rpnExpress Help

Version 1.2

Contents

1 Introduction

rpnExpress is a keyboard-centric calculator which uses the GMP and MPFR libraries to handle the mathematical operations. It grew out of our own needs for a fast versatile RPN calculator with small memory footprint that was controlled from the keyboard and supported large integers. The main features of this calculator are:

- All commands are entered from the keyboard, minimizing the need to move your hand to the mouse or trackpad.
- Due to the backend libraries GMP and MPFR, it can handle very large integers and floating numbers with extreme precision.
- Values and command sequences can be saved into named variables and recalled later for calculations.
- You can enter and display numbers in any base between 2 and 36 inclusive.
- You can group the digits for easier reading of e.g. hex and binary values.
- It has both a RPN mode and algebraic mode.
- It has over 60 powerful functions ranging from number theoretical functions like primality testing and the Riemann Zeta function to trigonometric functions and logical bitwise operations and calculator control functions.
- A declaration generator for exporting large integers to C arrays.

1.1 Notations

In this manual, we will use the term *top* of the stack to denote level 1 (L1) even though it is at the bottom of the stack view. The different levels of the stack is denoted L1, L2, L3 and so forth.

Commands that should be typed into the text input field will be written in `this monospaced font`. When describing keys on the keyboard, we will use this san serif font.
Example: Enter `0 cos` and press Return to get the result 1 on L1.

When describing the different commands and their effect on the stack, we will use the following figure:

stack	L4:		**Result** →	*stack*	L4:	
	L3:				L3:	1
	L2:	1			L2:	2
	L1:	2			L1:	21.86
input	`5 2 3 + * 3.14 -`			*input*		

This shows the effect of entering `5 2 3 + * 3.14 -` in the input area and pressing Return. The previous content of the stack has been pushed down (to L2 and L3) and the result of the calculation is now on L1.

2 The two modes, RPN and algebraic

rpnExpress handles two ways of entering expressions. Reverse Polish Notation (RPN) and normal algebraic syntax. You can see the current input mode in the lower right corner of the main window. If you have the toolbar visible, the input mode switch button will also indicate which mode you're in. You can switch mode by pressing ⌘I.

RPN mode.

In RPN mode, you first enter the operands, thereby pushing them on the stack. Then you enter the operator or command. A simple example is to calculate $2 + 3$. In RPN mode you would enter this as `2 3 +`. Each parameter is pushed onto the stack and when a command is entered it will pop the required parameters from the stack and push back the result of the command onto the stack. In RPN mode you can enter multiple commands in a single line and the calculator will break up the command line into tokens and push them on the stack. For example, to calculate $5 \cdot (2 + 3) - 3.14$ you can enter the command line:

```
5 2 3 + * 3.14 -
```

or you can enter each token separately (pressing ↩ after it) and watch the intermediate results on the stack. Note that some commands are insensitive to the order of the parameters, but for many commands, the order in which you push the parameters is important. For example, the operator $+$ is commutative which means that $a + b = b + a$ but the operator $-$ is not. To calculate $a - b$ you must first push a on the stack, then b, and finally the operator $-$. For most commands there is a "natural" order and it will be intuitive how to enter the parameters. Check out the Command Reference in Section 14 for the order of the parameters for each command.

Algebraic mode.

In algebraic mode, you enter the mathematical expressions in the natural way. The normal

rules for precedence and associativity applies.For example,

$$5 \cdot 2 + 3 = (5 \cdot 2) + 3$$

and

$$2\text{^}3\text{^}4 = 2^{(3^4)}$$

The following table gives the precedence and associativity of the algebraic operations in rpnExpress . The top row has the *least* precedence.

Operator	Associativity	Comment
+ -	Left	Addition and subtraction
* / %	Left	Multiplication, division and modulo
^	Right	Power
!	Right	Factorial
-		Unary minus
( )		Parenthesis

When using commands that cannot be written inline in an algebraic expression, the parameters are put inside parenthesis just like function calls in many programming languages. For example, to calculate the inverse of 12 modulo 17 you enter `invmod(12,17)`.

Mode mixing.

RPN expressions are enclosed in curly braces, { and }. If you explicitly enter the curly braces when in RPN mode, the expression will not be evaluated, but simply push to the stack. This is handy for creating programs as explained in 3.1. Similarly, algebraic expressions are enclosed in single quotes. If you are in RPN mode and want to evaluate an algebraic expression, simply enclose the input in single quotes and press Enter. It will be pushed to the stack and evaluated. However, you cannot use RPN expressions in algebraic mode.

3 Variables

You can save values in named variables and use the name in subsequent calculations. To store the value 42 into the variable *x*, use enter the command `42 #x STO` or `STO(42,#x)` depending on the input mode. The hash sign (#) before the name prevents the calculator to try to evaluate the variable. So when you enter `x` you refer to the value of #x. More formally, the command `STO` will take the value on L2 and store it into the variable name on L1. Whenever you like to use the value of the variable, just push the name (without the hash sign) on the stack and it will be evaluated. To delete a variable, you push the name, with the preceding hash sign, on L1 and enter `DEL`.

Tips:

If the variable has not been declared before (has no stored values) you do not need to put a hash

sign before the name when storing a value into it.

Variables can be *persistent*. A persistent variable can (depending on your preferences) be saved between launches of rpnExpress . In the Preferences Pane (accessed through ⌘,) you have the option of specifying if all, none or only the persistent variables should be restored when rpnExpress is launched. To make a variables persistent, you push the variable name on L1 and enter the command `MPER`. To make it non-persistent you push the variable name on L1 and enter the command `CPER`. rpnExpress can show all your variables and their values in a separate window, which is showed by pressing ⌘2.

Variable names are case sensitive (commands are not) and must start with a letter (A…Z or a…z) and the following characters may be digits (0…9) or letters.

3.1 Programs

You can not only store values in a variables, you can also store sequences of commands and values. In RPN mode, such a program is entered by explicitly enclosing the RPN expression in curly braces, { and }. The following program, to calculate the area of a circle, is stored into a variable called *#area*:

Example:

```
{ 2 pow pi * }  #area sto
```

`pow` is the built-in command for *to the power of* and `pi` is a built-in constant which returns the value of π. To use the program, you push the radius on the stack and enter `area`. rpnExpress will fetch the program and execute it by sequentially push each item from the program onto the stack.

In algebraic mode you cannot store programs that take additional inputs, but you can create expressions that depends on another variable's value. To calculate the area of a circle with a radius stored in variable #r, you enter:

```
STO('r^2*pi',#area)
```

Note that the algebraic expression needs to be enclosed in single quotes to prevent it from being evaluated. Now you can store a suitable value in the variable #r, for example `STO(2.3,#r)` and then use `area` to calculate the result. Algebraic programs with input parameters might be supported in a future version.

4 Local Variables

In RPN expressions, you can pop the stack at anytime and store that stack value into a local variable for later use in that RPN expression. A local variable is created by prefixing a variable

name with an ”at-sign” @. The life-span of this local variable is limited to the current RPN expression. Local variables can shadow global variables without affecting them.

Example:

stack	L4: L3: L2: 8 L1: 5	**Result** →	*stack*	L4: L3: L2: L1: 23	
input	`@a @b 3 a * b +`		*input*		

You can only assign a local variable in an RPN expression, but that local variable can then be used in an Algebraic expression embedded in the RPN expression. Let us look at the previous example were we instead use the Algebraic notation for the calculations.

Example:

stack	L4: L3: L2: 8 L1: 5	**Result** →	*stack*	L4: L3: L2: L1: 23	
input	`@a @b 'a*3+b'`		*input*		

Note that the input mode in both these examples are RPN.

The same local variable name can be used in nested RPN expressions without conflict. For example, assume you have stored a program to calculate the area of a circle

```
{ @r pi r 2 pow * } #area sto.
```

Then you define another program

```
{ @r 2 r * area 4 3 / pi * r 3 pow *} #circlefun sto
```

to calculate two things given a radius on level 1 of the stack. Firstly, it calculates the area of a circle which has twice the radius, secondly it calculates the volume of the sphere with the given radius.

The first local variable `r` will be assigned the value on level 1 and then the program doubles it and the result is now on the stack. When we then call the `area` function, it will locally assign a new value to `r` to perform its calculations. When our `circlefun` program continues, `r` will have it’s original value back as assigned when the program started. Calling `circlefun` with a value 1 on the stack, it will produce the area of a circle of radius 2 on level 2 and the volume of a sphere with radius 1 on level 1.

stack	L4: L3: L2: L1: 1	**Result** $\longrightarrow$	*stack*	L4: L3: L2: 12.56637061 L1: 4.18879020	
input	`circlefun`		*input*		

5 Stack Manipulations

There are basically three stack manipulation commands available:

- `DUP`(^D): Duplicate L1.
- `DROP`(⌘ ⌫): Deletes L1.
- `SWAP`(^S): Swap L1 and L2.

If you press ↩ in an empty input field, you will also duplicate L1. And if you haven't started to type anything into the input field since last ↩, you can drop L1 simply by pressing backspace or delete. Once you start to type into the input field, backspace and delete will function as expected.

There are additional stack manipulations that cannot be entered as a command to the calculator, but are available as menu items with keyboard shortcuts. To recall L1 to the current cursor position in the input field, you can press ^↩. This will drop L1 from the stack and paste it into the current input text.

Copying from the stack to the input area can be done using the command ⌘↩. The default is to copy L1 but you can change the source line my moving up and down using the shortcuts ⌘↑ and ⌘↓. To swap any two neighbor levels, use the commands swap up and swap down, accessible by the shortcuts ⌥⌘↑ and ⌥⌘↓.

6 Different Bases and Grouping

The calculator can display numbers in different bases. Valid bases are between 2 and 36 inclusive. For bases larger than 10, the letters A . . . Z will be used as digits for the numbers 10 . . . 35.

To change the displayed base, push the desired base on the stack and use the `BASE` command. This command is also available in the menu and on the toolbar using the shortcut ⌘B. Numbers displayed in base 2 are prefixed with the string `0b` and numbers in base 16 are prefixed with the string `0x`. Other bases (apart from 10) are prefixed with `nn:` where `nn` is the chosen base. Here is an example were the display base is changed to hex:

stack	L4: L3: L2: L1:	 1234 16	**Result** →	*stack*	L4: L3: L2: L1:	 0x4D2 0x10
input	`16 base`			*input*		

and then changed to base 2.

stack	L4: L3: L2: L1:	 0x10 0x4D2	**Result** →	*stack*	L4: L3: L2: L1:	 0b10000 0b10011010010
input	`2 base`			*input*		

The default base for entering numbers is 10, but you can enter numbers in any base you like, independently of the chosen display base. Simply prefix your number with the base (entered in base 10) followed by a colon. Base 2 and base 16 numbers can also be entered using the prefixes `0b` and `0x`.

Display grouping is a feature where the digits of the number are grouped, separated by an underscore character. This is very handy for separating the bytes of a large hex number, for example. To change the display grouping, you push the group size on the stack and use the `DGRP` command. This command is also available in the menu and on the toolbar using the shortcut ⌘D. To remove the grouping, set the group size to zero, `0 DGRP`. When display grouping is active, the output will be padded to the left with zeros to form a full group. Here is an example were a hex integer is grouped into bytes:

stack	L4: L3: L2: L1:	 0x123456	**Result** →	*stack*	L4: L3: L2: L1:	 0x12_34_56
input	`2 dgrp`			*input*		

You can also use the underscore to separate digits when you enter them in the input field. The underscore will be silently ignored by the number parser.

7 Float numbers

Floats are displayed using the standard format 1234.567e8 where the e8 means *times 10 to the power of 8*. If you are using any other base than 10, the exponent delimiter 'e' will be replaced with a '@' and analogously the interpretation is *times b to the power of* where *b* is your chosen base. The exponent is always given in base 10.

You can switch between *normal* and *scientific* mode for displaying floats. To change float mode display, you push 0 or 1 on the stack and use the `FMODE` command. If the value on L1 is zero, normal mode will be used, and if the value on L1 is one, scientific mode will be used.

In normal mode, floats that have exponent larger than -3 and smaller than the number of significant digits will be written without exponent delimiter and exponent. In scientific mode, floats in the range $[0 \ldots 1000[$ will be written without exponent delimiter and exponent, but all other values will be written using a exponent divisible by 3. Here are some examples:

Value	Normal mode	Scientific mode
1.23456	1.23456	1.23456
999.99	999.99	999.99
1234	1234	1.234e3
0.1234	0.1234	123.4e-3
0.001234	0.001234	1.234e-3
0.0001234	1.234e-4	123.4e-6

8 Command History and Undo

All commands that are entered into the input field are saved and you can move through the history of commands by using the keyboard shortcuts ^↑ and ^↓. The command history is not saved between launches of the calculator.

The calculator also keeps an Undo history. To undo the last input, simply press ⌘Z. The last 32 inputs can be reverted. Note that an input may contain several commands and the Undo feature restores the state as it were prior to executing the input. The saved states are deleted when you quit the calculator.

9 Workspaces

You can save and load workspaces. A workspace is the collection of the *stack*, the *variables*, and the *calculator mode*. The calculator mode is the current display base and grouping, float mode and input mode. When you later open a saved workspace, you have the choice to

restore only, for instance the stack. Or the stack and the variables. In this way you can save useful variables and programs in a workspace and load them later. Note that if you choose to restore the variables from a saved workspace, your current list of variables will be deleted. Same goes with the stack.

You can open and save workspaces using the menu items in the File menu, or the shortcut ⌘O and ⌘S respectively.

10 Preferences

There are three different panes on the preference panel. In the General pane you can define how you like rpnExpress to behave between launches. You can restore the stack and the calculator mode to its previous state or not. For the variables you can choose to have all, none or only the persistent variables restored between launches. Persistent variables are described in section 3.

In the Colors pane, you can select different colors for the stack. And in the Advanced pane you can specify the precision for the floating numbers.

11 Declaration Generator

A handy declaration generator for C array declarations of large integers is available by pressing ⌘4. The item currently on level 1 can be imported to the declaration generator by pressing ⌘L. In the declaration generator window, different options for endianess and MSW/LSW ordering of the array can be chosen. The resulting array declaration is shown in the text field and can be marked and copied into a C program.

12 Bug reporting and contact information

Even though we do rigorous unit testing, we have been coding for too long to believe that our code is perfect. If you find a bug, please report it to *rpnexpress@ciphic.com*. Questions can also be mailed to that address and we will try to help the best we can.

13 Uninstall

All necessary libraries and resources are included in the rpnExpress.app bundle. To uninstall it, simply delete the rpnExpress.app application bundle. In addition, rpnExpress stores each users preferences and current state in the container directory "~/Library/Containers/com.ciphic.rpnExpress/". Additional workspaces that you have saved have file-ending ".rpne" by default, so you should be able to search for them efficiently.

14 Commands reference

An alphabetic list of all available commands in the calculator.
The commands reference is also available by pressing ⌘3.

The following notation is used to describe the different possible inputs:

Notation	Meaning
'obj'	Any object on the stack.
a, b, c	Integer
n, s, t, k	Non-negative integer
x, y, z	Real number
'symb'	Algebraic symbolic expression, number, or undefined variable
#Vname	Prefixed variable
Vname	Variable

	Friendly name	:	Addition
+	**Type**	:	Operator
	Number of parameters	:	2

Description:

Returns the sum of its arguments. In algebraic mode, addition is a binary operator and it is used as expected within an algebraic expression.

Example:

Input	
RPN mode	Algebraic mode
`3 4 +`	`3+4`

Calculates $3 + 4$.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	$x + y$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1 + symb'_2$

Friendly name : Subtraction
- **Type** : Operator
Number of parameters : 2

Description:

Returns the difference of its arguments. In algebraic mode, subtraction is a binary operator and it is used as expected within an algebraic expression.

Example:

Input	
RPN mode	Algebraic mode
`10 4 -`	`10-4`

Calculates $10 - 4$.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	$x - y$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1 - symb'_2$

*

Friendly name	:	Multiplication
Type	:	Operator
Number of parameters	:	2

Description:

Returns the products of its arguments. In algebraic mode, multiplication is a binary operator and it is used as expected within an algebraic expression.

Example:

Input		
RPN mode	Algebraic mode	Calculates $3 \cdot 4$.
`3 4 *`	`3*4`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	$x \cdot y$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1 \cdot symb'_2$

/

Friendly name	:	Division
Type	:	Operator
Number of parameters	:	2

Description:

Returns the quotient of its arguments. The first argument is divided by the second. In algebraic mode, division is a binary operator and it is used as expected within an algebraic expression.

Example:

Input		
RPN mode	Algebraic mode	Calculates $12/4$.
`12 4 /`	`12/4`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	x/y
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1/symb'_2$

^

Friendly name	: Power of
Type	: Operator
Number of parameters	: 2

Description:

Calculates x^y. In RPN mode, the command `POW` can also be used. In algebraic mode, `^` is a binary operator and it is used as expected within an algebraic expression.

Example:

Input	
RPN mode	Algebraic mode
`3 4 ^`	`3^4`

Calculates 3^4.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	x^y
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1 \text{^} symb'_2$

Argument range:

Argument	Range
x	Real number
y	Real number

See also:

POW

!

Friendly name	:	Factorial
Type	:	Operator
Number of parameters	:	1

Description:

Returns the factorial of its argument. In algebraic mode, factorial is an operator and it is used as a postfix to the argument. The factorial of n is defined as $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, or recursively as

$$\begin{aligned} n! &= n \cdot (n-1)! \\ 0! &= 1 \end{aligned}$$

Example:

Input		
RPN mode	Algebraic mode	Calculates 10 factorial
`10 !`	`10!`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	$n!$
$'symb'$	$\rightarrow$	$'symb!'$

Argument range:

Argument	Range
n	$0 \leq n < 2^{32} - 1$

Note:

For very large arguments, in the order of one million, this operation can take considerable time.

See also:

FACT

%

Friendly name	: Modulo operation
Type	: Operator
Number of parameters	: 2

Description:

Returns the remainder when a is divided by n, written $a \mod n$. In algebraic mode, addition is a binary operator and it is used as expected within an algebraic expression. If a is negative, it follows the normal convention that $-1 = (n-1) \mod n$. The sign of n is ignored.

Example:

Input	
RPN mode	Algebraic mode
`37 5 %`	`37%5`

Calculates $37 \mod 5$.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
a	n	$\rightarrow$	$a \mod n$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'symb_1$ % $symb'_2$

Argument range:

Argument	Range
a	Integer
n	Non-zero integer

See also:

MOD

ABS

Friendly name : Absolute value
Type : Function
Number of parameters : 1

Description:
Returns the absolute value of its argument.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$\lvert x \rvert$
$'symb'$	$\rightarrow$	$'\text{ABS}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:
NEG

ACOSH

Friendly name : Inverse Hyperbolic Cosine
Type : Function
Number of parameters : 1

Description:

Returns the inverse hyperbolic cosine on its principle domain $(1, \infty)$. The inverse hyperbolic cosine on this domain is defined as

$$acosh(x) = ln(x + \sqrt{x^2 - 1})$$

Example:

Input	
RPN mode	Algebraic mode
`4 ACOSH`	`ACOSH(4)`

Calculates $acosh(4)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$acosh(x)$
$'symb'$	$\rightarrow$	'ACOSH($symb$)'

Argument range:

Argument	Range
x	$1 < x$

See also:

COSH, **ASINH**, **ATANH**

ACOS

Friendly name : Arc-cosine
Type : Function
Number of parameters : 1

Description:

Returns the *arc-cosine* function, the inverse function of cosine on its principle domain $[0 \ldots \pi]$. $acos(y) = x, -1 \leq y \leq 1$ returns $x, 0 \leq x \leq \pi$ such that $cos(x) = y$.

Example:

Input	
RPN mode	Algebraic mode
`0.5 ACOS`	`ACOS(0.5)`

Calculates at what angle the cosine function equals 0.5.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
y	$\rightarrow$	$acos(y)$
$'symb'$	$\rightarrow$	$'\text{ACOS}(symb)'$

Argument range:

Argument	Range
y	$-1 \leq y \leq 1$

See also:

COS, **ASIN**, **ATAN**

AND

Friendly name : Bitwise And
Type : Operator
Number of parameters : 2

Description:

Returns the logical bitwise AND of its arguments. If the number of bits needed to represent the two arguments are different, this operation will act as if the shortest argument is padded with zeros in the most significant bits.

Example:

Input	
RPN mode	Algebraic mode
`0x91 0x9 AND`	`AND(0x91,0x9)`

How to calculate 0x91 AND 0x09.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
s	t	$\rightarrow$	s AND t
$'symb'_1$	$'symb'_2$	$\rightarrow$	'AND$(symb_1, symb_2)'$

Argument range:

Argument	Range
s	Non-negative integer
t	Non-negative integer

See also:

OR, **XOR**

ASINH

Friendly name : Inverse Hyperbolic Sine
Type : Function
Number of parameters : 1

Description:

Returns the inverse hyperbolic sine of its argument. The inverse hyperbolic sine is defined as

$$asinh(x) = ln(x + \sqrt{x^2 + 1})$$

Example:

Input	
RPN mode	Algebraic mode
`4 ASINH`	`ASINH(4)`

Calculates $asinh(4)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$asinh(x)$
$'symb'$	$\rightarrow$	'ASINH($symb$)'

Argument range:

Argument	Range
x	Real number

See also:

SINH, **ACOSH**, **ATANH**

ASIN

Friendly name	:	Arc-sine
Type	:	Function
Number of parameters	:	1

Description:

Returns the *arc-sine* function, the inverse function of sine on its principle domain $[-\pi/2 \ldots \pi/2]$. $asin(y) = x, -1 \leq y \leq 1$ returns $x, -\pi/2 \leq x \leq \pi/2$ such that $sin(x) = y$.

Example:

Input	
RPN mode	Algebraic mode
`0.2 ASIN`	`ASIN(0.2)`

Calculates at what angle the sine function equals 0.2.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
y	$\rightarrow$	$asin(y)$
$'symb'$	$\rightarrow$	$'\text{ASIN}(symb)'$

Argument range:

Argument	Range
y	$-1 \leq y \leq 1$

See also:

SIN, **ACOS**, **ATAN**

ATANH

Friendly name : Inverse Hyperbolic Tangent
Type : Function
Number of parameters : 1

Description:

Returns the inverse hyperbolic tangent. The inverse hyperbolic tangent is defined as

$$atanh(x) = \frac{1}{2} ln(\frac{x+1}{x-1})$$

Example:

Input	
RPN mode	Algebraic mode
`-0.5 ATANH`	`ATANH(-0.5)`

Calculates $atanh(-0.5)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$atanh(x)$
$'symb'$	$\rightarrow$	$'\text{ATANH}(symb)'$

Argument range:

Argument	Range
x	$-1 < x < 1$

See also:

TANH, **ASINH**, **ACOSH**

ATAN

Friendly name : Arc-tangent
Type : Function
Number of parameters : 1

Description:

Returns the *arc-tangent* function, the inverse function of the tangent function on its principle domain $[-\pi/2 \ldots \pi/2]$. $atan(y) = x$ returns $x, -\pi/2 \leq x \leq \pi/2$ such that $tan(x) = y$.

Example:

Input	
RPN mode	Algebraic mode
`100 ATAN`	`ATAN(100)`

Calculates at what angle the tangent function equals 100.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
y	$\rightarrow$	$atan(y)$
$'symb'$	$\rightarrow$	$'\text{ATAN}(symb)'$

Argument range:

Argument	Range
y	Real number

See also:

TAN, **ACOS**, **ASIN**

BASE

Friendly name : Display base
Type : Command
Number of parameters : 1

Description:

Sets the base of the displayed stack numbers. Valid bases are $2 \ldots 36$.
For bases larger than 10, the letters A . . . Z will be used as digits for the numbers $10 \ldots 35$.
Numbers displayed in base 2 are prefixed with `0b`. Numbers displayed in base 16 are prefixed with `0x` and numbers in other bases (except 10) are prefixed with `nn:` where `nn` is the decimal representation of the base.
The base used for input is not related to the displayed base, but will always be assumed to be in base 10, unless it is prefixed with `0b`, `0x`, or `nn:`.

Example:

Input		
RPN mode	Algebraic mode	Sets the display base to 16.
`16 BASE`	`BASE(16)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	

Argument range:

Argument	Range
n	$2 \leq n \leq 36$

See also:

DGRP

BCNT

Friendly name : Bit count
Type : Function
Number of parameters : 1

Description:

Returns the number of 1 bits in the base 2 representation of its argument. This is sometimes also referred to as the *Hamming weight*.

Example:

Input	
RPN mode	Algebraic mode
`1234 BCNT`	`BCNT(1234)`

Counting the bits in 1234 that are set to 1.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	s

where s is the number of bits set to 1 in the base 2 representation of n.

Argument range:

Argument	Range
n	Non-negative integer

See also:

NOT

BIN

Friendly name : Binomial coefficient
Type : Function
Number of parameters : 2

Description:

Returns the binomial coefficient $\binom{n}{k}$. The binomial coefficient is defined as

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Example:

Input	
RPN mode	Algebraic mode
`24 8 BIN`	`BIN(24,8)`

Calculates $\binom{24}{8}$.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
n	k	$\rightarrow$	$\binom{n}{k}$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\text{BIN}(symb_1, symb_2)'$

Argument range:

Argument	Range
n	Non-negative integer.
k	Non-negative integer.

Note:

For large n, in the order of one million, this operation can take considerable time.

CEIL

Friendly name : Ceiling function
Type : Function
Number of parameters : 1

Description:

Returns the ceiling of its argument. The ceiling of x is the integer n which is larger or equal to x.

Example:

Input	
RPN mode	Algebraic mode
`3.5 CEIL`	`CEIL(3.5)`

Calculating the ceiling of 3.5 to a result of 4.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	n
$'symb'$	$\rightarrow$	$'\text{CEIL}(symb)'$

where n is the ceiling of x.

Argument range:

Argument	Range
x	Real number

See also:

ROUND, **FLOOR**

CLEAR

Friendly name : Clear stack
Type : Command
Number of parameters : 0

Description:

Clears the stack. In algebraic mode this commands is used without any parameters.

Example:

Input	
RPN mode	Algebraic mode
`CLEAR`	`CLEAR`

The command usage is the same in both input modes.

See also:

DROP

COSH

Friendly name : Hyperbolic Cosine
Type : Function
Number of parameters : 1

Description:

Returns the hyperbolic cosine of its argument. The hyperbolic cosine is defined as

$$cosh(x) = \frac{e^x + e^{-x}}{2}$$

Example:

Input	
RPN mode	Algebraic mode
`4 COSH`	`COSH(4)`

Calculates $cosh(4)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$cosh(x)$
$'symb'$	$\rightarrow$	$'\text{COSH}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

SINH, **TANH**, **ACOSH**

COS

Friendly name : Cosine
Type : Function
Number of parameters : 1

Description:

Returns the cosine of its argument. The argument is assumed to be given in radians. Due to the imperfect representation of π, trigonometric expressions that mathematically would evaluate to 0, such as $cos(\pi/2)$, will not necessarily evaluate to 0.

Example:

Input		
RPN mode	Algebraic mode	Calculates $cos(\pi/3)$
`pi 3 / COS`	`COS(pi/3)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$cos(x)$
$'symb'$	$\rightarrow$	$'\mathrm{COS}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

SIN, **ACOS**, **PI**

CPER

Friendly name : Clear persistency
Type : Command
Number of parameters : 1

Description:

Removes the persistent property from a variable. Depending on user preferences, the variable might be deleted when the user quits the program.

Example:

Input		
RPN mode	Algebraic mode	Clear the persistency of variable #a.
`#a CPER`	`CPER(#a)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
$\#Vname$	$\rightarrow$	

See also:

MPER, **DEL**

DELALL

Friendly name : Delete all variables
Type : Command
Number of parameters : 0

Description:
Deletes all variables, including persistent ones.

Example:

Input		
RPN mode	Algebraic mode	Deletes all variables
`DELALL`	`DELALL`	

Input/Output:

Level 1 / Argument 1	Level 1 / Result
→	

See also:
DEL

DEL

Friendly name : Delete variable
Type : Command
Number of parameters : 1

Description:

Deletes the variable on level 1. The name of the variable must be prefixed with a #-sign so that it is not evaluated.

Example:

Input		
RPN mode	Algebraic mode	Deletes the variable *a*
`#a DEL`	`DEL(#a)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
$\#Vname$	$\rightarrow$	

See also:

DELALL

DGRP

Friendly name : Display grouping
Type : Command
Number of parameters : 1

Description:

Separates the digits of the stack numbers into groups of digits using an underscore. Valid grouping size is $0 \ldots 32$, inclusive. Setting the grouping size to zero removes the grouping. Input numbers can be entered with the underscore symbol present anywhere between digits. It will be silently ignored by the input parser.

Example:

Input		
RPN mode	Algebraic mode	Sets the display grouping to 4 digits.
`4 DGRP`	`DGRP(4)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	

Argument range:

Argument	Range
n	$0 \leq n \leq 32$

See also:

BASE

DROP

Friendly name : Drop stack item
Type : Command
Number of parameters : 1

Description:

Removes the item on level 1 from the stack.
In algebraic mode this command is used without any parameters.

Example:

Input	
RPN mode	Algebraic mode
`DROP`	`DROP`

The command usage is the same in both input modes.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
obj	→	

See also:

DUP,**DROPI**, **DROPN**

DROPI

Friendly name : Drop stack item at index
Type : Command
Number of parameters : 1

Description:

First the n parameter is popped from the stack. Then this command removes the item on level n from the stack, where $n > 0$.

Example:

Input		
RPN mode	Algebraic mode	Drops level 3 from the stack.
`3 DROPI`	`DROPI(3)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	

Argument range:

Argument	Range
n	$n > 0$

See also:

DROP DROPN

DROPN

Friendly name : Drop multiple stack items
Type : Command
Number of parameters : 1

Description:

First the a parameter is popped from the stack. Then this command removes a item from the stack. The `DROPN` command is safe in that sense that it will not return an error if there is not enough items in the stack. It will just drop all in that case. If a is zero or negative, it will do nothing.

Example:

Input		
RPN mode	Algebraic mode	Drops 4 items from the stack.
`4 DROPN`	`DROPN(4)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
a	$\rightarrow$	

Argument range:

Argument	Range
a	Integer

See also:

DROP DROPI

DUP

Friendly name : Duplicate stack item
Type : Command
Number of parameters : 1

Description:

Duplicates the item on level 1 and pushes it onto the stack.
In algebraic mode this command is used without any parameters.

Example:

Input	
RPN mode	Algebraic mode
`DUP`	`DUP`

The command usage is the same in both input modes.

Input/Output:

Level 1		Level 2	Level 1
obj_1	$\rightarrow$	obj_1	obj_1

See also:

DROP

DUPI

Friendly name : Duplicates stack item at index
Type : Command
Number of parameters : 1

Description:

First the n parameter is popped from the stack. Then this command duplicates the item on level $n > 0$ and pushes it on the stack.

Example:

Input		
RPN mode	Algebraic mode	Duplicates the item at level 3.
`3 DUPI`	`DUPI(3)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	

Argument range:

Argument	Range
n	$n > 0$

See also:

DUP DROPI

ERFC

Friendly name : Complimentary error function
Type : Function
Number of parameters : 1

Description:

Returns the complimentary error function of its argument x, erfc(x), defined as

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

Example:

Input		
RPN mode	Algebraic mode	Calculates erfc(0.5).
`0.5 ERFC`	`ERFC(0.5)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	erfc(x)
$'symb'$	$\rightarrow$	'ERFC($symb$)'

Argument range:

Argument	Range
x	Real number.

See also:

ERF

ERF

Friendly name : Gauss error function
Type : Function
Number of parameters : 1

Description:

Returns the Gauss error function of its argument x, erf(x), defined as

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$$

Example:

Input		
RPN mode	Algebraic mode	Calculates erf(0.5).
`0.5 ERF`	`ERF(0.5)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	erf(x)
$'symb'$	$\rightarrow$	$'\text{ERF}(symb)'$

Argument range:

Argument	Range
x	Real number.

See also:

ERFC

EULER

Friendly name : The Euler constant
Type : Constant
Number of parameters : 0

Description:

Returns the Euler constant value $\gamma = 0.5772\ldots$, given with the current floating precision. The constant name is case insensitive, and can be use in place of a number in algebraic expressions.

Example:

Input	
RPN mode	Algebraic mode
`EULER NEG EXP`	`EXP(-EULER)`

Calculates $e^{-\gamma}$ using the Euler constant.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
	$\rightarrow$	γ

See also:

PI, **ZETA**

EVAL

Friendly name : Evaluate stack item
Type : Command
Number of parameters : 1

Description:

The EVAL command evaluates the level 1 item on the stack according to the following table:

Object	Action
Number	Pushes back the number on the stack.
Algebraic expression	Evaluates as much of the expression as possible. If undefined variables are encountered the result will be a new algebraic expression. If no undefined variables are present, a numeric result will be produced.
Variable	Recalls the definition of the variable on the stack and calls EVAL again. If the variable is undefined, it is pushed back on the stack.
Program	For each token in the program sequence, push the token on the stack and call EVAL again. Commands inside a program are executed.

Note:

Since rpnExpress is not a *Computer Algebra System* (CAS), evaluating an algebraic expression with undefined variables does not completely simplify the expression. Depending on the internal representation, you might see answers of the type '$4 * x + 17 - 24$', which could be further simplified. Support for simplification of algebraic expressions might be introduced in later versions.

EXP10

Friendly name : Base 10 exponential function
Type : Function
Number of parameters : 1

Description:

Returns the base 10 exponential of its argument, i.e., 10 to the power of the argument.

Example:

Input	
RPN mode	Algebraic mode
`2.3 EXP10`	`EXP10(2.3)`

Calculates $10^{2.3}$

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	e^x
$'symb'$	$\rightarrow$	$'\text{EXP10}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

EXP, **EXP2**, **LOG10**

EXP2

Friendly name : Base 2 exponential function
Type : Function
Number of parameters : 1

Description:

Returns the base 2 exponential of its argument, i.e., 2 to the power of the argument.

Example:

Input	
RPN mode	Algebraic mode
`2.3 EXP2`	`EXP2(2.3)`

Calculates $2^{2.3}$

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	e^x
$'symb'$	$\rightarrow$	$'\text{EXP2}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

EXP, **EXP10**, **LOG2**

EXP

Friendly name : Exponential function
Type : Function
Number of parameters : 1

Description:

Returns the natural base exponential of its argument, i.e., the natural base e to the power of the argument.

Example:

Input		
RPN mode	Algebraic mode	Calculates $e^{2.3}$
`2.3 EXP`	`EXP(2.3)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	e^x
$'symb'$	$\rightarrow$	$'\text{EXP}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

EXP2, **EXP10**, **LN**

FACT

Friendly name : Factorial
Type : Function
Number of parameters : 1

Description:

Returns the factorial of its argument. In algebraic mode, the operator variant $n!$ can also be used as a postfix operator. The factorial of n is defined as $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, or recursively as

$$\begin{aligned} n! &= n \cdot (n-1)! \\ 0! &= 1 \end{aligned}$$

Example:

Input		
RPN mode	Algebraic mode	Calculating 10 factorial
`10 FACT`	`FACT(10)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	$n!$
$'symb'$	$\rightarrow$	$'\text{FACT}(symb)'$

Argument range:

Argument	Range
n	$0 \leq n < 2^{32} - 1$

See also:

!

FLOOR

Friendly name : Floor function
Type : Function
Number of parameters : 1

Description:

Returns the floor of its argument. The floor of x is the integer n which is smaller or equal to x.

Example:

Input	
RPN mode	Algebraic mode
`3.5 FLOOR`	`FLOOR(3.5)`

Calculating the floor of 3.5 to a result of 3.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	n
$'symb'$	$\rightarrow$	'FLOOR($symb$)'

where n is the floor of x.

Argument range:

Argument	Range
x	Real number

See also:

ROUND, **CEIL**

FMODE

Friendly name : Float display mode
Type : Command
Number of parameters : 1

Description:

In normal mode, floats that have exponent larger than -3 and smaller than the number of significant digits will be written without exponent delimiter and exponent. In scientific mode, floats in the range $[0 \ldots 1000[$ will be written without exponent delimiter and exponent, but all other values will be written using a exponent divisible by 3.

Example:

Input	
RPN mode	Algebraic mode
`1 FMODE`	`FMODE(1)`

Sets the float mode to Scientific mode

Input/Output:

Level 1 / Argument 1		Level 1 / Result
n	$\rightarrow$	

Argument range:

Value	Float mode
0	Normal mode
1	Scientific mode

See also:

BASE, **DGRP**

FRAC

Friendly name : Fractional part
Type : Function
Number of parameters : 1

Description:

Returns the fractional part of its argument.

Example:

Input		
RPN mode	Algebraic mode	Fractional part of 1234.5678
`1234.5678 FRAC`	`FRAC(1234.5678)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	y

where y is the fractional part of x.

Argument range:

Argument	Range
x	Real number

See also:

TRUNC

FROUND

Friendly name : Fractional rounding
Type : Function
Number of parameters : 2

Description:

Rounds the argument x to have n significant digits in its fraction, rounding a tie away from zero. If $n = 0$ it is equivalent to calling `ROUND`.

Example:

Input	
RPN mode	Algebraic mode
`1234.56789 2`	`FROUND(1234.56789,2)`
`FROUND`	

Rounding 1234.56789 to a result of 1234.57.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	n	$\rightarrow$	y
$'symb'_1$	$symb_2$	$\rightarrow$	$'\text{FROUND}(symb_1, symb_2)'$

Argument range:

Argument	Range
x	Real number
n	Non-negative integer

See also:

ROUND

GCD

Friendly name : Greatest common divisor
Type : Function
Number of parameters : 2

Description:

Returns the greatest common divisor gcd(a, b) of its arguments. The gcd is always positive regardless of the signs of the arguments. Furthermore, this functions defines gcd$(0, 0) = 0$.

Example:

Input		
RPN mode	Algebraic mode	Calculates gcd$(63, 14)$
`63 14 GCD`	`GCD(63,14)`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
a	b	$\rightarrow$	gcd(a, b)
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'$GCD$(symb_1, symb_2)'$

Argument range:

Argument	Range
a	Integer.
b	Integer.

See also:

LCM

INVMOD

Friendly name	:	Inverse modulo
Type	:	Function
Number of parameters	:	2

Description:

Returns the inverse of a modulo n, written $a^{-1} \mod n$, such that $a \cdot a^{-1} \mod n = 1$. If a is negative, it follows the normal convention that $-1 = (n-1) \mod n$. The sign of n is ignored.

Example:

Input	
RPN mode	Algebraic mode
`5 37 INVMOD`	`INVMOD(5,37)`

Calculates the inverse of $5 \mod 37$.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
a	n	$\rightarrow$	$a^{-1} \mod n$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\text{INVMOD}(symb_1, symb_2)'$

Argument range:

Argument	Range
a	Integer
n	Non-zero integer

See also:

MOD, **POWMOD**

INV

Friendly name : Inverse
Type : Function
Number of parameters : 1

Description:
Returns the inverse of its argument.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$1/x$
$'symb'$	$\rightarrow$	$'\text{INV}(symb)'$

Argument range:

Argument	Range
x	Non-zero real number

See also:
INVMOD

ISPRIME

Friendly name : Primality testing
Type : Function
Number of parameters : 1

Description:

Performs a probabilistic test whether the argument is a prime number or not. It returns three possible values depending on the certainty of the outcome.

-1 Probably prime.
0 Definitely composite.
1 Definitely prime.

The sign of the argument is ignored.

Example:

Input		
RPN mode	Algebraic mode	Checks if 4001 is prime
`4001 ISPRIME`	`ISPRIME(4001)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
a	$\rightarrow$	$-1/0/1$
$'symb'$	$\rightarrow$	'ISPRIME($symb$)'

Argument range:

Argument	Range
a	Integer.

Note:

If this function is returning -1, the probability that the argument is composite is very low. The documentation for the GMP function that does the primality testing is not very clear on what the probability is, but internally, rpnExpress calls the `mpz_probab_prime_p` function with the parameter `reps`$= 80$.

See also:

NPRIME

LCM

Friendly name : Least common multiple
Type : Function
Number of parameters : 2

Description:

Returns the least common multiple lcm(a, b) of its arguments. The lcm is always positive regardless of the signs of the arguments. If any of the arguments is zero, it will return zero.

Example:

Input		
RPN mode	Algebraic mode	Calculates lcm$(7, 5)$
`7 5 LCM`	`LCM(7,5)`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
a	b	$\rightarrow$	lcm(a, b)
$'symb'_1$	$'symb'_2$	$\rightarrow$	'LCM$(symb_1, symb_2)$'

Argument range:

Argument	Range
a	Integer.
b	Integer.

See also:

GCD

LN

Friendly name : Natural logarithm
Type : Function
Number of parameters : 1

Description:

Returns the natural logarithm of its argument, i.e., the power to which e has to be raised to produce the argument.

Example:

Input		
RPN mode	Algebraic mode	Calculates $ln(12.5)$
`12.5 LN`	`LN(12.5)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$ln(x)$
$'symb'$	$\rightarrow$	$'\mathrm{LN}(symb)'$

Argument range:

Argument	Range
x	$0 < x$

See also:

LOG2, **LOG10**, **EXP**

LOG10

Friendly name : Base 10 logarithm
Type : Function
Number of parameters : 1

Description:

Returns the base 10 logarithm of its argument, i.e., the power to which 10 has to be raised to produce the argument.

Example:

Input		
RPN mode	Algebraic mode	Calculates $log_{10}(12.5)$
`12.5 LOG10`	`LOG10(12.5)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$log_{10}(x)$
$'symb'$	$\rightarrow$	'LOG10($symb$)'

Argument range:

Argument	Range
x	$0 < x$

See also:

LN, **LOG2**, **EXP10**

LOG2

Friendly name : Base 2 logarithm
Type : Function
Number of parameters : 1

Description:

Returns the base 2 logarithm of its argument, i.e., the power to which 2 has to be raised to produce the argument.

Example:

Input		
RPN mode	Algebraic mode	Calculates $log_2(12.5)$
`12.5 LOG2`	`LOG2(12.5)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$log_2(x)$
$'symb'$	$\rightarrow$	$'\mathrm{LOG2}(symb)'$

Argument range:

Argument	Range
x	$0 < x$

See also:

LN, **LOG10**, **EXP2**

MOD

Friendly name : Modulo operation
Type : Function
Number of parameters : 2

Description:

Returns the remainder when a is divided by n, written $a \mod n$. In algebraic mode, the infix operator % can also be used. If a is negative, it follows the normal convention that $-1 = (n-1) \mod n$. The sign of n is ignored.

Example:

Input		
RPN mode	Algebraic mode	Calculates $37 \mod 5$.
`37 5 MOD`	`MOD(37,5)`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
a	n	$\rightarrow$	$a \mod n$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\mathrm{MOD}(symb_1, symb_2)'$

Argument range:

Argument	Range
a	Integer
n	Non-zero integer

See also:

%, **INVMOD**, **POWMOD**

MPER

Friendly name : Make persistent
Type : Command
Number of parameters : 1

Description:

Makes a variable persistent. A persistent variable can (depending on user preferences) be restored between launches.

Example:

Input		
RPN mode	Algebraic mode	Make the variable #a persistent.
`#a MPER`	`MPER(#a)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
$\#Vname$	$\rightarrow$	

See also:

CPER, **STO**

NEG

Friendly name : Negation
Type : Function
Number of parameters : 1

Description:
Returns the negative of its argument.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$-x$
$'symb'$	$\rightarrow$	$'\text{NEG}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:
ABS

NOT

Friendly name : Bitwise inverse
Type : Function
Number of parameters : 1

Description:
Returns the bitwise inverse of its arguments.

Example:

Input	
RPN mode	Algebraic mode
`0x123456ED NOT`	`NOT(0x123456ED)`

Inverting the bits in $0x123456ED$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
s	$\rightarrow$	t
$'symb'$	$\rightarrow$	$'\text{NOT}(symb)'$

where t is the bitwise inverse of s.

Argument range:

Argument	Range
s	Non-negative integer

See also:
AND, **OR**

NPRIME

Friendly name : Next prime
Type : Function
Number of parameters : 1

Description:

Returns the next prime (using the probabilistic test of `ISPIME`) that is greater than then argument. The sign of the argument is ignored an the argument is treated as if it is positive.

Example:

Input		
RPN mode	Algebraic mode	Get the next prime number after 5000.
`5000 NPRIME`	`NPRIME(5000)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
a	$\rightarrow$	b
$'symb'$	$\rightarrow$	'NPRIME($symb$)'

where b is the next prime greater than $|a|$.

Argument range:

Argument	Range
a	Integer.

See also:

ISPRIME

NROOT

Friendly name : N:th root
Type : Function
Number of parameters : 2

Description:

Returns the n:th-root of a real number, written $\sqrt[n]{x}$.

Example:

Input	
RPN mode	Algebraic mode
`3 -17 NROOT`	`NROOT(3,-27)`

Calculates the 3:rd root of -27.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
n	x	$\rightarrow$	$\sqrt[n]{x}$
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\text{NROOT}(symb_1, symb_2)'$

Argument range:

Argument	Range
n	Positive integer (and odd for $x < 0$).
x	Real number.

See also:

POW, **SQRT**

OR

Friendly name : Bitwise Or
Type : Operator
Number of parameters : 2

Description:

Returns the logical bitwise OR of its arguments. If the number of bits needed to represent the two arguments are different, this operation will act as if the shortest argument is padded with zeros in the most significant bits.

Example:

Input	
RPN mode	Algebraic mode
`0x91 0x9 OR`	`OR(0x91,0x9)`

How to calculate 0x91 OR 0x09.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
s	t	$\rightarrow$	s OR t
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\text{OR}(symb_1, symb_2)'$

Argument range:

Argument	Range
s	Non-negative integer
t	Non-negative integer

See also:

AND, **XOR**

PI

Friendly name : Pi (π)
Type : Constant
Number of parameters : 0

Description:

Returns the constant value π, given with the current floating precision. The constant name is case insensitive, and can be use in place of a number in algebraic expressions.

Example:

Input	
RPN mode	Algebraic mode
`pi 4 / SIN`	`SIN(pi/4)`

Calculates $sin(\pi/4)$ using the constant pi.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
	$\rightarrow$	π

See also:

SIN, **COS**

POWMOD

Friendly name : Exponentiation modulo
Type : Function
Number of parameters : 3

Description:

Returns a to the power b modulo n, written $a^b \mod n$. If a is negative, it follows the normal convention that $-1 = (n-1) \mod n$. If b is negative, it will try to invert a mod n, and calculate $a^{-b} \mod n$. If the inverse of $a \mod n$ does not exist, it will fail for negative b. The sign of n is ignored.

Example:

Input	
RPN mode	Algebraic mode
`5 7 37 POWMOD`	`POWMOD(5,7,37)`

Calculates $5^7 \mod 37$.

Input/Output:

Level 3 / Arg. 1	Level 2 / Arg. 2	Level 1 / Arg. 3		Level 1 / Result
a	b	n	$\rightarrow$	$a^b \mod n$
$'symb'_1$	$'symb'_2$	$'symb'_3$	$\rightarrow$	'POWMOD$(symb_1, symb_2, symb_3)$'

Argument range:

Argument	Range
a	Integer
b	Integer
n	Non-zero integer

See also:

MOD, **INVMOD**

POW

Friendly name : Power of
Type : Function
Number of parameters : 2

Description:

Calculates x^y. In algebraic mode, the infix notation ^ can also be used. In algebraic mode, ^ is a binary operator and it is used as expected within an algebraic expression.

Example:

Input	
RPN mode	Algebraic mode
`3 4 POW`	`POW(3,4)`

Calculates 3^4.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
x	y	$\rightarrow$	x^y
$'symb'_1$	$'symb'_2$	$\rightarrow$	'POW$(symb_1, symb_2)$'

Argument range:

Argument	Range
x	Real number
y	Real number

Special cases which will result in an error are:

- $x = 0$ and $y < 0$.
- $x < 0$ and y not an integer.

See also:

^, **SQ**, **NROOT**

REC

Friendly name : Recall variable
Type : Command
Number of parameters : 1

Description:

The REC command recalls the un-evaluated value of the variable on level 1. The variable value is pushed to level 1. Assume that the expression '$3 * r$^2' has previously been stored in the variable `#a`. By the input `#a REC` or `REC(#a)` depending on the input mode, the expression '$3 * r$^2' will be pushed to the stack on level 1.

Example:

Input	
RPN mode	Algebraic mode
`#a REC`	`REC(#a)`

Recalling the value of variable #a to level 1

Input/Output:

Level 1 / Argument 1		Level 1 / Result
$\#Vname$	$\rightarrow$	obj

See also:

STO, **DEL**

ROLL

Friendly name : Roll stack items
Type : Command
Number of parameters : 2

Description:

Performs a circular shift of the n top items on the stack a times. If $a > 0$ then the circular shift is performed by removing the item at index n and pushing it back on top of the stack. If $a < 0$ then the circular shift is performed by removing the item at index 1 and inserting it at position n, pushing the elements at position n and above down in the stack (to higher levels). The command `SWAP` is a special case of `2 1 ROLL`.

Example:

Input	
RPN mode	Algebraic mode
`3 2 ROLL`	`ROLL(3,2)`

Rolls the 3 top stack items 2 steps

stack	L4:	4	**Result** →	*stack*	L4: 4
	L3:	3			L3: 1
	L2:	2			L2: 3
	L1:	1			L1: 2
input	`3 2 roll`			*input*	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
n	a	$\rightarrow$	

Argument range:

Argument	Range
n	$n > 0$
a	Integer

See also:

SWAP

ROUND

Friendly name : Round to integer
Type : Function
Number of parameters : 1

Description:

Rounds the argument to its nearest integer, rounding a tie away from zero.

Example:

Input	
RPN mode	Algebraic mode
`3.5 ROUND`	`ROUND(3.5)`

Rounding 3.5 to a result of 4.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	n
$'symb'$	$\rightarrow$	$'\text{ROUND}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

TRUNC

SINH

Friendly name : Hyperbolic Sine
Type : Function
Number of parameters : 1

Description:

Returns the hyperbolic sine of its argument. The hyperbolic sine is defined as

$$sinh(x) = \frac{e^x - e^{-x}}{2}$$

Example:

Input	
RPN mode	Algebraic mode
`4 SINH`	`SINH(4)`

Calculates $sinh(4)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$sinh(x)$
$'symb'$	$\rightarrow$	$'\text{SINH}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

COSH, **TANH**, **ASINH**

SIN

Friendly name : Sine
Type : Function
Number of parameters : 1

Description:

Returns the sine of its argument. The argument is assumed to be given in radians. Due to the imperfect representation of π, trigonometric expressions that mathematically would evaluate to 0, such as $sin(\pi)$, will not necessarily evaluate to 0.

Example:

Input		
RPN mode	Algebraic mode	Calculates $sin(\pi/2)$
`pi 2 / SIN`	`SIN(pi/2)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$sin(x)$
$'symb'$	$\rightarrow$	$'\text{SIN}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

COS, **ASIN**,**PI**

SQRT

Friendly name : Square root
Type : Function
Number of parameters : 1

Description:

Returns the square root of its argument, $\sqrt{x}$.

Example:

Input		
RPN mode	Algebraic mode	Square root of 16
`16 SQRT`	`SQRT(16)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$\sqrt{x}$
$'symb'$	$\rightarrow$	$'\text{SQRT}(symb)'$

Argument range:

Argument	Range
x	Non-negative real number

See also:

SQ

SQ

Friendly name : Squaring
Type : Function
Number of parameters : 1

Description:

Returns the square of its argument, x^2.

Example:

Input		
RPN mode	Algebraic mode	Square of 4.1
`4.1 SQ`	`SQ(4.1)`	

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	x^2
$'symb'$	$\rightarrow$	$'\mathrm{SQ}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

SQRT

STO

Friendly name : Store value
Type : Command
Number of parameters : 2

Description:

The STO commands stores values, programs or expressions into variables.

Example:

Input		
RPN mode	Algebraic mode	Storing the value 1024 into the variable #kb
`1024 #kb STO`	`STO(1024,#kb)`	

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
obj	$\#Vname$	$\rightarrow$	

See also:

REC, **DEL**

SWAP

Friendly name : Swap stack items
Type : Command
Number of parameters : 2

Description:

The SWAP command swaps the level 1 item with the level 2 item. In algebraic mode this command is used without any parameters.

Example:

Input	
RPN mode	Algebraic mode
`SWAP`	`SWAP`

The command usage is the same in both input modes.

Input/Output:

Level 2	Level 1		Level 2	Level 1
obj_2	obj_1	$\rightarrow$	obj_1	obj_2

See also:

DROP, **ROLL**

TANH

Friendly name : Hyperbolic tangent
Type : Function
Number of parameters : 1

Description:

Returns the hyperbolic tangent of its argument. The hyperbolic tangent is defined as

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Example:

Input	
RPN mode	Algebraic mode
`4 TANH`	`TANH(4)`

Calculates $tanh(4)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$tanh(x)$
$'symb'$	$\rightarrow$	'TANH($symb$)'

Argument range:

Argument	Range
x	Real number

See also:

COSH, **SINH**, **ATANH**

TAN

Friendly name : Tangent
Type : Function
Number of parameters : 1

Description:

Returns the tangent of its argument. The argument is assumed to be given in radians. Due to the imperfect representation of π, trigonometric expressions that mathematically would evaluate to infinity, such as $tan(\pi/2)$, will evaluate to a very large number.

Example:

Input	
RPN mode	Algebraic mode
`pi 4 / TAN`	`TAN(pi/4)`

Calculates $tan(\pi/4)$

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$tan(x)$
$'symb'$	$\rightarrow$	$'\text{TAN}(symb)'$

Argument range:

Argument	Range
x	Real number

See also:

SIN, **COS**, **ATAN**, **PI**

TRUNC

Friendly name : Truncate
Type : Function
Number of parameters : 1

Description:
Truncates its argument, keeping only the integer part.

Example:

Input	
RPN mode	Algebraic mode
`3.5 TRUNC`	`TRUNC(3.5)`

Truncating 3.5 to a result of 3.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	n
$'symb'$	$\rightarrow$	$'\text{TRUNC}(symb)'$

where n is the integer part of x.

Argument range:

Argument	Range
x	Real number

See also:
ROUND

XOR

Friendly name : Bitwise exclusive or
Type : Operator
Number of parameters : 2

Description:

Returns the logical bitwise XOR of its arguments. If the number of bits needed to represent the two arguments are different, this operation will act as if the shortest argument is padded with zeros in the most significant bits.

Example:

Input	
RPN mode	Algebraic mode
`0x91 0x9 XOR`	`XOR(0x91,0x9)`

How to calculate 0x91 XOR 0x09.

Input/Output:

Level 2 / Argument 1	Level 1 / Argument 2		Level 1 / Result
s	t	$\rightarrow$	s XOR t
$'symb'_1$	$'symb'_2$	$\rightarrow$	$'\text{XOR}(symb_1, symb_2)'$

Argument range:

Argument	Range
s	Non-negative integer
t	Non-negative integer

See also:

AND, **OR**

ZETA

Friendly name : Riemann Zeta function
Type : Function
Number of parameters : 1

Description:

Returns the Riemann Zeta function of its argument x, $\zeta(x)$, defined as

$$\zeta(x) = \sum_{n=1}^{\infty} n^{-x} = \frac{1}{1^x} + \frac{1}{2^x} + \frac{1}{3^x} + \ldots \ , x > 1$$

This can, by analytic continuation, be extended to all real numbers $x \neq 1$. For $x = 1$, the Riemann Zeta function is infinite, but the limit

$$\lim_{\epsilon \to 0} \frac{\zeta(1+\epsilon) + \zeta(1-\epsilon)}{2}$$

approaches the Euler constant $\gamma = 0.5772\ldots$.

Example:

Input	
RPN mode	Algebraic mode
`2 ZETA`	`ZETA(2)`

Calculates $\zeta(2)$.

Input/Output:

Level 1 / Argument 1		Level 1 / Result
x	$\rightarrow$	$\zeta(x)$
$'symb'$	$\rightarrow$	$'\text{ZETA}(symb)'$

Argument range:

Argument	Range
x	Real number, not equal to 1.

See also:

EULER