Hypatia

Open Source
Programmable Text-based RPN Calculator
for Windows

Version 2.5
www.hypatia-rpn.net

Preface

Hypatia is different. The two most obvious differences 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, 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.

Version 2.2 and later

If you've used prior versions of Hypatia, don't be confused by a change in terminology: "run files" are now called "scripts" or "script files," but they are still the same thing.

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'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.
- 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 does not know matrixes, vectors or arrays, imaginary or complex numbers, etc. While Hypatia knows 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. The numerical accuracy is ~15 digits, of which up to 12 digits are shown. 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 2.5, from March 20, 2023.
Keep checking www.hypatia-rpn.net for updates.
Regarding Linux, see "Hypatia for Linux", page 59.

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.

Installation

Actually, there isn't anything to install — all that 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.

Install for 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."

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.

In the "Target" field of the "Shortcut" tab you can also add command line options after hy.exe (for instance -d to set angle mode to degrees instead of radians, see page 57).

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 10).

Install for Windows Command Line

Add the program folder to the system path. (If you're familiar with the command line, you know how.)

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 58).

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 5 7 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 5 7 1 4 SUM -
```

calculates the sum of 3, 5, 7, 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 57).

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 again prompt for the next input line. This result will also be written into the file hy in the program folder (see page 25).

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 15)

Numbers can be written in several formats: decimal, hexadecimal, or binary — see chapter "Number Formats", page 20. 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 variable, or a constant. If Hypatia doesn't recognize it, you will get an error message. For a list of Hypatia's commands, see pages 52 to 54.

There is one exception: assigning a value to a variable through `$myvar =` is a command, too, even if it begins with a variable (see page 17).

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 26), 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 27 and 28). 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).

Editing the Input Line

You can edit your input before you press the ENTER key.

ARROW LEFT, ARROW RIGHT, HOME and END keys move the cursor.

The BACKSPACE key deletes the character left to the cursor, the DELETE key deletes the character at the cursor position.

Typing is always in insert mode — any character you type will be written at the cursor position, and existing text moved to the right.

The ENTER key works regardless of the cursor position.

When the cursor is at the beginning of the line, ARROW UP and ARROW DOWN let you scroll through the previous input lines, which you can re-use as they are, or edit before re-using them. ENTER always inputs the line as you are seeing it.

The File hyin

The most recent calculation line gets written to the file hyin. RUN commands (page 27), $myvar = commands (page 17) and the executable part of IF … THEN clauses (page 44) get written to hyin, too. You can copy the line to the clipboard with the command COPIN.

You can open the file hyin in Windows Notepad (or a different editor) with the EDIN command, edit and save it, and execute the edited line with the REPEAT command (see 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 43).

“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 15. At the start of Hypatia or after the INIT 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- INT FRAC SQ SQRT CUBE CBRT !
LOG LOG10 LOG2 EXP EXP10 EXP2 RAD DEG SIN COS TAN ASIN ACOS ATAN
```

1-argument unit conversion operators:

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

2-argument operators:

```
+ - * / // \ ^ % %+ %- %? %n %a ROUND ROUNDS GATE UPLIMIT LOLIMIT
MOD 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
```

Pseudo operators (see page 41):

```
WHISK A B DONE |
```

See also constants and pseudo-constants (page 16):

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

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)
INT	truncated integer value (for rounding to integer, use 0 ROUND, see 2-Argument Operators)
PRIME	1 if argument is prime number, 0 if not (argument must be 0 or positive integer)
FRAC	fractional part of a decimal number (negative, when argument is negative; the sum of INT and FRAC equals the original number) INT and FRAC round the argument to 12 significant digits before performing their function
SQ	square
SQRT	square root
CUBE	cube
CBRT	cube root (argument can be negative)
LOG	natural logarithm (base e)
LOGN	same as LOG (use LOGN or $ LOG at beginning of line, because LOG is also a command)
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 (the maximum argument is 170)
RAD	converts angle from degrees to radians (do not confuse with USERAD command)
DEG	converts angle from radians to degrees (do not confuse with USEDEG command)
SIN	sinus
COS	cosinus
TAN	tangens
ASIN	arcus sinus
ACOS	arcus cosinus
ATAN	arcus tangens

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

By default, SIGN and the six IS operators consider arguments within ± 1e-13 to be zero (you can change or disable this threshold, see “Zero or Not Zero”, page 19).

SIN, COS and TAN always set results within ± 1e-13 to zero.

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-13 (this is useful as a loop exit condition)
\	integer division, a \ b (this rounds off the absolute value of the result to the next integer)
^	power, a ^ b
EQUAL	1 if a and b are equal, 0 if they differ
UNEQUAL	1 if a and b differ, 0 if they are equal
MULTIPLE	1 if a is multiple of b (a / b is a positive integer, 0 if a or b are 0)
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 -6 to 9)
ROUNDS	round a to b significant digits (b can be 1 to 12)
GATE	zero if abs(a) is less than b, otherwise a (if threshold value b is 0, it is set to 1e-13)
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 must be 0 or positive integer, b must be positive integer)
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 = st.dev., 0 0 = standard normal dist.)

Of the six possible ways to compare two values, the EQUAL and UNEQUAL operators cover two. For the other four, you have to use *minus* and one of the IS operators:

a = b	`a b EQUAL`	or	`a b - IS0`
a ≠ b	`a b UNEQUAL`	or	`a b - ISNOT0`
a > b	`a b - IS+`		
a ≥ b	`a b - IS0+`		
a < b	`a b - IS-`		
a ≤ b	`a b - IS0-`		

In each case, the result is either 1 for true, or 0 for false.

By default, EQUAL and UNEQUAL consider differences below ± 1e-13 to be zero, and the IS operators consider absolute values below 1e-13 to be zero. You can change or disable this threshold, see “Zero or Not Zero”, page 19.

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.

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

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

The “insert” function ( ) allows statistical operators to read data from files (see chapter “Insert Files”, page 28).

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.

Chain Calculations

The symbol $ always represents the result of the previous calculation, you can use it in the next calculation wherever you want. By default, the first operator in a calculation uses the previous calculation result if it is short of one argument — that is, Hypatia supports chain calculations by automatically assuming $ at the beginning of the line, if needed. (Note that this does not apply to n-argumant operators, which have no way to decide whether an argument is missing or not.)

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

Constants

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

PI	3.141592653589793
E	2.718281828459045
PHI	1.618033988749895 (the “golden ratio”)

Pseudo Constants

These aren’t really constants at all, but in Hypatia’s syntax they look like constants.

RAND	Random decimal number in the range of 0 to 1 (do not confuse with RANDINT and RANDND, which are 2-argumant 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 loop: always 1 see “The Loop Command REPEAT n”, page 32
ISLOOP	in DO and REPEAT loops 1, outside of loops (when a script is called directly) 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 4.3825 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 three 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 28.

You can then convert 160 hp(M) to kW by writing `160 (hp) *` or convert 100 kW to hp(M) by writing `100 (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):

`$hp = 0.7355` (spaces before and after = are required!)
You can then convert 160 hp(M) to kW by writing `160 $hp *`

Or you can save $hp to a file, and load it when you need it — see the following chapter “Variables”.

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 an INIT 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 — e.g., $myvar = 2, $myvar = 2 SQRT, $myvar = $myvar SQ 2 *

After the equal sign = the assign command accepts 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 $ nor the result file hy (see page 25). Unlike other commands, though, it does get written to hyin (see page 30).

The variables $zero and $loop have special meanings — do not use them except for their intended purposes! (For $zero see page 19, for $loop see page 34.)

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 $$
SHOW $var1 $var2 …	show these variables
DEL $variable	delete the variable
SAVE filename	save all variables to a file (not including $ and $$)
_filename	(or RUN filename) retrieve variables that were saved with SAVE

`STO $var`, `$var =` and `PROMPT $var` create the variable, if it does not already exist.
Neither `$var =` nor `PROMPT $var` update $ and hy.
`$var =` updates hyin if not in a script, `PROMPT $var` never does.

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 45 and 47).

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.

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

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 file in Hypatia's program folder. 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.

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 (see chapter "Script Files", page 27).

Zero or Not Zero

Because of unavoidable rounding differences, zero is not always exactly zero. For instance, in theory `2 SQRT SQ 2 -` is 0, but when you perform that calculation, you get 4.4408920985e-16. While this is an extremely small number (with 15 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-13 (if you're not familiar with scientific notation, this is 10^{-13} or 0.0000000000001).

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 EQUAL and UNEQUAL operators regarding the difference between two values.

Considerations of what is, or is not, zero are particularly relevant in the context of IF … THEN clauses (page 44). You can change the threshold or disable it by setting the variable $zero, which overrides the default 1e-13 value. Note that to IF any deviation from 0 means true, 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-13

If you set a negative value, an error will occur when one of the SIGN, IS or EQUAL/UNEQUAL operators are used.

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-13. 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-13, but you are free to specify any threshold value instead of 0, including $zero. Unlike with the SIGN and IS operators, you cannot disable the threshold, though — if $zero is 0, `a $zero GATE` equals `a 0 GATE` and uses the default 1e-13 threshold. (Disabling it wouldn't make sense, it would be the same as simply not using the GATE operator.)

SIN, COS and TAN always round their results to 0 if they are less than 1e-13.

INT and FRAC round their arguments to 12 significant digits before determining the integer and fractional parts, this is due to only 12 digits being visible. This way, the value 0.9999999999999 will give an integer part of 1 and a fractional part of 0 — without rounding the integer part would be 0, which would be technically correct, but the fractional part would show as 1.

Number Formats

You can enter numbers in decimal, heximal and binary format. Decimal numbers can also be written in scientific notation. Hexadecimal and binary numbers can only be positive integers. As always, input is case insensitive. Examples:

`161`	decimal
`&hA1`	&H precedes hexadecimal numbers
`&b10100001`	&B precedes binary numbers
`1.61e6`	1.61 times 10 to 6th power, that is, 1.61 million, or 1610000

To enter a negative number, write it with a leading minus sign.

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.

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

To convert a hexadecimal number into decimal format, simply enter it:

```
? &hA1
= 161
```

Output Format Commands

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

By default, Hypatia shows result in decimal or scientific notation. To display results in hexadecimal or binary format, use the commands FHEX and FBIN. The command FDEC sets output back to decimal format. With numbers that are not positive integers FHEX and FBIN will be ignored.

Optionally, you can add the number of hexadecimal or binary digits in which the result should be displayed, after the FHEX and FBIN commands:

```
? 161
= 161
? FHEX
= &hA1
? FHEX 8
= &h000000A1
? FBIN 16
= &b0000000010100001
? FDEC
= 161
```

Once you have chosen the desired format with the FHEX or FBIN command, all results will be shown in this format. To convert numbers to hexadecimal or binary, just enter them. The command == shows the last result in default format, the command = shows it in the currently specified format.

Decimal Digits

With the decimal number format you can specify how many digits will be shown after the decimal point, by adding that number after the FDEC command. If that number is negative, the value will be displayed with as many zeros before the decimal point.

```
? 2 SQRT
= 1.41421356238
? FDEC 2
= 1.41
```

Do not confuse this with the ROUND or ROUNDS operators, which round the actual value — the format commands only affect how values are displayed.

Enter FDEC without a number to restore output to the default number of decimal digits. This is not the same as FDEC 0, which sets the number of decimal digits to 0.

If you copy the calculation result to the clipboard with the COPY command (see page 22), then it will be copied in the currently chosen format.

List of Output Format Commands

FDEC	decimal format, no specified number of digits after the decimal point (this is the default)
FDEC n	decimal format, n digits after the decimal point (n can be 0, max. = 10)
FDEC -n	decimal format, n digits (with leading zeros) before decimal point (max. 9)
FHEX	hexadecimal format
FHEX n	hexadecimal format, n digits (or more, if the number is larger)
FBIN	binary format
FBIN n	binary format, n digits (or more, if the number is larger)

Any number that is not a positive integer, or that is larger than 4294967295 — (&hFFFFFFFF, 8 hexadecimal digits, or &b11111111111111111111111111111111, 32 binary digits) will be displayed in decimal format.

Angle Format

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)

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

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 more than a single result. For details, see "Result File hy and Result Variable $" (page 25).

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.

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 28.

Input and Output Files

Hypatia can read and process files that can contain data, parts of calculations, or what in other programs would be called macros — see the two chapters “Insert Files” and “Script Files”.

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 by Hypatia automatically:

hy has a copy of the last calculation result, using current format settings (see page 25)
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.

`EDIN` opens the file hyin, which contains the most recent calculation input line (including the last $myvar = assign or RUN command), in an external editor — by default, Windows Notepad. You can edit it there, save it, and use REPEAT to perform the edited calculation (see page 30 — though you will usually scroll up in the input editor, and edit your previous input there).

Two kinds of files are written optionally: files in which variables are saved, and Hypatia’s log file.

`SAVE filename` writes all current variables into a file with the specified name. They can later be restored using the RUN command (see chapter “Variables”, page 17). The file name must not contain spaces or parentheses.

`LIST` 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 a 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):

```
? 47 37 46.94 :msdec +
= 47.6297055556
? 15 51 37.07 :msdec + &
= 15.8602972222
? COPY
```

With Ctrl/V you can now paste `47.6297055556 15.8602972222` 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 27) 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 45).

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 even 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.

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 (though $ would be easier to use):

```
? 2 3 +
= 5
? (hy) $ *          hy is used as an "insert file" (see page 28), therefore the parentheses
= 25                in the calculation, both (hy) and $ have the value of 5
```

There are some differences, though. One is that $ has the full internal accuracy of Phix variables, approximately 15 digits, while the result saved to hy has a maximum of 12 significant digits.

When you have specified a number format (see page 20), then hy has the result in this format, while $ always has the unformatted value.

The other differences are:
- Accumulation mode allows hy to have more than one result (see pages 23 and 37).
- Silent mode lets you update $ without changing hy (see page 24).
- The command & clears hy without changing the value of $ (see page 37).
- Within a script (see page 27) $ 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 22), 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, and overwrites $$.

Simply entering `(hy)` sets the value of $ to the value saved in hy (unless hy holds more than one result after you have used accumulation mode, in which case you will get an error message). Do not confuse this with the command HY, which only 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 default format (as does `SHOW $`). 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 42), 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.

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 INIT 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 57).

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

Important:
Do not confuse the LOG command with the LOG operator (natural logarithm). When the line begins with LOG (that is, you want to use the previous result as the operator's argument), then the LOG command is assumed. For the operator, you can use the alternative name LOGN, or you begin the line with $ LOG.

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 command automatically creates to save variables, which you can still edit (see “Variables”, page 17). The file name can not contain spaces or parentheses. Like all other files that Hypatia uses, script files are located in its 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 define user constants (see chapter “Constants”, page 16), to change program mode settings, or to perform calculations. Scripts can be used in loops (see page 32).

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 24) for the script’s last calculation line prevents the result from being written to hy.

If you set accumulation mode for a calculation line, though (see page 23), 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, INIT, 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 executed as a script when Hypatia starts, or when you reset Hypatia with the command INIT.

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 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 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 facilitate recurring operations by defining your own operators
- to input data for statistical (n-argument) operators

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.

Defining Your Own Constants

This has already been discussed in the chapter “Constants” (page 16). 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.

Alternatively, you can use variables as your “constants,” save them to a file with SAVE filename, and load them with _filename (see chapter “Variables”, page 17).

Facilitating Recurring Mathematical Operations

Since an insert file can be anything that you can type into the input line, it can be used for recurring operations. Suppose you have to calculate the masses of iron balls — using m and kg as units, for diameter d this mass is calculated by the formula d 2 / 3 ^ pi * 4 * 3 / 7874 *

If you save the expression `2 / 3 ^ pi * 4 * 3 / 7874 * 3 rounds` as a file ironball, then you can calculate the mass of an iron ball, rounded to 3 significant digits, by writing (ironball) after the diameter:

```
? 0.1 (ironball)
= 4.12
```

(a 10 cm iron ball has a mass of 4.12 kg)

This way, you have created your own 1-argument operator. For how to create your own 2-argument operators, see the chapter “WHISK, A and B” (page 41).

Entering Data for Statistical Operators

You may want to use statistical operators, for instance SUM, MEAN or SDEV, with far more arguments than can be written into the input line. Also, you may want to use the same data for different calculations. For that purpose, Hypatia can read data from a text file.

Numbers in the insert file can be separated by spaces, or can be written in separate lines, or any combination thereof.

If your file myfile.dat has the numbers for which you want to calculate the mean value and the standard deviation, you can enter

```
? (myfile.dat) MEAN
? (myfile.dat) STD
```

Insert (hy)

The fily hy always has the most recent calculation result, which also is the variable $. It is easier to type $ than to type (hy) — but, $ is lost when you exit Hypatia, while (hy) is still available the next time you start it.

We have already discussed differences between $ and (hy) in the chapter “Result File hy and Result Variable $” (page 25). What interests us here, though, is that thanks to the accumulation mode, hy can hold a number, 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 40), “DONE, and Generalized Fibonacci Sequences” (page 42), or “Centimeters, Feet and Inches” (page 45).

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.
- `(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 =` assign statements and in responses to `PROMPT $var` commands.

Some files can be used as either script or insert files. Look at the “iron ball” example on the previous page — we have used it as an insert file, inserting the formula after the diameter. But you could also use it as a script file, in which case it would use the previous result as its argument:

```
? 0.1
? _ironball
= 4.12
```

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 file hyin contains the most recent calculation or RUN command input line (other input lines are not written to hyin). Some reasons why you may want to re-use it:

- 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 previous calculation input 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
= 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 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 script file.

Loop commands are not allowed within scripts.

Loop commands can not be nested — see "Nested Loops", though (page 49).

The REPEAT loop command is:

```
REPEAT n [?]
```

with n being an integer between 1 and (unless changed) 99999. By default only the final result will be shown, with the optional symbol ? results are shown for each pass. Note that this is not the same as the "debugging" question mark (see page 56).

REPEAT n repeats the previous calculation or RUN command n times — for more about this, see the following page.

The DO loop command is:

```
DO n [?] :: calculation or RUN command
```

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.

Each time a calculation is performed the variables $ and $$ are updated, allowing each calculation to build upon the result of the previous one(s). Within a script, you can also use your own variables.

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 23 and 24, and "Loops and Accumulation Mode", page 37.

If you use a loop command with a script, then note that $ and $$ get updated by each calculation line, 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, they 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 37).

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, for instance, don't), in that case it remains what it had been before. 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.

The Loop Command REPEAT n

REPEAT n, with n being a number between 1 and 9999, 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 loop command:

```
? 0                 set result to 0
= 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”).

The Loop Command DO n

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

```
? 0                 set result to 0
= 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. One advantage of the REPEAT loop is that you can see the original result, and verify that your calculation line or your script works as intended, before you execute the loop.

“Infinite” Loops

Hypatia’s loops always need a number of passes to be specified, but this number can be very high — by default, the maximum is 100000, but the command MAXLOOP n lets you change this limit to any value from ten to ten million (easier to write in scientific notation, 1e7). You can use these high numbers of passes in iterations or Monte Carlo experiments, but you can also use them in the sense of an unspecified number of passes, when your script contains an exit condition (see below, page 34).

The DO command has a shortcut for DO 100000 (or what other maximum you may have specified):

`DO * :: ...` (DO * ? :: is not permitted)

`DO * :: _filename` (executing a script in a loop) has a shortcut version:

`*_filename` (and because this can easily be mis-typed, `_*filename` works, too)

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, by using accumulation mode — see page 37). If you want the results of each pass to be shown while the loop is performed, you have to add ? to the loop command, after the number of loop passes:

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

Note that this is not the same as the “debug” question mark at the end of a calculation line (see “Debugging”, page 56).

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

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 37)
- in a script, you can use your own variables which you can update at each pass (see page 17)
- you can use the loop index “pseudo constant” I

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

Outside of a loop I always has the value 1.

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
= 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 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 behave any way you want them to.

Defining an Exit Condition

There are two ways to exit a loop before the specified number of passes is reached: by setting the variable $loop to 0, or with the IF … THEN ENDLOOP command — for that, see “Conditional IF … THEN Clauses”, page 44.

$loop is meant to be used in a script. 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.

Note two facts: First, the value of $loop is ignored if it is not changed *within* the loop. And second, you can define a condition for *not* exiting the loop by setting $loop to 0 at the beginning of the script, and changing its value to anything not zero as long as the loop is to be continued.

The exit condition is only met when the value of $loop is *exactly* zero. To use either the default zero threshold of 1e-13 or one you define via $zero, you can use the operators SIGN or IS0 (see “Zero or Not Zero”, page 19). The operators EQUAL, UNEQUAL and the other IS operators, which can all be used for defining the exit condition, also regard the zero threshold, while the operator // has its zero threshold fixed to 1e-13.

The GATE operator lets you set $loop to zero when a certain threshold is reached or exceeded, and by following it with IS0, when it is *not* reached. To exit the loop when the absolute value of a variable $v has become less than 1, you can write `$loop = $v 1 GATE IS0`

Exit conditions can be based on calculation results, on the loop index I, on user input, or any combinations thereof. An example of user input as exit condition you can find in “Scripts, Prompts, Loops, Insert Files” (page 47).

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
= 10.6309782609
```

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-13, 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 6
= 9.94987437107           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 simply set

the maximum number of passes to 100000, using DO *. 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 = 1.0872790404
? REPEAT                       let's look at the next one
  run babroot                  REPEAT displays the line it repeats
= 10.6309782609
  $loop = 0.141592917
? REPEAT                       and another one
  run babroot
= 9.971692801
  $loop = 0.00439047795546     now we are satisfied that the iteration works as intended,
? REPEAT 100                   and can give the REPEAT loop command
  run babroot
Loop exit condition met in pass 3
= 9.94987437107
```

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-13 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-13 threshold may not be adequate — you can change it by setting the variable $zero (see page 19). Or you use the GATE operator: `$loop = $ $$ - b GATE`, replacing b with your chosen threshold (which defaults to 1e-13 if b = 0, since an actual gate value of 0 would make no sense).

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, and 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. To avoid this you have to use silent mode (page 24):

```
? (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
```

To prevent accidentally overwriting hy, you can 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. With Windows Notebook you could also just open hy with EDIT, keep it open, and save it again to restore it — Notebook doesn't notice it when an open file gets changed by another application, but other editors may behave differently.

Another example: suppose you want to see a value of 1000 increase by 7.5% ten times, and you want the numbers to be shown with 2 decimal digits. && writes each result in a new line.

```
? 1000
= 1000
? FDEC 2
= 1000.00
? DO 10 :: 7.5 %+ &&
= 2061.03
```

in this example you do not need to clear hy with the & command, because you want the start value to be included in the list of results

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

```
1000.00
1075.00
...
1917.24
2061.03
```

Note that calculations are performed with the exact values, the format commands (like FDEC 2) only affect the output. To base each calculation on the rounded result of the previous one, you have to use ROUND — in this example there is hardly any difference, but in financial mathematics for instance you may have to consider this issue:

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

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 need 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 49). 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.)

Since the calculation is done in a single line we wouldn't need to use a script, but let's do it anyway:

Type `EDIT mwpairs` to create a file of that name, and write this line:

```
I1: &
178 7.6 RANDND 164 6.35 RANDND - 1.5 GATE &&
```

This creates a random male height, then a random female height, then subtracts the latter from the former, and finally arbitrarily sets the difference to zero if its absolute value is less than 1.5.

&& at the end of the line activates accumulation mode (page 23).

Now lets do this 1000 times:

```
? DO 1000 :: _mwpairs
= 9.5981819084              (for instance) this is just the last random difference
```

At the end of a loop the last calculation result always gets displayed, which is the value of $. As in our example, this is not always the content of hy — hy now holds a list of 1000 numbers, the differences of 1000 random pairs, with differences below the arbitrarily chosen limit of ± 1.5 cm set to 0.

```
? (hy) N- #
? (hy) N0 #
? (hy) N+ #
```

prevents hy from being overwritten (silent mode, see page 24). To protect against this happening by mistake, you can use the command EDIT to open the file hy in an editor, save it under a different name, and use that file name instead of (hy), without the need for #.

For an example of a more complex Monte Carlo experiment see "Nested Loops in Monte Carlo", page 50.

Pseudo Operators

WHISK, A and B, and User-Defined 2-Argument Operators

In “Facilitating Recurring Mathematical Operations” (page 28) you have seen how you can use insert files to create your own 1-argument operators — for your own 2-argument operators, you may need to use the WHISK pseudo operator. (You need WHISK when an operation has to be performed on argument a — in infix notation, $a^2 + b^2$ need WHISK, $a + b^2$ does not.)

WHISK removes (whisks away) the two values to its left (remember, when an operator is encountered, left of it 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 + SQRT
= 5
```

This means, to calculate the hypotenuse of a rectangular triangle you can, for instance, create a file hypot with the line `WHISK A SQ B SQ + SQRT`, which you can then use as an insert file, which works as your own 2-argument operator:

```
? 6 8 (hypot)
= 10
```

In this example you might have done the 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 43). WHISK does not have this restriction, and also makes much 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 *

A file ellips with the line

```
WHISK a b + 3 * a b * 10 * a sq b sq + 3 * + sqrt - pi *
```

would then serve as a 2-argument operator to calculate the approximate circumferences of ellipses:

```
? 3 6 (ellips)
= 29.065263293
```

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 more flexible way of using the same value several times than the pseudo constant DUP (page 16) — for differences between DUP and WHISK, see “Processing Input Lines” (page 60).

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.

DONE, and Generalized Fibonacci Sequences

The pseudo operator DONE is useful in the context of accumulation mode and REPEAT loops (see pages 32 and 37).

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 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 generalizations 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 after the initial 0 and 1 — but you will easily be able to generalize this method.

```
? EDIT                          open hy, write  0 1  and save the file
? DO 20 :: (hy) + DONE &
  0 1 + done &                  the line including the insert file is shown once
= 10946                         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.

Instead of writing the first two numbers of the sequence to hy with the help of an editor, we could do everything in 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 beginning the script with `I1: 0` instead of the two lines `I1: &` and `I1: 0 &` would not work, 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 0 and 1.

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

```
? DO 20 :: _fibonacci
= 10946                 the 20th result
? EDIT                  open hy to view the sequence
```

There are limits to what Hypatia can do with generalized Fibonacci sequences, but they are not reached soon, and with the help of variables and scripts you may be able to push them even a bit further.

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.

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 INIT 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` exits a 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 following lines of a script. If this happens in a loop, the loop itself is not afffected. See “Scripts, Prompts, Loops, Insert Files” (page 47) for an example.

Small numbers below the zero threshold (by default 1e-13, see chapter “Zero or Not Zero”, page 19) 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. Be aware that all six IS operators 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, like ELSE:, executed when the 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. They can be used in I0: 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 you can use Hypatia's :FT unit conversion operator, convert the meters to feet, and assign to variable $ft. Multiply the fractional part of $ft with 12 and you have the inches, take the integer part of $ft and you have the feet. The problem starts when you want to round the inches to integers — you can end up with, for instance, 5 feet 12 inches. To deal with that case, you have to add a condition: if you get 12 inches, then increase feet by one. Also, if you have 12 inches, set inches to 0.

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

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

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

Line 6: Instead of IF … THEN we could use: `$ft = $ft $in 12 / INT +` (add the integer part of $in divided by 12 to $ft). Hypatia often offers several ways of achieving a result, just use the first one you happen to think of.

Line 7: 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 calculation result (the value of $ft) to hy.

Line 8: `$inch 12 MOD &` (that is, $inch modulo 12) sets $inch to 0 if its value is 12, and writes the result to hy. Alternatively, we could say `ALSO: $inch = 0` but that would need a separate line to write the result to hy, `$inch &`

`? _cm-fi` 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`. If a script contains calculations, the value of $ (the most recent calculation result) is always shown at its end.

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

The second line adds 149 to the loop index, which lets the table start at 150 cm (you could make this more flexible by using a variable). At the end of the loop, the list gets copied to the clipboard, and the according message is displayed:

```
I1: &
I 149 + &&
$ft = $ 100 / :FT
$inch = $ft FRAC 12 * 0 ROUND
$ft = $ft INT
IF $inch 12 EQUAL THEN $ft = $ft 1 +
$ft &
$inch 12 MOD &
I*: COPY
I*: # List copied to clipboard
```

Line 1 clears hy at the beginning of the loop, line 2 writes the centimeters (index plus 149) to a new line in hy. Instead of using a variable $cm, this file uses $ for the centimeters (the results of line 2 in the loop). 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 51 passes:

```
? DO 51 :: _cm-fi-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) — anyway, you'll get:

```
150 4 11
151 4 11
152 5 0
153 5 0
154 5 1
155 5 1
... ...
197 6 6
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
```

If you write `12 / + :M 100 * 0 ROUND` to a file named fi-cm, you can use this as an insert file that serves as a feet-and-inches-to-cm operator:

```
? 5 9 (fi-cm)
= 175
```

Scripts, Prompts, Loops, Insert Files

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 — this shows how insert files and scripts can be used together (for the insert file fi-cm, see above):

```
? _bmi
# Enter weight in kg, then height in cm
? $kg = 145 :kg
  $kg = 65.77089365
? $cm = 5 6 (fi-cm)
= 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 end loop), then height in cm
PROMPT $kg
IF $kg IS0 THEN ENDLOOP
ALSO: SKIP
PROMPT $cm
$kg $cm 100 / SQ / 1 ROUND
IF ISLOOP THEN =
```

This script can be used for single calculations 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 end loop), then height in cm
```

Lines 3 and 4: Entering a weight of 0 triggers the ENDLOOP command — but this only ends the loop after the end of the script. The SKIP command skips the rest of the script, so that the end is reached immediately. Note that SKIP has to come after ENDLOOP, or the ENDLOOP command itself would be skipped.

Lines 6 and 7: 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 7 determines whether the script is run by a loop — if the pseudo constant ISLOOP is 0 this line does nothing because the result has already been shown, 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 several advanced computer languages in which you can do this, but Hypatia lets you do it, too:

```
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 example, 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 (with DO, * stands for 10000).

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 - IS+ 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 - IS+ THEN ENDLOOP         if $j is greater than 12, the loop is exited
I*: $sum
```

The 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:

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

The "Loop exit condition met …" line tells you that the loop 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 - IS- 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 true if neither of the EQUAL results were 0, 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, what will the average sum of eyes be? There are more ways that one to do this, for instance:

```
I1: &                              at the start of the loop, clear hy …
I1: $eyes = 0                      … and set $eyes to 0
$roll = 1 6 RANDINT                $roll is assigned the result of one roll of the dice
$eyes = $eyes $roll +              add that to $eyes
IF $roll 6 EQUAL THEN $eyes &      after we've rolled a six, write $eyes to hy …
ALSO: $eyes = 0                    … and reset it to 0, thus closing the inner loop
I*: (hy) MEAN #                    mean of the $eyes in hy (don't overwrite hy)
```

If we call that file roll6, then …

```
? *_roll6
= 21.2046737518                    but even with 100000 rolls of the dice this result varies
```

The above script is short and elegant, but the large number of write operations slows it down (and, to be honest, when it comes to loops, Hypatia isn't that fast to begin with). Here is a version that uses two variables, $sum and $n, instead of writing intermediate results to hy:

```
I1: $eyes = 0
I1: $sum = 0
I1: $n = 0
$roll = 1 6 RANDINT
$eyes = $eyes $roll +
IF $roll 6 EQUAL THEN $sum = $sum $eyes +
ALSO: $eyes = 0
ALSO: $n = $n 1 +
I*: $sum $n / &&
IF I 10000 MULTIPLE THEN SHOW I
```

At the end of the loop the average number of eyes until you've rolled a six is calculated. The last line is only there to show you the loop index every 10000 passes, so you know that Hypatia is busy and how long you still have to wait for the result.

By default, the maximum number of passes in a loop is 100000, but you can change this limit with the MAXLOOP n command:

`? MAXLOOP 1000000` (or, easier to write in scientific notation, MAXLOOP 1e6)

lets you roll the dice up to one million times. The highest value you can specify is ten million, or 1e7, but ten million passes of a loop will take their time.

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

List of Commands

Mode Settings

AUTO$ ON | OFF	set auto include $ mode ON (default) | OFF	15
AUTO$	show current auto include $ mode	
ECHO ON | OFF	set echo mode ON | OFF (default)	27
ECHO	show current echo mode	
LOG ON | OFF	start | stop (default) logging input and output to file hy.log	26
LOG	show current logging mode	
USE DEG	(or USEDEG) angle unit is degrees (do not confuse with the DEG operator)	21
USE RAD	(or USERAD) angle unit is radians (default; do not confuse with the RAD operator)	
FDEC	use decimal format for displaying results (default)	21
FDEC n	show results with n digits after decimal point (unlike the ROUND operator, this does not affect the value of results, only the way they are displayed)	
FHEX	use hexadecimal format for displaying results (only valid for positive integer numbers)	20
FHEX n	use hexadecimal format, with at least n digits	
FBIN	use binary format for displaying results (only valid for positive integer numbers)	20
FBIN n	use binary format, with at least n digits	
INIT	delete all variables, reset all options to default or to command line parameters, execute hy.ini (unless Hypatia was called with -i option), this does not delete hy and hyin	

For accumulation mode, silent mode and debug mode see “List of Control Symbols”, page 55

Variables

STO $var	assign value of last result to variable (create variable if it doesn’t already exist)	18
$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)	
SAVE filename	save variables to file (anything after file name is a comment line)	
_filename	or RUN filename: retrieve variables from file	

Results

=	show last result (value of $) in currently chosen format	
==	show last result (value of $) in default decimal format	
HY	show content of result file hy	25

Clipboard

COPY	copy the result file hy to the clipboard	22
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

RUN filename	execute a script file, each line is treated as an input line	27
_filename	same as RUN filename, but without displaying intermediate results	
(filename)	insert content of file in input line	28
&	clear result file hy	37
&&	add a line break to result file hy	
HY	show content of result file hy	25
LIST	show all files in Hypatia's program folder	23
EDIT	open result fily hy to view or edit (short for EDIT hy)	23
EDIT filename	open editor to view, edit or create a file in Hypatia's program folder	23
EDIN	open editor to view or edit file hyin (short for EDIT hyin)	23
	default editor is Windows Notepad (notepad.exe)	
EXTEDITOR filename	replace notepad.exe with an editor of your choice	23

Repeat, Loops and Scripts

REPEAT	repeat the most recent calculation	30
REPEAT and DO	loop commands	31
MAXLOOP n	sets max. number of passes in a loop (by default 100000)	31
REPEAT n	loop command, repeat the most recent calculation n times	32
DO n :: …	loop command	32
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	34
I1: …	execute line only when loop index is 1, or when not in a loop	39
I*: …	execute line only at the end of loop	39
IF … THEN …	execute command or calculation only when condition is met	44
IF … THEN ENDLOOP	exit loop when condition is met	
ALSO: …	execute when preceding IF/THEN condition was met	44
ELSE: …	execute when preceding IF/THEN condition was not met	
IF … THEN SKIP	skip the rest of the script when condition is met	44

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) 10
Comments following I1:, I*:, IF … THEN, ALSO: or ELSE: will always be displayed if true

#: comment line 10
Comment lines in a script that start with #: will be displayed in loops

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

calculation line **&** 23
& 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 **&&** 23
Same as &, but sets "new line" accumulation mode for this line:
new results will be added to hy as new lines

calculation line **#** 24
at the end of a calculation line sets silent mode for this line:
result file hy will not be updated

calculation line **?** 56
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 **?** 53
DO n **?** :: … …
Show results for each loop pass

_filename 27
Same as RUN filename, but only final result will be shown

… **(**filename**)** … 28
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 again, 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 mode 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 INIT 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 58).

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:\ ... >= 1.41421356238
```

To see the sinus of 45 degrees, enter

```
C:\ ... >hy -d 45 sin
C:\ ... >= 0.707106781187
```

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 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 compiled with Phix 1.0.2, 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 INIT 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. The input line is split into its elements, separated by spaces.
3. The first element determines if it is a command or a calculation line (for exception see step 5).
4. If it is a command, it gets executed. Done.
5. If the second element is = and the first element is a variable name, this is a command that assigns a calculation result to a variable. A flag is set, and the first two elements $variable = are removed from the stack before step 6.
6. 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 line, which is now a sequence of words, contains only numbers and operators.
7. 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).
8. If the assign flag was set (step 5), the result is assigned to the variable. Step 9 is omitted.
9. If no error has occurred, the result is displayed, $ and $$ are updated, and the file hy is updated (unless silent mode is set).
10. 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 6, before calculation begins. In Hypatia's terminology, these are pseudo constants.

A and B, though, along with WHISK, make it to step 7. 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 6.

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 5 sigma are rejected (the probability of which is about one in a million).

Accuracy

Numbers in Phix (32 bit) have an accuracy of up to 15 digits, but Hypatia only shows 12 digits. This can lead to a small difference between $ — the actual value of the recent calculation result — and (hy), which has the number you get to see, with a maximum of 12 digits. To demonstrate, enter a 14-digit number, and subtract (hy):

```
? 123456789.12345
= 123456789.123
? $ (hy) -
= 0.000450000166893
```

This shows us that the 13th and 14th digits, though usually hidden, are actually there. Because numbers are stored in binary format, there will usually be a tiny discrepancy between the decimal and the binary representations of non-integer numbers — this is where the …0000166893 comes from — but Hypatia's calculations are actually more accurate than they would be if they were performed with only 12 digits.

```
? 2 SQRT              square root of 2
= 1.41421356238       these are the 12 digits that Hypatia shows
? STO $s              let's store the result in variable $s
? (hy) SQ             the 12-digits result squared
= 2.00000000001       this is what you get with 12 digits
? $s SQ               we square the actual result of square root of 2, which we have stored in $s
= 2                   this is indeed more accurate, even if there is, of course, some rounding
                      involved. Let's see how much the result really differs from 2:
? 2 $ -
= -4.4408920985e-16   this is -0.00000000000000444…
```

To sum it up, you can safely rely on the accuracy of Hypatia's calculations for the 12 digits that you get to see. If you use (hy) in your calculations, some of that accuracy will get lost.

Note: the operators INT and FRAC round their arguments to 12 significant digits before determining the integer and fractional parts (see "Zero or Not Zero", page 19).

By default, SIGN and the six IS operators consider values less than ± 1e-13 to be zero. EQUAL and UNEQUAL consider two values with differences less than ± 1e-13 to be equal (see page 19).
SIN, COS and TAN always set results less than ± 1e-13 to zero.
x 0 GATE always gives zero for x less than ± 1e-13 .

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.

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 2.5, March 20, 2023
- New n-argument delimiter |
- Fixed bug introduced in 2.4, which made some numbers < 1e-10 displayed as 1
- Fixed bug that disabled EXP2 operator

Version 2.4, March 19, 2023
- Results are now shown with up to 12 instead of 10 digits
- Fixed FDEC format command bug
- Hypatia now uses Phix version 1.0.2

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
- Bugfix: DO * did not work with upper case DO

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 line with accumulation mode in scripts now updates 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