Hypatia

Open Source
Programmable Text-based RPN Calculator
for Windows

Version 4.0
www.hypatia-rpn.net

Preface

Hypatia is different. The two most obvious ways in which it is different are:

- Hypatia uses RPN (Reverse Polish Notation) — instead of `2 + 3 =` you have to type `2 3 +`
- Hypatia is a text-based console program. There is no graphic interface, and you need to use the keyboard instead of the mouse.

As unfamiliar as these features may seem at first glance, they do have their advantages. After a little getting used to them (and to Hypatia's idiosyncratic terminology), you will find Hypatia not only to be well suited for complex calculations, but also to be fast and easy to use for simple everyday tasks.

You may want to use Hypatia because of the ways it is different — RPN may appeal to your sense of logic, and you may appreciate apps that are puristic, portable, small and fast — or you may be willing to put up with Hypatia's peculiarities because of some of the features it offers, which you will not easily find anywhere else.

With variables, scripts, prompts, user defined elements, count- and condition controlled loops and versatile If/Then/Else clauses Hypatia offers elements of a programming language. While these features make Hypatia a powerful tool, you can use Hypatia for a lot of things without ever bothering about more than the basics.

Skim through this documentation, then read the parts that are interesting to you. It is perfectly fine if you need only a small part of Hypatia's features, you can safely ignore the rest — but if you ever need more, Hypatia may be able to provide it.

What is New?

(For a detailed list of updates see "Release Notes", page 80).

Version 4.0

Hypatia is now a 64-bit program, requiring a 64-bit Windows system — this allows integer numbers to have up to 18 digits, and floating point numbers to be shown with up to 15 digits.

New operators: ISPOSINT, DIGITS, LOG^ and IMOD, new oputput format commands: FSHORT / FLONG and F' ON / F' OFF, and new commands: EDINI, BUFFER DISCARD and INTEGER ON / OFF. Particular effort has gone into tackling rounding issues and making the results of Hypatia's floating point calculations accurate to the 15th digit (18 digits for integers).

The operator LOG (for natural logarithm) has been renamed LN, to avoid ambiguities.

Versions 3.x

The most important additions in versions 3.0 to 3.4 were "user defined elements" as a versatile alternative to insert files, buffer mode to drastically increase the speed of loops with thousands of write-to-disk operations, the compare operators >>, >=, << and <=, the ability of the ITEM operator to accept negative index values (counting backwards through the list of items), the TIME pseudo constant to measure the running time of loops, and the lakh, million and crore formats for large numbers.

Table of Contents

Introduction

Hypatia is a powerful programmable calculator for simple and complex calculations that uses its own enhanced version of RPN (Reverse Polish Notation).

Hypatia is a Windows console program — it can be started from the Windows desktop or from the Windows command line prompt. You should have a basic understanding of folders and files to install Hypatia, and a basic ability to use a text editor to make use of Hypatia's more advanced features. You do not need to be familiar with the Windows command line, but if you are, there are some features of Hypatia which you may find particularly useful.

Hypatia requires a 64-bit Windows system.

Hypatia's main features are:
- It uses a greatly expanded version of RPN, offering a wide range of built-in functions.
- It allows the use, storage and retrieval of variables.
- It knows decimal, hexadecimal and binary numbers.
- It can read data from files and write multiple results to files.
- You can easily define your own functions and routines.
- Integer numbers can have up to 18 digits, floating point numbers up to 15 significant digits.
- Its strictly text-based approach lets you scroll through past inputs, edit and re-use them, log inputs and results, etc. Your own functions and scripts use exactly the same syntax as your calculations and seamlessly integrate with them.
- With count- and condition controlled loops and If/Then/Else clauses you can do iterative calculations, compute generalized Fibonacci sequences, perform Monte Carlo experiments, etc.

Hypatia has only a very limited understanding of one-dimensional arrays, does not know matrixes and vectors, and does not know imaginary or complex numbers. While Hypatia can easily calculate arithmetic, geometric and harmonic means, medians and standard deviation, it is not a tool for statistical analysis. Hypatia's loop commands and scripts cannot be nested. There is no syntax checking while typing.

Hypatia is written in the Phix language — http://phix.x10.mx/ — which is based on EUPHORIA.

The name "Hypatia" is a tribute to the teacher, philosopher, astronomer and mathematician Hypatia of Alexandria. For details about her, see http://en.wikipedia.org/wiki/Hypatia

The current version is 4.0, from April 2, 2024. Keep checking www.hypatia-rpn.net for updates.
Regarding Linux, see "Hypatia for Linux", page 75.

Disclaimer: I've done what I can to make this program work correct and reliably, but always bear in mind that computer programs are fallible. If much depends on the results, please double-check them.

Feedback is essential. Please report all bugs, and please do not hesitate to ask questions or make suggestions, regarding the program or the documentation.

Contact: robert@drs.at — please include "Hypatia" in the subject line. If you don't receive a reply, I'm either not able to write, or my mail has landed in your spam folder.

Installation

This is quickly done — all you need is a basic understanding of folders and files, and of how to use Windows File Explorer (or a better file manager, such as Total Commander).

Create a new folder, wherever you want to have it (this can also be in your documents folder), and unpack hypatia.zip to that folder. Delete the \source folder, unless you're interested in it, though there is no harm in keeping it.

Hypatia will create an empty file hy.ini in its program folder, and the files hy and hyin. All other files that you may tell Hypatia to read or write will, by default, be located in that folder too.

To update Hypatia, simply overwrite the files in your Hypatia folder with the new versions.

Installation for the Windows Desktop

Create a shortcut on the desktop: in your file manager right-click hy.exe, go to "Send to," click "Desktop (create shortcut)," and then rename the icon "Hypatia." (Depending on your Windows version, details may differ.)

To define a shortcut key combination, for instance Ctrl+Alt+H, right-click the icon, open "Properties," click the "Shortcut" tab, then into the "Shortcut key" field, and press the desired key combination. You can now use this key combination to open Hypatia.

You can customize Hypatia's console window — size, font, font size, background color, text color, etc. Right-click Hyptia's icon on your desktop, and click "Properties" to edit them. "Layout" lets you define the size of Hypatia's console window, "Font" lets you choose a font and its size, "Colors" lets you set background and text colors. "Options" lets you deal with some advanced features, though you will probably want to leave them at their default states.

Depending on the version of Windows, and which console features are enabled, you can resize Hypatia's console window with the mouse at any time (see page 12).

Installation for the Windows Command Line

Add the program folder to the system path. (If you're familiar with the command line, you know how, if not, you do not need this.)

Running Hypatia from the command line allows you to start Hypatia with command line options, and allows you to use Hypatia for quick one-line calculations (see chapter "Command line calculation mode", page 74).

Uninstall Hypatia

Since Hypatia never touches the registry, there is no uninstall process. If you do not need Hypatia anymore, simply delete the program folder with hy.exe and any other Hypatia-related files, and delete the desktop icon. If you have added the program folder to the Windows system path, remove it.

Hypatia's RPN (Reverse Polish Notation)

Hypatia uses an enhanced version of RPN. RPN is a form to write mathematical expressions that is an alternative to the much better known infix notation. Admittedly it's a matter of taste, but while RPN can seem a bit confusing at first sight it follows a more concise logic, and it does have advantages for simple as well as for complex calculations.

RPN does not use, or need, parentheses, it only knows arguments and operators. It is strictly written from left to right, with all arguments and operators being separated by spaces.

Operators are written after their arguments. An operator performs a calculation upon one or more argument(s) which it finds to its left, and returns a single value. The operator "knows" how many arguments it has to process (usually either one or two). After the operator has performed its calculation, it replaces the entire expression (arguments and itself) with the result, which can then become an argument for the next operator.

Example: The + operator takes two arguments, and returns their sum. So, to perform an addition, you write the two arguments, and the + sign after them (always write spaces between the elements of an expression):

```
5 4 +
```

The result, 9, will replace those three elements. This can be the end of the calculation, or it can become an argument for further calculations:

```
5 4 + SQRT
```

SQRT, square root, is an operator that takes one argument. 5 and 4 are added up, making 9, then the operation "square root" is performed upon that result, so the final result is 3.

Conventional infix notation relies upon operator hierarchies and upon parentheses — RPN knows neither, but relies exclusively upon the order in which arguments and operators are written:

```
5 4 SQRT +
```

In this example, the first operator that is encountered, reading from the left, is SQRT. It takes one argument, 4, and replaces it (and itself) with the square root, 2. Then, reading on, the operator + is encountered. It looks for two arguments to its left, and finds 5, still untouched by any previous operation, and 2, which now stands where "4 SQRT" had stood. The final result, therefore, is 7.

Another example:

```
2 3 + 5 7 + *
```

Here, the first + operator adds 2 and 3, returning the result 5. Then the next + operator adds 5 and 7, returning 12. The * operator, which is encountered next, finds those two results to its left, and therefore multiplies 5 and 12, returning the result 60.

For some operators that take two arguments the order of those arguments is significant: it makes a difference whether you divide 5 by 2, or 2 by 5. For those operators you have to know the meaning of the argument order, but usually it will be rather obvious:

`5 2 /` means 5 divided by 2
`2 5 ^` means 2 to the 5th power

If you are not familiar with RPN it may take a while to get used to it, but by now you have already learned all the rules there are!

When writing, or debugging, an RPN expression, always have this in mind:

- The expression must start with an argument (which can be a number, a variable, or a constant).
- The expression, unless it simply consists of one argument (in which case the argument is also the result), must end with an operator.
- When all operations have been performed, which, from left to right, one after the other replace one or more argument(s) and their operator with the result of that operation, then exactly one argument must remain, which is the final result.

Hypatia's Enhanced RPN

Compared to standard RPN, Hypatia not only offers a large number of additional 1- and 2-argument operators, it also knows n-argument operators (or statistical operators), which take a variable number of arguments: all the arguments they find to their left.

(Since expressions are read from left to right, and all operations are immediately executed, left of any operator, when it is encountered, can only be arguments, no other operators.)

Example:

```
3 7 5 1 4 SUM
```

Since n-argument-operators can count the number of their arguments, they can calculate statistical parameters:

```
3 7 5 1 4 MEAN
```

gives the mean value 4, calculated by dividing the sum of the arguments by their number.

N-argument-operators can be applied to data read from a file, with the parentheses file inclusion syntax (not to be confused with the parentheses of infix notation):

```
(mydata) SUM
```

By default n-argument operators take everything to their left as their arguments, but you can use the vertical bar | as a delimiter — anything to its left will not be seen by the n-argument operator:

```
100 | 3 7 5 1 4 SUM -
```

calculates the sum of 3, 7, 5, 1 and 4, and subtracts it from 100.

More about all this in the appropriate chapters!

How to Use Hypatia

You can start Hypatia from the Windows desktop like any other program. If Hypatia's program folder is included in the system path, you can start Hypatia from the Windows command line with `hy` or from the Windows PowerShell with `hy.exe` (see also chapter "Command Line Options", page 73).

When you start Hypatia, it greets you with a question mark as the input prompt. Your input will be processed after you have pressed the ENTER key.

The length of the input line is limited by the width of the screen. Only standard ASCII characters (character codes < 128) can be written. Except in comment lines, all input will be converted to lower case characters.

There are three kinds of input lines:
- Calculations
- Commands
- Comments

To exit Hypatia, use any of the commands Q, QUIT or EXIT, or close the console window.

The question marks and equal signs at the beginnings of lines in this documentation are displayed by Hypatia as input prompts and to indicate calculation results, do not type them.

Calculations

A line that starts with a number, variable or constant is an expression that Hypatia will try to calculate.

Hypatia will display the result in the next line, preceded by an equal sign, and prompt for the next input line. This result will also be written into the file hy in the program folder (see page 32).

You can immediately use the result of a calculation for your next calculation:

```
? 3 4 +
= 7
? 2 *
= 14
```

(more about this in chapter "Chain Calculations", page 19)

Numbers can be written in several formats: decimal, hexadecimal, or binary — see chapter "Number Formats", page 26. Internally, numbers are stored and processed independent of their input format.

Commands

A command line in Hypatia (not to be confused with the operating system's command line) is a line that starts with a command. Hypatia considers everything at the beginning of the line as a command that isn't either a number, a constant, a variable, or a user defined element. If Hypatia doesn't recognize it, you will get an error message. For a list of Hypatia's commands, see pages 68 to 70.

Assigning values to variables with $myvar = and assigning content to user defined elements with @ude = are commands, too, even though they begin with variable or UDE names (pages 21 and 40).

Comments

Any line that begins with # is a comment line. When you enter a comment line, nothing happens — only if "logging" is on (see page 33), then the comment line will be written to the log file.

You can add comment lines to files that you use as script or insert files (see pages 34 and 35). Comment lines in script files will be displayed, but not in loops, or when they begin with ##.

In loops, comment lines that begin with #: will always be displayed — that is, at each pass. Comments following I1:, I*:, IF … THEN, ALSO: or ELSE: will be displayed when that condition is met (all this will be explained later, in the context of scripts and loops).

You can add a comment at the end of any command or calculation line after ##, which is useful for documentation purposes in script files. Note that in this case ## needs to be preceded by a space.

Editing the Input Line

- ARROW LEFT, ARROW RIGHT, HOME and END keys move the cursor
- BACKSPACE deletes the character to the left of the cursor position
- DELETE deletes the character at the cursor position
- ENTER inputs the line as you see it, regardless of the cursor position

When the cursor is at the beginning of the line:
- ARROW UP and ARROW DOWN scroll through previous input lines, you can edit and reuse them

ESC (like HOME) moves the cursor to the beginning of the line. If you have scrolled up, pressing ESC twice, or pressing ESC at the beginning of the line, returns you to the current line.

Input is always in insert mode. The length of the input line is limited by the width of the program window. Comments apart, all input will be converted to lower case characters.

Numbers

You can write numbers in different formats (note that all letters are case insensitive):

`12345`	numbers may not contain spaces or commas …
`12'345'600`	… but you can use apostrophes ' to structure large numbers
`123.45`	decimal point (decimal commas are not allowed)
`.12`	leading 0 is not required, but you could also write 0.12
`-.12`	a leading minus sign makes any number negative (leading + is allowed)
`1.2e6`	scientific notation (but also 12e5, for instance, is allowed)
`1.2e-6`	scientific notation, exponent can be negative, must be in the range of ±1000
`1.2 lakh`	number is multiplied by one hundred thousand (1e5), space is required!
`1.2 million`	number is multiplied by 1 million (1e6), space is required!
`1.2 crore`	number is multiplied by 10 million (1e7), space is required!
`&hA0FF`	hexadecimal numbers are preceded by &h (apostrophes are allowed)
`&b1001'0010`	binary numbers are preceded by &b (apostrophes are allowed)

About how Hypatia can display results, see chapter "Number Formats" (page 26).

Integer numbers can have up to 18 digits (± 1e18 minus 1). Floating point (non-integer) numbers are shown with up to 15 significant digits.

The permtted range of (positive or negative) numbers is from 1e1000 to 1e-1000. Any number larger than ± 1e1000 will produce an error, any number less than ± 1e-1000 is zero.

Files

By default, all files that Hypatia reads or writes are located in the program folder.

Some rules apply to Hypatia's file names: they cannot contain spaces (with the exception of "insert files"), they cannot contain parentheses, they cannot use characters with ASCII codes above 127, and they are always written in lower case.

With the exception of hy.ini, Hypatia's files do not need to have file extensions, and do not have them by default, though you can give them any extensions you want — files can, for instance, have the names myfile, myfile.txt, my.data, my.script, etc.

Three files are essential for Hypatia's operations: hy, hyin, and hy.ini.

hy — the file that Hypatia writes results to (see page 32)
you can copy hy to the clipboard with the command COPY

hy.ini — the script file that gets executed when Hypatia starts (see page 34)
Hypatia creates an empty hy.ini file, you can add your own command lines

hyin — has a copy of the most recent calculation line, and some command lines (see page 44)
you can copy hyin to the clipboard with the command COPIN

The optional log file hy.log gets written by Hypatia when you turn log mode on (see page 33)

Other files can be written by Hypatia with the SAVE commands (SAVE, SAVE@ and BUFFER SAVE), can be executed as scripts with the RUN command, or can contain data and be read as "insert" files.

All of Hypatia's files are in plain text format, and can be opened, viewed and edited with any text editor — Hypatia offers the EDIT command that (by default) calls Windows Notepad.

EDIT without file name is short for EDIT hy, EDIN is short for EDIT hyin, EDINI is short for EDIT hy.ini.

More about all this in the chapter "Input and Output Files" (page 30).

Changing the Console Window Size

When you change the size of the console window with the mouse while you use Hypatia, Hypatia will only become aware of the new size the next time it waits for an input — the current input line will assume an incorrect maximum line length. It is a good idea, therefore, to change the window size when the input line is empty, and then press ENTER to let Hypatia adjust to the changed screen width.

Operators

All operators are written after their argument(s). Operators “know” how many arguments they have — in Hypatia, there are 1-, 2- and n-argument operators.

N-argument operators consider everything to their left as their arguments, unless their range is restricted by the delimiter | (see “N-Argument Delimiter |”, page 18).

“Measurement unit conversion operators” work like all other operators; that they begin with : is only a naming convention. With (currently) one exception, conversion operators are 1-argument operators.

When the first element of a calculation line is a 1- or 2-argument operator, and it is short of one argument, it uses the previous calculation result as its (first) argument — see “Chain Calculations”, page 19. At the start of Hypatia or after the RESET command, the previous calculation result is 0.

The line between arithmetic and unit conversion operators is a bit blurred. If you ask why RAD and DEG are arithmetic operators, but :MSDEC is a unit conversion operator, I do not have a good answer.

Note that all input is case insensitive, you can write sqrt instead of SQRT.

A “numerical expression” is an RPN expression, a constant, or a variable.

Note that calculations are processed from left to right, with each operator replacing itself and its arguments with its result — so, by the time an operator is reached, all elements to its left, and therefore all its arguments, are numbers.

Hypatia knows the following operators — for details, see the following pages:

1-argument operators:

```
+- RCP ABS SIGN IS0 ISNOT0 IS+ IS0+ IS- IS0- ISPOSINT ISPRIME DIGITS
INT FRAC SQ SQRT CUBE CBRT ! LAKH MILLION CRORE
LN LOG10 LOG2 EXP EXP10 EXP2 RAD DEG SIN COS TAN ASIN ACOS ATAN
```

1-argument unit conversion operators:

```
:F :C :MI :KM :NAUTMI :NAUTKM :IN :CM :FT :M
:LB :KG :OZ :G :TOZ :TG :GAL :L :MPG
```

2-argument operators:

```
+ - * / // \ ^ LOG^ EQUAL UNEQUAL >> >= <= << % %+ %- %? %n %a
ROUND ROUNDS GATE UPLIMIT LOLIMIT MOD IMOD BIN RANDINT RANDND
```

2-argument unit conversion operators:

```
:MSDEC
```

n-argument (statistical) operators:

```
N N+ N0 N- SUM MEAN GMEAN HMEAN MED SDEV SQSUM RCPSUM PROD MIN MAX ITEM
```

Pseudo operators (see page 59):

```
WHISK A B DONE |
```

See also constants and pseudo-constants (page 20):

```
PI E PHI RAND DUP I ISLOOP TIME
```

1-Argument Operators

Syntax: `a operator`, where a is a number or a numerical expression

+-	changes algebraic sign
RCP	gives reciprocal value
ABS	absolute value
SIGN	1 if argument is positive, 0 if zero, -1 if negative
IS0	1 if argument is zero, 0 if not zero (positive or negative)
ISNOT0	1 if argument is positive or negative, 0 if zero
IS+	1 if argument is positive, 0 if zero or negative
IS0+	1 if argument is positive or zero, 0 if negative (you can also write IS+0)
IS-	1 if argument is negative, 0 if positive or zero
IS0-	1 if argument is negative or zero, 0 if positive (you can also write IS-0)
ISPOSINT	1 if argument is positive integer, 0 if not (error if argument ≥ 1e18)
ISPRIME	1 if argument is prime number, 0 if not (error if argument not 0 or positive integer)
DIGITS	number of digits before decimal point, if negative: number of zeros after dec. point
INT	truncated integer value (for rounding to integer use 0 ROUND, error if argument ≥ 1e18)
FRAC	fractional part of a decimal number (rounded to total of 18 digits, error if arg. ≥ 1e15)
SQ	square
SQRT	square root
CUBE	cube
CBRT	cube root (argument can be negative)
LN	natural logarithm (base e), see 2-argument operator LOG^ for logarithm base b
LOG10	logarithm base 10
LOG2	logarithm base 2
EXP	e to the power of argument
EXP10	10 to the power of argument
EXP2	2 to the power of argument
!	factorial (argument must be 0 or positive integer, max. 449, rounded for arguments > 19)
RAD	converts angle from degrees to radians (do not confuse with USE RAD command)
DEG	converts angle from radians to degrees (do not confuse with USE DEG command)
SIN	sine (results less than ± 1e-18 are rounded to zero)
COS	cosine (results less than ± 1e-18 are rounded to zero)
TAN	tangent (abs. value of argument must be less than π/2 or 90 degrees)
ASIN	arcsine
ACOS	arccosine
ATAN	arctangent

By default, all angles are in radians.
You can change this to degrees by using the USE DEG command, or the -d command line option.

SIGN and the first six IS operators observe the zero threshold, by default ± 1e-16, see page 23).

FRAC rounds its result to a total of 18 digits of the argument because more decimal digits would be random, and then down to 0.999999999999999 (15 digits) if it is larger, to avoid the result 1.

Large Number Units

LAKH	multiplies value by one hundred thousand (Indian numbering system)
MILLION	multiplies value by one million
CRORE	multiplies value by ten million (Indian numbering system)

Note that LAKH, MILLION and CRORE are 1-argument operators:
`2 1 MILLION +` is 1000002, `2 1 + MILLION` is 3000000.

LAKH, MILLION and CRORE differ from other 1-argument operators in two ways:
- you can use them in MAXLOOP n, DO n and REPEAT n commands
- you can use them after numbers at the beginning of lines in chain calculations (see page 19)

1-Argument Unit Conversion Operators

:F	degrees Celsius to Fahrenheit
:C	degrees Fahrenheit to Celsius
:MI	km to international miles
:KM	international miles to km
:NAUTMI	km to nautical miles, or km/h to knots
:NAUTKM	nautical miles to km, or knots to km/h
:IN	cm to inches
:CM	inches to cm
:FT	meter to feet (1 ft = 12 in)
:M	feet to meter (1 ft = 12 in)
:LB	kg to avoirdupois pounds
:KG	avoirdupois pounds to kg
:OZ	g to avoirdupois ounces (1 lb = 16 oz)
:G	avoirdupois ounces to g (1 lb = 16 oz)
:TOZ	g to troy ounces
:TG	troy ounces to g
:GAL	liters to US liquid gallons
:L	US liquid gallons to liters
:MPG	miles per gallon to liters per 100km, and vice versa

2-Argument Unit Conversion Operator

:MSDEC	minutes and seconds to decimal hours or degrees

2-Argument Operators

Syntax: `a b operator`, where a and b are numbers or numerical expressions

+	addition, a + b
-	subtraction, a - b
*	multiplication, a * b
/	division, a / b
//	(a - b) / b, a if b = 0, set to 0 if within ± 1e-16 (this is useful as a loop exit condition)
\	integer division, a \ b (absolute value of result is rounded off to the next integer)
^	power, a ^ b
LOG^	logarithm of a base b (a must be positive, b must be > 1)
EQUAL	(or ==) 1 if a and b are equal, 0 if they differ
UNEQUAL	(or <>) 1 if a and b differ, 0 if they are equal
<<	1 if a is less than b, 0 if greater or equal
<=	1 if a is less than b or equal, 0 if greater
>>	1 if a is greater than b, 0 if less or equal
>=	1 if a is greater than b or equal, 0 if less
MULTIPLE	1 if a is positive multiple of b, a and b must be integer, 0 if not the same sign
ROUND	round a to b digits after decimal point; if b is 0, round to integer if b is negative, then round to b zeros before decimal point (b can be -9 to 15)
ROUNDS	round a to b significant digits (b can be 1 to 17)
GATE	zero if abs(a) is less than b, otherwise a (if threshold value b is 0, it is seen as 1e-16)
UPLIMIT	b is upper limit for a (result is a, but not higher than b)
LOLIMIT	b is lower limit for a (result is a, but not lower than b)
%	percent, b% of a
%+	percent +, a increased by b percent
%-	percent -, a diminished by b percent
%?	how many percent of a is b?
%N	net: if a is net amount plus b percent, what is the net amount?
%A	agio: if a is gross amount including b percent, what is the agio amount?
MOD	remainder, a modulo b (a and b must have the same sign)
IMOD	remainder, a modulo b (a and b must be integer and have the same sign)
BIN	binomial coefficient (a must be positive integer, b must be 0 or positive integer)
RANDINT	random integer, within the range of a to b (do not confuse with RAND pseudo constant)
RANDND	normal distributed random number, a is mean, b = standard deviation 0 0 = standard normal distribution, same as 0 1

By default, the compare operators (EQUAL or ==, UNEQUAL or <>, <<, <=, >> and >=) consider differences of less than ± 1e-16 to be zero. You can change or disable this threshold, see “Zero or Not Zero”, page 23.

Do not confuse the compare operator == (same as EQUAL) with the == command!

n-Argument Operators

Syntax: `a1 a2 a3 … an operator`, where a_1 to a_n are numbers or numerical expressions.

n-Argument operators always consider everything to their left as their arguments, so, the input line starts with a_1, or a_1 follows the delimiter | (see page 18).

N	number of arguments
N+	number of positive values
N0	number of zeros
N-	number of negative values
SUM	sum
MEAN	mean
GMEAN	geometric mean (all arguments must be > 0)
HMEAN	harmonic mean (all arguments must be > 0)
MED	median
SDEV	standard deviation
SQSUM	sum of squares
RCPSUM	sum of reciprocals
PROD	product
MIN	minimum
MAX	maximum
i ITEM	item i in argument list (see “Special n-Argument Operator ITEM” below)

By default, N+, N0 and N- operators consider arguments within ± 1e-16 to be zero (you can change or disable this threshold, see “Zero or Not Zero”, page 23).

N-argument-operators (also called statistical operators) need at least 1 argument, except for N, which can have 0 or more, and SDEV and ITEM, which need at least 2.

The “insert” function ( ) allows statistical operators to read data from files (see chapter “Insert Files”, page 35).

N-argument operators can be useful with only a few arguments, too. For instance, if you want to calculate the hypotenuse of a rectangular triangle with the famous “square root of a square plus b square” formula, then instead of writing `6 SQ 8 SQ + SQRT` you can write `6 8 SQSUM SQRT`.

As is true for all operators, arguments do not have to be numbers, they can be numerical expressions. `2 3 * 8 SQSUM SQRT` is 10, because when the SQSUM operator is reached, the expression `2 3 *` has already been replaced with its result 6.

Special n-Argument Operator ITEM

a_1 a_2 a_3 ... a_n i ITEM returns the i^{th} item in the list of arguments. ITEM is meant to be used in loops. The index i can be a number (which will rarely be useful), a variable or a numerical expression, but usually it will be the loop index I, or will be calculated from it.

If the ITEM index value i is negative, items in the list will be counted backwards.

i must be an integer in the range from 1 to n or from -1 to -n. Note that the the number of arguments that ITEM processes (removes from the stack and replaces them with the result) is n plus 1, that is, the data items in the list plus the index of the referenced item.

Referencing the last item in the list (or the first item, when the index is negative) exits the loop after completing the script, see “Defining an Exit Condition” (page 49).

Do not confuse the ITEM operator’s index argument (the number of the item in the list) with the loop index (see “Pseudo Constants”, page 20, and “Loops and the Loop Index”, page 48), even though the loop index I may often be used for the ITEM index argument.

To process the items of the argument list in reverse order, you can use the expression I +- for a negative ITEM index.

For examples of how to use ITEM see chapter “Loops, Data Lists, and the ITEM Operator” (page 56).

n-Argument Delimiter |

While n-argument (statistical) operators want to take anything to their left as arguments, you can use the vertical bar | as a delimiter, which hides anything to its left from the next n-argument operator. As a simple example, you can subtract a sum of several values from an initial value.

```
? 100 | 17 33 20 SUM -
= 30
```

The delimiter works with all n-argument operators. There are no restrictions to what stands left of the delimiter — it remains hidden until an n-argument operator is encountered, and then becomes visible again. There can be more than one delimiter in a calculation line, but each delimiter must be followed by a corresponding n-argument operator.

Pseudo Operators WHISK and DONE

These are special features that will be discussed later. They are called “pseudo operators” because like operators they manipulate the stack, but do not perform any calculations.

WHISK (see page 41) removes the two values left of it from the stack, and stores them under the names A and B, which can then be used in the calculation line.

DONE (see page 58) preserves the value immediately to its left, but deletes all prior values from the stack.

Chain Calculations

After each completed calculation — that is, when a calculation line was processed without an error — the symbol $ represents its result, and you can use it in the subsequent calculation. For instance:

```
? 3 5 +
= 8
? 2 $ ^
= 256
```

$ is a “special variable” — this will be discussed below (see “Variables”, page 21, and “Result File hy and Result Variable $”, page 32).

In some cases, though, you can use the previous result for the next calculation without the help of $: when the *first* operator in a calculation line is a 1- or 2-argument operator, and it is short of one argument, then it automatically includes the result of the previous calculation as its (first) argument. This sounds more complicated than it is, as the following example shows:

```
? 2 8 *
= 16
? SQRT              the same as  $ SQRT          (the previous result was 16)
= 4
? 3 * 2 /           the same as  $ 3 * 2 /       (the previous result was 4)
= 6
```

And so on …

You can control this behavior with the commands AUTO$ ON (default) and AUTO$ OFF. Disabling the “auto include $” mode forces you to explicitly use $ for all chain calculations.

Note that this does not apply to n-argument operators, which have no way to decide whether an argument is missing or not, and never automatically include the previous result. If you want to include the previous result, you have to use $.

The operators LAKH, MILLION and CRORE can be used with auto including of $:

```
? 3
= 3
? MILLION
= 3000000
? 2 MILLION +
= 5000000
```

The last but one result, by the way, is represented by $$ — like $ you can use it in a calculation line any way you want to, but it will never be automatically included.

If you need to preserve a calculation result for more than the next calculation, you have to assign it to a variable — see the next but one chapter “Variables”.

Constants

You can use constants the same way you use numbers. Hypatia currently has four built-in constants:

PI	3.141592653589793238
TAU	τ = 2π = 6.28318530717958648
E	2.718281828459045235
PHI	1.618033988749894848 (the "golden ratio")

Pseudo Constants

These aren't actually constants, but in Hypatia's syntax they look like constants.

RAND	Random decimal number in the range of 0 to 1 (do not confuse this with RANDINT and RANDND, which are 2-argument operators)
DUP	Duplicates the preceding number in the input line (or equals $ if there isn't any) Constants and variables are treated by DUP like numbers
I	in DO loops: loop index, in REPEAT n loops: loop index + 1, outside of loops always 1 see "The Loop Command REPEAT n", page 47
ISLOOP	in DO and REPEAT loops 1, outside of loops (when a script is called directly) 0
TIME	in DO and REPEAT loops the time since start in seconds, outside of loops always 0

The DUP pseudo constant is useful for calculations where the same number appears in the formula twice. For instance, in infix notation $a^2 + a$ — here is how you can write this in Hypatia's syntax:

```
? 4.1825 SQ DUP +
= 21.67580625
```

Note that the pseudo constant DUP duplicates the number 4.1825, not, as you would expect from an operator, the result of 4.1825 SQ. (Compare pseudo operators WHISK, A and B, page 41). DUP can be used several times in one calculation line, duplicating the same or different numbers.

User Constants

Hypatia offers two ways in which you can use your own constants. For instance, if you need to convert between metric horse power (PS in German) and kW you could write the number 0.7355 into a file which you can give the name hp (no extension needed) — this file has to be in Hypatia's program folder. (For creating and editing files, you can use the EDIT command. For inserting data using parentheses see chapter "Insert Files", page 35.)

You can then convert 160 hp(M) to kW by writing `160 (hp) *` or convert 120 kW to hp(M) by writing `120 (hp) /`

Or you can define a variable (see below) with the name $hp and assign it its value by adding this line to the file hy.ini in Hypatia's program folder (do not get confused by the fact that your constant is a variable, see the following chapter "Variables"):

`$hp = 0.7355` (spaces before and after = are required!)

You can then convert 160 hp(M) to kW by writing `160 $hp *`

Variables

You can use variables the same way you use numbers or constants, as elements in a calculation line. All variable names begin with $. The names must not contain spaces or parentheses.

Hypatia uses two special variables, $ and $$ — $ is the result of the previous calculation, $$ the result of the one before. At each start of Hypatia, and after a RESET command, both have the value 0.

Any other variables are created by assigning them a value, in one of two ways (if the variable already exists, it will receive the new value):

`STO $myvar`
assigns the previous calculation result to the variable $myvar (stores it in the variable).

`$myvar =` number or calculation — for instance:
```
$myvar = 2
$myvar = 2 SQRT
$myvar = $myvar 2 *
```

`$myvar = $` does the same as `STO $myvar`

After the equal sign = you can write anything that could be a calculation line. Spaces before and after = are required!

Important: the = assign command does not give a "calculation result," it does not update $ or the result file hy (see page 32). Unlike other commands, though, it does get written to hyin (see page 44).

The variables $zero and $loop have special meanings — do not use them except for their intended purposes! (For $zero see page 23, for $loop see page 49.)

Variable Commands

STO $variable	assign the previous calculation result to a variable
$variable = …	assign value or result of calculation to a variable
PROMPT $var	prompt user for value of variable (number or calculation result)
SHOW	show all variables and their values, including $ and $$
SHOW $var1 $var2 …	show these variables
DEL $variable	delete the variable
SAVE filename	save all variables to a file (not including $ and $$) — see next page
_filename	(or RUN filename) retrieve variables that were saved with SAVE

STO $var, $var = and PROMPT $var create the variable, if it does not already exist.

In loops, the SHOW list of variables can also contain the pseudo constants I and TIME.

SHOW shows numbers with up to 15 digits, or up to 9 digits in short mode (see FSHORT, page 24). Other output format commands do not affect it.

$ and $$ cannot be used with the STO, PROMPT and DEL commands.

The PROMPT Command

The PROMPT command is meant to be used in scripts, it makes no sense outside of scripts where you can simply use `$var =` (for examples see "Centimeters, Feet and Inches" and "Script, Prompt, Insert File", pages 61 and 63).

In response to being prompted for a variable value you can enter a number, or a calculation.

Simple pressing ENTER as a response to the PROMPT command leaves the variable unchanged.

In a script called by a loop, the PROMPT command can be used with I1: to enter start values, or for instance with IF … ELSE to ask whether the loop should be aborted — this will be discussed in the respective chapters.

Neither `$var =` nor `PROMPT $var` update $ and hy.

`$var =` updates hyin if not in a script, `PROMPT $var` never does.

Saving and Retrieving Variables

Variables are lost when you exit Hypatia, unless you save them with the SAVE command:

```
SAVE filename [comment]
```

This command saves the values of all user defined variables (not $ and $$) into a script file in Hypatia's program folder (script files will be discussed later, see chapter "Script Files", page 34). No file extension is needed. The file name must not contain spaces or parentheses, and it will be converted to lower case. If a file of this name already exists, you will be asked whether you want to overwrite it. You can edit the file at any time, add or delete variables, or change their values.

Values are saved with up to 18 digits.

Anything you write after the file name will be written into the file as a comment line. For technical reasons, it gets converted to lower case along with the file name.

You can retrieve these variables at any later time:

```
_filename
```

This is short for `RUN filename` (execute the script — again, see chapter "Script Files", page 34).

You can also manually create files that contain `$var = ...` lines, as an alternative to defining your user constants in hy.ini — like variables in files created with the SAVE filename command, they will not be automatically available at program start, but only when you "run" that file.

Zero or Not Zero

Because of unavoidable rounding differences, zero is not always exactly zero. For instance, in theory `1.2 SQRT SQ 1.2 -` is 0, but the actual result is -1.084…e-19. While this is an extremely small number (with 18 zeros after the decimal point), it isn't really zero — in most cases, though, when you ask "is the result zero?", you will want to disregard such minor deviations.

This is what, by default, the SIGN and the IS operators do. By default, their "zero threshold" is ± 1e-16 (if you're not familiar with scientific notation, this is 10^{-16} or 0.0000000000000001).

If the absolute value of a is less than the zero threshold, then:

```
a IS0, IS0+, IS0-     = 1
a ISNOT0, IS+, IS-    = 0
a SIGN                = 0
```

The zero threshold is also used by the six compare operators regarding the difference between two values, and is used by the n-argument operators N+, N0 and N-.

Considerations of what is, or is not, zero are particularly relevant in the context of IF … THEN clauses (page 60). You can change the threshold or disable it by setting the variable $zero, which overrides the default 1e-16 value. Note that the IF condition is true for all values not exactly 0, regardless of the zero threshold.

`$zero = 0`	disables the threshold, only the exact value of 0 is treated as 0
`$zero = ...`	the value of the variable $zero is now the threshold (it must not be negative) absolute values *less* than that threshold are considered to be 0
`DEL $zero`	resets the threshold to the default value of 1e-16

If $zero is negative, an error will occur when an operator attempts to use it.

If you want the zero threshold to be changed or disabled by default, add the line $zero = … to hy.ini.

The following operators are *not* affected by $zero:

`a b //` (a minus b divided by b) — the result is set to 0 if its absolute value is less than 1e-16. Being a quotient, the result is independend of the order of magnitude of the numbers involved.

`a 0 GATE` — the result is 0 if the absolute value of a is less than 1e-16, but you are free to specify any threshold value instead of 0, including $zero.

`a $zero GATE` sets absolute values less than the zero threshold to 0. Unlike with the SIGN and IS operators, you cannot disable the threshold. If $zero is 0, `a $zero GATE` equals `a 0 GATE` and uses the default 1e-16 threshold. (Disabling it wouldn't make sense, it would be the same as simply not using the GATE operator.)

SIN and COS always round their results to 0 if they are less than ± 1e-18.

Independent of the zero threshold, any operator's smallest possible result is ± 1e-1000 — anything smaller is zero.

Accuracy, Digits, and “Integer Bias”

While the internal accuracy is 18 digits, by default Hypatia shows non-integer numbers with 15 significant digits (trailing zeros after the decimal point are, again by default, not shown). For output format commands see the following chapter “Number Formats” (page 26).

Integer numbers can have up to 18 digits, and by default are shown completely. 18 digits are enough to count the seconds since the beginning of the universe some 13.8 billion years ago.

Computers store and process numbers in binary format, and with non-integer numbers the necessary conversions from decimal to binary and back from binary to decimal unavoidably bring about tiny discrepancies. Also, some arithmetic operations can by their very nature never be performed with absolute precision. Hypatia takes care to resolve or work around those issues, helped by the three “spare” digits — for most practical purposes, you can trust Hypatia’s results. (See also “Rounding”, page 77.)

The command == shows you the recent result with up to 18 digits. This can be closer to its “true” internal value, but is not necessarily more correct — the restriction to 15 digits is there for a reason. The 18-digits result does not get written to hy, but it does get written to the log file if logging mode is on (page 33). The SAVE command (see page 22) always saves variables with 18 significant digits, so that the full internal accuracy is preserved when variables are retrieved.

To make it easier to enter numbers with many digits, you can freely use apostrophes to structure them: 728'625'211000. Apostrophes within numbers (also hexadecimal and binary ones) get ignored.

Numbers can be entered in scientific notation with exponents up to ± 1000. Calculation results, including intermediate results, must not exceed ± 1e1000.

When Nine Digits are Enough

Often you may not be interested in results with an accuracy of 15 digits, and may find a long row of digits after the decimal point distracting.

The command FSHORT puts Hypatia in “short” mode, where only up to 9 instead of up to 15 digits are shown. FLONG sets the maximum number of digits back to 15 (18 digits for integers).

These are output format commands (see page 26), and like all oputput format commands they have no effect on the actual values. FSHORT does not round any results and does not make Hypatia less accurate, only what is *shown* gets rounded to 9 digits. FSHORT *does* affect how results are saved to the file hy, though.

Unlike the other output format commands, FSHORT also affects how the values of variables are displayed by SHOW (page 21), and how intermediate results are shown in debug mode (page 72).

FSHORT does not change how numbers are displayed with the FDEC n format or by the == command.

If you want to make 9 digits your default, add a line FSHORT to the file hy.ini — you can then switch to 15 digits (18 digits for integers) with FLONG.

Integer or Not Integer

A calculation result can be meant to be integer, can look like an integer, and still, due to the internal workings of the computer, not *be* an integer. To illustrate this, let's take the cubic root of 3:

```
? 27 CBRT
= 3
```

So far, so good, but

```
? ISPOSINT
= 0
```

It is not an integer! What is it, then? Let's find out, by subtracting 3:

```
? 27 CBRT 3 -
= -2.16840434497101e-19
```

The deviation from 3 is in the 19th digit after the decimal point — for most practical purposes this is negligible, but obviously the result *should* be 3, not 2.9999999999999999998…

To deal with this issue, Hypatia employs a mechanism that it idiosyncratically calls "integer bias" — a bias towards integer results: when a result is *very* close to an integer (deviating only after the 17th digit), then Hypatia assumes it *should* be an integer, and makes it an integer.

This mechanism is enabled by default — in fact, when you try the above example without having integer bias disabled, you actually get the result you'd expect:

```
? 27 CBRT 3 -
= 0
```

The commands INTEGER ON and INTEGER OFF enable/disable Hypatia's integer bias. INTEGER ON is default, INTEGER alone shows the current status. For the original example, integer bias was disabled.

The integer bias is not applied to the end result of a calculation line, but immediately to each operator's results. Therefore, with integer bias ON (as by default), you can calculate the factorial of the cubic root of 27 — with integer bias OFF, you get an error message:

```
? 27 CBRT !
= 6
? INTEGER OFF
? 27 CBRT !
Error: factorial of non-integer number
```

Unless you have a specific reason to disable integer bias, it is best to leave it ON.

Note that "deviating after the 17th digit" refers to the 17th significant digit, not to the 17th digit after the decimal point. Integer bias is only applied to numbers less than 10^{15} (max. 15 digits before the decimal point).

Also note that the zero threshold deals with an entirely different issue — integer bias is *not* applied to results close to zero!

Number Formats

You can enter numbers in decimal, heximal and binary format, in scientific notation, and large numbers can be entered as lakh, million or crore — see “Numbers” (page 11). You can also use these formats for displaying calculation results.

You can mix different formats in one input line, and use them for your calculations. The way in which numbers are stored and processed is independent of how you have entered them.

All format commands only affect how results are shown, they have no effects on the actual numbers.

Output Format Commands

By default output is in decimal format, which automatically switches to scientific notation for very large or very small numbers.

To force scientific notation for all results, use the command FSCI. To display results in hexadecimal or binary format, use the commands FHEX and FBIN. The command FDEC sets output back to the default format. For numbers that are not positive integers, or are too large, FHEX and FBIN will be ignored.

You can add the number of digits to be shown after the decimal point after the FDEC command, and the number of hexadecimal or binary digits after the FHEX and FBIN commands:

```
? 161
= 161
? FDEC 3
= 161.000
? FHEX
= &hA1
? FHEX 8
= &h0000'00A1
? FBIN 16
= &b00000000'10100001
? FDEC
= 161
```

for “apostrophe format” see next page

The command FMILLION lets results larger than 1e6 be shown in million. The command FLAKH lets results larger than 1e5 be shown in lakh, and results larger than 1E7 in crore.

Once you have chosen the desired format, all results will be shown in this format, unless they are out of that format’s range.

After a format command the recent result is shown in the specified format, but hy is *not* updated. To update hy with the result in the new format, enter $ (this does not overwrite $$).

The command = shows the last result in the currently specified format, the command == shows it in decimal format with up to 18 digits. These commands do *not* update the result file hy, but if logging is ON, they write the result as shown to the log file.

Decimal Digits

By default a maximum total of 15 digits before and after the decimal point will be shown, except for integer numbers, which can have up to 18 digits.

With the FDEC n command you can specify how many digits will be shown after the decimal point. If the total number of digits exceeds 15, the number of digits after the decimal point will be reduced.

FDEC 0 sets the number of decimal digits to 0.

If the number of digits in FDEC n is negative, the value will be displayed with at least as many digits (including leading zeros) before the decimal point. Positive numbers will have a space before them, so that positive and negative numbers remain aligned.

FDEC without a number restores output to the default decimal format, automatically switching to scientific notation for very large or very small numbers. Only this format shows integers with 18 digits.

All format commands only affect the way that numbers are shown, they have no influence on any actual numbers. Also, format commands do *not* update hy — you can write the result in the newly chosen format to hy by entering $ (which, unlike any other single value or variable, is a command).

Apostrophe Format

With “apostrophe format” apostrophes get inserted to make large numbers easier to read.

49'483'031'477.7823	every 3 digits before the decimal point
&h260C'F198'5759	every 4 digits for hexadecimal numbers
&b1110'00001100'11110001'10011000'01010111'01011001	every 8 digits for binary numbers

The commands F' ON and F' OFF turn apostrophe format on and off, F' toggles it. Default is ON.

List of Output Format Commands

FSHORT	show only up to 9 digits (does not affect FDEC n)
FLONG	show up to 15 digits, up to 18 digits for integer numbers (default)
FDEC	decimal format, scientific notation for very large or small numbers
FDEC n	decimal format, n digits after the decimal point (n can be 0, max. = 15)
FDEC -n	decimal format, at least n digits (with leading zeros) before decimal point (max. 15)
FSCI	scientific format, 15 digits and exponent (9 digits and exponent with FSHORT)
FHEX	hexadecimal format
FHEX n	hexadecimal format, n digits (or more if the number is larger, min. 2, max. 12)
FBIN	binary format
FBIN n	binary format, n digits (or more if the number is larger, min. 2, max. 48)
FLAKH	decimal format, Indian notation, results ≥ 1e5 are shown in lakh, results ≥ 1e7 in crore
FMILLION	decimal format, results ≥ 1e6 are shown in million
F' ON | OFF	apostrophe format ON (default) | OFF
F'	toggles apostrophe format ON/OFF

Angle Units

By default, angles are entered and displayed in radians. You can switch between radians and degrees using the angle format commands — these settings only affect the trigonometric functions, which have to "know" in which unit their arguments are given:

USE DEG (or USEDEG)	use degrees for angles
USE RAD (or USERAD)	use radians for angles (default)
USE	show currently set angle format

Do not confuse these angle unit commands with the RAD and DEG operators, which transform degrees to radians and vice versa. Before calling a trigonometric function, be sure that your numbers conform to the selected angle unit.

```
? 90 RAD SIN          converts 90 degrees to radians, and calculates sine of π/2 in radians mode
= 1
```

or

```
? USE DEG             sets angle unit to degrees
? 90 SIN              calculates sine of 90° in degrees mode
= 1
```

Also:

```
? 0.5 ASIN            arcsine of 0.5 (this is independent of chosen angle unit)
= 0.523598776         π/6 (in default radians mode)
? DEG                 convert result to degrees
= 30
```

or

```
? USE DEG             changing angle unit to degrees
? 0.5 ASIN            arcsine of 0.5
= 30                  the result is now given in degrees
```

If you want to use degrees by default, add the line USE DEG to hy.ini (you can also start Hypatia with the -d command line option).

Copy and Paste

Hypatia can interact with the operating system's clipboard.

Writing to the Clipboard

You can copy calculation results and calculation input lines to the Windows clipboard.

COPY	copy the recent calculation result to the clipboard, in the currently chosen format
COPYALL ON	copy all subsequent calculation results to the clipboard
COPYALL OFF	end COPYALL
COPYALL	show current copyall mode (on or off)
COPIN	copy the last calculation input line to the clipboard

Copying all calculation results means that after each calculation the result is copied to the clipboard, overwriting the previous result. COPYALL does not copy the result of the latest already performed calculation, for this you have to use COPY.

COPY actually copies the content of the file hy to the clipboard (this also applies to COPYALL) — as we will see later, in the context of "accumulation mode," this is not necessarily the same, and can include far more than a single result. For details, see "Result File hy and Result Variable $" (page 32).

COPYALL copies hy to the clipboard after a single calculation was performed, at the end of a script, or at the end of a loop — these will be explained later.

COPY still works when COPYALL mode is ON.

COPIN always copies the last calculation input line (this includes $myvar = assign commands, RUN commands, and the executable part of IF … THEN clauses), even if an error has occurred.

Pasting from the Clipboard

The Windows paste command Ctrl/V works in the Hypatia input line. Note that the length of the input line is limited by the width of the console window.

If your calculation requires a larger amount of data than fits into the input line, you can write the data to a file, and use this file as an insert file — see chapter "Insert Files", page 35.

Input and Output Files

Hypatia can read and process files that can contain data, parts of calculations, or scripts — see the two chapters “Insert Files” (page 35) and “Script Files” (page 34).

But first, let us look at the files that Hypatia writes, or can write — by default, all files are in Hypatia’s program folder. Two files are written and updated by Hypatia automatically:

hy has a copy of the last calculation result, using current format settings (see page 32)
hyin in this file Hypatia stores a copy of the last calculation input line

The COPY and COPIN commands copy these files to the clipboard.

hy.ini is created as an empty file by Hypatia, you can edit it to set your own defaults, etc.

`EDIT` opens the fily hy in an external editor — by default, Windows Notepad.

`EDIN` (short for EDIT hyin) opens the file hyin, which contains the most recent calculation input line (including the last $myvar = assign or RUN command). You can edit it there, save it, and use REPEAT to perform the edited calculation (see page 44) — though you will usually scroll up in the input editor, and edit your previous input there.

`EDINI` (short for EDIT hy.ini) opens the file hy.ini.

Optionally Hypatia can write files in which results, variables or user defined elements are saved, and a log file.

`SAVE filename` writes all variables to a file (see page 22)
`SAVE@ filename` writes all user defined elements to a file (see page 37)
`BUFFER SAVE filename` in buffer mode, writes results to a file other than hy (see page 43)

`FILES` displays the names of all files in Hypatia’s program folder.

`EDIT filename` calls an editor, by default Windows Notepad, to let you view or edit any file in Hypatia’s program folder. You can specify an existing file, or create a new one. The file name is converted to lower case. A file extension is optional. The name cannot contain spaces or parentheses. (When you create a new file, it is created by Hypatia, before it is opened with the editor.)

Specifying an Editor

EXTEDITOR filename lets you specify a text editor that will be used instead of notepad.exe. This is the only instance where a file name can include spaces and parentheses — if necessary, you can specify the full path. Do not enclose it in quotation marks! Example:

EXTEDITOR C:\Program Files (x86)\IDM Computer Solutions\UltraEdit\Uedit32.exe

It would be a good idea to create a script file with this command. If you want to make the choice of a different editor permanent, add the EXTEDITOR command line to the file hy.ini

Accumulation Mode

Adding & or && at the end of a calculation line sets the accumulation mode, which means that the result will be written to the file hy after the previous result, instead of replacing it. If COPYALL is active then the accumulated results will be copied to the clipboard, if not then you can use the COPY command.

… & separates results by spaces, … && adds the new result as a new line.

Example: You want to convert the GPS data from a photo you've taken with your cell phone (given in degrees, minutes and seconds) to the format that most digital maps expect: degrees with decimals, latitude first (:msdec converts minutes and seconds to decimal, and + adds the full degrees):

```
? FSHORT                          9 digits are enough here
? 47 37 46.94 :msdec +
= 47.6297056
? 15 51 37.07 :msdec + &
= 15.8602972
? COPY
```

With Ctrl/V you can now paste `47.6297056 15.8602972` into the search field of your favourite digital map. (Note that you will have to add a minus sign in front of the number for latitudes south of the aequator and longitudes west of Greenwich.)

The accumulation mode only applies to the current line, but you can set it in several consecutive lines, adding more results to the same line. The first calculation line in which you do not set it deletes the previously accumulated results from the file hy and replaces them with its own result.

Scripts (see page 34) update hy only at the end of the file, but if you set accumulation mode in any calculation line within the script, this line's result will be added to hy, too. In the last calculation line accumulation mode (or silent mode) must also be set, or any previous results will overwritten at the end of the script.

For an example, see "Centimeters, Feet and Inches" (page 61).

Commands & and && (Clear hy, Add Line Break)

& on its own, unlike at the end of a calculation line, is a command — it clears the result file hy (or the buffer in buffer mode, page 43). The result variables $ and $$ are not affected.

&& on its own is a command that adds a line break to hy or the buffer.

These commands will be important in the context of scripts, accumulation mode and loops.

Silent Mode

By adding # at the end of a calculation line you set silent mode, meaning that the result file hy will not be overwritten with the new calculation result.

The variables $ and $$ will still be updated.

Like accumulation mode, silent mode only applies to the current calculation line.

Note that just entering a number or a variable name is a calculation and overwrites the file hy.

```
? 0 #
```

sets $ to 0, without changing the content of hy (you could also achieve this with $ = 0).

Result File hy and Result Variable $

Both the variable $ and the file hy have the most recent calculation result. You could use either for the next calculation. $ is easier to type and has a higher accuracy, but there can be reasons to use (hy).

```
? 2 3 +
= 5
? (hy) $ *          hy is used as an "insert file" (see page 35), therefore the parentheses
= 25                in this calculation, both (hy) and $ have the value of 5
```

An important difference is that hy has the result in the currently specified format (see page 26), while $ always represents the actual value with the full internal 18 digits accuracy.

The other differences are:
- Accumulation mode allows hy to have more than one result (see pages 31 and 52).
- Silent mode lets you update $ without changing hy (see page 32).
- The command & clears hy without changing the value of $ (see page 52).
- Within a script (see page 34) $ gets updated by each calculation line, while hy only gets updated at the end, or by calculation lines that have accumulation mode set.
- hy can be opened in an editor with the command EDIT (which is short for EDIT hy), can be read by other applications, and can be copied to the clipboard with the COPY command (see page 29), while there is no way to access $ from outside of Hypatia.
- When you exit Hypatia $ is lost, while hy is still there at the next start of Hypatia.

Simply entering $ writes the value of $ to hy in the currently specified format, so you can copy it to the clipboard with COPY. (The last-but-one result variable $$ is not affected.)

Simply entering (hy) sets $ to the result saved in hy. If hy contains more than one number after you have used accumulation mode, (hy) without operator(s) will cause an argument count mismatch error. Do not confuse (hy) with the command HY, which displays the content of hy.

The command `=` shows the value of $ in the currently specified format (this can be useful in scripts), the command `==` shows it in decimal format with up to 18 digits. Neither hy nor $$ are affected.

Viewing Accumulated Results

After you have used accumulation mode the file hy will hold a sequence of two, more, or many more results, in a single line, in separate lines, or a combination thereof. You can use that sequence of results for the next calculation (see for instance “Generalized Fibonacci Sequences”, page 58), but there are three other ways you can view or use the content of hy:

- HY displays the content of hy.
- EDIT (short for EDIT hy) opens hy in an external editor.
- COPY copies the content of hy to the clipboard, from where you can paste it to any application.

HY only shows the content of the file hy when it does not have more than 40 lines or 4000 characters. EDIT and COPY have no size limits.

The Log File

Hypatia can log all input lines and calculation results in the file hy.log. The log file can not be processed by Hypatia, it is meant to be viewed with a text editor.

The commands LOG ON and LOG OFF start and end logging.

When logging begins, the current date and time are written to the log file.

When you resume logging, the log will be appended. To start a new log, you have to delete or rename the existing hy.log file.

You can add comments to the logfile by entering comment lines — any input line that starts with # is a comment line.

Exiting Hypatia stops logging, as does the RESET command — if you want to continue logging when you use Hypatia the next time, you have to give the LOG ON command again. You can add LOG ON to the file hy.ini, if you want it enabled by default, or you can call Hypatia with the command line option -l (see page 73).

The command LOG displays the logging mode (on or off).

Script Files

Hypatia can process — “run” — scripts, treating each line in the script as an input line. These lines can be command or calculation lines. Lines beginning with # are comment lines, they will be displayed when the file is run, unless they begin with ##.

A script is a text file. You have to create and edit script files using a text editor (you can easily do this with the command EDIT filename), except for files which the SAVE and SAVE@ commands automatically create to save variables and user defined elements, which you can still edit. The file name can not contain spaces or parentheses. Script files are located in Hypatia’s program folder.

```
RUN filename
```

will run the specified file.

You can use an abbreviated form of the RUN command:

`_filename` is the same as RUN filename (no space between _ and filename!), with one difference: only the end result will be shown (if there is one), unless echo mode is on (see below).

Script files can be used to assign values to variables (see chapter “Constants”, page 20), to change program mode settings, or to perform calculations. Scripts can be used in loops (see page 47).

When the script is executed, the variables $ and $$ get updated by each calculation line.

By default, the result file hy only gets updated at the end of the script. Setting silent mode (see page 32) for the script’s last calculation line prevents the result from being written to hy.

If you set accumulation mode (see page 31) for a calculation line, though, its result gets written to hy. In that case, the last calculation line must have either accumulation mode or silent mode set, or its result would overwrite the existing content of hy.

Within a script, the commands RUN, RESET, REPEAT and DO are not allowed. A script can end with the quit command Q, QUIT or EXIT, though, which closes Hypatia. Scripts can not be nested.

hy.ini

The file hy.ini is a script file that gets executed when Hypatia starts, or when you reset Hypatia with the command RESET.

An empty file hy.ini is created when you start Hypatia for the first time. You can leave it empty, or you can use the command EDINI (short for EDIT hy.ini) to modify it according to your needs, for instance to define constants or to change default settings. Hypatia itself does not modify the hy.ini file.

Echo Mode

The command ECHO ON sets echo mode to on — when a script is run, its lines get dispayed as they are processed. The command ECHO OFF deactivates echo mode. These commands can also be used within scripts. Comment lines get displayed even when echo mode is off, unless they begin with ##.

ECHO displays the current echo mode (on or off).

“Insert” Files

The content of a file can be inserted into the input line by writing its name in parentheses.

The inserted file can have a single number, or a large number of numbers, or mathematical expressions — anything that you can write into an input line. Other than for the input line itself, there is no limit to its length. The elements in the insert file can be separated by spaces and/or by line breaks.

EDIT filename lets you create and edit insert files, but you can create them any way you like. Insert files are the only Hypatia files whose names can include spaces (but not parentheses).

You can use insert files …

- to define your own constants
- to define your own operators — more about this in chapter “User Defined Operators”, page 40
- to input any amount of data

An input line can have up to ten inserts. Inserts can be nested — that is, an insert file can itself refer to another insert file. By default, all files are in Hypatia’s program folder.

When your input line contains an insert file, the line including the inserted text will be displayed.

Insert files can include comment lines for documentation purposes: any line in the insert file that begins with # is a comment line. Comment lines in insert files will not be displayed.

Note that, apart from comment lines, line breaks in insert files are ignored — no matter how many lines the insert file has, when it is used in an input line its content will become a part of that line.

If you have a file myfile with the following lines:

```
# this is an example for the use of insert files
5
3 11 6
9 8
```

then you can use those numbers for your calculations:

```
? (myfile) SUM
= 42
? (myfile) MEAN
= 6
```

As this simple example demonstrates, you can use all n-argument operators like mean, median, standard deviation etc. on data in insert files. Individual data items can be referenced with the ITEM operator (page 18). Insert files can be large, much larger than fits into an input line, and can hold thousands of numbers. Hypatia is not a tool for statistical analysis, though.

An insert file can not have the name buffer, this is reserved for Hypatia’s result buffer (page 43).

Defining Your Own Constants

This has already been discussed in the chapter “Constants” (page 20). If you repeatedly need a certain number for your calculations, you can create a file, write that number into it, and in the input line write the name of that file in parentheses — it will be replaced with the number that you have saved.

It can be easier to use variables as your “constants,” and either save them to a file with SAVE filename and load them with _filename (see chapter “Variables”, page 21), or if you need them often, to add them as variables to the hy.ini file, using `$myvar = ...` assign commands. If their values change often, though — for instance currency exchange rates — having them as insert files will make it easier to update them.

Insert (hy)

The file hy always has the most recent calculation result, which usually is the same as the variable $ — see “Result File hy and Result Variable $” (page 32). $ is lost when you exit Hypatia, while (hy) is still available the next time you start it.

Thanks to accumulation mode, hy can hold more than one, even a large number, of results, which you can use for subsequent calculations. Hypatia’s ability to use series of calculation results as data for new calculations offers a lot of possibilities. You will find examples of how to make use of this feature below, for instance in the chapters “Monte Carlo Experiments” (page 55), “DONE, and Generalized Fibonacci Sequences” (page 58), or “Centimeters, Feet and Inches” (page 61).

Script Files vs. Insert Files

Note the difference between script and insert files:

- `RUN filename` or `_filename` expects a file in which each line is a valid input line.
- Line breaks in script files matter, while in insert files they are converted to spaces.
- `(filename)` inserts the content of the file into the current input line — this file can contain numbers and/or operators, which can be separated by spaces or line breaks.
- Insert files can be used in scripts.
- Insert files can also be used in `$var =` and in `@ude =` assign statements (for the latter, see next page), and in responses to `PROMPT $var` commands.
- The contents of an insert file can be assigned to a user defined element (again, see next page), and a user defined element can be used instead of an insert file.

It is possible to have single line files that can be used both as script files and as insert files, but this is of little practical relevance. Most scripts will have more than one line, and their purposes are very different from those of insert files.

User Defined Elements (UDEs)

A user defined element looks like a variable, but works like an insert file — without a file, though.

Like insert files, user defined elements can contain anything that can be written into an input line — data, operators, or (probably less often useful) commands.

Like variables, user defined elements are created with an assignement command. While names of variables begin with $, those of user defined elements begin with @. (@ on its own is a valid name.)

While a variable is always a number, a user defined element is always a text, though this text can consist of, or contain, numbers. Unlike script and insert files, user defined elements always consist of single lines.

User defined elements have two main purposes:

- to let you create your own operators. You can do that with insert files, as discussed in the previous chapter, but UDEs may be more convenient — see chapter “User Defined Operators”, page 40.
- to hold data, entered manually or read from a file — see next page.

UDE Commands

The UDE commands, ending on @, are similar to the variable commands:

@ude = …	assign a text to a user defined element
SHOW@	show all user defined elements
SHOW@ @ude1 @ude2 …	show these user defined elements
DEL@ @ude	delete the user defined element
SAVE@ filename	save all UDEs to a file
_filename	(or RUN filename) retrieve UDEs that were saved with SAVE@

The SHOW@ command only displays one screen line for each UDE — if the UDE’s text is longer, it will be truncated. (The SAVE@ command does not truncate.)

Like variables, UDEs are lost when you exit Hypatia, or use the RESET command. In analogy to the command SAVE filename, which writes all variables to a file, SAVE@ writes all UDEs to a file, and as with SAVE, you can add a comment text after the file name. (No spaces or parentheses are allowed in the file name, all characters are converted to lower case, and no file extension is added if you do not specify one.)

You can edit saved UDEs with EDIT filename (or you can use any other editor), and you can load them from the file with `_filename` or `RUN filename`.

UDEs that you need more often will best be added to the hy.ini file, so that from the user’s point of view they become part of Hypatia’s vocabulary.

Data UDEs, UDEs and the Result File hy

Like insert files, user defined elements can not only contain operators, but also data. You can enter a list of vaues manually, like this:

```
? @mydata = 3 8 4 9 5 2 7 11 6
```

or you can assign the data from a file to a UDE, and then use this UDE instead of inserting that file:

```
? @mydata = (myfile)
```

You can then use n-argument operators on the data that's stored in the UDE:

```
? @mydata MEAN          instead of (myfile) MEAN
? @mydata SDEV
```

Reading data from a file into a user defined element can make loops faster, when that data is used in that loop hundreds or thousands of times. (Loops will be discussed below, page 45.)

There is one important difference between using an insert file and using a UDE that has that file's data: when the file changes, the data in the UDE remain the same. This can be useful in the context of "accumulation mode" (page 31), when a loop has written a number of values to the result file hy (for instance in a Monte Carlo experiment) and you want to use them as arguments for subsequent calculations.

```
? (hy) MEAN
```

will compute the arithmetic mean of all values in hy, but the result will overwrite the data.

One way to avoid losing the data is to use silent mode (page 32):

```
? (hy) MEAN #
? (hy) STDEV #
? (hy) N+ #
```

etc., but if you forget the silent mode symbol # just once, your data is lost. Also, you cannot copy the results to the clipboard, because that depends on writing them to the file hy (page 29).

These problems can be avoided when you assign the data from the result file hy to a user defined element, and then use this UDE as the argument for the subsequent calculations with n-argument operators — whatever gets written to the file hy will not change the content of the UDE:

```
? @ = (hy)              assign the content of hy to @ — yes, @ on its own is a valid UDE name!
? @ MEAN
? @ STDEV               etc. …
```

Notes on User Defined Elements

You can not edit the content of a user defined element within Hypatia, with one exception: you can add something before and/or after the existing text:

```
? @myude = 3 5 7 11
? @myude = 1 @myude 13
? SHOW@ @myude
@myude = 1 3 5 7 11 13
```

The only way to actually edit a UDE is to use the command SAVE@ filename to save all UDEs to a file, and then open that file in an editor — from within Hypatia you can do this with EDIT filename. Take care not to insert line breaks — they are allowed in insert files, but not in UDE assign commands! You can then reload the edited UDE(s) with _filename, or copy/paste the edited UDE for instance to hy.ini.

Apart from numbers and operators, a UDE assign command can include insert files, variables, and UDEs — you must be aware of how they are processed, though:

? @myude = ... (somefile) ...

The insert file gets read and, as we've seen on the previous page, its current content will be assigned (or will be part of what is assigned) to @myude. If the file does not exist at the time of the assign command, you get an error message. When the file later changes or is deleted, @myude will remain unaffected.

? @myude = ... $somevariable ...

A variable name in a UDE assign command will *not* be replaced by the variable's current value, but the variable *name* will be part of the UDE's content. The variable does not even need to exist when you define your UDE, but if it does not exist when you *use* the UDE, you will get an error message.

? @myude = ... @someude ...

Use this with caution if at all, because it can have two different results, depending on whether the UDE to the right of the equal sign exists or not. If it exists, it is treated like an insert file — if it later gets changed or deleted, @myude will remain unaffected. (It is possible to copy a UDE — the new UDE will be independent of the original one: @ude2 = @ude1)

If the UDE to the right of the equal sign does *not* exist, though, then it is treated like a variable: its *name* will be part of @myude, and it will be replaced by that UDE's current content at the time when @myude is used.

User Defined Operators

Everything that Hypatia is able to calculate, can be calculated using Hypatia's operators. You cannot add new features, but you can create your own composite operators that suit your needs.

For instance, to calculate the area of a circle with a given diameter, you would have to divide the diameter (for instance 7) by 2 to get the radius, square it, and multiply the result with π (we use FSHORT in these examples):

```
? 7 2 / SQ PI *
= 38.48451
```

If you have to calculate circle areas more often, you can create your own operator that performs this calculation. You can do this in two ways:

As an insert file: you write `2 / SQ PI *` to a file which you may call, for instance, circlarea — the easiest way to do this is:

```
? EDIT circlarea
```

Or you can create a user defined element: you assign `2 / SQ PI *` to a UDE:

```
? @circlarea = 2 / SQ PI *
```

To have this UDE always available, add this line (of course without the question mark) to hy.ini.

In each case, it is up to you to remember the name of your operator, and how to use it — for instance, that it requires the diameter, not the radius, as its argument. (In the insert file, you could add this information as a comment line.)

According to which method you have chosen, you can now calculate the area of a circle with a diameter of 7 in one of these ways:

```
? 7 (circlarea)
= 38.48451
```

```
? 7 @circlarea
= 38.48451
```

Or, to give a somewhat more complex example, an operator to calculate the volume of a sphere. In conventional infix notation, the formula is ((d/2)^3*π*4)/3 — in Hypatia's RPN, you can write:

```
? @sphvol = 2 / 3 ^ PI * 4 * 3 /
```

and then, for instance,

```
? 7 @sphvol
= 179.59438
```

to calculate the volume of a sphere with a diameter of 7 (again, you could as well have used an insert file for your operator).

You can use your user defined operators in any calculations, the same way you can use Hypatia's "native" operators. Whether you use insert files or user defined elements for your user defined operators is entirely up to you.

Pseudo Operator WHISK, and User-Defined 2-Argument Operators

With one argument, there are no restrictions to user defined operators. With two arguments, it gets a bit more complicated: some can be easily defined, others need recourse to a special feature, the WHISK pseudo operator.

Let's look at these examples — formulas in infix notation, followed by corresponding user defined operators:

$a + b^2$	`SQ +`	square b, then add the result to a
$(a + b)^2$	`+ SQ`	add up a and b, then square the result
$a^2 + b^2$		???

WHISK performs the necessary trick: it removes (whisks away) the two values to its left (remember, when an operator is encountered, to its left can only stand numbers), and stores them under the names A and B, which you can then use in your calculation. (Note that WHISK deliberately does not auto-include $ at the beginning of a line.)

```
? 3 4 WHISK A SQ B SQ
= 25
```

This lets us, for instance, create a user defined 2-argument operator that calculates the hypotenuse of a right-angled triangle from its legs — the famous $a^2 + b^2 = c^2$ or $c = SQRT(a^2 + b^2)$ formula:

```
? @hypot = WHISK A SQ B SQ + SQRT
? 6 8 @hypot
= 10
```

True, you might have done this calculation with an n-argument operator, 3 4 SQSUM SQRT, for which you wouldn't need WHISK, but n-argument operators only work at the beginning of a line, or with the help of the delimiter | (page 18). WHISK does not have this restriction, and also makes more complex calculations possible, in which A and B are used repeatedly.

Take, for instance, Srinivasa Ramanujan's approximation formula for the circumference of an ellipse —
in infix notation: $(3 (a + b) - sqrt(10 a b + 3 (a^2 + b^2))) * pi$
in RPN notation: a b + 3 * a b * 10 * a SQ b SQ + 3 * + SQRT - pi *

We can assign this to a user defined 2-argument operator:

```
? @ellips = WHISK A B + 3 * A B * 10 * A SQ B SQ + 3 * + SQRT - PI *
```

and use this to calculate the approximate circumferences of ellipses:

```
? 3 6 @ellips
= 29.0652633
```

(Instead of user defined elements, we could also use insert files for our user defined operators.)

In some cases you may want to "whisk away" only one value — you can do this by putting a dummy value (for instance, 0) before WHISK, and then only use A as often in your calculation line as you need.

WHISK offers a far more flexible way of using the same value several times than the pseudo constant DUP (page 20) — for differences between DUP and WHISK, see "Processing Input Lines" (page 76).

You can use WHISK several times in one calculation line, overwriting the previous values of A and B.

Note that WHISK, A and B can only be used within one calculation line, the values of A and B are not preserved after the line has been processed — this is deliberate. Where WHISK doesn't cover your needs, you need to use variables and scripts.

More than 2 Arguments

With the help of WHISK you can create any conceivable 2-argument operator, but it is also possible to create operators with three or even more arguments — but only, when the operator can work through its arguments from right to left.

A very simple example: to calculate the volume of a cuboid, you have to multiply length, width and height. You can write this as `l w * h *` but you could also write it as `l w h * *` — in that case, `* *` would work as a 3-argument operator:

```
? @cubvol = * *
? 3 4 2 @cubvol
= 24
```

Slightly less trivial: a to b equals c to x — you have, for instance, a rectangle with sides a and b, and want to know side x of a rectangle with the same aspect ratio and side c instead of a.

In infix notation:

```
c / x = a / b
x / c = b / a
x = b * c / a
```

but, if you want to write the arguments in the order a b c, you cannot create a 3-argument operator that does this calculation — unless you turn the fraction on its head:

```
x = 1 / (a / (b * c))
```

or in Hypatia's RPN:
`a b c * / RCP` — multiply b and c, divide a by their product, get reciprocal of that result.

And for this, we can create a 3-argument operator, let's call it @abcx:

```
? @abcx = * / RCP
? 3 4 12 @abcx
= 16
```

The possibilities for 3- and more argument operators are restricted, but where they are possible they can be useful. WHISK can be used in them, too, and in fact we could have solved this problem (less elegantly) this way: `* WHISK B A /`

And, of course, you can also use variables in your used defined operators …

The Result Buffer

Hypatia's buffer has the purpose to make loops (which we haven't yet discussed) faster, in case they write hundreds or thousands of results to hy. A large number (it can theoretically be up to ten million) of write-to-disk operations take their time — much of that time can be saved by writing all results to Hypatia's internal buffer, and only write them to the disk (that is, to the file hy) in a single operation at the end of the loop.

Buffer Commands

BUFFER START	turn buffer mode on, clear buffer (if buffer mode is already on, clear buffer)
BUFFER SHOW	display buffer content (only when not more than 40 lines or 4000 characters)
BUFFER SAVE filename [comment]	save buffer to file (anything after filename is a comment line)
BUFFER FLUSH	write buffer to hy replacing its prior content, clear buffer, turn buffer mode off
BUFFER DISCARD	delete content of buffer and turn buffer mode off
BUFFER	display buffer mode (on or off)

Using the Buffer

When buffer mode is on, everything that would otherwise be written to the result file hy will be written to the buffer, in exactly the same way.

Accumulation mode (& or && at the end of a calculation line) and silent mode (# at the end of a calculation line) work with the buffer exactly as they do with hy (see page 31).

The commands & and && also work the same way: in buffer mode, the command & clears the buffer, while && adds a line break — see "Loops and Accumulation Mode" below, page 52. (Do not confuse the commands & and &&, which stand on their own, with the & and && control symbols at the end of a calculation line!)

After turning buffer mode on with the BUFFER START command the buffer is empty. The result file hy will remain unaffected and can still be copied with COPY, opened in an editor with EDIT, or inserted into an input line with (hy). Only the BUFFER FLUSH command will replace it with the content of the buffer.

As long as buffer mode is on, (buffer) can be used exactly as you might use (hy):

… (buffer) …	insert buffer into input line as if it were an insert file
… (buffer) i ITEM …	the i^{th} item in the buffer
@myude = (buffer)	assign data in buffer to user defined element

When buffer mode is on, the BUFFER SAVE command lets you export the current content of the buffer to a file at any time.

BUFFER FLUSH replaces hy with the buffer, clears the buffer, and turns buffer mode off. If the buffer is empty, BUFFER FLUSH only turns buffer mode off, but does not overwrite hy. To end buffer mode without writing the content of the buffer to hy (because you do not need it anymore, or because you have already saved it to a different file), use BUFFER DISCARD.

The REPEAT Command

REPEAT has two purposes: to repeat a calculation with possible modifications, for instance with different values or after fixing an error, or as a loop command, repeating a calculation a specified number of times or until an exit condition is met.

The File hyin and the REPEAT Command

The most recent calculation line gets written to the file hyin. RUN commands (page 34), $myvar = commands (page 21) and the executable part of IF … THEN clauses (page 60) get written to hyin, too. You can copy the line to the clipboard with the command COPIN, you can open, edit and save it with the EDIN command, and you can execute the line again with the REPEAT command.

Some reasons why you may want to re-use the previous input line:

- to repeat a calculation that uses the previous result $ as an argument
- to repeat a calculation with changed content of inserted files
- to repeat a calculation after you have edited it

The command REPEAT reads the line from the file hyin, and treats it as a new input line.

Alternatively to using REPEAT you could scroll up through the list of recent input lines by pressing the ARROW UP key, edit the line if necessary, and execute it by pressing ENTER.

When the calculation line contains an insert file, you can change the content of that file before you let REPEAT perform the calculation with different data.

When the calculation line contains $ or $$, they will get updated, so that their values keep changing:

```
? 3              sets the value of $ to 3 (you could also write $ = 3)
= 3
? SQ             short for  $ SQ  because $ can be omitted at the start of the line
= 9
? REPEAT
  sq             the REPEAT command displays the line it repeats
= 81
```

If you want to correct an error, or perform a calculation with some modifications, you can edit the recent input line. There are two ways of doing this:

- you can scroll up and use Hypatia's own editing features, which would also allow you to go back to older input lines
- or you can use the EDIN command to edit the file hyin in a text editor (this is the same as EDIT hyin), save it, and use REPEAT to perform the edited calculation.

The loop command REPEAT n (see next page) depends on the file hyin.

Loops

Hypatia knows two kinds of loops, REPEAT and DO loops. Actually they are very similar, but they differ in how you use them. Hypatia's loops can be count- or condition controlled — that is, you specify how many passes of the loop will be performed, but you can also specify an exit condition. The executable part of a loop can be a calculation line, or it can be a RUN command that calls a script file.

Loop commands can not be nested — see "Nested Loops", though (page 65). Loop commands are not allowed within scripts.

The REPEAT loop command is:

```
REPEAT n [?]
```

with n being an integer between 1 and (by default) 99999 — see "The MAXLOOP Command", page 46. This repeats the previous calculation or RUN command n times — see page 47.

The DO loop command is:

```
DO n [?] :: calculation
DO n [?] :: RUN filename
```
or
```
_filename
```

Different from the REPEAT loop command, DO does not depend on the previous input line — again, this will be discussed below. The number of passes n has to be between 1 and (by default) 100000.

Instead of a number, you can put an asterisk `*` after DO or REPEAT, which stands for the highest permitted number of passes — see "The MAXLOOP Command" (page 46).

By default only the final calculation result will be shown, though if you put a question mark after the number, calculation results will be shown for up to 40 passes (see page 49). Note that this question mark is not the same as the "debugging" question mark (see page 72).

Regarding exit conditions see "Defining an Exit Condition" (page 49) and "Conditional IF … THEN Clauses" (page 60).

Within a loop, the time in seconds since the start of the loop is given by the pseudo constant TIME. You can assign TIME to a variable ($t = TIME), you can display it with the SHOW command (SHOW TIME), or you can use it in an exit condition (IF TIME >> 100 THEN ENDLOOP).

I*: SHOW TIME at the end of a script will show you how many seconds the loop needed to complete.

Loops and the Result Variable $

By each calculation line, unless it results in an error, the variables $ and $$ are updated, allowing each calculation to easily build upon the result of the previous one(s).

Results are also written to the file hy, overwriting the previous result. You can modify this behavior by using accumulation mode (all consecutive results are written to hy, separated by spaces or line breaks), or silent mode (hy does not get updated) — see pages 31 and 32, and "Loops and Accumulation Mode", page 52 (silent mode does not prevent $ and $$ from getting updated). Note that variable assignment commands do *not* update $ or hy, even if they perform calculations.

If your loop runs a script, $ and $$ get updated by each calculation line in the script, but the result file hy only gets updated once at each pass, at the end of the script — except for calculation lines that are set to accumulation mode, these write their results to hy immediately, before the end of the script is reached. Accumulation mode and silent mode for the end result of the script can be set in its last calculation line (see also “Loops and Accumulation Mode”, page 52).

At the end of a loop the value of the most recent result, that is the value of $, is always shown. A loop does not necessarily update it (variable assignment commands don’t!), in that case $ remains what it had been before, but is still shown.

If the loop was aborted due to an error, the most recent result is reset to what it had been before the loop, and this value is written to hy. (Do not confuse this with the ABORT command, page 60.)

One more thing: As examples will show, you may need to set $ to a certain value, for instance 0, before executing a loop command. This can be done in two ways: either just enter 0 (which is a minimalistic calculation line that updates $ with the result 0), or directly assign the value 0 to the variable $ with $ = 0. The difference is what happens to $$: in the first case, it is updated with the prior value of $, in the second case, it stays unchanged.

The MAXLOOP Command

By default the maximum number of loop passes is one hundred thousands. You can change this value with the MAXLOOP command to any number from 10 to ten million.

```
? MAXLOOP 1000
Max. number of passes set to: 1000
```

For large numbers, you can use million, or the Indian number system units lakh (one hundred thousand) and crore (ten million). MAXLOOP 1 LAKH sets the maximum number of loops back to its default value of 100000, MAXLOOP 1 CRORE sets it to its highest possible value of ten million.

Apart from being able to use lakh, million or crore, you can also write the number in scientific notation, e.g. 1E4 for 10000. You can *not* use an arithmetic expression or a variable, though.

The maximum number of loop passes serves two purposes: it prevents you from accidentally starting loops with a too high number of loop passes, but you can also easily use it in the REPEAT n and DO n commands by writing an asterisk * instead of a number (see below) — this also applies to the short version of the DO command for scripts.

MAXLOOP without a number will show you the current value.

The Loop Command REPEAT n

REPEAT n, with n being a number between 1 and by default 99999 (one less than the maximum number of passes), repeats the most recent calculation or the most recently called script n times.

The following example makes no sense, it is only meant to demonstrate the REPEAT loop command:

```
? $ = 0              set result to 0 (you could also just write 0)
? 1 +                add 1 (the same as $ 1 + because $ can be omitted at the start of the line)
= 1
? REPEAT 99          repeat the calculation 99 times
  1 +                the REPEAT command displays the line it repeats
= 100
```

REPEAT reads the most recent calculation from the file hyin. You can edit this file with an editor, before giving the REPEAT command.

Note that REPEAT 99 means that the calculation gets performed a total of 100 times — first by the original input line, followed by 99 repetitions (see also the following chapter “Loops and Loop Index”).

Instead of the number of passes you can write an asterisk — REPEAT `*` — it stands for the current maximum number of passes (see “The MAXLOOP Command” above) minus one.

The Loop Command DO n

With the DO loop command, the above example would look like this:

```
? $ = 0              set result to 0 (you could also just write 0)
? DO 100 :: 1 +      add 1 to the previous result a hundred times
= 100
```

This is considerably shorter, and in most cases you will probably prefer the DO n over the REPEAT n command. The advantage of the REPEAT loop is that you can see the result of the first pass, or a few more, to verify that your calculation line or script works as intended, before you execute the loop.

In most cases DO loops will be used with scripts:
`DO n :: RUN filename` or `DO n :: _filename`

You can write the number of passes in scientific notation (e.g. 1E4 for 10000), and you can also use lakh or million. The number can not be higher than the current maximum (one hundred thousand by default, or the number you have set with the MAXLOOP command). You can *not* use an expression or a variable.

Instead of the number of passes you can write an asterisk — it stands for the current maximum number of passes.

```
DO * :: _filename        this has a shortcut version:
*_filename               (to make it easier to remember, _*filename works, too)
```

Loops and the Loop Index I

Loops allow you to perform calculations repeatedly, whether they are single lines or scripts.

Of course, it makes no sense to repeat the exact same calculation over and over again — something has to chance each time the calculation is performed. You can do this in four ways:

- you can use the previous result $ and/or the penultimate result $$ for your calculation, as we've seen in the above example. $ and $$ get updated by each calculation line, this is also the case within a script
- you can use (hy)
  In a script, hy only gets updated at the end of the script, or by lines with accumulation mode set
  Using (hy) in a script will usually happen in combination with accumulation mode (see page 52)
  When you use buffer mode (see page 43), then (buffer) is used instead of (hy)
- in a script, you can use your own variables which you can update at each pass (see page 21)
- you can use the loop index "pseudo constant" I (outside of a loop I always has the value 1)

You can use I in a calculation the same way you would use a number or a variable. Suppose you want to add up the numbers 1 to 100 (of course there is a simple formula by which you can compute the result, but we want to see the loop index at work):

```
? $ = 0              set $ to 0 (or just write 0)
? DO 100 :: I +      (short for  $ I +  because $ can be omitted at the start of the line)
= 5050
```

Let's stay with this simple example but use a REPEAT instead of a DO loop, to illustrate a difference between the two regarding the loop index I:

```
? $ = 0              set $ to 0  (or just write 0)
? I +                add I — as already said, outside of a loop I always has the value of 1
= 1
? REPEAT 99          add the numbers 2 to 100 — in a REPEAT loop, I is loop index plus 1
  i +                the REPEAT command displays the line it repeats
= 5050
```

With I in the loop starting at 1 and ending up at 99, the loop would have added the numbers 1 to 99 to the previous result instead of the numbers 2 to 100, as we want it to do. But REPEAT assumes that you repeat the previous calculation, which was meant to be the first pass of the loop — therefore, in a REPEAT loop the value of I is always the actual loop index plus 1.

While I always is an integer that grows by 1 with each pass, with simple arithmetics you can emulate a loop with any start value, end value, and increment. And within scripts you can set up your own loop variables that can behave any way you want them to.

Showing Loop Pass Results

By default, only the final result at the end of the loop is shown (this is independent of whether results get written to hy at each pass of the loop by using accumulation mode — see page 52). If you want the value of $ to be shown after each pass while the loop is performed, you have to add ? to the loop command, after the number of loop passes. (Note that this is not the same as the "debug" question mark at the end of a calculation line — see "Debugging", page 72.)

```
REPEAT n ?
DO n ? :: ...
```

If your loop calls a script, the results that are shown depend on whether you use RUN filename, or the short version, _filename.

Only the results of the first 40 passes will be shown. Should you need more, you can use accumulation mode, or you can use the SHOW command in a script. If logging mode is ON (see page 33), then the results will also be written to the log file.

Defining an Exit Condition

Apart from when an error has occurred in a calculation, a loop ends after the n-argument operator ITEM has referenced the last element of a list of values (or the first one when the index is negative, see page 18), or, in the most ordinary case, after the specified number of passes.

A loop can also end when a certain condition is met. In that case, the number of passes serves as a safety net that prevents the loop from running forever — when you use * for that number, the loop will terminate after 100000 passes, or what value you have set with the MAXLOOP command.

Exit conditions can be based on calculation results, on the loop index I, on user input, or any combinations thereof, and even on the loop timer pseudo constant TIME.

Setting an exit condition requires a script, and can be done in two ways: by setting the variable $loop to 0, or by using the IF … THEN ENDLOOP or IF … THEN ABORT commands — for more about those, see "Conditional IF … THEN Clauses", page 60.

When $loop is set to zero within a loop, and *remains* zero, the loop is exited after the script has been completed. The loop is always executed at least once. The value of $loop is ignored if it is not changed *within* the loop.

The exit condition is only met when the value of $loop is *exactly* zero. To use the zero threshold (by default, 1e-16), use the operators SIGN or ISNOT0 after $loop (see "Zero or Not Zero", page 23). The other IS operators, the operators EQUAL, UNEQUAL and the other compare operators, which all can be used for defining an exit condition, also observe the zero threshold, while the operator // has its zero threshold fixed to 1e-16.

In many cases, though, it will be easier to define an exit condition with IF … THEN ENDLOOP or IF … THEN ABORT than with $loop.

Iterative Calculations

Hypatia is an excellent tool for performing iterative calculations — to demonstrate this, let us calculate square roots using the "Babylonian method."

The mathematics behind it are in fact a bit more sophisticated, but the principle is this: when z is the number whose square root you want to draw, you start with an estimate. Then you divide z by that estimate, take the arithmetic mean of your estimate and that quotient, and use this as your new estimate. Let's do this for the number 99:

```
? $z = 99                 assign the value to variable $z
? 4                       enter an estimate for the square root (it can be inaccurate)
= 4
? $ $z $ / mean           this calculates the next estimate: the mean of $ and the quotient $z/$
= 14.375
```

Now you can repeat this — each new iteration uses the previous result $ as its starting point and gets you closer to the desired square root of z:

```
? REPEAT
= 10.6309782608696
```

Instead giving one REPEAT command after the other, we can use a loop command to execute the calculation repeatedly. By using a script we can define a condition that exits the loop when we are as close to the desired end result as Hypatia can take us.

To do this, type `EDIT babroot` to create the file that will perform our Babylonian root method, and write these two lines (without the comments):

```
$ $z $ / mean             this is the iteration
$loop = $ sq $z //        the exit condition
```

For the exit condition, we compare the square of the current estimate with the number whose root we are computing. The a b // operator checks how close a and b are, by dividing their difference by b — when the absolute value of this quotient is less than 1e-16, it is set to zero. And, when the variable $loop becomes zero within a loop, the loop is exited.

After you have saved the babroot file, you can calculate the square root of 99 as follows:

```
? $z = 99                 assign the value of which you seek the square root to variable $z
? 4                       enter an estimate for the square root
= 4
? DO * :: _babroot        or the abbrevated form: ? *_babroot
Loop exit condition met in pass 7
= 9.9498743710662         this is the square root of 99
? SQ                      just to check if it is correct …
= 99
```

We used the DO loop command because we knew that we had a well established method, could rely on it to be convergent, and hadn't made a mistake in our script — this is also why we could use the

asterisk (meaning 100000 by default) for the maximum number of passes. If we were not that sure, it might be a good idea to use REPEAT instead of DO, and to be able to observe the first few passes.

```
? $z = 99                    assign the value of which you want the square root to variable $z
? 4                                enter an estimate for the square root
= 4
? RUN babroot                      the first iteration
= 14.375
  $loop = 2.59375
? REPEAT                           let's look at the next one
  run babroot                      REPEAT displays the line it repeats
= 10.6309782608696
  $loop = -0.260453686200378
? REPEAT                           and another one
  run babroot
= 9.97169280100377
  $loop = -0.0620155025894923      now we are satisfied that the iteration works as intended,
? REPEAT *                         and can give the REPEAT loop command
  run babroot
Loop exit condition met in pass 4
= 9.9498743710662
```

A different way to define the exit condition would be to compare the result of the latest iteration with the previous one — when they are close enough to being equal, the iteration process has reached its end (do not use this with an iteration process that you do not trust). The second line in the babroot script would then be:

```
$loop = $ $$ //
```

Or instead of (a–b)/b, as the // operator does, you could directly check whether they still differ:

```
$loop = $ $$ UNEQUAL
```

(as long as they are unequal, $loop is 1 and the loop continues)

Note that the fixed 1e-16 threshold of the // operator always refers to a quotient, and is therefore independent of the order of magnitude of the numbers you're working with. When you look at a difference, as with the EQUAL or UNEQUAL operators, the default 1e-16 threshold may not be adequate — you can change it by setting the variable $zero (see page 23). Or you use the GATE operator: `$loop = $ $$ - b GATE`, replacing b with your chosen threshold (which defaults to 1e-16 if b = 0, since an actual gate value of 0 would make no sense).

Or, as discussed below in the chapter "Conditional IF … THEN Clauses" (page 60) you could write:

```
IF $ $$ EQUAL THEN ENDLOOP
```

(Again, EQUAL observes the zero threshold, which may need to be adjusted if you are working with very large or very small numbers.)

Loops and Accumulation Mode

Accumulation mode lets you save calculation results within a loop, instead of retaining only the final result. You have to activate accumulation mode by adding & (separated by space) or && (separated by line breaks) to the calculation line whose result you want to append to hy.

In a script, accumulation mode can be set for any calculation line whose result you want to be saved (by default, only the last result is saved to hy) — in that case, you need to set either accumulation or silent mode for the last calculation line in the script, or the existing content will be overwritten!

The command & clears the result file hy, but does not affect the variables $ and $$.

The command && inserts a line break to hy — this is very different from the command &, which deletes the content of hy.

Suppose you want a sequence of 8 random 8-bit words (these are hexadecimal numbers in the range of 0 to 255, or &h00 to &hFF):

```
? FHEX 2                set output format to hexadecimal with 2 digits
= &h00                  or whatever the current value of $ is
? &                     clear hy (this does not set $ to 0, but we do not need that here)
? DO 8 :: 0 255 RANDINT &
= &h8F                  (for instance) — only the last result is shown
? HY                    show the content of hy
  &h80 &hEA &hFA &h9B &h7E &h50 &h36 &h8F
```

Line 4 executes a loop that writes 8 random integer numbers in the range of 0 to 255 to hy, in the currently specified format. Instead of `0 255`, you could also have written `0 &hFF`.

HY displays the list of results; with EDIT (short for EDIT hy) you can open it in a text editor, or with COPY you can copy it to the clipboard from where you can paste it to any application. The &h… notation may not be what you want, but it is trivial to change this to `80 EA FA 9B 7E 50 36 8F` in any text editor, by replacing &h with nothing.

The following makes little sense, but it demonstrates how you can make use of (hy):

`(hy) mean` will calculate the mean value of the results — this works whether you use spaces or new lines for accumulation mode, and has no problems with hexadecimal numbers. With only 8 random numbers between 0 and 255, their mean may deviate considerably from the expected value of 127.5.

This calculation, like any new calculation, would overwrite the list of results and replace it with its own result. One way to avoid this is to use silent mode (page 32):

```
? (hy) min #            silent mode, do not overwrite hy
= &h36
? (hy) max #            silent mode, do not overwrite hy
= &hFA
? (hy) mean             without specifying silent mode, the result becomes the new content of hy
= 146.25                result is shown in decimal format, because it is not a positive integer
```

A more reliable way would be to assign the data in hy to a user defined element (see page 37), and use that as the list of arguments:

```
? @u = (hy)
? @u min              the result file hy gets overwritten, but @u still has its former content
= &h36
? @u max              etc. …
= &hFA
```

A third way to preserve the data in hy would be to use EDIT to open hy, save it under another file name, and use that file instead of hy to calculate (in our example) minimum, maximum and mean.

Another example for using accumulation mode: suppose you want to see a value of 5000 increase by 5.25% thirty times, and you want the numbers to be shown with 2 decimal digits. && writes each result in a new line.

```
? 5000                in this example you do not need to clear hy with the & command,
= 1000                because you want the start value to be included in the list of results
? FDEC 2
= 5000.00
? DO 30 :: 5.25 %+ &&
= 23207.76
```

We could have gotten the end result with the simple formula `1000 1.075 10 ^ *`, but the loop, together with accumulation mode, gives us a list of all intermediate values. When you look at hy with the command HY, or open it in an editor with EDIT, you see

```
5000
5262.50
...
22050.12
23207.76
```

With only a few values written to hy there is no need to use the buffer (see page 43), but should there be hundreds or thousands, the loop will be much faster if you use buffer mode.

Note that calculations are performed with the exact values, the format commands (like FDEC 2) only affect the output. Should you want to base each calculation on the rounded result of the previous one, you'd have to use ROUND (the actual differences are minimal):

```
? DO 10 :: 7.5 %+ 2 ROUND &&
= 23207.73
```

Note that while Hypatia does its best to round accurately, it may not fully meet the requirements of financial mathematics.

Start and End Lines of Loops

When using a script in a loop you may have to take some preliminary steps, like clearing the result fily hy, defining variables, or assigning them initial values. You can do this manually, before you use DO or RUN/REPEAT, but you can also do it in the script itself: when you put `I1:` at the beginning of a line, whether it is a command or a calculation line, then this line is only executed when the loop index is 1 — that is either when the script is called directly, not in a REPEAT loop (there, I starts with 2), or at the first pass of a DO loop.

Also, there may be steps you want to take at the end of the loop, like displaying the end values of some variables, adding a final result to hy, or displaying the content of hy. Again, you can do this manually, or you can do it within the script: when you start a line with `I*:` then it only gets executed at the end of the loop.

I*: lines also get executed when the loop is ended due to the $loop = 0 or IF … THEN ENDLOOP exit condition being met, but only if they come after the line in which that happens.

When a script is called directly, I1: lines are executed, I*: lines are not.

As a simple example, let's write a file that lets us roll a dice repeatedly, writing the individual results to hy — let's name the file rolldice:

```
I1: &                    this clears the fily hy at the start of the loop
1 6 RANDINT &            a random value 1 to 6 gets appended to hy
I*: HY                   at the end of the loop hy gets displayed, and …
I*: (hy) SUM #           the sum gets calculated and displayed, but not written to hy
```

To roll the dice ten times, we use a DO loop, and we'll see something like:

```
? DO 10 :: _rolldice
  2 6 4 5 2 1 4 4 6 3
= 37
```

If we're not interested in the individual rolls of the dice but only want to know the total sum, then instead of writing the numbers to hy, we better use a variable to add them up:

```
I1: $sum = 0
$sum = $sum 1 6 RANDINT +
I*: $sum
```

REPEAT vs. REPEAT 1

REPEAT and REPEAT 1 are not exactly the same, even though both repeat the preceding calculation or RUN command one time. REPEAT without a number of repeats is exactly the same as performing the calculation or calling the script file again — the loop index I is 1, a script displays all the results and variable assignments, I1: lines are executed, I*: lines are not.

REPEAT 1, though, is a loop — the loop index I is 2, a script only displays the final result, I1: lines are not executed, I*: lines are.

Monte Carlo Experiments

A Monte Carlo experiment is a mathematical method that uses a series of random numbers to solve a problem for which no direct algebraic solution is known; the name alludes to playing roulette at the Monte Carlo casino.

Hypatia's ability to perform Monte Carlo experiments is limited by the fact that loops can not be nested — there is a workaround, though, as described in chapter "Nested Loops" (page 65). Here is an example of how Hypatia can be used for simple Monte Carlo experiments:

Human body heights are approximately normal distributed. I cannot vouch for the correctness of these numbers, but allegedly in the U.S. the mean height of men is 178 cm with a standard deviation of 7.6, while the mean height of women is 164 cm with a standard deviation of 6.35.

In random pairs of men and women, what is the probability of men or women being taller? And let us exclude minimal differences in height, by saying that if their heights differ by less than 1.5 cm, we consider their sizes to be equal. (It is probably possible to calculate the result if you know the proper statistical formulas, but I don't.)

Let's use the buffer to make it run faster (see page 43), without using the buffer we would have to begin with the command `&` to clear the file hy:

```
? BUFFER START
? DO 10000 :: 178 7.6 RANDND 164 6.35 RANDND - &&
= 11.8353077
```

The loop creates 10000 differences between random heights of men and women, && at the end of the line activates accumulation mode (page 31). The result that is shown has no meaning, it is just the last random difference (and thus the value of $), the 10000 values are stored in the buffer.

Let's assign the 10000 data points in the buffer to a user defined element, and turn buffer mode off:

```
? @mw = (buffer)
? BUFFER DISCARD
```

The BUFFER DISCARD command ends buffer mode without writing anything to hy.

To consider any difference smaller than 1.5 as zero, we have to set the zero threshold to 1.5 (page 23) — then we can use the n-argument operators N+, N0 and N- to count the positive, zero and negative values:

```
? $zero = 1.5          sets the zero threshold
? @mw N+               counts the positive values
= 8989                 and then @mw N0 and @mw N- to count zero and negative values

? DEL $zero            reset the zero threshold, if needed
```

If you repeat the test you will get slightly different results, but this lies in the nature of Monte Carlo experiments. Hypatia can do more complex Monte Carlo experiments than this one — see for instance "Nested Loops in Monte Carlo", page 66.

Loops, Data Lists, and the ITEM Operator

The ITEM operator lets you address individual items from a list of values — in combination with loops and the loop index I, this allows you to process one value from a list after the other.

The syntax (see page 18) is `a1 a2 a3 ... an i ITEM`, where of course `a1 ... an` can be replaced with an insert file, or a user defined element: `@mydata i ITEM`.

As a simple example, let's suppose we want to calculate the sum of the square roots of a series of numbers which we assign to a user defined element:

```
? @mydata = 5 7 9 12
```

We can calculate the sum of their square roots with a simple loop command — first we need to set the result variable $ to 0, then we use a loop to calculate the square roots of the values in @mydata and one by one add them to the result. The loop terminates after the last element in the list has been referenced.

```
? 0                          $ needs to be set to 0, or it would be added to the result —
= 0                          this also applies to the two examples in the next paragraph!
? DO * :: @mydata I ITEM SQRT $ +
Loop exit condition met in pass 4
= 11.3459209037021
```

With this small number of arguments we could write the numbers directly into the input line:

```
? DO * :: 5 7 9 12 I ITEM SQRT $ +
```

or, to show that insert files are a very versatile tool, we could write this whole line (without the question mark, of course) into the file mydata, and just tell Hypatia:

```
? (mydata)
```

Or we could write a script file for this calculation, let's name it sqrtsum. Let's read the data from a file, and, for a change, let's use a variable for the summation. Line 1 initializes it and sets it to 0, line 2 assigns the data from file mydata to a UDE with the same name (it could have any name), line 3 adds the square roots of the values in @mydata to $sum, and line 4 updates Hypatia's result variable $ (and also the result file hy) with the value of our variable $sum when the loop ends:

```
I1: $sum = 0
I1: @mydata = (mydata)
$sum = @mydata I ITEM SQRT $sum +
I*: $sum
```

To use this script we have to call it in a loop (`*_sqrtsum` is short for `DO * :: RUN sqrtsum`):

```
? *_sqrtsum
Loop exit condition met in pass 4
= 11.3459209037021
```

The n-argument delimiter | (see page 18) can be used the same way as with any other n-argument operator — for instance, in the above script the third line could be written as:

```
$sum = $sum | @mydata I ITEM SQRT +
```

Here is a somewhat more complex example, with little practical relevance except to demostrate how the ITEM operator can be used.

Suppose we have a number of boxes — that is, rectangular cuboids — and want to calculate both the sum of their volumes and that of their surfaces.

We need two files — one for the measurements of the boxes, let's call it boxes, and one for the script, let's call it boxscript.

The file boxes may look like this:

```
35 42 12.5
60 35 20
15  8  2.5
```

The line breaks and the additional spaces make the file easier to write and read, but Hypatia ignores them and just sees a one-dimensional list of numbers — the sides a, b and c of the first cuboid are items 1, 2 and 3 of this list, the sides of the second one items 4, 5 and 6, etc. The script has to reference them accordingly, the loop index I representing the consecutive cuboids, or data triplets:

```
I1: @boxes = (boxes)                    to avoid having to read the file 3 times at each pass
I1: $vol = 0                                set volume variable 0 at the start of the loop
I1: $sur = 0                                set surface variable 0 at the start of the loop
$a = @boxes I 3 * 2 - ITEM                  get side a (loop index times 3 minus 2) …
$b = @boxes I 3 * 1 - ITEM                  …  side b (loop index times 3 minus 1) …
$c = @boxes I 3 * ITEM                      … and side c (loop index times 3)
$vol = $a $b $c PROD $vol +                 add product of a, b and c to $volume
$sur = $a $b * $a $c * $b $c * $sur SUM     add the three different surfaces to $surface
I*: $sur = $sur 2 *                         total surface area is twice that sum
I*: $n = I                                  save loop index to variable $n
I*: SHOW $n $vol $sur                       display the results
```

For assigning the data from file boxes to the user defined element @boxes in line 1, see page 38.

When we have these files, we run the script — it automatically stops after the last item in the list got addressed:

```
? *_boxscript
$n = 3
$vol = 60675
$sur = 13220
Loop exit condition met in pass 3
= 0
```

On exiting the loop the final value of the loop index I (corresponding to the number of data triplets in the boxes file) gets assigned to the variable $n in case it may be needed (for instance to calculate the average volume), because I reverts to 1 once the loop has ended. Whatever the last result had been before running the script (for instance 0) is shown at the end, because the script assigns all calculation results to variables, which means that it never updates the variable $ or the result fily hy.

Pseudo Operator DONE, and Generalized Fibonacci Sequences

The pseudo operator DONE is useful in the context of accumulation mode and REPEAT loops (see pages 47 and 52).

DONE preserves the value immediately to its left, but deletes all prior values from the stack:

```
? 3 4 5 6 + DONE
= 11
```

Without DONE you would have gotten the error message “Argument count mismatch, remaining on stack: 2” — DONE tells Hypatia that the calculation has been completed, and to ignore all values that are left unused.

(DONE does not need to be the end of the calculation line: 3 4 5 6 + DONE 2 * would give 22.)

This seems weird — why would you enter numbers, which you then deliberately ignore? — but it is very useful for calculating generalized Fibonacci sequences, or for similar tasks.

The Fibonacci sequence starts with 0 and 1, and each following element is the sum of the two elements before it: 0 1 1 2 3 5 8 13 21 34 and so on.

Calculating Fibonacci numbers is not particularly interesting, as their values are known, but almost any generalization of Fibonacci sequences can easily be computed (tribonacci and n-nacci sequences, the wide field of Lucas sequences, random Fibonacci sequences, etc.). Generalized Fibonacci sequences can start with different numbers, can use non-integer numbers, can use more than two elements to calculate the following one, or can use other arithmetic functions than addition, etc. Padovan and Narayana’s cows sequences, for instance, can be computed with the help of WHISK.

To demonstrate the principle, let’s see how to calculate the first 20 Fibonacci numbers (usually the 0 at the beginning is not counted) — you will easily be able to generalize this method.

```
? 0                          here we could not say $ = 0, because we have to write 0 to hy
= 0
? 1 &                        hy is now 0 1
= 1
? DO 19 :: (hy) + DONE &     20 Fibonacci numbers, because we already have number 1
  0 1 + done &               the line including the insert file is shown once
= 6765                       the last result is shown
? HY
  0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 ...
```

(hy) + DONE & inserts the content of hy, calculates the sum of the two numbers before the + operator, disregards the numbers before them, and adds the result to the numbers already in hy.

With && instead of & the numbers would have been written in separate lines, instead of a single line of text — for Hypatia, this makes no difference.

Instead of displaying the sequence with HY, you can open it in your editor with EDIT, or copy it to the clipboard with COPY.

We could do all this with a script — lets call the script file fibonacci:

```
I1: &                   clears hy
I1: 0 &                 writes 0 to hy
I1: 1 &                 hy is now 0 1
(hy) + DONE &           adds the next Fibonacci number to the sequence
```

Note that it would not work to begin the script with `I1: 0` instead of the two lines `I1: &` and `I1: 0 &` because in a script results only get written to hy by the last calculation line, and by lines in accumulation mode — it is therefore necessary to clear hy with the command & and use accumulation mode to write first 0 and then 1.

To calculate the first 20 Fibonacci numbers we use a DO loop:

```
? DO 19 :: _fibonacci
= 6765                  the 19th result, which is the 20th Fibonacci number
? EDIT                  open hy to view the sequence
```

This works fine with 20 numbers, and even with more, but at some point the script will become slow. We can make it faster by using buffer mode (page 43) — this saves all the separate disk read and write operations:

```
I1: BUFFER START        starts buffer mode with empty buffer
I1: 0 &                 writes 0 to buffer (in a script, & is needed to write to hy)
I1: 1 &                 buffer is now 0 1
(buffer) + DONE &       adds the next Fibonacci number to the sequence
I*: BUFFER FLUSH        writes the content of the buffer to hy and ends buffer mode
```

You use this version of the script exactly like the original one — it works the same, just faster.

To eliminate the confusion between 20th result and 21st Fibonacci number, we can tell the script not to add a new number at the first loop pass — instead of `(buffer) + DONE &` we can write:

```
IF I 1 >> THEN (buffer) + DONE &
```

Now the count is unambiguous:

```
? DO 20 :: _fibonacci
= 6765                  the 20th result, which now is also the 20th Fibonacci number
```

Be aware, though, that Hypatia is not designed for number theoretical tasks. Independent of buffer mode, the results will become rounded and inexact, though still useful as approximations, once the numbers have more than 18 digits — this happens after the 87th Fibonacci number. In generalized Fibonacci sequences this can happen earlier or later, and, depending on your problem definition, may or may not be relevant.

Accuracy apart, there are limits to what Hypatia can do with generalized Fibonacci sequences, but with the help of variables and scripts they are not reached soon.

Conditional IF … THEN Clauses

The execution of a command or a calculation can be made to depend on whether a condition is met, that condition being a number which is either zero (false) or not zero (true).

```
IF variable or calculation THEN command, calculation or comment
```

The part after IF can be simply a variable, or any calculation — its result will not be displayed, not saved to hy, and $ and $$ will not be updated, it will only be used to decide whether the part that follows THEN will be executed or ignored.

If the condition is met (that is, if the result of the IF … THEN calculation is not zero), everything that follows THEN will be treated exactly as if it were a regular input line, with these minor exceptions:

IF … THEN clauses can not be nested. The RESET command, REPEAT commands and the Q/QUIT/EXIT command can not be used. The RUN command can be used, but not within a script, because scripts can not be nested.

In case that THEN is followed by a comment (anything that begins with #) this will be displayed and written to hy.log.

`IF ... THEN ENDLOOP` completes the script and then exits the loop. This has the same effect as IF … THEN $loop = 0, except that it does not set the variable $loop. This is the only way in which the ENDLOOP command can be used.

`IF ... THEN SKIP` skips the rest of the lines of a script when the conditon is met. If this happens in a loop, without the ENDLOOP command or $loop = 0 preceding it, it does *not* exit the loop.

`IF ... THEN ABORT` combines ENDLOOP and SKIP — it skips the rest of the script, and then exits the loop. See “Scripts, Prompts, Loops, Insert Files” (page 63) for an example.

Small numbers below the zero threshold (by default 1e-16, see chapter “Zero or Not Zero”, page 23) are *not* taken to be zero (false) by IF. If you want that to happen, add `ISNOT0` at the end of the calculation. If you want to invert the result — so that zero means true — add `IS0`. With `IS+` and `IS-` only positive/negative numbers mean true, with `IS0+` and `IS0-` positive/negative numbers and zero. IS0, ISNOT0, IS+, IS0+, IS– and IS0– consider absolute values below the zero threshold to be zero.

IF … THEN, ALSO: and ELSE:

If in a script you need more than one command or calculation executed (or comment displayed) when a condition is met, you can add one or more ALSO: lines after the IF … THEN condition:

```
IF variable or calculation THEN command, calculation or comment
ALSO: command, calculation or comment
```

You can also execute commands and calculations, and display comments, when the IF condition has *not* been met, by adding one or more ELSE: lines:

```
IF variable or calculation THEN command, calculation or comment
ELSE: command, calculation or comment
```

ALSO: and ELSE: lines can be used together. The ELSE: line reverts the result of the IF condition, so that any ALSO: line that *follows* ELSE: is executed when the original IF condition had *not* been met (another ELSE: would revert it back to the original result of IF). ALSO: and ELSE: do not need to follow IF … THEN immediately, there can be any number of lines in between.

IF … THEN, ALSO: and ELSE: can be used in I1: and I*: lines. At the end of the script, the result of the IF condition is lost.

In a loop, IF … THEN can be used in combination with PROMPT to let you decide whether to continue or to end the loop. For instance:

```
IF I 1000 MULTIPLE THEN SHOW $...
ALSO: PROMPT $loop
```

Every 1000 passes of the loop this shows the value of one or more variables, and prompts for a value of $loop — entering 0 ends the loop, everything else (including just pressing ENTER) lets it continue.

Centimeters, Feet and Inches

Here is an example in which a combination of Hypatia's features are used to do something that is less trivial than it may at first seem: convert centimeters to feet and inches.

The basic principle is simple: convert centimeters to meters so we can use Hypatia's :FT unit conversion operator, convert the meters to feet, and assign the value to variable $ft. Multiply the fractional part of $ft with 12 and we have the inches, take the integer part of $ft and we have the feet. The problem is, after rounding the inches to integers we can end up with, for instance, 5 feet 12 inches. To deal with that case, if we get 12 inches, we increase feet by one, and set inches to 0.

We can do this with a script — let's call it cm2feet, and create it by entering `EDIT cm2feet`

```
&
PROMPT $cm
$ft = $cm 100 / :FT
$inch = $ft FRAC 12 * 0 ROUND
$ft = $ft INT
IF $inch 12 EQUAL THEN $ft = $ft 1 +
ALSO: $inch = 0
$ft &
$inch &
HY
```

Line 1 clears the result fily hy, line 2 requests a value that will be stored in the variable $cm.

Lines 8 and 9: Because this is a script, where by default only the final result gets written to hy, setting accumulation mode by adding & is needed to write the values of $ft and $inch to hy.

`? _cm2feet` will prompt you for the centimeters — enter 176 and you will get 5 feet 9 inches:

```
  5 9
= 9
```

The command HY in the last line of our script shows the content of hy, `5 9` — this is the result we are looking for. The `= 9` result just repeats the inches — when a script contains calculations, the value of $ (the most recent calculation result) is always shown at its end, whether we need it or not. We could insert the line `$ = $cm` before HY, then the script's output would look nicer:

```
  5 9
= 176
```

With a few additions we can use our centimeter-to-feet-and-inches conversion routine with a loop to create a list of centimeter and corresponding feet and inches values — let's call this file cm2feet-list. Even if this may be of limited practical purpose, it shows what Hypatia can do.

The second line adds 99 to the loop index, which lets the table start at 100 cm. At the end of the loop, the list gets copied to the clipboard, and the according message is displayed:

```
I1: BUFFER START
I 99 + &&
$ft = $ 100 / :FT
$inch = $ft FRAC 12 * 0 ROUND
$ft = $ft INT
IF $inch 12 EQUAL THEN $ft = $ft 1 +
ALSO: $inch = 0
$ft &
$inch &
I*: BUFFER FLUSH
I*: COPY
I*: # List copied to clipboard
```

Line 1 sets buffer mode at the beginning of the loop, line 2 writes the centimeters (index plus 99) to a new line. The rest is still the same, except for the `I*:` lines at the end.

Now we can create a cm-feet-inches list — let's end it at 200 cm, after 101 passes:

```
? DO 101 :: _cm2feet-list
```

You can now paste the list to any Windows application (or you can open it with the command EDIT, for which we wouldn't have needed the `I*: COPY` line at the end). Or, instead of the last three lines we could write

```
I*: BUFFER SAVE cm-feet-and-inches
```

to write the list to a file (remember: no spaces in file names!) — ayway, we get:

```
100 3 3
101 3 4
102 3 4
... ...
198 6 6
199 6 6
200 6 7
```

By the way, converting feet and inches to centimeters is easy: divide inches by 12, add to feet, convert to m, multiply by 100 to get cm, and round to 0 decimal digits — let's do this for 5 feet 9 inches:

```
? 5 9 12 / + :M 100 * 0 ROUND
```

We can create a user-defined 2-argument operator (see page 40) to convert feet and inches to cm, let's call it @feet2cm:

```
@feet2cm = 12 / + :M 100 * 0 ROUND
```

And if we add this line to hy.ini, our operator will be always available:

```
? 5 9 @feet2cm
= 175
```

(cm being finer graded, 174, 175 and 176 cm each correspond to 5'9")

Scripts, Prompts, Loops, User Defined Operators

The following example will show several of Hypatia's features at work.

Let's write a simple script that calculates the body mass index — let's call it bmi — which requires input of the weight in kg and the height in cm.

```
# Enter weight in kg, then height in cm
PROMPT $kg
PROMPT $cm
$kg $cm 100 / SQ / 1 ROUND
```

(The body mass index is weight in kg divided by the square of body height in meters, therefore centimeters have to be converted to meters by dividing them by 100. The result is rounded to one decimal digit.)

If you happen to know your weight in pounds (let's say, 145), and your height in feet and inches (let's say, 5'6"), you can do the necessary conversions at the input prompts (about @feet2cm see above):

```
? _bmi
# Enter weight in kg, then height in cm
? $kg = 145 :kg
  $kg = 65.7708936
? $cm = 5 6 @feet2cm
= 5 6 12 / + :M 100 * 0 round
  $cm = 168
= 23.3
```

Now let's assume that you want to calculate body mass indices for a group of people. Instead of repeating the process several times by entering REPEAT or again calling _bmi, you could use bmi in a loop — but if you do that with the above script, there are three problems:

- in a loop, the comment is not shown,
- in a loop, the result of the calculation is not shown,
- and finally, you would need to specify how many calculations you want to do in the DO command, for otherwise you could only end the loop by force (closing the console window, or pressing Ctrl/C).

To address these problems, we have to modify the script:

```
#: Enter weight in kg (0 to quit), then height in cm
PROMPT $kg
IF $kg IS0 THEN ABORT
PROMPT $cm
$kg $cm 100 / SQ / 1 ROUND
IF ISLOOP THEN =
```

This script can be used for a single calculation by entering `_bmi`, or for a number of consecutive calculations by entering `*_bmi` (which is short for `DO * :: _bmi`.)

Line 1: When a comment line begins with #: it is shown regardless of whether the script is called directly or in a loop.

Should you want the comment to be displayed only once, the line would be:

```
I1: # Enter weight in kg (0 to quit), then height in cm
```

Line 3: Entering a weight of 0 triggers the ABORT command, which combines ENDLOOP and SKIP — ENDLOOP declares the exit condition to be met, and SKIP jumps to the end of the script.

Lines 5 and 6: If the script is called directly, its result is shown, but in a loop the result is only shown at the end of the loop. Line 6 determines whether the script is run in a loop — if the pseudo constant ISLOOP is 0, then this line does nothing because the result has already been shown, but if ISLOOP is 1, the command = shows the result.

Nested Loops

Hypatia's loop commands can not be nested, but you can achieve the effect of nested loops by using your own loop index variables.

In a programming language, a nested loop may look something like this:

```
FOR j = startj TO endj BY incrementj DO
    FOR k = startk TO endk BY incrementk DO
    ...
    END FOR
END FOR
```

When you need nested loops, you are probably familiar with computer languages in which you can do this, but, while it needs a bit more effort, Hypatia lets you do it, too — here is the basic principle:

```
I1: $j = startj
I1: $k = startk
...
$k = $k incrementk +
IF $k endk - IS+ THEN $j = $j incrementj +
ALSO: $k = startk
IF $j endj - IS+ THEN ENDLOOP
```

In an actual script the start, end and increment values for the two loop variables can be numbers, variables, or arithmetic expressions.

In line 4 the inner loop variable k gets increased — if its value exceeds the end value the outer loop then variable j is increased (line 5), and the innler loop variable k is reset to its start value (line 6). When the outer loop variable exceeds its end value, ENDLOOP exits the loop.

Note that this has nothing to do with Hypatia's own loop index I — both the DO and REPEAT commands require you to specify the (maximum) number of repeats, but you just have to give a number that is high enough for your nested loops to reach the end (by default * stands for 100000).

Let's do a simple example — let's look at a 12 x 12 multiplication table, and ask, what is the total sum of the results it contains? Start values and increments of both loop variables are 1, and their end values are 12. Let's call our script file mtable:

```
I1: $sum = 0
I1: $j = 1                          start value of outer loop variable $j
I1: $k = 1                          start value of inner loop variable $k
$sum = $sum $j $k * +
$k = $k 1 +                         inner loop variable $k gets increased by 1
IF $k 12 >> THEN $j = $j 1 +        if $k is greater than 12, then $j is increased by 1 …
ALSO: $k = 1                        … and $k is reset to 1
IF $j 12 >> THEN ENDLOOP            if $j is greater than 12, the loop is exited
I*: $sum
```

Note that the last but one line says ENDLOOP, not ABORT — with ENDLOOP the following line is still executed before the loop is ended, with ABORT it would be skipped and we wouldn't see the result.

This last line is the first calculation line — that is, only there, at the last pass of the loop, we have a result that updates $ and hy, and that gets displayed at the end of the loop. Let's do our calculation:

```
? *_mtable
Loop exit condition met in pass 144
= 6084
```

The "Loop exit condition met …" line tells you that the task was completed (if, for instance, you had said DO 100 :: _mtable, it wouldn't be — you'd still see a result, but it would be wrong).

It isn't necessary here, but you could also add a comment line that tells you when at the end of the loop a certain condition has been met — in our case, that the outer loop variable has reached 13:

```
I*: IF $j 13 EQUAL THEN # Loop completed
```

or that it has not been met ($j is less than 13):

```
I*: IF $j 13 << THEN # Warning: Loop not completed!
```

or you could do both:

```
I*: IF $j 13 EQUAL THEN # Loop completed
I*: ELSE: # Warning: Loop not completed!
```

or — just to demonstrate how this works — you could display a line during the loop, when a condition is met. This line might be inserted before line 5, which increases $k = $k by 1:

```
IF $j 7 EQUAL $k 1 EQUAL * THEN # Half way through the loop
```

(The multiplication at the end of the IF clause makes the result false when either of the EQUAL results is zero, it is the equivalent of a logical *and*. The equivalent of a logical *or* would be an addition.)

Nested Loops in Monte Carlo

To illustrate how Hypatia can perform Monte Carlo experiments that involve nested loops, here is an (not very exciting) example: When you roll a dice until you've rolled a six — let's call that a round — what will the average number of eyes per round be? Let's roll the dice, add up the eyes, and count the number of rounds — a new one each time we roll a six. Let's call this script roll6:

```
I1: $sum = 0
I1: $n = 0
$roll = 1 6 RANDINT
$sum = $sum $roll +
IF $roll 6 EQUAL THEN $n = $n 1 +
I*: $sum $n /
```

Now we run this script in a loop. The number of passes, or how often the dice is rolled, is 100000 by default, but you can change it with the MAXLOOP command (page 46):

```
? _*roll6
= 21.2775408
```
(for instance — even with 100000 rolls of the dice the result varies)

There is a small inaccuracy, though: the loop stops at the maximum number of passes — the last round remains incomplete, but the eyes we've rolled since the last six has been added to $sum. The effect will be minor, but let's fix this error:

```
I1: $eyes = 0                                   sum of eyes in current round
I1: $sum = 0                                    total sum of eyes
I1: $n = 0                                      number of rounds
$roll = 1 6 RANDINT                             roll the dice
$eyes = $eyes $roll +                           update sum of eyes in this round
IF $roll 6 EQUAL THEN $sum = $sum $eyes +       update total sum after a "6" …
ALSO: $eyes = 0                                 … reset sum of eyes in current round …
ALSO: $n = $n 1 +                               … and add 1 to number of rounds
I*: $sum $n /                                   at the end, calculate eyes per round
```

The total sum only gets updated when a round is completed, the eyes rolled after the last "6" will not be counted.

But what if we want to see the results of all the throws? Here is a modified version of the above script that writes them to hy, one line per round. Because a total of 100000 write-to-disk operations would take their time, we use buffer mode, write the results to the buffer, and only at the end write the content of the buffer to hy — or even better, we could use a different output file:

```
I1: BUFFER START                                turn buffer mode on
I1: $eyes = 0                                   sum of eyes in current round
I1: $sum = 0                                    total sum of eyes
I1: $n = 0                                      number of rounds
$roll = 1 6 RANDINT                             roll the dice
$roll &                                         this writes $roll to the buffer
$eyes = $eyes $roll +                           update sum of eyes in this round
IF $roll 6 EQUAL THEN $sum = $sum $eyes +       update total sum after a "6" …
ALSO: $eyes = 0                                 …  reset sum of eyes in current round …
ALSO: $n = $n 1 +                               …  add 1 to number of rounds …
ALSO: &&                                        …  and add a line break to the buffer
I*: BUFFER SAVE filename                        at the end, write buffer to a file …
I*: BUFFER DISCARD                              … exit buffer mode …
I*: $sum $n                                     … and calculate eyes per round
```

Hypatia can handle more complex tasks than we have discussed, involving any number of variables and IF THEN ELSE conditions, but we'll stop here.

List of Commands

Mode Settings and Output Format Commands

AUTO$ ON | OFF	set auto include $ mode ON (default) | OFF	19
ECHO ON | OFF	set echo mode ON | OFF (default)	34
LOG ON | OFF	start | stop (default) logging input and output to file hy.log	33
INTEGER ON | OFF	set integer bias ON (default) | OFF	25
USE DEG	angle unit is degrees (do not confuse with the DEG operator)	28
USE RAD	angle unit is radians (default; do not confuse with the RAD operator)	
AUTO$, ECHO, LOG, INTEGER, USE	— show current mode	
FSHORT, FLONG	show up to 9 digits (does not affect FDEC n), show up to 15 digits (default)	24
FDEC	(default) decimal format, scientific notation for large or small numbers	26
FDEC n	show results with n digits after decimal point (before decimal point if n is negative)	
FSCI	show results in scientific notation, 15 digits (9 digits with FSHORT) and exponent	
FHEX	use hexadecimal format for displaying results (only valid for positive integer numbers)	
FHEX n	use hexadecimal format, with at least n digits (2 to 12)	
FBIN	use binary format for displaying results (only valid for positive integer numbers)	
FBIN n	use binary format, with at least n digits (2 to 48)	
FLAKH	decimal format, numbers ≥ 1E5 are shown in lakh, numbers ≥ 1E7 in crore	
FMILLION	decimal format, numbers ≥ 1E6 are shown in million	
F' ON | OFF	apostrophe format ON | OFF (F' toggles apostrophe format ON/OFF)	
RESET	delete all variables, reset all options to default or to command line parameters	

For accumulation mode, silent mode and debug mode see “List of Control Symbols”, page 71

Variables

STO $var	assign value of last result to variable (create variable if it doesn’t already exist)	21
$var = …	assign number or calculation result to variable (create if it doesn’t already exist)	
PROMPT $var	prompt user for value of variable (number or calculation)	
DEL $var	delete variable	
SHOW	display all variables; if angle mode is set to degrees, this will be displayed	
SHOW $var1 $var2 …	display these variables (can include loop index I and loop timer TIME)	
SAVE filename	save variables to file (anything after filename is a comment line)	22
_filename	or RUN filename: retrieve variables from file	

Results

=	show last result (value of $) in currently chosen format	26
==	show last result (value of $) in decimal format with up to 18 digits	26
HY	show content of result file hy	32
$	write last result (value of $) to hy in the currently specified format	32

User Defined Elements

@ude = …	assign numbers and/or operators to UDE (create UDE if it doesn't exist)	37
DEL@ @ude	delete user defined element	
SHOW@	display all user defined elelements	
SHOW@ @ude1 @ude2 …	display these user defined elements	
SAVE@ filename	save user defined elements to file (anything after filename is a comment line)	
_filename	or RUN filename: retrieve user defined elements from file	

Clipboard

COPY	copy the result file hy to the clipboard	29
COPYALL ON	set copy to clipboard mode ON (all results will be copied to the clipboard)	
COPYALL OFF	set copy to clipboard mode OFF (default)	
COPYALL	show current copyall mode	
COPIN	copy last calculation input line to clipboard	

Files

FILES	show all files in Hypatia's program folder	30
EDIT	open result fily hy to view or edit (short for EDIT hy)	30
EDIT filename	open editor to view, edit or create a file in Hypatia's program folder	
EDIN	open editor to view or edit file hyin (short for EDIT hyin)	
EDINI	open editor to view or edit file hy.ini (short for EDIT hy.ini) default editor is Windows Notepad (notepad.exe)	
EXTEDITOR filename	replace notepad.exe with an editor of your choice	30
RUN filename	execute a script file, each line is treated as an input line	34
_filename	same as RUN filename, but without displaying intermediate results	
(filename)	insert content of file in input line	35
&	clear result file hy (if buffer mode is on, clear buffer)	31
&&	add a line break to result file hy (if buffer mode is on, to buffer)	31
HY	show content of result file hy	33

Buffer

BUFFER START	turn buffer mode on, clear buffer	43
BUFFER SHOW	display buffer content	
BUFFER SAVE filename [comment]	save buffer to file (anything after filename is a comment line)	
BUFFER FLUSH	write buffer to hy, clear buffer, turn buffer mode off	
BUFFER DISCARD	clear buffer, turn buffer mode off	
BUFFER	dispay buffer mode (ON, ON but empty, or OFF)	
(buffer)	if buffer mode is on, insert content of buffer in input line	

Scripts

RUN filename	execute script file	34
_filename	same as RUN filename, but only end result will be displayed	

Repeat and Loops

REPEAT	repeat the most recent calculation	44
MAXLOOP n	set max. number of passes in a loop (by default 100000, max. 1E7)	46
REPEAT n	loop command, repeat the most recent calculation n times	47
REPEAT *	loop command, n = maximum (by default 100000) minus 1	
DO n :: …	loop command	47
DO * ::	loop command, n = maximum (by default 100000)	
*_filename	short for DO * :: _filename (run script in loop with max. n of passes)	
	loops are exited when variable $loop is set to 0 within the loop	49
I1: …	execute line only when loop index is 1, or when not in a loop	54
I*: …	execute line only at the end of loop	54
IF … THEN …	execute command or calculation only when condition is met	60
IF … THEN ENDLOOP	exit loop after last line in the script when condition is met	
IF … THEN SKIP	skip the rest of the script when condition is met	
IF … THEN ABORT	combines ENDLOOP and SKIP	
	skip the rest of the script when condition is met, then exit loop	
ALSO: …	execute when preceding IF/THEN condition was met	60
ELSE: …	execute when preceding IF/THEN condition was not met	
$loop =	setting variable $loop to 0 exits the loop, the same as ENDLOOP	49

IF does *not* observe the zero threshold, it only returns false for values exactly zero.
$loop only exits the loop when its value is exactly zero.

Help

	display program version and basic help info
?	display program version and settings
HELP	display help overview
HELP ?	display available help topics

Quit

Q or QUIT or EXIT quits the calculator.
You can also just close the console window, Hypatia keeps no files open that could be corrupted.

List of Control Symbols

comment line (will not be displayed in a loop) 11
Comments following I1:, I*:, IF … THEN, ALSO: or ELSE: will always be displayed if true

#: comment line 11
Comment lines in a script that start with #: will be displayed in loops

comment line 11
Comment lines in a script that start with ## will not be displayed, except in echo mode

… … **##** comment 11
Anything in a command or calculation line after ## is a comment

calculation line **&** 31
& at the end of a calculation line sets accumulation mode for this line:
instead of overwriting hy the present result will be appended, separated by a space

calculation line **&&** 31
Same as &, but sets "new line" accumulation mode for this line:
the new result will be added to hy in a new line

calculation line **#** 32
at the end of a calculation line sets silent mode for this line:
result file hy will not be updated

calculation line **?** 72
Debug mode, intermediate results will be shown

Permitted combinations (&& can be used instead of &):
… … **? #** … … **? &**… … **& ?**

Accumulation, silent and debug modes are only set for the particular line
In scripts hy only gets updated at the end, or by lines with accumulation mode set
& or # can be used in the script's last calculation line

REPEAT n **?** 45
DO n **?** :: … … 45
Show results for each loop pass

_filename 34
Same as RUN filename, but only final result will be shown

… **(**filename**)** … 35
Insert content of file into the input line

Do not confuse the & and && control symbols at the end of calculation lines with the commands & and && (& clears the file hy, && adds a line break), nor with the &h and &b notation of hexadecimal and binary numbers.
Do also not confuse the control symbol ? at the end of a calculation line, or that after the REPEAT command, with the command ? which shows current program settings.

Debugging

When you add a question mark at the end of a calculation line, Hypatia tells you in detail how the line is processed — each operator is shown followed by its result, and by its position in the stack. This can be helpful when you get an error message which may not be obvious, when you get an unexpected result, or when you just want to watch Hypatia do its calculations.

If you want to see this for the line you had just entered, you can use (hyin) to repeat that line — using it as an insert file in an otherwise empty input line — and add the question mark after it.

Instead of using (hyin), you can also press the ARROW UP key to scroll through the list of past input lines, go to the end of the line by pressing the END key, add a space and the question mark, and press ENTER.

Note that `REPEAT ?` would not work!

Here is a simple example, calculating (once again) the hypotenuse of a rectangular triangle:

```
? 3 SQ 4 SQ + SQRT
= 5

? (hyin) ?
[ op: sq | result: 9 | position: 1 ]
[ op: sq | result: 16 | position: 2 ]
[ op: + | result: 25 | position: 1 ]
[ op: sqrt | result: 5 | position: 1 ]
= 5
```

When you have found an error in your calculation line, you can scroll up, edit it and press ENTER, or you can use EDIN to open, edit and save the fily hyin and then use the REPEAT command.

In the unlikely case that you want to combine debugging and accumulation modes, ? and & (or &&) can be used in either order, but need to be separated by a space.

In a loop, the debug question mark is ignored.

Command Line Options

When you start Hypatia from the Windows command line or the Windows PowerShell, you can use the following options. If you start Hypatia from the Windows desktop, you can add them to the "Target" in the desktop icon's "Properties" dialogue.

As all Hypatia input, the options are case insensitive.

`-c`	files location is current folder (default: program folder)
`-d`	set angle unit to degrees (default: radians) — the same as USE DEG
`-e`	echo mode is on (default: off) — the same as ECHO ON
`-i`	do NOT execute ini file hy.ini
`-l`	logging to file hy.log is on (default: off) — the same as LOG ON
`-r filename`	run filename

The command line options -d, -e and -l have the same effect as the respective commands would have in the hy.ini file.

The file specified with the -r option is executed after hy.ini is run. If you want to use that file instead of hy.ini, you have to use both -i and -r.

The script file called by -r can end with a QUIT command (or Q, or EXIT).

If Hypatia is started with the -i option, then a hy.ini file is not created if it does not exist.

The -c option applies to all files that Hypatia reads or writes, inlcuding hy.ini, hy, hyin, and hy.log.

The command RESET resets Hypatia to the state when it was started with the current command line options, but does not execute the -r option.

See also "Command Line Calculation Mode" (page 74).

Command Line Calculation Mode

From the operating system's command line you can call Hypatia in "command line calculation mode," which means that you can write a calculation which Hypatia will execute, display the result, and then terminate. The calculation result is also written to the file hy, but the file hyin is not updated.

Hypatia's program folder needs to be included in the system path. If you use Hypatia from the Windows PowerShell instead of the console command line, you have to write hy.exe instead of hy.

All input is case insensitive.

hy.ini is executed before the calculation is performed, but if it does not exist, it is not created.

Hypatia's full syntax is:

```
hy [options] [calculation]
```

For instance, to see the square root of 2, enter

```
C:\ ... >hy 2 sqrt
C:\ ... >= 1.4142135623731
```

To see the sine of 45 degrees, enter

```
C:\ ... >hy -d 45 sin
C:\ ... >= 0.707106781186548
```

To roll a dice — that is, get a random integer number in the range of 1 to 6 — enter

```
C:\ ... >hy 1 6 randint
C:\ ... >= 2
```

(for instance)

In a command line calculation you can use insert files. For instance, you can create a file dice that consists of the line `1 6 randint` (you can add a comment line), and you can then roll a dice from the operating system's command line:

```
C:\ ... >hy (dice)
```

Or — to demonstrate how this works — you can roll the dice twice, and add up the two results:

```
C:\ ... >hy (dice) (dice) +
```

(In this case, because our example file can be used both as an insert and a script file, you could also use dice as a script from the command line: `C:\ ... >hy -r dice` — this would open Hypatia, run dice, and keep Hypatia open.)

Because hy.ini gets executed before the command line calculation is performed (unless you use the -i option), in case that any variables or UDEs get defined in hy.ini, you can use them in the calculation.

Commmand line calculation mode and option -r can not be used together.

You cannot use a Hypatia command instead of a calculation. If you try to, Hypatia will open and give you an "option … not recognized" warning.

Command line calculation mode will not give you detailed error messages. When the calculation can not be performed, Hypatia will display "Command line calculation error" and terminate.

Hypatia for Linux

While in principle the Phix source code should be cross-platform, there is currently no Linux version of Hypatia.

Under Linux Hypatia's input line editor does currently not work, which means that you cannot move the cursor in the input line to edit the text, nor can you scroll through past input lines.

Under Linux Hypatia does not know the size of its console window and assumes a width of 80 characters.

To use Hypatia with Linux, you need to install Phix and compile Hypatia, which is quickly done. The current version of Hypatia is written for Phix 1.0.4, it is possible that later versions of Phix may require minor adjustments of the code.

Under Linux you need to add two lines to the hy.ini file: the Hypatia commands EDIT and EDIN will only work if you specify an editor, and the commands COPY, COPYALL and COPIN only work if you specify a copy-to-clipboard command.

```
EXTEDITOR editor
CCCOMMAND copy text file to clipboard
```

Under Windows, defaults are `notepad.exe` and `clip <`, but under Linux no such defaults exist. These commands will probably work, if you have gedit and xclip installed:

```
EXTEDITOR gedit
CCCOMMAND xclip -sel c <
```

Be aware that under Linux, some of Hypatia's features may not (yet) work as expected. Please let me know about your experiences — robert@drs.at

Notes on Hypatia's Internal Workings

Processing Input Lines

You do not need to know any of this to use Hypatia, but it may help to understand why certain things work the way they do.

When Hypatia processes an input line, the following lines are taken care of immediately:
Blank lines, Comment lines, the RESET command, the REPEAT command, the Q/QUIT/EXIT command.

For other lines, Hypatia takes the following steps, in this order (details regarding scripts, loops and if/then conditions are omitted here). Actual calculations are performed in steps 6 and 7.

1. Names of insert files are replaced with the contents of the files. If an error occurs, end.
2. UDEs are resolved, except for arguments of the commands del, del@, show, show@, save, save@, run, and edit. In an assign command, the first word (UDE name) is preserved. Unresolved UDE names remain in input line.
3. The input line is split into its words — this is the stack.
4. The first word determines if it is a command or a calculation line (for exception see step 5).
5. If it is a command, it gets executed. Done.
6. If the second word is = and the first word is a UDE name, the text following = is assigned to the UDE, and the file hyin gets updated. Done. If the second word is = and the first word is a variable, this is a command that assigns a calculation result to a variable. A flag is set, and the first two words ($variable =) are removed from the stack before step 7.
7. All variables, constants and pseudo constants are replaced with their values. Control symbols at the end of the line (for accumulation, silent, and debug mode) are read and removed. From that point on, the stack contains only numbers and operators.
8. The calculation is now performed, each operator replacing 1, 2 or all numbers to its left, and itself, with the result of the calculation that it stands for (except for the pseudo operator WHISK, which only removes two numbers, and the pseudo operators A and B, which only replace themselves).
9. If the assign flag was set (step 6), the result is assigned to the variable. Step 10 is omitted.
10. If no error has occurred, the result is displayed, $ and $$ are updated, and the file hy is updated (unless silent mode is set).
11. The file hyin gets updated even when an error has occurred, unless the line has started with an unrecognized operator, in which case it is assumed to have been a mis-spelled command.

Along with the variables (including $ and $$) and the constants PI, E and PHI, also DUP, RAND, I and ISLOOP are replaced by numbers in step 7, before calculation begins. In Hypatia's terminology, these are pseudo constants.

A and B, though, along with WHISK, make it to step 8. From the user's perspective, A and B look like variables, or maybe pseudo constants, but to Hypatia everything in step 7 is either a number or an operator. A and B are zero-argument operators, replacing themselves with the values stored by WHISK, which, unlike variables, constants and pseudo constants, are not yet known or can be determined in step 7.

Rounding

The rule says that values ≥ …5…, that is including …5, are to be rounded up. Because of tiny inaccuracies that are unavoidable with a limited number of digits and a binary number system, it is possible that a value that should end with …5, and is shown as ending with …5, actually ends with …49999999 etc., and would get rounded down. (This is somewhat related to, but independent of the "Integer or Not Integer" issue, see page 24.)

Hypatia's own rounding function deals with this issue by adding 1 at the 18th digit of the number that is to be rounded (except when rounding to more than 15 significant digits, in which case the position at which 1 is added is moved further back).

I see currently no reason not to use this mechanism (its general effect on results is negligible), but you can use an (otherwise) undocumented command to switch to the programming language's (that is, Phix's) built-in rounding routine:

USE STDROUND	use standard rounding function
USE HYROUND	use Hypatia's rounding function (default)

To illustrate the effect:

```
? USE STDROUND      standard rounding function
? 42.875 CBRT       cubic root of 42.875 is 3.5 …
= 3.5               … but actually the computed value is some 4e-19 less than 3.5
? 0 ROUND           rounding to integer
= 3                 rounded off, while 3.5 should be rounded up

? USE HYROUND       Hypatia's rounding function
? 42.875 CBRT
= 3.5
? 0 ROUND           rounding to integer
= 4                 Hypatia's rounding function rounds up as expected, not down
```

Rounding not only takes place when you use a rounding operator (ROUND or ROUNDS), but also when numbers are shown that (rightly or wrongly) have more digits than the output format permits (a maximum of 15 for non-integer values). FDEC and FDEC n use Hypatia's rounding (unless you have used the USE STDROUND command), FSCI always uses standard (that is, Phix's) rounding.

```
? FDEC 0            output format is decimal with 0 digits after decimal point
? 42.875 CBRT
= 4                 result 3.5 is rounded up, as expected
? USE STDROUND      standard rounding function
? =                 display last result
= 3
```

The actual value is still the same (unlike with the above rounding examples), approximately 3.5, but the rounding function determines how it is shown. With default FDEC format (up to 15 digits), though, the rounding method will rarely make a difference, and if, then only at the 15th digit.

Writing to the Clipboard

Hypatia calls the Windows utility clip.exe to copy the contents of the files hy or hyin to the clipboard. This dates back to an old problem, still with Euphoria 3.1 — there was a routine to write to the clipboard, and it worked fine, but after calling it repeatedly it tended to cause the program to malfunction. This was difficult to test and even more difficult to explain, and I dealt with it by warning users to use COPY only sparsely. With Phix now, the copy to clipboard routine is part of a deprecated library — letting the operating system do the work is clearly the safest solution, even if it looks a bit clumsy.

Loops

Some things can only be explained historically. The ability of the input editor to scroll back to earlier input lines came rather late — before that, I added REPEAT to allow repeating a calculation or a script, and, as a rather clumsy way to edit the input, you could edit hyin before using REPEAT. Once the REPEAT command existed, the possibility suggested itself to enhance it with a loop function. That you had to perform a calculation or execute a script before repeat-looping it looked like a feature — this way, you always made sure that it worked correctly. When your script is already well tested, it seemed needlessly cumbersome, though — so, I added the one-line loop statement DO — and, with REPEAT loops already in place, the simplest way to implement it was to graft it to the REPEAT command: split the input line at the :: separator, write the second part to hyin, in the first part replace DO with REPEAT, then feed it to the code that processes REPEAT, together with a flag that says it's a DO loop. Not elegant, but didn't take much effort and avoided introducing additional complexity, and it works.

Normal Distributed Random Numbers

Hypatia uses the Marsaglia polar method to deliver normal distributed random numbers. It produces a pair of random numbers — to further randomize the result, one of them is chosen randomly.

To avoid extremely unlikely values of normal distributed random numbers, those that deviate from the mean value by more than 6 sigma are rejected (the probability of which is about one in a billion).

License

No legalese, just very simple rules.

You may do with the executable file, the documentation, and the source code whatever you want, under three conditions:

- that you credit the original author,
- that you document the changes you have made, at least in general terms, and that
- under no circumstances you are allowed to modify this program in a way that could be potentially harmful to the user, or to distribute it in a potentially harmful way, for instance with an installer that installs other software or makes undocumented or unrequested changes to the user's computer.

The executable, the source code and the documentation are offered “as is.” No warranty of any kind is provided or implied. All support is given voluntarily.

If you improve or otherwise modify the program, port it to Linux etc., please let me know.
I’d also like to hear if you find the program useful, or if you have any suggestions or complaints.

Important:
Please tell me if you’ve found or suspect a bug, if you found an error in the documentation, or if you find parts of the documentation to be unclear, incomplete, or difficult to understand.

Contact: Robert Schaechter, Vienna/Austria, robert@drs.at
Please put Hypatia in the subject line of your message.

Release Notes

Minor changes and improvements and older bug fixes are not listed

Version 4.0, April 2, 2024
- Hypatia is now a 64-bit program, and requires a 64-bit Windows system
- Floating point numbers are shown with up to 15 digits, integers with up to 18 digits
- Default zero threshold is now 1e-16
- Many internal changes to make use of the enlarged range of numbers, and to ensure the accuracy of results
- Large numbers can be written with inserted apostrophes to structure them
- New oputput format commands F' ON/OFF, FSHORT, FLONG
- Output format commands do not update hy
- New operators ISPOSINT, DIGITS, LOG^
- Operator LOG (also alternative name LOGN) renamed LN
- Operator MOD now accepts non-integer argument, new operator IMOD requires integer arguments
- New commands INTEGER ON/OFF, EDINI, BUFFER DISCARD, ==
- New constant TAU
- ESC key can be used in input editor to move to beginning of line and back to current line

Version 3.4, February 12, 2024
- Numbers can now be written as lakh, million and crore (new LAKH, MILLION and CRORE 1-argument operators)
- New number output format commands FLAKH, FMILLION and FCRORE
- New pseudo constant TIME (time in seconds since start of a loop)
- REPEAT * is now permitted
- DO * ? and REPEAT * ? are now permitted, only the first 40 results are shown
- PRIME operator renamed ISPRIME
- Attempt to calculate tangent of 90° now reliably prevented from crashing Hypatia

Version 3.3, January 24, 2024
- New command BUFFER SAVE
- Save commands (SAVE, SAVE@ and BUFFER SAVE) now check for prohibited file names

Version 3.2, January 20, 2024
- New compare operators <<, <=, >>, >= (also, == and <> can now be used instead of EQUAL and UNEQUAL)
- ITEM operator now accepts negative index arguments (counts backwards from end of list)

Version 3.0, January 4, 2024
- New feature “user defined elements,” with new commands @ude = …, SHOW@, DEL@ and SAVE@
- New feature result buffer, new commands BUFFER START, BUFFER SHOW, BUFFER FLUSH, and BUFFER
- New feature inline comments
- New output format command FSCI
- INIT command renamed RESET (INIT still works)
- Command HY only displays content of hy when max. 40 lines / 4000 characters, the same applies to BUFFER SHOW
- Tab characters in script or insert files caused errors, they are now converted to spaces
- n-argument operators N+, N0 and N- now observe the zero threshold
- Max. number of digits after decimal point in FDEC n command is now 12 instead of 10
- Hypatia now uses Phix version 1.0.4

Version 2.6, September 7, 2023
- New n-argument operator ITEM
- New command ABORT combines the commands ENDLOOP and SKIP
- LIST command renamed FILES
- COPY command now works independent of COPYALL mode (that is, it also works when COPYALL is ON)

Version 2.51, March 28, 2023
• Basic info text displayed at first start (when hy.ini not found)

Version 2.5, March 20, 2023
• New n-argument delimiter |
• Fixed bug introduced in 2.4, which made 1E-n displayed as 1 for n > 10
• Fixed bug that disabled EXP2 operator

Version 2.4, March 19, 2023
• Results are now shown with up to 12 instead of 10 digits

Version 2.3, March 9, 2023
• New commands SKIP, = and ==
• New pseudo constant ISLOOP
• #: comment lines are shown in loops
• *_filename is short for loop command DO * :: _filename

Version 2.2, March 2, 2023
• New command PROMPT
• Changed the term “run file” to “script” or “script file”

Version 2.1, February 22, 2023
• New operator MULTIPLE
• New command MAXLOOP
• SHOW list of variables can include loop index I

Version 2.0, February 2, 2023
• New loop commands DO, ENDLOOP. Max. number of loop passes increased to 100000
• New conditional commands ALSO: and ELSE:
• New command AUTO$
• New operators EQUAL, UNEQUAL, UPLIMIT and LOLIMIT
• New operator PRIME
• New operators IS0+, IS0- and ISNOT0, operators SIGN+, SIGN0 and SIGN- renamed IS+, IS0, and IS-
• New unit conversion operators :NAUTMI and :NAUTKM
• SHOW can be followed by list of variables to be displayed

Version 1.2, January 14, 2023
• Conditional IF … THEN clauses
• Loop index conditions I1: and I*:
• New command DEL
• New operator SIGN- (renamed IS- in version 2.0)
• Variable zero threshold for SIGN operators
• New constant PHI (golden ratio)

Version 1.0, December 14, 2022
• Some command line calculation issues fixed

Version 0.993, December 4, 2022
• Command line calculation mode didn’t work when hy.ini wasn’t empty

Version 0.992, November 22, 2022
• LN and LG operators renamed LOG and LOG10
• New operators LOG2, EXP, EXP10, EXP2, CUBE, CBRT
• New operators //, GATE, SIGN0
• New operators GMEAN, HMEAN

• Assign command $variable = can now directly assign calculation results to variables
• Variabe $loop provides loop exit condition

Version 0.991, November 12, 2022
• Alternatively to USEDEG and USERAD also USE DEG and USE RAD are permitted
• Source code ready for Linux, as far as currently possible

Version 0.99, November 2, 2022
• New pseudo-operator DONE
• New n-argument operator MED
• New command HY
• Calculation lines with accumulation mode in scripts now update hy
• ## comment lines in scripts will not be displayed

Version 0.95, October 24, 2022
• New RANDND operator for creating normal distributed random numbers
• New N+, N0 and N- operators
• New SIGN+ operator

Version 0.94, October 22, 2022
• Input editor allows scrolling through prior input lines and re-use them
• New pseudo-operator WHISK makes user-defined 2-argument operators possible
• SIGN now returns 0 when argument is 0, not +1 (this had been deliberate, but was not a good idea)

Version 0.93, October 18, 2022
• Command & to clear fily hy
• “new line” accumulation mode &&
• Pseudo constant I (REPEAT loop index +1)
• Console window can now safely be closed to exit program, even when logging mode is on

Version 0.92, October 12, 2022
• New commands EDIN, EXTEDITOR, and REPEAT
• Loop command REPEAT n
• Shortcut _filename for RUN filename
• ECHO, LOG and COPYALL commands changed: ECHO [on|off], LOG [on|off], COPYALL [on|off]
• Silent mode to keep result fily hy from being updated
• Only the most recent calculation line is saved to file hyin, command lines are not saved
• hyin now contains the original input line, not data inserted with ()
• SAVE command now gives error message when no variables are defined
• File and variable names are now consistently lower case

Version 0.9, October 6, 2022
• EDIT command calls Notepad editor to view, edit or create files in program folder
• RANDI operater renamed to RANDINT, to distinguish more clearly from RAND pseudo-constant

Version 0.73, August 16, 2022
• Accumulation mode combines results of two or more calculation in one line of text
• Comment lines allowed in “insert” files

Version 0.72, June 7, 2022
• Input line can now be edited
• LIST command shows files in the Hypatia program folder

Version 0.7, May 30, 2022 — This is a major step towards version 1.0, one day …
• Switched from Euphoria to Phix, http://phix.x10.mx/
• COPY and COPYALL can now be used without problems

- SAVE does not save the values of $ and $$ anymore
- New file hyin has last input line
- New operator ROUNDS, rounding to n significant digits
- The rounding operator is now ROUND instead of RD
- n-argument operators now SQSUM and RCPSUM instead of SQSM and RCPSM
- Conversion operator grams to ounces now :TOZ instead of :TOC
- The behavior of SHOW, ? and empty input line have been changed
- Results of trigonometric functions are now rounded down to 0 when < 1e-13
- The -c command line option now changes all file input and output to the current folder, including hy.ini and hy
- The option to use decimal comma instead of decimal point has been removed
- The operator FLOOR has been removed
- FDEC -n displays results with n zeros before decimal point

Version 0.4, May 6, 2022
- New operator :MSDEC - converts minutes and seconds to decimal degrees or hours
- New command COPIN - copies input line to clipboard
- Conversion operator grams to ounces now :OZ instead of :OC
- Empty hy.ini file is created if it doesn't exist

Version 0.31, March 2021
- Bugfix: conversion feet/meter (operators :FT and :M)

[…] Version 0.27, October 2007 — First published version