature
V_m	m^3/mol	Molar volume of an ideal gas at standard conditions $\approx 22.4\ l/mol$
Z_0	Ω	Characteristic impedance of vacuum
α	1	Fine-structure constant
γ	$m^3/kg\ s^2$	<i>Newtonian</i> constant of gravitation; also known as G from other authors. See also GM_⊕ above.
γ_{EM}	1	<i>Euler-Mascheroni</i> constant
γ_p	Hz/T	Proton gyromagnetic ratio
$\Delta\nu_{Cs}$	Hz	Hyperfine transition frequency of ¹³³ Cs
ϵ_0	F/m	Electric constant or vacuum permittivity
λ_c	m	<i>Compton</i> wavelengths of the electron, neutron, and proton, respectively
λ_{cn}		
λ_{cp}		
μ_0	H/m	Magnetic constant or vacuum permeability
μ_B	J/T	<i>Bohr</i> magneton
μ_e		Electron magnetic moment
μ_e/μ_B	1	Ratio of electron magnetic moment to <i>Bohr's</i> magneton
μ_n	J/T	Neutron magnetic moment
μ_p		Proton magnetic moment
μ_u		Nuclear magneton
μ_μ		Muon magnetic moment
σ_B	W/m^2K^4	<i>Stefan-Boltzmann</i> constant

Name	Unit ¹⁷⁶	Remarks
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 special numeric values in your WP 43S, however.
∞		

Employ the constants stored here for further useful equivalences, e.g.:

- express *joules* in *electron-volts* ($1 \text{ J} = 1 \text{ A s V} = 1 \text{ eV}/e \approx 6.24 \times 10^{18} \text{ eV} = 6.24 \times 10^9 \text{ GeV}$),
- calculate the wavelength from the frequency of electromagnetic radiation via $\lambda = c/\nu$ (so 1000 THz correspond to ca. 300 nm),
- determine the energy of electromagnetic radiation from its frequency via $E = h\nu$ (so $1 \text{ THz} \times h = 6.63 \times 10^{-22} \text{ J} = 4.14 \times 10^{-3} \text{ eV}$). Thus, 1 eV corresponds to 241.8 THz (or a wavelength of 1.240 μm).

Another **example**:

If you want to see the energy equivalent (in *electron-volts*) of one of the small masses given in kg above, multiply its mass by

$$c^2/e \approx 5.610 \times 10^{35} \text{ m}^2/\text{A s}^2$$

and you are done: m_e corresponds to 511.0 keV, m_p to 938.3 MeV, etc.

One more **final example**:

Assume American advanced scientists will succeed in producing a tiny bit of anti-matter in one of their high-tech laboratories one day – e.g. 0.1 μg of anti-hydrogen, carefully stored isolated in ultra-high vacuum. Although in future, most probably American power transmission lines will still look like they do today since this is a well-tried American (first) standard.

Thus, under slightly extreme weather conditions, however, an accidental blackout may easily happen for some days:¹⁷⁷ the electric vacuum pumps will stop working, and a subsequent vacuum breakdown will let atmospheric gas leak into the shiny vacuum vessel where it will interact with the precious anti-matter and annihilate immediately. How much energy is going to be released then?

Solution:

You only need the same tiny amount of (usual) matter, so 0.2 µg will annihilate in total within the vessel. 1 µg equals 10^{-9} kg. Enter:

DSP	ENG	3	0.000
.2	E	+/-	$.2 \times 10^{-9}$
CNST	c		299.8×10^6
x^2			89.88×10^{15}
x			17.98×10^6

... resulting in 18 MJ set free. The odds are frightening high this lab will need no cleaning anymore.¹⁷⁸

On the other hand, 0.1 µg of anti-matter requires e.g. $\frac{N_A}{10^7}$ atoms of anti-hydrogen (with N_A being Avogadro's number); this means 6×10^{16} atoms or 3×10^{16} molecules of this gas. Luckily, this amount is far from being produced in any lab for the time being.¹⁷⁹

¹⁷⁷ Think of a thunderstorm, blizzard or alike, maybe even fostered by climatic change. Though do not be afraid, this is all fake news created by insane minds according to the greatest president of that blessed nation.

¹⁷⁸ For comparison, 1 kg of TNT releases 4.6 MJ. The official definition is some 10% less than this value for historical reasons. Anyway, 18 MJ are equivalent to some 4 kg of TNT, enough for a great blast.

¹⁷⁹ And proper UHV vessels show very low leak rates as well, so the annihilation energy may be released in little bits over a longer time interval – power supply may be re-established in time and vacuum pumps operating again. For crucial applications, however, uninterruptible power sources based on batteries and/or generators should be installed locally wherever supplies are threatened by the actual state of public infrastructure.

Anyway, ordering antipasti and pasta together in an Italian ristorante is strictly at your own risk. You have been warned!

Unit Conversions

Your WP 43S features fourteen angular conversions stored in $\Delta \rightarrow$ (as shown on p. 122) and 86 unit conversions in $U \rightarrow$. The latter *menu* mainly provides means to convert local to common units and vice versa.¹⁸⁰



Also the constant T_0 may be useful for converting centigrade temperatures to *kelvin*. It is found in CNST and is not repeated in $U \rightarrow$ because it is only added or subtracted.

In an attempt to bring some order in that heap of units, $U \rightarrow$ is structured like a tree. Press $U \rightarrow$ 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 $U \rightarrow$ 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:

¹⁸⁰ 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 $U \rightarrow$ 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, $U \rightarrow$ 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 $U \rightarrow$, too.

$U \rightarrow$ may also give you a slight idea of the mess we had in the world of measuring before going metric following the French Revolution over 200 years ago. In the ReM, you find comprehensive explanations of all conversions provided.

Without *Imperial* and *US-American* units, $U \rightarrow$ would contain eighteen entries only.

							Remarks
<u>U:</u>	E: $^{\circ}\text{C} \rightarrow ^{\circ}\text{F}$	P: $^{\circ}\text{F} \rightarrow ^{\circ}\text{C}$	year \rightarrow s s \rightarrow year	F&p: $\text{Btu} \rightarrow \text{J}$	m: $\text{Wh} \rightarrow \text{J}$	x: $\text{J} \rightarrow \text{Wh}$	units of time, temperature, and ratios – the latter in an extra-high menu row
	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>	$\text{hp}_{\text{E}} \rightarrow \text{W}$	$\text{W} \rightarrow \text{hp}_{\text{E}}$	$\text{hp}_{\text{UK}} \rightarrow \text{W}$	$\text{W} \rightarrow \text{hp}_{\text{UK}}$	$\text{hp}_{\text{M}} \rightarrow \text{W}$	$\text{W} \rightarrow \text{hp}_{\text{M}}$	units of power
<u>F&p:</u>	$\text{lbf} \rightarrow \text{N}$	$\text{N} \rightarrow \text{lbf}$	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>	lb \rightarrow kg	kg \rightarrow lb.	cwt \rightarrow kg	kg \rightarrow cwt	oz \rightarrow kg	kg \rightarrow oz	units of mass
	stone \rightarrow kg	kg \rightarrow stone	short cwt \rightarrow kg	kg \rightarrow sh.cwt	tr.oz \rightarrow kg	kg \rightarrow tr.oz	
	ton \rightarrow kg	kg \rightarrow ton	short ton \rightarrow kg	kg \rightarrow short ton	carat \rightarrow kg	kg \rightarrow carat	
<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.			yd. \rightarrow m	m \rightarrow yd.	
	fathom \rightarrow m	m \rightarrow fathom	point \rightarrow m	m \rightarrow point	survey foot \rightarrow m	m \rightarrow survey foot	
<u>A:</u>	acre \rightarrow m ²	m ² \rightarrow acre	ha \rightarrow m ²	m ² \rightarrow ha	acre _{us} \rightarrow m ²	m ² \rightarrow acre _{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}	barrel \rightarrow m ³	m ³ \rightarrow barrel	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 combine conversions as you like (DSP ENG 2) will do for all examples following):

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:

$$30 \text{ [U→]} \text{ F&p: } \text{psi} \rightarrow \text{Pa} \quad \text{returns} \quad 207 \times 10^3 \text{ Pa.}$$
$$\text{Pa} \rightarrow \text{bar} \quad 2.07 \text{ bar.}$$

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 (yes, the dams in the Mississippi delta turn out being of less use than once thought). What does this mean in real units?

$$1 \text{ [U→]} \text{ x: } \text{ft.} \rightarrow \text{m} \quad \text{returns} \quad 305 \times 10^{-3}$$
$$3 \text{ [y}^x\text{]} \quad 28.3 \times 10^{-3}$$
$$10 \text{ [x]} \quad 283 \times 10^{-3} \text{ m}^3.$$

OK, some work – but manageable.

Example 3:

A network switch is specified for 3 320 Btu/h. What?!?

$$3320 \text{ [U→]} \text{ E: } \text{Btu} \rightarrow \text{J} \quad \text{returns} \quad 3.50 \times 10^6 \text{ J/h.}$$

Since $1J = c_c Wh \Leftrightarrow 1J/h = c_c W$ applies, you can use

$J \rightarrow Wh$ 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	U→	x:	in.→ m	returns	25.4×10^{-3}	
3	y^x				16.4×10^{-6}	
19	.	11	.	16	x	$323. \times 10^{-6} \text{ m}^3$.

Since $1 \text{ m}^3 = 1000 \text{ liter}$, this volume is almost $1/3 \text{ liter}$.

And to help those enduring life on the *British Imperial* islands or ex-territories, you must (!) ask them for their location first. Then choose either **U→ v: m³ → floz_{UK}** or **m³ → floz_{US}** and give them the respective result, i.e. **11.4** or **10.9**, 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 this translate for air pilots?

.1	U→	x:	pc → m	returns	$3.09 \times 10^{15} \text{ m.}$
	EXIT				returns to the top view of U→ .
1 year	→ s	(/)		returns	$97.8 \times 10^6 \text{ m/s.}$
	ENTER↑				pushes the result on the stack.
CNST	c			recalls $c =$	$300. \times 10^6 \text{ m/s.}$
(/)				returns	$326. \times 10^{-3} = 32.6\% \text{ of } c.$

Since $1 \text{ h} = 3600 \text{ s}$ and $1 \text{ km} = 1000 \text{ m}$, you can see directly that

$$3.6 \frac{\text{km}}{\text{h}} = \frac{3600 \text{ m}}{60 \times 60 \text{ s}} = 1 \frac{\text{m}}{\text{s}}$$

Thus, **R↓ 3.6 x** returns $352. \times 10^6 \text{ km/h.}$

This corresponds to

$$1000 \times \text{U→} \times \text{m} \rightarrow \text{nmi.} \quad 190. \times 10^6 \text{ nmi/h}$$

or 190 *megaknots*.

Supported by your *WP 43S*, you will find further easy ways to produce whatever conversions you may need personally in addition.

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.¹⁸¹ Remember also the example on pp. 89ff.

In this field, some outdated units may be found in older literature as well:

- Pour les fidèles amis de Madame *Marie Skłodowska Curie* (1903 Nobel laureate in physics and 1911 in chemistry), there was a unit *curie* with $1 \text{ Ci} = 3,7 \cdot 10^{10} \text{ Bq} = 3,7 \cdot 10^{10} \text{ decays/s}$. You can deduct from this unit that larger pieces of radioactive material were ‘absolutely no problem’ for the pioneers in this field.¹⁸²
- For those admiring the very first (1901) Nobel laureate in physics, *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* ($1 \text{ R} = 2,58 \cdot 10^{-4} \text{ A s/kg}$).¹⁸³
- A few decades ago, *rem* (i.e. *roentgen equivalent in men*¹⁸⁴) measured what *sievert* does today ($1 \text{ rem} = 10 \text{ mSv}$).
- And $1 \text{ Gy} = 100 \text{ rad}$ (i.e. *radiation absorbed dose*), which is pretty much since you will find almost nothing greater than *millirad* in literature.

¹⁸¹ Our warmest regards go to Algeria*, Australia*, Belarus*, Canada, China, France, India, Japan, Kazakhstan*, Kiribati*, the Marshall Islands* (e.g. Bikini, Eniwetok), North Korea, Pakistan, Russia, the Ukraine*, the United Kingdom, the USA, and Xinjiang-Uigur* (in alphabetical order) so far. The countries marked with a star suffer from actions of their respective ‘mother states’ at those times (the task to find out about those colonialists is left for the reader). The states without marks controlled their industry and/or military in a way that activated areas within their own territory could happen, too. At the bottom line, mankind gathers experience with radioactivity.

¹⁸² *Marie Curie* died from aplastic anemia, aged 66.

¹⁸³ *Conrad Röntgen* died from carcinoma of the intestine, aged 77.

¹⁸⁴ This unit must be outdated – it is not gender equitable.

SECTION 6: CREATING YOUR VERY PERSONAL WP 43S

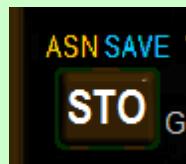
Your WP 43S is the first calculator worldwide allowing for fully customizing the user interface; i.e. you may assign an arbitrary function to almost any location, unshifted or shifted, on the keyboard or in a *menu*. *User mode* will then bring your personal assignments to the front, so you can interact 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	/	:	÷ ¹⁸⁵
Multiplication	×	•	
Radix mark	.	,	

Note this manual prints the default key labels throughout its text.

Use ASSIGN (ASN) for storing your personal favorite assignments. It allows for reassigning the entire keyboard except the top row of keys – these will stay *softkeys* always. Keep basic functionalities accessible (see pp. 285f for some caveats).



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*),

¹⁸⁵ Though see ISO 80000-2: “The symbol ÷ should not be used.”

- **menu** is the name of an arbitrary *menu* defined, either
 - entered directly by keying in **f** **α** (or **α** , cf. p. 55) and the necessary characters terminated by **ENTER↑**, or
 - picked from CATALOG by entering **f** **CATALOG** **MENUS**, browsing to the target *menu*, and pressing the respective , or
 - called from the keyboard by pressing the respective key headed by the associated *prefix*, if applicable; and
- **name** is the name of an arbitrary *item*. Remember an *item* may be an operation, function, digit, character, routine, variable, system *flag*, or a (*sub*) *menu* defined. **The name of an item consists of up to seven characters and must be unique within its set.** There are two sets:
 - One contains every operation, function, global label, and (*sub*) *menu* defined at execution time of ASSIGN.
 - The other set contains all the variables and system *flags* – a variable undefined at execution time of ASSIGN will be created (as explained on p. 60).

Note upper and lower case letters are checked, so the system will regard **Menu1** and **MENU1** as being different names. Superscripts and subscripts are not discriminated from normal characters, so e.g. **data1T** and **data₁T** are seen as the same name by your *WP 43S* but the latter may ease reading for you. Where a name is required, it may be either

- entered directly by keying in **f** **α** (or **α**) and the necessary characters terminated by **ENTER↑**, or
- picked from CATALOG by entering **f** **CATALOG**, choosing the respective branch, browsing to the target *item*, and pressing the respective , or
- called from the keyboard by pressing the respective key (optionally headed by a *prefix*).

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** **[key]** will assign that named *item* to the respective position in the bottom *menu row* displayed at the time you press **[key]**. In analogy, **[ASN] name** **f** **[key]** and **[ASN] name** **g** **[key]** 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. 285f) – except the *items assigned to top row of keys in two user menus* (see MyMenu and Mya below): **they will be displayed as long as no other menu is called.**

Example 1:

Let's assign the statistical sample standard error to **g + CC** (this location is assigned to **4** in *startup default*). There are three different ways to do this (specified here printing all keystrokes necessary):

1) **f ASN** **α** **▼** **S M** **ENTER↑** **g CC**

2) **f ASN** **f CATALOG** **FCNS** **S M** **s_m** **g CC**

This way will be shown step by step overleaf.

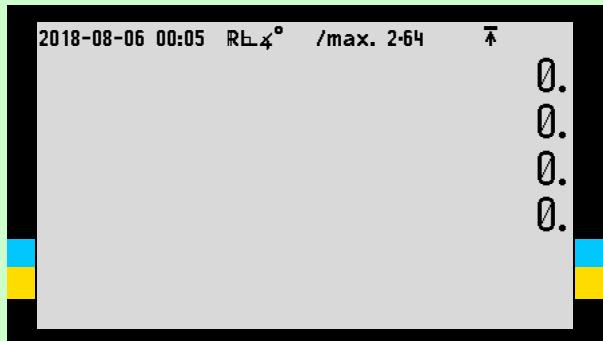
3) **f ASN** **f STAT** **s_m** **g CC**

On the other hand,

f ASN **ENTER↑** **g CC**

will reset **g-shifted CC** to factory default **4** as explained at the very end of last chapter.

We will demonstrate solution 2 step by step here, starting with a clear stack for sake of clarity:



Only the *menu section* and the command echo row will be shown in the following since all action will take place there:

The image shows three separate calculator screens arranged vertically, each with a title bar and a menu section.

- ASN**: Shows the "ASSIGN" command in the menu section, with the echo row showing "0.". The menu bar below includes "FCNS", "PROGS", "DIGITS", "CHARS", "VARS", and "MENUS".
- CATALOG**: Shows the "ASSIGN" command in the menu section, with the echo row showing "0.". The menu bar below includes "FCNS", "PROGS", "DIGITS", "CHARS", "VARS", and "MENUS".
- FCNS**: Shows the "ASSIGN" command in the menu section, with the echo row showing "0.". The menu bar below includes "AGM", "AGRAPH", "ALL", "AND", "arccos", and "arcosh". A detailed table below lists various functions:

2COMPL	$\sqrt[3]{x}$	ABS	ACOS	$ac \rightarrow m^2$	$ac_{us} \rightarrow m^2$
$^{\circ}C \rightarrow ^{\circ}F$	$^{\circ}F \rightarrow ^{\circ}C$	10^x	1COMPL	$1/x$	2^x

This is the top view of the FCNS submenu in CATALOG. Now enter the 1st letter of the requested command:

S

0. 

SETEUR	SETIND	SETJPN	SETSIG	SETTIM	SETUK	
SDL	SDR	SEED	SEND	SETCHN	SETDAT	
SAVE	SB	SCI	SCI0VR	scw→kg	SDIGS?	

Quickly entering the 2nd letter helps significantly:

- (M) (if you find you waited too long before pressing **M**, just wait another few *seconds*, then enter **SM** quickly here instead)

0. 

STOEL	STOIJ	STOS	STO+	STO-	ST0x	
STATUS	STO	STOCFG	STOEL	STOIJ	STOP	
s _m	SMODE?	s _{mw}	SOLVE	SPEC?	SR	

Now, press the leftmost  for the function to be assigned:

s_m

0. 

STOEL	STOIJ	STOS	STO+	STO-	ST0x	
STATUS	STO	STOCFG	STOEL	STOIJ	STOP	
s _m	SMODE?	s _{mw}	SOLVE	SPEC?	SR	

g

0. 

STOEL	STOIJ	STOS	STO+	STO-	ST0x	
STATUS	STO	STOCFG	STOEL	STOIJ	STOP	
s _m	SMODE?	s _{mw}	SOLVE	SPEC?	SR	

CC

0. 

STOEL	STOIJ	STOS	STO+	STO-	ST0x	
STATUS	STO	STOCFG	STOEL	STOIJ	STOP	
s _m	SMODE?	s _{mw}	SOLVE	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 **f** will stay accessible in *user mode* via **CATALOG FCNS ...** – or via default **g**  when leaving *user mode* shortly.

Example 2:

Assign the weighted arithmetic mean to the 1st key in MyMenu (assume *startup default* settings):

ASN

ASSIGN	_	0.

STAT

ASSIGN	*	0.
CLΣ	\bar{x}_g	ε
$\Sigma -$	\bar{x}_w	s_w
$\Sigma +$	\bar{x}	s
		PLOT
		ε_p
		ε_m
		s_{mw}
		SUM

\bar{x}_w

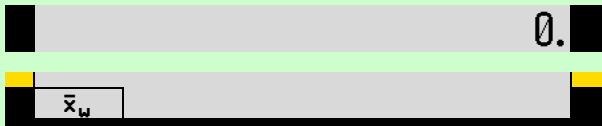
ASSIGN	\bar{x}_w	0.
CLΣ	\bar{x}_g	ε
$\Sigma -$	\bar{x}_w	s_w
$\Sigma +$	\bar{x}	s
		PLOT
		ε_p
		ε_m
		s_{mw}
		SUM

USER

ASSIGN	\bar{x}_w	0.

Note that pressing **USER** will exit all *menus* being open at that time so MyMenu (which is empty still) can slip on the screen.

(press the leftmost softkey)



Summarizing,

[ASN] [STAT] [x_w] [USER]

did this assignment.

Note that MyMenu will show up whenever all other *menu* are exited completely. It will remain on screen as long as no other *menu* is called, unless your *WP 43S* is in *alpha input mode*.¹⁸⁶ 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*. You may, for instance, put the six trigonometric functions into the unshifted row of MyMenu and will have them almost always at hand.

Creating Your Own Menus

[ASN] [USER] new_name [ENTER↑] will define a new user *menu*. Therein, **[ASN] [USER]** 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] [USER] F [▼] A [V] [▲] F [▼] U [N] [ENTER↑]

The new name will be inserted in CATALOG'MENUS (ASSIGN will throw an error if the 'new' *menu* name specified will turn out being defined already). The new *menu* itself will be created with 18 blank entries – its size is fixed. You may fill it now.

¹⁸⁶ In A/M, Mya will appear instead when no other *menu* is called and will stay on screen until these conditions will change.

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]

ASSIGN _ _

0.

\bar{x}_w

[STAT]

ASSIGN * _

0.

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

[▲]

ASSIGN * _

0.

	\bar{x}_{RMS}	
	\bar{x}_H	
L.R.	r	s_{xy} cov \hat{y} \hat{x}

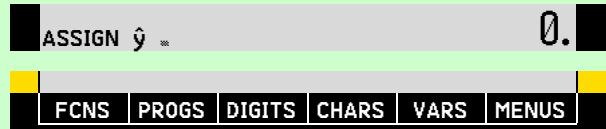
[ŷ]

ASSIGN \hat{y} _

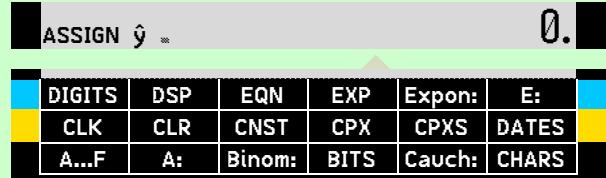
0.

	\bar{x}_{RMS}	
	\bar{x}_H	
L.R.	r	s_{xy} cov \hat{y} \hat{x}

[CATALOG]



MENUS



Here you see the first view on all the *menus* defined on your WP 43S. Now, enter **F** and the view jumps to the corresponding position in this submenu:

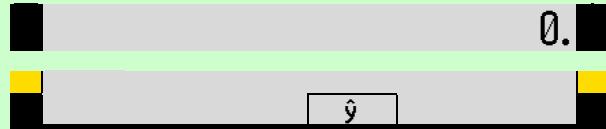


Press the leftmost *softkey*

FavFun



Since FavFun was just created above there is nothing to be seen in the *menu section* of the display yet. Pressing the fourth *softkey*, however, you will get now



Summarizing,

[ASN] [STAT] ▲ ♫ [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 up to seven characters long and must be unique (cf. p. 275).

CATALOG'PROGS contains all programs currently defined. Thus,

[CATALOG] PROGS ...] allows for deleting the program selected (cf. CLP and CLPALL).

Assigning Special Characters

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 (on p. 188). **[key]** may be any legal label location, shifted or unshifted, except

USER, **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 1st key in Myα (assume *startup default* settings once again):

α ASN **g** **ASSIGN _ _** **0.**

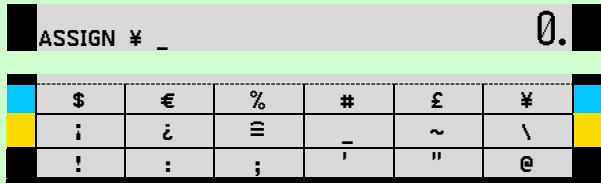
g **ASSIGN _ _** **0.**

0. **ASSIGN _ _** **0.**

	\$	€	%	#	£	¥	
!	:	;	'	"	\	@	

g

¥ (press the rightmost softkey)



USER



Note that **USER** will exit all *menus* being open at that time so Myα can slip on the screen (being empty still).

(press the leftmost softkey)



Summarizing,

(ASN) **g** **.** **¥** **USER**

did this assignment.

User Mode

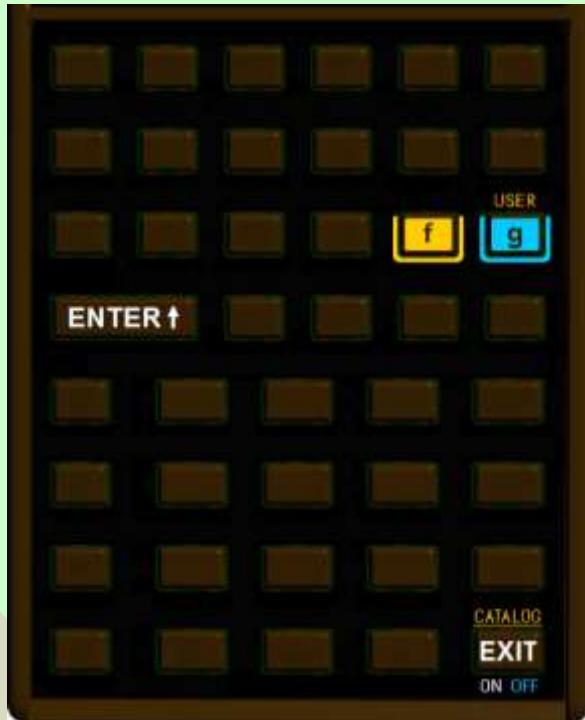
USER toggles *user mode*. Therein, your (user) assignments become valid wherever they apply. Except the top row of keys (being controlled by your MyMenu and Myα already, cf. pp. 279ff), everything is wide open for your ideas.

This mode gives you unexcelled freedom for creating your personal calculator layout and user interface. Enjoy *user mode* – and play with the world of opportunities you have. For obvious reasons, we recommend leaving **f**, **g**, **ENTER↑**, **USER**, **CATALOG**, **EXIT** / **ON**,



and **OFF** untouched.¹⁸⁷

WARNING: Do not remove said functionalities from the keyboard – you will not be happy with the result. Checking all consequences meticulously before reassigning operations is highly recommended; please note all reassessments you do are strictly at your own risk. In case of emergency, a hard reset will be the only escape – erasing all your precious programs and data but those you saved in *flash memory*.



Pressing any function key (or a *prefix plus a key*) displays a preview of the operation currently assigned to it left in the **T** numeric row – if you realize you have picked the wrong key, simply keep it pressed until the display falls back to **NOP** after 1 second.¹⁸⁸ This preview is particularly helpful in *user mode*, when the function executed by a key may not be the one indicated on the keyboard.

Once you have reached a stable user layout, we recommend storing it in a *register or variable* (using **STO CFG**), together with the other settings mentioned on p. 78. This applies especially if you plan having further alternative layouts – you can load any of them using

¹⁸⁷ Note **EXIT** and **ON** are connected.

¹⁸⁸ Preview and fallback apply for all keys except **f**, **g**, **0** ... **9**, **.**, **=**, **EXIT**, and all menu calls. Same holds for **±** and **CC** in numeric entry and for **C** in any entry. (On the *HP-42S*, preview and fallback are absent also for **PRGM**, **ASSIGN**, **STO**, **RCL**, **XEQ**, **SHOW**, and **OFF**.)



RCLCFG.¹⁸⁹ Think of storing a dedicated set of assignments for working with *short integers* featuring Boole's operations as primary functions, for instance.

It may pay well to print keyboard overlays for your favorite layouts, especially if you reassign just functions printed on the key plate. Adapting the surface of your *WP 43S* to such customizing is easy: note you can fix your overlays in the three slots provided on either side of the keyboard – see *App. F* of the *ReM* for the geometrical specifications.

Should you get lost in various user assignments, however, look for **U** top right in the *status bar* and remember that pressing **USER** will return immediately from *user mode* to the factory default keyboard of your *WP 43S* as you know it from the very beginning. And if you want to get rid of outdated user layouts and free the memory allocated for the respective assignments, simply clear the respective *registers* or delete the allocated variables as described on p. 283.

We sign off wishing you long lasting joy and benefit working with your very own, personalized *WP 43S*!

¹⁸⁹ RCLCFG will throw an error if you try recalling something different than a configuration.

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:

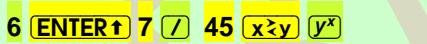
$$1 - 2 \cdot 3^4 : 5 + \sin(6 - \sqrt[3]{7^2}) \cdot 8! + \ln \left[\left(-9^{2^3} \cdot 45^{(6/7)} \right)^2 \right]$$

(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* with *startup default* settings:

 calculates -9^{2^3} ; note the arguments automatically fill in correctly.

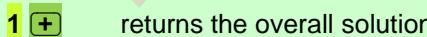
 calculates $45^{(6/7)}$.

 solves the rightmost term.

 solves the sine.

 solves the 3rd term and adds it to the fourth.

 solves the 2nd term and subtracts it from said sum.

 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>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 softkey row of the LCD.
		Call the function displayed at the corresponding position in the golden softkey row, respectively.
		Does nothing if there is no function displayed at this position. Cf. pp. 26f.
2		Inverts the number x or all elements of the matrix x .
		Enters fraction display mode (<i>FDM</i>), i.e. displays all <i>reals</i> as <i>proper fractions</i> or mixed numbers. If <i>FDM</i> was active already, toggles display between <i>proper</i> and <i>improper fractions</i> .
		Opens the <i>menu</i> of operations for <i>alpha string</i> manipulation. Cf. pp. 192f.
		Enters the letter A or a
		Opens the <i>catalog</i> of all Latin letters provided (also accented ones) in AIM (cf. pp. 187ff).
		Enters the Greek letter Α or α

R	Keystrokes	Meaning
2	y^x	Raises y to the power of x .
	#	If pressed trailing integer input, defines its base. Else converts x into a <i>short integer</i> of the base specified, cf. pp. 132ff.
	EXP	Opens a <i>menu</i> containing x^3 , roots, 2^x , logarithms, hyperbolic and some exponential functions more (cf. p. 26).
	B	Enters the letter B or b
	#	Enters the character # in AIM (cf. pp. 187ff).
	g B	Enters the Greek letter B or β
2	TRI	Opens the <i>menu</i> containing SIN, COS, TAN, hyperbolic functions, and their inverses (cf. p. 28)..
	d.m.s	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. Cf. pp. 121ff.
	π	Recalls the number π into X.
	C	Enters the letter C or c in AIM.
	g C	Enters the Greek letter Γ or γ
2	In	Returns the natural logarithm of x . ¹⁹⁰
	.d	If pressed trailing numeric input, enters a <i>date</i> (cf. p. 185). Else leaves fraction display mode (see a b/c above) and converts <ul style="list-style-type: none"> • an integer to a <i>real number</i> (cf. p. 131), • a sexagesimal <i>angle</i> to a decimal number (cf. p. 124), • a sexagesimal <i>time</i> to a decimal number (cf. p. 184).
	Ig	Returns the (common) decadic logarithm of x . ¹⁹⁰
	D	Enters the letter D or d in AIM.
	g D	Enters the Greek letter Δ or δ

¹⁹⁰ I.e. either of the number x or of all elements of the matrix x .

R	Keystrokes	Meaning
2	e^x	Raises e to the power of x . ¹⁹⁰
	h.ms	If pressed trailing numeric input, enters a sexagesimal <i>time</i> . Else converts x to such a <i>time</i> . Cf. pp. 183f.
	10^x	Raises 10 to the power of x . ¹⁹⁰
	E	Enters the letter E or e
	g E	Enters the Greek letter E or ϵ in AIM (cf. pp. 187ff).
2	x²	Returns the square of x . ¹⁹⁰
	α	Sets AIM for entering characters (cf. pp. 187ff).
	✓x	Extracts the square root of x . ¹⁹⁰
	F	Enters the letter F or f
	f x²	Enters the character \checkmark in AIM.
3	STO	Stores (copies) x in the destination specified (cf. pp. 52ff).
	ASN	Assigns an <i>item</i> to a key, allowing you to create your very personal user keyboard layout (cf. pp. 274ff).
	SAVE	Saves all your data in the backup region (cf. p. 227) from where they may be recovered by LOAD.
	G	Enters the letter G or g
	g G	Enters the Greek letter Γ or γ in AIM.

R	Keystrokes	Meaning
3	RCL	Recalls (copies) a stored object into X (cf. pp. 52ff). If pressed in RBR, leaves RBR after recalling the object at the bottom line or entering a corresponding step (cf. pp. 254ff).
	VIEW	Views the destination, i.e. displays its address and contents directly below the <i>status bar</i> until next keystroke (cf. p. 58).
	CNST	Opens a catalog of fundamental physical, mathematical, astronomical, and surveying constants. Cf. pp. 263ff.
	H	Enters the letter H or h _____ in AIM (cf. pp. 187ff).
	g H	Enters the Greek letter X or χ
3	R↓	Rolls the <i>stack</i> contents one level down (cf. p. 38).
	RBR	Calls the <i>register browser</i> (cf. pp. 254ff).
	R↑	Rolls the <i>stack</i> contents one level up.
	I	Enters the letter I or i _____
	f R↓	Makes next character a subscript (if applicable) _____ in AIM.
	g I	Enters the Greek letter I or ι
3	CC	<i>Complex</i> closing, composing, cutting, and converting, see pp. 149ff and 300.
	 x 	Returns the absolute (unsigned) value of x . ¹⁹⁰
	∠	Either returns the phase of x ¹⁹⁰ or the angle between the vectors x and y .
	J	Enters the letter J or j _____ in AIM.
	g J	Enters the Greek letter H or η
3	f	Prefix to reach a secondary gold function label.
	CPX	Opens the <i>menu</i> of commands operating on <i>complex numbers</i> like CONJ, CROSS, DOT, and Re>Im. Cf. pp. 149ff.

R	Keystrokes	Meaning
3		Prefix to reach a secondary blue function label.
		Toggles <i>user mode</i> (see pp. 285ff).
4		Context sensitive key, see p. 300.
		Returns free space available, memory currently used, <i>user</i> and <i>system flags</i> set (cf. pp. 256f).
		Drops x from the <i>stack</i> (cf. p. 38).
4		Swaps the contents of X and Y (cf. p. 38).
		Fills all <i>stack registers</i> with x (cf. p. 38).
		Opens the <i>menu of stack related operations</i> (drop, swap, and shuffle commands). Cf. pp. 37ff.
		Enters the letter K or k
		Enters the character \gtrless in AIM (cf. pp. 187ff).
4		Enters the Greek letter Κ or κ
		If pressed during input of mantissa or exponent, changes its sign (cf. p. 24). Else multiplies x times -1.
		Returns $x - y$ % of y . Leaves y unchanged.
		Opens the <i>menu of financial functions</i> (i.e. % functions and the application TVM – see pp. 259ff and 303ff).
		Enters the letter L or I
		Enters the character \pm in AIM.
		Enters the Greek letter Λ or λ

R	Keystrokes	Meaning
4	E	Allows entering an exponent of ten for convenient entry of very large or very small numbers (cf. p. 24).
	SHOW	Shows the number x with its maximum precision until next key-stroke.
	DSP	Opens a <i>menu</i> containing FIX, SCI, ENG, and more commands for numeric display formatting. Cf. pp. 78ff.
	M	Enters the letter M or m
	f E	Makes next character a superscript (if applicable) in A/M (cf. pp. 187ff).
4	g M	Enters the Greek letter M or μ
	⬅	Context sensitive key, see p. 302.
	↶	Undoes the last command executed (cf. p. 50).
5	CLR	Calls a <i>menu</i> containing commands for clearing; cf. p. 51.
	/	Divides y by x . For matrices, multiplies y times x^{-1} .
	RMD	Returns the remainder of y divided by x .
	MOD	Returns y modulo x .
	N	Enters the letter N or n
5	f /	Enters the character $/$ in A/M.
	g N	Enters the Greek letter N or ν
	7	If there is an open question like Are you sure? , enters N for 'no'. Else enters the digit 7 .
5	O	Enters the letter O or o
	f 7	Enters the character 7 in A/M.
	g O	Enters the Greek letter Ω or ω

R	Keystrokes	Meaning
5	8	Enters the digit 8 .
	P	Enters the letter P or p
	f 8	Enters the character 8 in AIM (cf. pp. 187ff).
	g P	Enters the Greek letter Π or π
5	9	Enters the digit 9 .
	RTN	Returns to the caller. Cf. pp. 196ff.
	Q	Enters the letter Q or q in AIM.
	f 9	Enters the character 9
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 the specified location in program memory.
	LBL	Enters a label for a particular location in program memory.
6	x	Multiplies y times x .
	x!	Returns the factorial of x (or $\Gamma(x + 1)$ for non-integer x).
	PROB	Opens a <i>menu</i> containing combinations, permutations, the Gamma function, a random number generator, and all probability distributions supported. Cf. pp. 93ff.
	R	Enters the letter R or r
	f x	Enters the character × or - in AIM.
	g R	Enters the Greek letter Ρ or ρ

R	Keystrokes	Meaning
6	4	Enters the digit 4 .
	STAT	Opens the <i>menu</i> of sample statistics operations: $\Sigma+$, $\Sigma-$, CL Σ , various means and measures for scattering, as well as curve fitting functions and settings. Cf. pp. 96ff.
	Σ	Opens the <i>menu</i> of accumulated statistical sums, cf. p. 114.
	S	Enters the letter S or s
	f 4	Enters the character 4 in AIM (cf. pp. 187ff).
	g S	Enters the Greek letter Σ or σ
6	5	Enters the digit 5 .
	R\leftrightarrow	Calls \rightarrow REC, converting polar coordinates r (in X) and θ (in Y) to rectangular (<i>Cartesian</i>) coordinates x and y (cf. p. 124).
	\rightarrowP	Calls \rightarrow POL, converting rectangular coordinates (x and y) to polar coordinates r (in X) and θ (in Y, cf. pp. 19f).
	T	Enters the letter T or t
	f 5	Enters the character 5 in AIM.
	g T	Enters the Greek letter T or τ
6	6	Enters the digit 6 .
	U\rightarrow	Opens the <i>menu</i> of unit conversions. Cf. pp. 269ff.
	L\rightarrow	Opens the <i>menu</i> of angular conversions. Cf. p. 122.
	U	Enters the letter U or u
	f 6	Enters the character 6 in AIM.
	g U	Enters the Greek letter Θ or ϑ
6	▲	Context sensitive key, see p. 302.
	\equiv▲	Moves the program pointer one step back. Cf. pp. 196ff.
	FLAGS	Opens the <i>menu</i> of flag commands. These are of most use in PEM. Cf. pp. 196ff.

R	Keystrokes	Meaning
7		Subtracts x from y .
		Opens a <i>menu</i> of advanced operations for solving arbitrary equations, finding roots, integrating, deriving, computing sums and products (cf. pp. 229ff).
		Opens the <i>menu</i> of all equations currently defined (cf. pp. 232ff).
		Enters the letter V or v
		Enters the character $-$ in AIM (cf. pp. 187ff).
7		Opens a <i>menu</i> of math symbols
		Enters the digit 1 .
		Opens a <i>menu</i> containing Boole's operations (AND, OR, NOT, etc.) as well as bit manipulating commands.
		Opens a <i>menu</i> of operations for integers as well as sign mode settings.
		Enters the letter W or w in AIM.
7		Enters the Greek letter Ψ or ψ
		Enters the digit 2 .
		Opens the <i>menu</i> of matrix operations including e.g. $[M]^{-1}$, $ M $, $[M]^T$, CROSS, DOT, and the <i>Matrix Editor</i> (cf. pp. 158ff).
		Opens a <i>menu</i> of advanced mathematical (extra) functions. See the ReM.
		Enters the letter X or x
7		Enters the character 2 in AIM.
		Enters the Greek letter Ξ or ξ

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 (cf. pp. 257ff).
	CLK	Opens the menu of <i>time</i> and <i>date</i> commands. Cf. pp. 183ff.
	Y	Enters the letter Y or y
	f 3	Enters the character 3 in <i>AIM</i> (cf. pp. 187ff).
	g Y	Enters the Greek letter Υ or υ
7	▼	Context sensitive key, see p. 302.
	≡▽	Moves the program pointer one step forward (cf. pp. 196ff).
	MODE	Opens a <i>menu</i> of operations for setting modes like angular display format, max. denominator, etc. (cf. pp. 121ff and 146ff).
8	+	Adds x to y .
	I/O	Opens the <i>menu</i> of I/O-related operations. Cf. pp. 226f.
	PRINT	Opens the <i>menu</i> of print-related operations.
	Z	Enters the letter Z or z
	f +	Enters the character + in <i>AIM</i> .
	g +	Enters the printer character ¤
8	0	Enters the digit 0 .
	LOOP	Opens a <i>menu</i> containing INC and DEC and the related loop control commands ISG, DSE, etc. Cf. pp. 211f.
	TEST	Opens the <i>menu</i> of comparisons, conditionals, and other binary tests. Cf. pp. 208ff.
	?	Enters the character ?
	f 0	Enters the character 0 in <i>AIM</i> .
	g Z	Enters the Greek letter Ζ or ζ

R	Keystrokes	Meaning
8		Usually enters a decimal radix mark in numeric input. If pressed twice in numeric input, allows for entering a fraction (cf. pp. 67f and 146ff). In register or flag addressing, heads a local address (cf. pp. 56ff).
		Opens a <i>menu</i> containing FP, IP, SIGN, DECOMP, etc.
		Opens a <i>menu</i> of commands to return system information. Cf. p. 211.
		Enters a comma
		Enters a point
		Enters a blank space in AIM
		Enters a punctuation mark etc.
8		Context sensitive key, see p. 301.
		Toggles <i>program-entry</i> and <i>run mode</i> .
		Opens a <i>menu</i> of dedicated programming functions. These are of most use in PEM. Cf. pp. 196ff.
		Enters a blank space in AIM.
8		Context sensitive key, see p. 301.
		Opens the <i>catalog</i> of everything (functions, variables, menus, programs, etc.). See the ReM for its structure and contents.
		Turns your WP 43S off unless in PEM, where it inserts OFF behind the <i>current step</i> (cf. p. 198).

Seven context sensitive keys need longer explanations – find them in the table below, 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 next input.

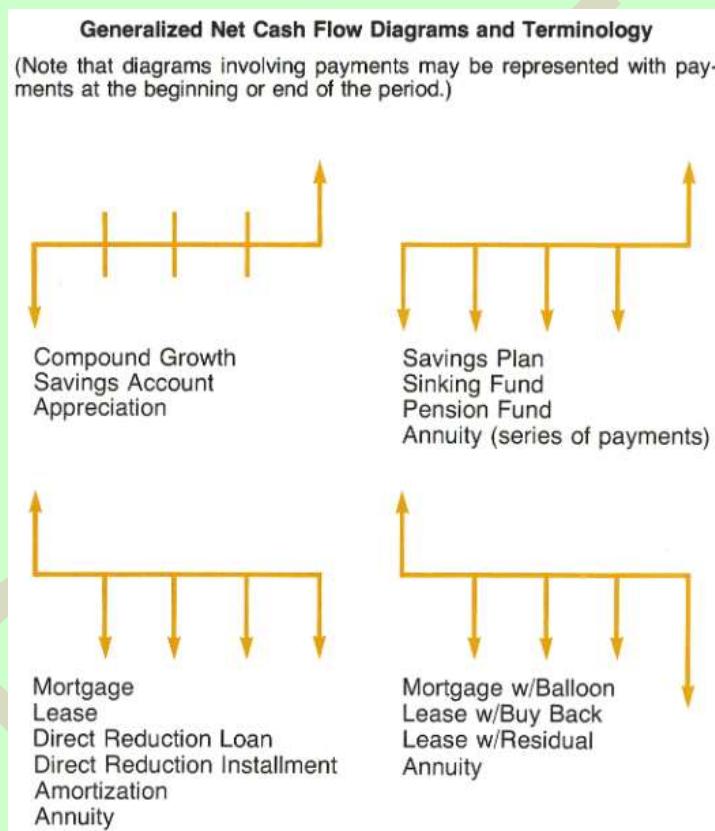
Key	Condition(s)	Meaning
CC	X contains an open (input) number, cf. p. 24	If RECTN is set, CC closes input, checks, and saves it as <i>real</i> part of a forthcoming <i>complex number</i> , then waiting for your input of its <i>imaginary</i> part. Else CC closes, checks, and saves the input as magnitude and waits for your input of the phase. Cf. pp. 149ff for more.
	X contains a closed <i>complex number</i> , vector, or matrix	If RECTN is set, CC splits ('cuts') <i>x</i> into its <i>real</i> and <i>imaginary</i> part, returning the <i>real</i> part in Y and the <i>imaginary</i> part in X . Else CC splits <i>x</i> into its magnitude r and phase θ , returning r in Y and θ in X .
	X and Y contain two closed <i>reals</i>	Interprets <i>y</i> and <i>x</i> either (for RECTN clear) as magnitude and phase, or (for RECTN set) as <i>real</i> and <i>imaginary</i> parts. CC combines <i>y</i> and <i>x</i> to compose one <i>complex number</i> <i>x</i> , then drops <i>y</i> .
	X and Y contain two closed <i>real</i> vectors (or matrices) of identical dimension	Returns one <i>complex</i> 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 (cf. p. 62 for more).
	Asking for confirmation	Confirms the question.
	In TIMER	Is honored as described on pp. 257f.
	In RBR, STATUS	Does nothing.
	Else	Closes alphanumeric input and enters data in the <i>stack</i> (cf. pp. 32f and 38 for details).

Key	Condition(s)	Meaning
EXIT / ON	WP 43S turned off	Works as ON turning your WP 43S on.
	<i>Waiting for parameter input</i>	Cancels the pending command.
	Waiting for alphanumeric input	Closes input (note alphanumeric includes numeric 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. Cf. p. 67.
	Asking for confirmation	Denies the question.
	In RBR, STATUS, TIMER	Leaves the application (cf. pp. 254ff).
	In a (<i>sub-</i>) menu or browser	Leaves the current (<i>sub-</i>) menu or <i>browser</i> without executing anything, returning to the status of your WP 43S as it was before.
	 flashing	Stops executing the running program immediately.  will be lit until next keystroke.
	In PEM	Leaves <i>program-entry mode</i> like P/R .
	A or α	Closes <i>x</i> and leaves <i>alpha input mode</i> .
	Else	Does nothing.
R/S	In TIMER	Starts or stops the timer without changing its value (cf. pp. 257f).
	 flashing	Stops executing the running program immediately.  will be lit until next keystroke.
	In PEM	Enters the command STOP.
	Else	Runs the <i>current routine</i> (cf. pp. 196ff) or resumes its execution starting with the step after the <i>current step</i> .

Key	Condition(s)	Meaning
▲ or ▼	After STO or RCL	Honored as described on pp. 57ff.
	In RBR, STATUS, TIMER	Honored as described in Sect. 5 (pp. 254ff).
	A & in (<u>aINTL</u> or A...Ω)	▼ sets lower case.
	α & in (<u>aINTL</u> or A...Ω)	▲ sets upper case.
	A else	▼ sets lower case. Else continue testing.
	α else	▲ sets upper case. Else continue testing.
	In EQN	▲ goes to next and... ▼ to previous equation, if applicable.
	In a <i>multi-view menu</i>	▲ goes to next and... ▼ to previous view in the current <i>menu</i> .
	In PEM	▲ goes to previous and... ▼ to next program step. Will repeat with 2Hz when pressed longer than 0.5s.
	In <i>run mode</i>	Browses the <i>current routine</i> with... ▲ going to previous program step and... ▼ executing the current program step and going to next step.
⬅	Open alphanumeric input	Deletes the last character entered. If none is left, cancels pending command like EXIT .
	<i>Temporary information</i> displayed	Clears the information returning to the calculator state as was before this (e.g. an error message) was thrown. See p. 67.
	Asking for confirmation	Denies the question.
	In TIMER	Resets the timer (cf. pp. 257f).
	In PEM	Deletes the current program step.
	Else	Calls the command CLX.

APPENDIX 3: FURTHER APPLICATIONS OF TVM

Throughout TVM pictures, amounts received are represented by arrows pointing up, money laid out (paid, invested) by arrows pointing down. Various types of financial problems can be sketched like this then:¹⁹¹



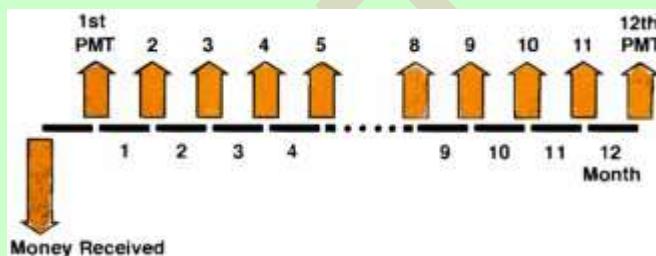
The following examples as well as all the other text printed blue in this appendix are quoted from the *HP-27 OH*. All calculations are executed in FIX 2. Enjoy the boundary conditions of that time – those were the days ...

¹⁹¹ Translator's note: You can use this picture as a dictionary of some financial terms in English. The word "with" is abbreviated by "w/" although this does not save any space here. Abbreviomania ...

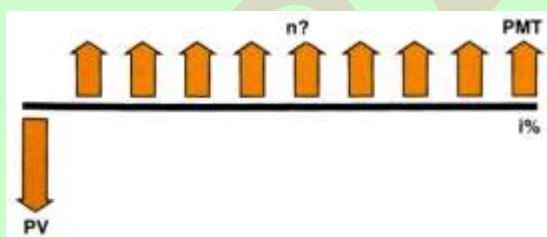
Ordinary Annuities (a.k.a. Payments in Arrears)

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 semiannually?

Solution:

[FIN] [TVM] [End]
5 **[ENTER]** 2 **[/]** **i%/*a***
50000 **PV**
3000 **PMT** **n_{PER}**

withdrawals are due at the end of each period,
2.50 % semiannual interest rate,
50 000.00 principal (capital),
21.83 semiannual withdrawals, so
your savings will last for almost 11 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	12.00	months per year,
12 [ENTER↑] 2 × nPER	24.00	periods in total,
1775 PV	1 775.00	principal (capital),
83.65 PMT	83.65	payment;
i%/a	12.11	% APR.

Borrowers are sometimes charged fees related to the issuance of a mortgage, which effectively raises the interest rate. Given the basis of the fee charge, the true annual percentage rate may be calculated.

Example 2 for finding the interest rate for an ordinary annuity:

A borrower is charged 2 points for the issuance of his mortgage. If the mortgage amount is \$50 000 for 30 years, and the interest rate is 9% per year, with monthly payments, what annual percentage rate is the borrower paying? (1 point is equal to 1 % of the mortgage amount.)

Solution:

First, compute the payment amount which is based on \$50 000

9 i%/a	9.00	annual interest rate,
12 per/a	12.00	months per year,
12 [ENTER↑] 30 × nPER	360.00	periods in total,
50000 PV	50 000.00	principal (capital);
PMT	83.65	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]	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 – usually considerably larger although it could also be smaller than the others – is called a *balloon payment* or *balloon*.

By subtracting the present value of the balloon payment from the loan amount, the problem effectively becomes "What is the monthly payment on a direct reduction loan?"

Example (finding the payment for an ordinary annuity with *balloon*):

Yellowstone Sam is heading north, and will invest in an \$8 000 dog sled and team. His loan specifies 60 monthly payments at 10% with a *balloon payment* in the 60th month of \$3 000. What will his monthly payments be?

Solution:

12 per/a	12.00	months per year,
60 nPER	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 <i>balloon</i> ;

RCL	i%/a	i%/a	10.00	recall & reuse interest rate,
RCL	n _{PER}	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:

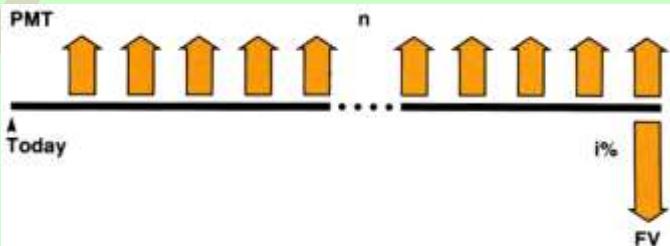
Yellowstone Sam decides to purchase a snowmobile. He plans to pay \$80 per month for 3 years, and he's willing to pay 10% annual interest. How much can he afford to pay for the snowmobile?

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?

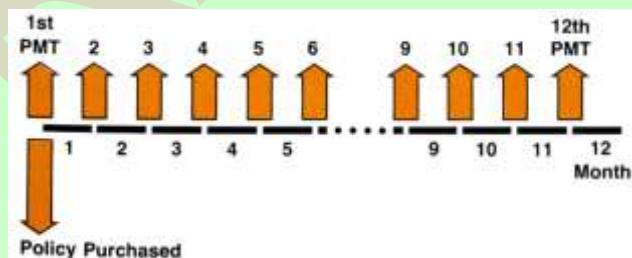
Solution:

2 per/a	2.00	halves per year,
10 ENTER↑ 2 × nPER	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 policy¹⁹² – looks like this

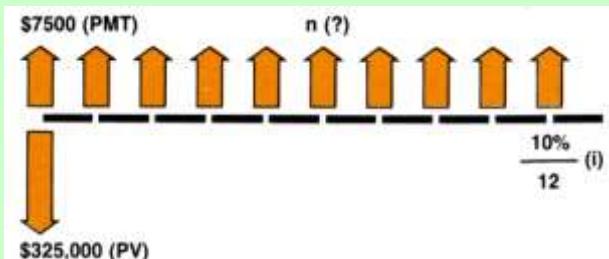


Notice that with an annuity due, you have a payment right away at the beginning of the first interval (with an ordinary annuity, your payment is not due until the end of the first period, but you also have a payment at the end of the entire term).

The following calculations all deal with *annuity due* problems, e.g. savings, insurance, leases, and rents.

¹⁹² Translator's note for German readers: "Policy" entspricht hier einer *Police*.

Example 1 for finding the number of periods for an annuity due:



Given an investment possibility of \$325 000 that will immediately produce rental income of \$7 500 per month, how long must the investment be held to yield 10% per annum?¹⁹³

[FIN] TVM Begin
12 per/a
10 i%/a
325000 PV
7500 PMT nPER

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?

Solution:

12 per/a
6 i%/a
0 PV
1000 FV
50 PMT nPER

12.00 months per year,
6.00 % annual interest rate,
0.00 start balance,
1 000.00 future value;
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?

¹⁹³ I frankly admit I understand neither this problem nor its solution.

Solution:

12	per/a	12.00	months per year,		
8	ENTER↑	12	× nPER	104.00	payment periods in total,
12000	PV	12000.00	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:

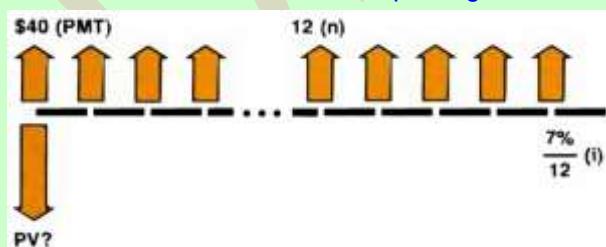
The owner of a building presently worth \$70 000 intends to lease it for 20 years at the end of which time he assumes the building will be worthless (i.e., has no residual value). How much must the quarterly payments (in advance) be to achieve a 10% annual yield?

Solution:

4	per/a	4.00	quarters per year,		
20	ENTER↑	4	× nPER	240.00	payment periods in total,
10	i%/a	10.00	% annual target yield,		
70000	PV	70000.00	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	12.00	months per year,
----	-------	-------	------------------

12 n_{PER}

7 $i\% / a$

40 PMT

PV

12.00

7.00

40.00

464.98

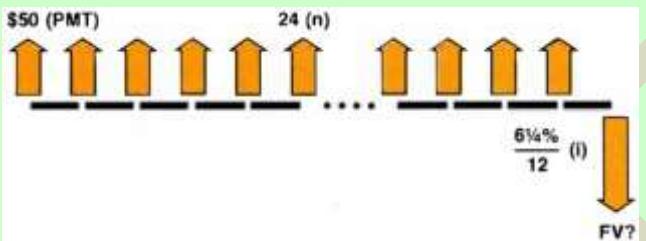
payment periods in total,

% annual target yield,

monthly payments;

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 $\text{ENTER} \uparrow$ 12 \times n_{PER}

6.14 $i\% / a$

50 PMT

0 PV

FV

12.00

24.00

6.25

40.00

0.00

1 281.34

months per year,

payment periods in total,

% annual target yield,

monthly payments;

start balance;

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. Watch p. 16 and see the *ReM*, App. A for more.

WARNING: Removing the battery for longer than **xxx seconds** may erase all data in RAM – only data in *flash memory* will remain.

See what sufficed for explaining the basic functionality of the *HP-45* on its back label in 1973:



Though it featured neither advanced operations (just four statistical sums, means, and standard deviations), programming, *menus*, *catalogs*, *data types*, browsers, applications, named variables, nor customizing – but your *WP 43S* does.

Introducing the first
pocket programmable
that remembers your
programs and data even
when it's turned off.
The new HP-25C. \$200.00*



Note the first scientific Programmable calculator that lets you take a break without turning it off without having to enter all over again when you get back. The reason? It memory and storage registers can CN even when it's off! - during a phone call, a shopping spree, or a weekend.

We call it the HP-25C. (disclaimer: the "C" stands for "Continuous Memory CMOS" technology.) *Continuous Memory*

There's one key to your favorite programs - defining it - and reuse it as many times as you want.

1. Select one key as your favorite program - defining it - and reuse it as many times as you want.
2. You can add additional functions via the keyboard (probability, scientific constants, etc.) simply by writing programs and saving them permanently in memory.

3. It makes it easy to gather data one place (such as at a survey point) and reuse it at another until you're back to work with data from four blocks of the time you'll need and the accuracy you'll gain because you won't have to re-enter information.

4. You don't lose data when you lose battery power. That's why after the thermal power display - following a low battery - most data and programs will be lost provided the calculator is turned OFF and the recharge is promptly commenced. The HP-25C will store more user data and programs for at least 5 seconds without any battery drain while you are replacing a discharged battery with a charged one.

But that's not all. With built-in Continuous Memory, the HP-25C is a different kind of pocket HP-25. Both give you a total of 72 pre-programmed functions and capabilities, complete keyboard (probability, branching and conditional test capability) and linear, arithmetical, and engineering functions. In



addition, "Dynamically,"

you can enter up to 270 logic entries with 8 memory levels and 3 storage registers.

And every program written for the HP-25 will work without modification for the HP-25C.

The HP-25C. There's more than a scientific Programmable Calculator like a before.

But of all the HP-25C's new HP's name on it. And you know what that means: Design, performance and a tracking support system that can't be beat.

Why not "buy touch" one for yourself? Or if Continuous Memory is not big enough for your particular application, we've standard HP-25, 3491A/3492. For complete HP-25C details and the name of your nearest dealer, call toll free 800-538-7922 (in Calif., 800-467-0882).

THE HP-25C AT A GLANCE

- Program, memory and storage registers use CN at all times.
- 72 functions and operations.
- Keystroke programmability.
- Full editing capability.
- Branching and conditional test capability.
- 8 addressable registers.
- Fixed decimal and scientific notation.
- Engineering notation.
- RPN logic entries with 8 memory levels.
- 3000 CF.

HEWLETT *hp* PACKARD

Write and wire to: 1211 Mission St., San Francisco,
CA 94103. 415/567-1000. Telex 321-2000.

In 1976, *Continuous Memory* was an innovation; and a grand total of 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. Note 200 US\$ of 1976 correspond to 911 US\$ of today! Linear regressions, correlations, and forecasting had to be programmed by you if you needed them – the respective routine as recommended by *HP* took 44 precious steps.

APPENDIX 5: RELEASE NOTES

	Date	Release notes
0	29.11.12	Official project start with first publication of the 43S concept and a layout on one of the forums of the <i>Museum of HP Calculators</i> (https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685). Though there are found far older traces of a '43S' denoting a 'Super HP-42S', though in various more or less fictional cases – pure vapourware™.
0.1	2.2.14 23.5.15	Manual setup based on the one of WP 34S. Passed to <i>Jake Schwartz</i> , <i>Eric Smith</i> , and <i>Richard Ottosen</i> for first information.
0.2	3.10.15	Update based on Jake's feedback and further thoughts, distributed to <i>Eric</i> , <i>Jake</i> , <i>Marcus</i> , and <i>Pauli</i> .
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 <i>Michael Steinmann</i> 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 keyboard layout.
0.6	22.8.17	Merged the Applications and Owner's Manual. Changed the input order of complex number parts on Pauli's request. Changed keyboard layout introducing D.MS, SST, BST, and % while removing 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.

Date	Release notes
0.7 2.4.18	Changed keyboard layout. Replaced the labels BST by , 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 HP-42S. 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 S/ 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 HP-42S.
0.8 7.5.18 20.9.18	Changed keyboard layout: introduced TRG containing trigonometric functions, removed HYP into EXP and to g-shifted , swapped some shifted labels. Refined the chapters about register arithmetic, Command Parameter Input, Alphanumeric Input, Matrix Calculations, and Orthogonal Polynomials. Introduced CLCVAR and more vintage examples. Rearranged temporary information on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE. Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9 3.1.19	Removed angle data type. Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded App. B. Added SHOW for displaying full precision of DP numbers and FBR for browsing our two fonts. Split a chapter. Expanded some titles. Added the overlay drawing. Modified functionalities of EXIT and to match HP-42S. Added a chapter about curve fitting. Modified functionalities of ENTER↑ and . Expanded App. K. Renamed DOUBLE to →DP. Added →SP and conversions of quarts. Rearranged X.FN. Replaced USR by UM. Changed keyboard moving UM, , and TRI. Moved to R/S. Added XIN and XOUT. Added a chapter in App. E and information about infinite integers. Extended the domain of GCD and LCM. Refined and corrected.

	Date	Release notes
0.10	3.3.19	Returned <i>angle</i> data type and αSR . Added IDIVR and VANGLE. Refined FP, IP, IMPFRC, PROFRC, SDIGS?, $\rightarrow DP$, $\rightarrow HR$, $\rightarrow INT$, $\rightarrow REAL$, $\rightarrow SP$, explanation of ALL, the summary of integer functions, and handling of long alpha strings. Modified contents of CPX, MATX, and $\alpha \bullet$. Added a summary of matrix functions. Removed the [ON] -key combinations. Modified MEM?. Rewrote the angular conversions. Renamed infinite and finite integers to <i>long</i> and <i>short integers</i> . Added a chapter about $\pm\infty$ and NaN. Modified RBR and the menu for STO and RCL. Removed \blacksquare from the keyboard. Renamed X_u to X_e for the distributions. .
0.11	8.5.19	Changed keyboard making [CC] primary and user mode shifted, removing x^2 , $x\zeta$, and DSP, adding $ x $, DROP, and SHOW, and moving some shifted labels. Modified BITS, CLREGS, CNST, CPX, DISP, EXP, INTS, MODE, PARTS, SHOW, STAT, U \rightarrow , aMATH, the division matrix, data type conversions, and the Quick Reference Guide. Added conversions of barrels, carats, and fathoms. Deleted DSP. – Separated predefined variables. Refined Sect. 6. Added \bar{x}_H , \bar{x}_{RMS} , nine statistical sums and five curve fit models. Split STAT in STAT and SUMS; renamed RMDR to RMD, L_n to L_m , L_{na} to L_{ma} , Π to Π_n , Σ to Σ_n , and some constants to avoid search ambiguities. Refined App. J, Sect. 3 and 4, $\rightarrow INT$, CLR, and the functions of Δ and ∇ . Put SUMS instead of RMD on the keyboard, moved ADV, BITS, CATALOG, EQN, FILL, INTS, MATX, MODE, PROB, RTN, SHOW, STAT, and αFN . Rearranged A... Ω and Sect. 2 of the OM.
0.12	16.10.19	Rearranged the appendices of the ReM from App. D on. Expanded App. A of the OM and App. K. Deleted the standardized normal distribution Φ and rearranged PROB. Updated CNST following CODATA 2018. Renamed the angular conversions. Changed the composing and cutting functionality of [CC] . Refined exiting short integer input. Expanded App. D. Specified maximum size of long integers. Changed keyboard adding \times, moving CPX, FIN, RBR, RT, and SHOW, removing %. Renamed VANGLE to V \times . Modified CPX, MATX, TRI, and X.FN. Rearranged Section 1 of the OM. Added some internal data types to App. B; reduced the range of long integer results and DP real inputs to 10^{-999} . Defined the domains of e^x-1 , IDIVR, LN(1+x), MOD, and RMD according to the HP-42S; modified PLOT and $\Sigma+$. Refined the Addressing Tables. Added a data type matrix for IDIVR. Refined the Special Results in App. B.
0.13	30.11.19	Expanded the alpha keyboard and App. I. Modified CPX, INTS, MODE, PROB, STK, TEST, $\alpha \bullet$, SHOW, and STATUS. Refined the sorting order of items, ALL, CX \rightarrow RE, MEM?, RE \rightarrow CX, RBR, RM, SLVQ, and U \rightarrow . Started filling App. F and G. Refined App. 2. Added a long integer example, CPXR?, LZ?, Δv_{CS} , conversions of hectares, and a proposal for system status information.

Date	Release notes
0.14 7.3.20	Introduced system flags for status information. Split I/O. Added CATALOG/SYS.FL, PRINT, PROG, RANI#, VAR, auxiliary constants, some predefined variables, and an index in App. I. Changed keyboard swapping MODE and FLAGS, U→ and 4→, moving CPX, FILL, RBR, R↑, USER, a.FN, aINTL, √x, and ∑, displaying PRINT, RMD, STATUS, x², and '·, and removing c/d, ∑x, →SP, and →DP. Renamed DISP to DSP and SUMS to Σ, changed 5 to 5. Refined the addressing tables and catalog access, a b/c, ADV, BATT?, BITS, CATALOG'CHARS and 'MENUS, CLALL, CLFALL, CPX, EXP, GAP, INTS, I/O, MODE, NEIGHB, PARTS, PRIME?, P.FN, SHOW, STAT, STK, X.FN, aINTL, and a•. Deleted all 16-digit (i.e. SP) data types as well as A...Z and the commands CLK12, CLK24, CPXi, CPXj, CPXRES, CPXR?, DBL?, DENANY, DENFAC, DENFIX, ENGOVR, FAST, IMPFRC, LZOFF, LZON, LZ?, MULTx, MULT-, POLAR, PROFRC, QUIET, RDX., RDX,, REALRE, RECT, SCIOVR, SLOW, SSIZE4, SSIZE8, →DP, and →SP. Corrected.
0.15 11.3.20	Changed keyboard shifting N, O, P, and Q, swapping ? and Z, moving FLAGS, RTN, and ∑, and adding MOD.

INDEX

This index lists special terms and keywords used in this manual. Furthermore, it points to the most prominent of the 167 examples included, 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.

42 (ex.) 131
about to die (ex.) 189
account balancing (ex.) 31
address space 52
ADM 121
advertising pictures (ex.) 93
AIM 187
air pressure and altitude (ex.) 84
aircraft navigation (ex.) 126
altimeter (ex.) 84
archery statistics (ex.) 97
ASSIGN 274
automatic stack lift 35
balloon trip (ex.) 82
bicycle gearing (ex.) 83
black or white cats (ex.) 190
bubble sort (ex.) 212
cannery (ex.) 199
carry 136
CDF 95
chain calculation (ex.) 34
chi-square statistic (ex.) 104
cleaning a veranda (ex.) 271
closing numeric input 24
complex aircraft navigation (ex.)
 153
compound interest (ex.) 47

confidence limits (ex.) 105
connecting peaks (ex.) 92
continuous distribution 95
cross product (ex.) 155, 171
cubic inches (ex.) 271
current step 198
data type 67
dating (ex.) 184
debugging 206
dice from *Las Vegas* (ex.) 104
diffraction pattern (ex.) 252
digit group separation 77
discrete distribution 95
dyadic functions 31
earthquakes (ex.) 87
electron-volts (ex.) 267
enter exponent 24
ENTER 33
error probability 95
extrapolation (ex.) 103
falling around the earth (ex.)
 220
falling in *Pisa* (ex.) 100
falling with drag (ex.) 218
fencing land (ex.) 19
FILL 38
filling tires (ex.) 271

forecasting (ex.) 103
free fall (ex.) 100
Fukushima accident (ex.) 89
general purpose registers 54
Golden Bow (ex.) 97
great circle distance (ex.) 123
Horner scheme (ex.) 48
improper fraction 147
indirect addressing 60, 61
interpolation (ex.) 102
ISM 133
item 26
long integers 131
Mach number (ex.) 44
markup and margin (ex.) 118
measuring capability (ex.) 111
measuring system analysis (ex.)
 110
menu 26
menu section 27
menu view 26
monadic functions 30
Mother's Kitchen (ex.) 199
Mt. Everest (ex.) 85
MyMenu 279
Mya 284
navigating in space (ex.) 128
overflow 134
parachutist (ex.) 218
PDF 95
PMF 95
prefix 18
primary function 17
process capability 98
program pointer 198
proper fraction 147

proton in magnetic field (ex.)
 171
 $R\downarrow, R\uparrow$ 37
radix mark setting 77
Rigel Centaurus (ex.) 49
RPN 31
satellite orbits (ex.) 220
scrap rate (ex.) 105
secondary function 17
short integers 132
significant change (ex.) 112f
significant improvement (ex.) 98
Sirius (ex.) 49
skiing (ex.) 81
skydiving (ex.) 218
softkey 27, 28
sorting numbers (ex.) 212
special registers 53
squaring circles (ex.) 80
stack 32
stack overflow 43
status bar 73
submenu 27
surfaces of *Jupiter's* moons (ex.)
 21
tax deduction (ex.) 119
temporary information 67
Tower of Pisa (ex.) 100
triadic functions 36
Upper Lagunia (ex.) 92
user flags 54
variables 55
vector operations in 2D 173
VIEW 58
virtual keyboard 56
Willie's Widget Works (ex.) 93