Multiprecision Calculator User Manual v1.2

# Index

[0. Index 1](#_Toc63983432)

[1. Introduction 5](#_Toc63983433)

[1.1. Visual aspect of the screen 6](#_Toc63983434)

[2. Quick start 7](#_Toc63983435)

[2.1. Requirements 7](#_Toc63983436)

[2.2. Quick start 7](#_Toc63983437)

[2.3. Examples 8](#_Toc63983438)

[3. User interface 9](#_Toc63983439)

[3.1. Main screen 9](#_Toc63983440)

[3.2. Application menu 11](#_Toc63983441)

[3.2.1. File menu 11](#_Toc63983442)

[3.2.2. View menu 11](#_Toc63983443)

[3.2.3. Tools menu 12](#_Toc63983444)

[3.2.4. Help menu 13](#_Toc63983445)

[3.3. Configuration 14](#_Toc63983446)

[3.3.1. Calculation engine 14](#_Toc63983447)

[3.3.2. User interface 15](#_Toc63983448)

[3.3.3. Application language configuration parameters 16](#_Toc63983449)

[3.3.4. Application view configuration parameters 17](#_Toc63983450)

[3.4. Autocomplete Window 18](#_Toc63983451)

[3.5. About … 19](#_Toc63983452)

[3.6. License 20](#_Toc63983453)

[4. List of Operations 21](#_Toc63983454)

[4.1. Reserved terms 21](#_Toc63983455)

[4.1.1. print 21](#_Toc63983456)

[4.1.2. Constants 21](#_Toc63983457)

[1.1.1.1. e 21](#_Toc63983458)

[1.1.1.2. pi 21](#_Toc63983459)

[1.1.1.3. ln2 21](#_Toc63983460)

[1.1.2. User variable commands 22](#_Toc63983461)

[1.1.2.1. Assigning variables 22](#_Toc63983462)

[1.1.2.2. clearvars 22](#_Toc63983463)

[1.1.2.3. erasevar 22](#_Toc63983464)

[1.1.3. User function commands and functions 23](#_Toc63983465)

[1.1.3.1. Assigning functions 23](#_Toc63983466)

[1.1.3.2. Function composition 24](#_Toc63983467)

[1.1.3.3. val 24](#_Toc63983468)

[1.1.3.4. subst 25](#_Toc63983469)

[1.1.3.5. derivative 25](#_Toc63983470)

[1.1.3.6. simplify 25](#_Toc63983471)

[1.1.3.7. eraseafunc 25](#_Toc63983472)

[1.1.3.8. clearfuncs 26](#_Toc63983473)

[1.1.4. Operators 27](#_Toc63983474)

[1.1.4.1. Sum operator (+) 27](#_Toc63983475)

[1.1.4.2. Subtraction operator (-) 27](#_Toc63983476)

[1.1.4.3. Multiplying Operator (\*) 27](#_Toc63983477)

[1.1.4.4. Division Operator (/) 27](#_Toc63983478)

[1.1.4.5. Power Operator (^) 27](#_Toc63983479)

[1.1.4.6. Parenthesis ( ( ) ) 28](#_Toc63983480)

[1.1.4.7. Operator precedence 28](#_Toc63983481)

[1.1.5. Other commands 29](#_Toc63983482)

[1.1.5.1. exit 29](#_Toc63983483)

[1.1.6. Predefined mathematical functions 30](#_Toc63983484)

[1.1.6.1. abs 30](#_Toc63983485)

[1.1.6.2. absolute 30](#_Toc63983486)

[1.1.6.3. acos 30](#_Toc63983487)

[1.1.6.4. acosh 30](#_Toc63983488)

[1.1.6.5. add 30](#_Toc63983489)

[1.1.6.6. arccos 31](#_Toc63983490)

[1.1.6.7. arccosh 31](#_Toc63983491)

[1.1.6.8. arcsin 31](#_Toc63983492)

[1.1.6.9. arcsinh 31](#_Toc63983493)

[1.1.6.10. arctan 31](#_Toc63983494)

[1.1.6.11. arctanh 32](#_Toc63983495)

[1.1.6.12. asin 32](#_Toc63983496)

[1.1.6.13. asinh 32](#_Toc63983497)

[1.1.6.14. atan 32](#_Toc63983498)

[1.1.6.15. atanh 32](#_Toc63983499)

[1.1.6.16. ceil 33](#_Toc63983500)

[1.1.6.17. ceiling 33](#_Toc63983501)

[1.1.6.18. cos 33](#_Toc63983502)

[1.1.6.19. cosh 33](#_Toc63983503)

[1.1.6.20. divide 33](#_Toc63983504)

[1.1.6.21. down 34](#_Toc63983505)

[1.1.6.22. exp 34](#_Toc63983506)

[1.1.6.23. floor 34](#_Toc63983507)

[1.1.6.24. gcd 34](#_Toc63983508)

[1.1.6.25. ln 35](#_Toc63983509)

[1.1.6.26. log 35](#_Toc63983510)

[1.1.6.27. max 35](#_Toc63983511)

[1.1.6.28. min 35](#_Toc63983512)

[1.1.6.29. multiply 35](#_Toc63983513)

[1.1.6.30. power 36](#_Toc63983514)

[1.1.6.31. quotient 36](#_Toc63983515)

[1.1.6.32. root 36](#_Toc63983516)

[1.1.6.33. round 36](#_Toc63983517)

[1.1.6.34. sgn 36](#_Toc63983518)

[1.1.6.35. sin 36](#_Toc63983519)

[1.1.6.36. sinh 37](#_Toc63983520)

[1.1.6.37. sqrt 37](#_Toc63983521)

[1.1.6.38. subtract 37](#_Toc63983522)

[1.1.6.39. sum 37](#_Toc63983523)

[1.1.6.40. tan 37](#_Toc63983524)

[1.1.6.41. tanh 38](#_Toc63983525)

[1.1.6.42. up 38](#_Toc63983526)

[1.1.6.43. value 38](#_Toc63983527)

[1.2. Help command 39](#_Toc63983528)

[1.3. Summary table of reserved terms 40](#_Toc63983529)

[2. Invoking from the command interface 42](#_Toc63983530)

# Introduction

The multiprecision calculator is an application that allows calculations to be performed with adjustable precision.

In reality, it is not particularly useful to provide accuracy beyond 15 decimal digits in standard calculations. However, programming this type of calculator is interesting and challenging.

A few years ago, I programmed the first version of the multiprecision calculator. Looking back, I thought it a shame for it to stay the way it was, with architecture that could be greatly improved.

I recovered from my distant past the challenge of the algebraic derivation and decided to program a new version of the application.

The previous functionality of the application still works, but new user options have been added.

The functionality offered by the previous version is:

* Calculation engine
	+ User functions in one or multiple variables.
	+ User variables.
	+ Numeric calculations with variable precision.

Improvements introduced by version v1.1 are:

* New functionalities:
	+ Calculation of function derived algebraically.
	+ Greatest common divisor
	+ Calculation using multiple threads in parallel.
	+ Version that can be invoked from the command interface.
* Improved user interface:
	+ Multiple languages.
	+ Adjustable zoom level.
	+ AutoComplete.
	+ Improved support.
	+ Text formatting
	+ Undo/redo within input text pane.
* Code Enhancements:
	+ Best-in-class architecture
	+ Possibility of easy and structured growth.

The improvements of version v1.2 are the following:

* New functionalities:
	+ Search for a new version in the server.

## Visual aspect of the screen

Visual aspect of the screen:



The screen has the following components:

* Main menu at the top.
* On the bottom:
	+ Precision. Allows precision for calculations to be set.
	+ Cancel button. Allows pending calculations to be cancelled.
* Text Panes:
	+ Panel **1**. Input Panel. Commands to be executed must be written here.
	+ Panel **2**. Exit Panel. Results of operations are written by the application in this panel.
	+ Panel **3**. Functions panel and user variables. In this panel, functions and variables that have been defined by the user are shown.

# Quick start

## Requirements

To execute the application is necessary to have the java version 8 or later environment installed (Java Runtime Environment).

With this, the file .jar in folder ./binary/ can be opened.

## Quick start

To begin the application, complete the following steps:

* Opens the multiprecision calculator by double-clicking on: el .jar of the application: which is located in .../\_binary/multicalcu-gui-main-v1.2-SNAPSHOT-all.jar
* Choose the precision of the calculations. (text component: “**Precisión:**
* Execute a command (typing in the lower right text panel):
	+ Get help. P. ej. help + return
	+ Define a function. Ex. f(x) = 2 \* x ^ 2 + return
	+ Write a numerical operation. Ex. print( 2 \* 4 ^ 2 ) + return
	+ Calculate a derived function. Ex. g(x) = subst( derivative( x, 1, f(x) ) ) + return

## Examples

* help

Displays application support in the chosen language.

* help functions

Displays keyword functions help.

* print( 2 \* pi )

Displays the result of the operation 2 \* pi with the selected precision.

* a = 1.1

Assigns the value 1.1 to variable a, which could be used in numeric calculations of expressions, but not with the definition of functions.

* fun(A, f, alpha, t ) = A \* cos( 2 \* pi \* f \* t + alpha )

Defines the function fun as a function of four variables.

* g(f, t ) = fun( 2.7, f, pi / 2, t )

Defines function g as a funtion of fun, but without replacing the expression.

That is to say, that whenever a value of function g is calculated latest fun definition will be taken, and the defined composition of variables will be carried out.

This is to say, that if fun changes after defining g, g is also affected.

* h(f, f2, t) = subst( fun( 2.7 \* sin( 2 \* pi \* f2 \* t + pi / 3), f, pi/2, t ) )

Defines function h as a function of fun, carrying out functional composition.

That means the relationship between h and fun is only at the moment of defining the h function.

If fun changes after the definition of h, h will remain unchanged, because the expression had already been substituted (thanks to the function subst ).

* print( h( 10, 0.26, 3.7 ) )

Displays the result of function h for its input values.

* print( 10 \* log( 10, ( 25.2 \* h( 10, 0.26, 3.7 ) ) ^ 2 ) )

Displays the result of the operation. For more details, consult item: ***1.1.6.26-log***

* print( 10 \* arccos( tanh( 3 \* atan( 500 / pi \* 3.6 ) ) ) + sqrt( exp( 5.3 ) ^ ( 30 \* sin( 1 / 30.7 ) ) ) / ( 7.3 + sin( 27 + pi ) + sin( 27 ) ) )

Calculates the numerical result of the complex expression.

* i(f, f2, t) = subst( derivative( t, 2, h(f, f2, t ) ) )

Calculates the second order partial derivative of h based on t and assigns it to function i.

# User interface

In this chapter you will learn how to use the main window panels.

## Main screen

When starting the graphic application, the following screen will appear:



The screen has the following components:

* Main menu at the top.
* On the bottom:
	+ Precision. Allows precision for calculations to be set.
	+ Cancel button. Allows pending calculations to be cancelled.
* Text Panes:
	+ Panel **1**. Input Panel. Commands to be executed must be written here.
	+ Panel **2**. Exit Panel. Results of operations are written by the application in this panel.
	+ Panel **3**. Functions panel and user variables. In this panel, functions and variables that have been defined by the user are shown.

Example of window in normal use:



## Application menu

The menu is the option bar that is located at the top of the window.

Next we will see the different options available.

### File menu

The File Menu looks like this:



This menu has one option:

* Exit. Allows you to exit the application.

### View menu

The view menu looks like this:



This menu has only one option:

* Zoom. Allows the size of the visual part of the application to be chosen with a percentage indicating the size of the components in comparison with the normal display of the windows (100%).

### Tools menu

The tools menu looks like this:



This menu has two options:

* Language. The languages available appear when this submenu is opened. It is used to change the language of the application.

The following default languages are available:

* "EN". English
* "ES". Spanish
* "CAT". Catalan
* Configuration. When this menu item is selected a form appears that allows the general configuration parameters of the application to be modified. See item: ***3.3-Configuration***

### Help menu

The help menu looks like this:



This menu has three options:

* **Help**. Opens the help file.
* **Search for a new version**. This option searches the server to see if there is a new version of the application.
* **What’s new**. This option displays the document the explains what is new in this version.
* **License**. Shows the license that was accepted the first time the application was launched. See section: ***3.6-License***
* **About**. When this menu item is selected, a form appears with the application data, acknowledgements, and contact info. See section: ***3.5-About …***

## Configuration

This form allows the general configuration parameters of the application to be modified.

The form has several tabs that we will detail below.

### Calculation engine

The tab looks like this:



The following parameters can be configured:

* Precision for calculations. This is the precision in which calculations made will be displayed. Internally, calculations are performed with a slightly higher accuracy to try to prevent inaccuracies in the result shown (something that is not always achieved).
* Maximum number of threads used for calculations. Indicates the maximum number of threads to be used for calculations.

It is recommended to use a number equal to the number of simultaneous threads supported by the processor minus one.

For example, if you have a i7 with 4 kernels and 2 threads per kernel the recommend number would be 7 (unless you want to reserve the processing capacity for other programs running simultaneously).

### User interface

The tab looks like this:



The following parameters can be configured:

* Display auto completion of commands.

When this option is checked a window with the autocomplete options appears as you type in the input text pane.

When this window is displayed, and there are options, cursors can be moved up and down in order to choose another possible option or continue typing (the search will close). If you press return while this window has possible options, the option selected is typed in the text panel and that window closes.

It is also possible to select the Autocomplete option desired using the mouse pointer.

* Display current parameter support.

When this option is checked, as you type within the input text pane, a windows appears with the current parameter being filled in.

It is useful to know, for example, which parameter we are entering in a function.

### Application language configuration parameters

The tab looks like this:



The following parameters can be configured:

* Language. This is the language of the text written by the application.

The languages that can be chosen are:

* EN. English
* ES. Spanish
* CAT. Catalan
* Language locale. This is the Java locale the application will use for this language.

The application uses it to convert numbers to formatted numerical strings.

* Web language. Indicates the language that is going to be used with the server in the new version inquiries.
* Additional language. This is the new additional language that appears at the time of being able to change language.

For a new language you have to select the java locale to be used.

If you want to enter a language in which none of the application text exists you can add your language by translating files within the directory that is created when you press the Accept button.

The files to be translated are copied to the directory indicated in : "Additional language directory"

The format of these text files is that of java properties .

For anyone unfamiliar with this format, know that a properties file has a title and after this a variable number of labels with its value similar to the following:

#TITLE

# xxxxxxxxxx

LABEL1=text 1

LABEL 2=text 2

...

Labels should remain unmodified and texts should be modified depending on the translation in the chosen language.

In addition, there are also some files in RTF format that you will have to translate with an RTF editor (a typical editor for this format is Office Word).

If you create the translation for an additional language different from those available in the application, you can send it to me at (frojasg1@hotmail.com), if you would like, and I will include it in upcoming versions of the application.

### Application view configuration parameters

The tab looks like this:



* Window size. This parameter allows you to control the appearance of the windows that are displayed in the application with the possibility of chosing small-sized, regular-sized, or large-sized.

## Autocomplete Window

The autocomplete window is a help window that appears as you are typing and it is always set to display.

The idea was drawn from the functionality many integrated development environments have in which while you are typing, options appear that refresh as the syntax of what you want to write, as well as which are the arguments that are taken in case the option is a function.

While that window with options (top panel) is being displayed, you can move between these options with the cursors or by clicking on the desired option with the mouse.

The way to select an option is by clicking with the mouse pointer or by pressing return. Then the complete option will be typed in the input text panel, and this window will disappear.

The window also disappears when the focus comes from the window itself or from the main window, and also when the main window is minimized. .

This is an example of the autocomplete screen:



In this case, both configuration parameters are active.

The autocomplete window has two panels

* Top panel: Indicates the autocomplete options.
* Bottom panel, indicates the current parameter in which we are typing.

## About …

The About option …,displays a window with a summary of what's new in this version. Acknowledgements are also included..

It looks like this:



## License

The license option within the help menu allows the license that was accepted the first time the application was launched to be displayed.

It looks like this:



# List of Operations

The multiprecisión calculator allows calculations with configurable accuracy to be performed, as well as performing some operations with functions and variables.

All of this is done by entering instructions via the keyboard.

These instructions are made up of different reserved terms and parameters.

##  Reserved terms

Reserved terms are words or signals that have a specific meaning for the expression analyzer.

Many of those reserved terms are commands and functions although there are other types.

In this section we will take a look at all of the these terms.

### print

The print command allows the numerical result of the expression that it has as an argument to be displayed.

Format: **print**( **expression** )

Example: print( 2 + 2 )

Result: 4

### Constants

Constants are elements like variables, but they have a fixed value automatically calculated by the application.

### e

Euler number.

Result: **2.718281828459045235360287471352662**…

### pi

Pi constant.

Result: **3.141592653589793238462643383279502**…

### ln2

Napierian logarithm of 2.

Result: **0.6931471805599453094172321214581**…

### User variable commands

This type of command allows the definition/deletion of variables with a specific numerical value to be managed.

### Assigning variables

Allows a numerical value to be assigned to a variable.

Format: **VariableName** = numeric expression

Example: aa = 2 + 2

Result: The value of 4 is assigned to variable aa.

### clearvars

Deletes all user variables.

Format: **clearvars**

Example: clearvars

Result: deletes all variables.

### erasevar

Deletes the indicated user variable.

Format: **erasevar VariableName**

Example: **erasevar aa**

Result: eliminates the variable aa.

### User function commands and functions

This type of command allow the definition/deletion of variables with a specific numerical value to be managed. Some are not really commands but functions that must form part of an expression. This is because by themselves they do not form a complete instruction.

### Assigning functions

Allows a function to be saved in the list of user functions.

Format: **FunctionName( variable1, …, variableN )** = expression( variable1, …, variableN )

Example: **fun**( x, y, z ) = 2 \* x ^ 2 \* y ^ 2 \* z ^ 2

Result: The expression is assigned according to three variables to the function name **fun**.

Notes: The expression of the function allows relationships between functions to be defined. In other words we could define the function as: fun2(x, y) = fun( x, y, x\*y )

 In this case, if we want to calculate the value of fun2( 1, 1 ) the calculation engine will take the function fun (x, y, z), and it will create the operations dynamically.

 The expression fun can be changed after the definition fun2, but we have not explicitly placed any expression in the definition of fun2. Therefore it would be resolved in a satisfactory manner at the time of execution, getting the latest definition of fun(x, y, z).

 If we want fun2 to have a specific expression that isn't dependent on the definition of other functions, we will have to use the command (function) subst which causes the expression that it takes as a parameter at the time it is invoked to be substituted.

 For example, if we want to define: fun2(x, y) = subst( fun( x, y , x\*y ) ), we will directly obtain a specific expression for fun2 ,and therefore after the defintion of funchanges, we would not observe any change in the definition of fun2.

 In that case, the result would directly be:

 fun2(x, y) = 2 \* x ^ 2 \* y ^ 2 \* ( x \* y ) ^ 2

 and the expression of fun2 would no longer be related to the expression of fun.

 Something similar occurs with the derivative command (function).

 This command allows the derivative function of the expression it takes as an argument to be obtained.

 Therefore, we can define a function taking other functions, or even their derived functions as input, as the expression of a differential equation, although in this case, the left part of the expression would be a simple function (it cannot be derived).

 (The application does not solve differential equations!!).

 For example, in order to illustrate what I'm talking about , we could define:

 fun3(x, y, z) = derivative( x, 2, fun( x, y, z ) ) – f(x, y, z) \* derivative( x, 1, fun( 2\*x, x\*y, x\*z ) )

 The mathematical expression of this function would be:

 $fun3\left(x, y, z\right)= \frac{δ^{2}fun\left(x,y,z\right)}{δ^{2}x}-fun\left(x,y,z\right)\* \frac{δfun\left(2\*x, x\*y,x\*z\right)}{δx}$

Nevertheless, if what we want is to specify the expression of fun2 at the time of the definition, we could put:

fun3(x, y, z)= subst( derivative( x, 2, fun(x, y, z ) ) – subst( f(x, y, z ) ) \* subst( derivative( x, 1, fun( 2\*x, x\*y, x\*z ) ) )

In that case, the result obtained for the definition of the function would be:

fun3( x, y, z ) = 4 \* y ^ 2 \* z ^ 2 – 2 \* x ^ 2 \* y ^ 2 \* z ^ 2 \* d fun( 2\*x, x\*y, x\*z ) / dx =

4 \* y ^ 2 \* z ^ 2 – 2 \* x ^ 2 \* y ^ 2 \* z ^ 2 \* d ( 2 \* ( 2 \* x ) ^ 2 \* ( x \* y ) ^ 2 \* ( x \* z ) ^ 2 ) / dx =

4 \* y ^ 2 \* z ^ 2 – 2 \* x ^ 2 \* y ^ 2 \* z ^ 2 \* 48 \* x ^ 5 \* y ^ 2 \* z ^ 2

In that case, the expression for fun3 would be obtained from the current definition of fun,but it would no longer be contingent on fun.

### Function composition

This possibility allows an expression to be calculated by substituting the variables of the function for expressions.

Format: **nameFunction( expression1, …, expressionN )**

Example: **fun( 2\*x, x\*y, x\*z )**

Result: if fun( x, y, z ) = 2 \* x ^ 2 \* y ^ 2 \* z ^ 2

 Then: fun( 2\*x, x\*y, x\*z ) = 2 \* ( 2 \* x ) ^ 2 \* ( x \* y ) ^ 2 \* ( x \* z ) ^ 2

### val

This command (which in reality is a function), allows an expression that gives a number as the result for that number to be substituted.

This is useful because in case we are defining a function, it could be useful to substitute a complex numerical expression for its result (so that fewer operations are carried out every time a value of the function is calculated).

The downside is that this operation is linked to the precision set at the moment of using it. So, if we then change the accuracy, the calculated value is not recalculated.

Format: **val( expression )**

Example: f( x ) = **val( tan( pi / 4 ) / 2 )** \* x ^ 2

Result: f( x ) = 0.5 \* x ^ 2

### subst

This command (which in reality is a function), allows a functional composition or a derivative calculation to be substituted for its specific expressions.

If this function is not used, the result will be expressed based on the functions referenced in the composition or the derivative calculation.

( For more detail, see item Notes, in: ***1.1.3.1-Assigning functions***)

Format: **subst( function composition) ó**

 **subst( derived function calculation )**

Example: f( x ) = **subst( g( 2 \* x ) )**

Result: if g(x) = 2 \* x then f( x ) = 2 \* 2 \*

### derivative

This command (which in reality is a function) allows the nth derivative to be calculated based on one of the variables of a function that can be defined according to multiple variables.

The possibility of calculating crossed partial derivatives has not be implemented as of yet .

If this function is not used as an argument of function subst, then the result will be expressed based on the functions referenced in the derivative calculation.

( For more detail, see item Notes, in: ***1.1.3.1-Assigning functions***)

Format: **derivative( variable, order, expression )**

Example: g( x ) = **derivative( x, 1, f(x) )**

Result: if f( x )= x ^ 2 then g( x ) = 2 \*

### simplify

This command (which in reality is a function) allows you to simplify an expression. It doesn't substitute functions even though they are an argument of a subst command.

It was created for the purpose of debugging, since this simplification function is invoked every time a derivative is calculated with the derived function.

Format: **simplify( expression )**

Example: f( x ) = **simplify( x ^ 2 / ( 2 \* x ) - 3 + 1 / 2 )**

Result: f( x )= x / 2 - 5 / 2

### eraseafunc

This command allows you to delete a user function.

Format: **erasefunc( nameFunction )**

Example: erasefunc( f )

Result: deletes the user function f

### clearfuncs

This command allows you to delete all user functions.

Format: **clearfuncs**

Example: clearfuncs

Result: deletes all user functions

### Operators

The operators are strings, usually formed by only one character, that allow you to concatenate expressions.

### Sum operator (+)

Adds two numbers.

Format: **summand1 + summand2**

Example: **1.01 + 3.45**

Result: 4.46

### Subtraction operator (-)

Subtracts two numbers.

Format: **minuend + subtrahend**

Example: **7.25 - 3.743**

Result: 3.507

### Multiplying Operator (\*)

Multiplies two numbers.

Format: **factor1 \* factor2**

Example: **7.25 \* 3**

Result: 21.75

### Division Operator (/)

Divides two numbers.

Format: **dividend / divisor**

Example: **16 / 4**

Result: 4

### Power Operator (^)

Raises one number to another.

Format: **base ^ exponent**

Example: **16 / 4**

Result: 4

### Parenthesis ( ( ) )

Gives priority to the operation between the parenthesis.

Format: **( expression )**

Example: **( 3 + 4 ) \* ( 2 - 7 )**

Result: 7 \* ( -5 )

### Operator precedence

Operators will be evaluated in the following order:

1. Expression between parentheses ( ... )
2. The power operator ^
3. Multiplication and division: \*, /
4. Addition and subtraction: +, -

### Other commands

Other commands, have to do with implemented commands that have nothing to do with the calculation engine.

### exit

Exit application.

Format: **exit**

Example: **exit**

Result: Exit the application.

### Predefined mathematical functions

They are mathematical functions that exist without the user having to define them.

They produce a result according to the input parameters.

### abs

Calculates the absolute value of the argument.

Format: **abs( argument )**

Example: **abs( -2.35 )**

Result: 2.35

### absolute

Calculates the absolute value of the argument.

Format: **absolute( argument )**

Example: **absolute( -2.35 )**

Result: 2.35

### acos

Calculates the arc cosine of the argument, giving the result in radians.

Format: **acos( argument )**

Example: **acos( 1 )**

Result: 0

### acosh

Calculates the hyperbolic cosine of the argument.

Format: **acosh( argument )**

Example: **acosh( 1 )**

Result: 0

### add

Calculates the sum of the two summands.

Format: **add( summand 1, summand 2 )**

Example: **add( 2, 3 )**

Result: 5

### arccos

Calculates the arc cosine of the argument, giving the result in radians.

Format: **arccos( argument)**

Example: **arccos( 1 )**

Result: 0

### arccosh

Calculates the hyperbolic cosine of the argument.

Format: **arccosh( argument )**

Example: **arccosh( 1 )**

Result: 0

### arcsin

Calculates the arcsine of the argument, giving the result in radians.

Format: **arcsin( argument)**

Example: **arcsin( 0 )**

Result: 0

### arcsinh

(CHECK)Calculates the hyperbolic sine arc of the argument.

Format: **arcsinh( argument )**

Example: **arcsinh( 0 )**

Result: 0

### arctan

Calculates the arctangent of the argument, giving the result in radians.

Format: **arctan( argument )**

Example: **arctan( 0 )**

Result: 0

### arctanh

Calculates the hyperbolic arctangent of the argument.

Format: **arctanh( argument )**

Example: **arctanh( argument )**

Result: 0

### asin

Calculates the arcsine of the argument, giving the result in radians.

Format: **asin( argument )**

Example: **asin( 0 )**

Result: 0

### asinh

(CHECK)Calculates the hyperbolic sine arc of the argument.

Format: **asinh( argument )**

Example: **asinh( 0 )**

Result: 0

### atan

Calculates the arctangent of the argument, giving the result in radians.

Format: **atan( argument )**

Example: **atan( 0 )**

Result: 0

### atanh

Calculates the hyperbolic arctangent of the argument.

Format: **arctan( argument )**

Example: **atanh( 0 )**

Result: 0

### ceil

Calculates the following number with n decimals greater or equal to the argument (in the direction of infinity).

Format: **ceil( n, argument )**

Example: **ceil( 0, 1.01 )**

Result: 2

Example: **ceil( 0, -1.01 )**

Result: -1

### ceiling

Calculates the following number with n decimals greater or equal to the argument (in the direction away from infinity).

Format: **ceiling( n, argument )**

Example: **ceiling( 0, 1.01**

Result: 2

Example: **ceiling( 0, -1.01 )**

Result: -1

### cos

Calculates the cosine of the argument in radians.

Format: **cos( argument )**

Example: **cos( 0 )**

Result: 1

### cosh

Calculates the hyperbolic cosine of the argument.

Format: **cos( argument )**

Example: **cosh( 0 )**

Result: 1

### divide

Calculates the division of the arguments.

Format: **divide( dividend, divisor )**

Example: **divide( 16, 4 )**

Result: 4

### down

Calculates the following number with n decimals less than or equal to the absolute value of the argument. (direction to 0).

Format: **down( n, argument )**

Example: **down( 0, 1.01 )**

Result: 1

Example: **down( 0, -1.01 )**

Result: -1

### exp

Exponential. Calculates the power of the number e raised to the argument.CHECK THIS!!!

Format: **exp( argument )**

Example: **exp( 0 )**

Result: 1

### floor

Calculates the following number with n decimals less or equal to the argument. (direction to less infinity)

Format: **floor( n, argument )**

Example: **floor( 0, 1.01 )**

Result: 1

Example: **floor( 0, -1.01 )**

Result: -2

### gcd

Calculates the maximum common divisor of the arguments.

Format: **gcd( argument1, ..., argumentN )**

Example: **gcd( 26, 39 )**

Result: 13

### ln

Napierian logarithm.

Format: **ln( argument )**

Example: **ln( 1 )**

Result: 0

### log

Calculates the logarithm of the argument in the base indicated.

Format: **log( base, argument )**

Example: **log( 10, 100 )**

Result: 2

### max

Calculates the maximum of the arguments.

Format: **max( argument1, ..., argumentN )**

Example: **max( 7, 16, 5 )**

Result: 16

### min

Calculates the minimum of the arguments.

Format: **min( argument1, ..., argumentN )**

Example: **min( 1, 7, -1 )**

Result: -1

### multiply

Calculates the product of the factors.

Format: **multiply( factor1, factor2 )**

Example: **multiply( 1, 7 )**

Result: 7

### power

Power. Calculates the power: base raised to the exponent.

Format: **power( base, exponent )**

Example: **power( 3, 4 )**

Result: 81

### quotient

Division. Calculates the division: dividend divided by divisor.

Format: **power( base, exponent )**

Example: **quotient( 16, 4 )**

Result: 4

### root

Calculates the nth root of the argument.

Format: **root( n, argument )**

Example: **root( 4, 81 )**

Result: 3

### round

Calculates the number closest to the argument with n decimals.

Format: **round( n, argument )**

Example: **round( 2, 0.513 )**

Result: 0.51

### sgn

Calculates the sign of the argument (returns -1 if it is negative, 0 if it is 0, and 1 if it is positive).

Format: **sgn( argument )**

Example: **sgn( -0.1 )**

Result: -1

### sin

Calculates the sine of the argument in radians.

Format: **sin( argument )**

Example: **sin( pi / 2 )**

Result: 1

### sinh

Calculates the hyperbolic sine of the argument.

Format: **sinh( argument )**

Example: **sinh( 0 )**

Result: 0

### sqrt

Calculates the square root of the argument.

Format: **sqrt( argument )**

Example: **sqrt( 9 )**

Result: 3

### subtract

Subtraction. Calculates the subtraction: minuend minus subtrahend.

Format: **subtract( minuend, subtrahend )**

Example: **subtract( 3, -2 )**

Result: 5

### sum

Calculates the sum of the two summands.

Format: **sum( summand1, summand2 )**

Example: **sum( 2, 3 )**

Result: 5

### tan

Calculates the tangent of the argument in radianes.

Format: **tan( argument )**

Example: **tan( pi / 4 )**

Result: 1

### tanh

Calculates the hyperbolic tangent of the argument.

Format: **tanh( argument )**

Example: **tanh( 0 )**

Result: 0

### up

Calculates the following number with n decimals with an absolute value greater or equal to the argument (direction less infinite if the argument is negative or more infinite if the argument is positive).

Format: **up( n, argument )**

Example: **up( 1, 1.01 )**

Result: 1.1

### value

Calculates the numerical value of the argument.

Format: **value( argument )**

Example: **value(** sqrt(2)/ 2 ^ (1/2) **)**

Result: 1

##  Help command

The help<t1/ command, displays support from the application. Its´ summary is found in the table detailed in the following section. Reference-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Format: **help** name

Example: help functions

Result: Display help for functions, along with a summary of each one.

Support is organized in sections by hierarchy.

If the help command doesn´t take arguments, global support for the application is shown, which is a summary of every one of the sections that has support.

If the help command takes a name as an argument , a description of the tag name is shown along with a summary of all the content derived from it, all derived content, in turn, with a **name** in blue or bold, that allows you to get more detailed support from each one of those tags, simply making: help name

When using the help command, new derived content for those can be navigated normally appear.

If you move the mouse pointer in the exit panel over the derived names (**name**), you will see that when the pointer is over a **name** “child” , the mouse pointer displays the icon of the type "follow link" icon, and if at that moment you press the left button of the mouse (left click), you get the support of that tag (that tag´s support) directly, as if you typed: help **name** + return<t11/

This feature makes it much easier to navigate otherwise very time consuming help. .

Possible names that are recognized by the help<t1/ command are those in the first column of the table of the following section.

##  Summary table of reserved terms

Next we will summarize the meaning of the reserved terms in a table.

|  |  |  |
| --- | --- | --- |
| **Reserved Term** | **Format** | **Description** |
| **help** |  | Displays application support (summarized in this table).  |
| **help** | help name | Displays specific support of the tag “name”.  |
|  **Commands** |  |  |
|  **Print** |  print( expression ) |  Print |
|  **user\_variables** |   |   |
|  **Assignvariable** |  variable = expression |  Assign variable |
|  **Clearvars** |  clearvars |  Cleans variables |
|  **Erasevar** |  erasevar( variable ) |  Eliminate variable |
|  **user\_functions** |   |   |
|  **Assignfunction** |  f( x, y ) = expression |  Define function |
|  **Composition** |  f( x ) = g( h( x ), i( x ) ) |  Composition |
|  **Val** |  val( expression ) |  Value |
|  **Subst** |  subst( f( x ) ) |  Substitute |
|  **Derivative** |  derivative( variable, n, function ) |  Derived function |
|  **Simplify** |  simplify( expression ) |  Simplify expression |
|  **Erasefunc** |  erasefunc( f ) |  Erase functions |
|  **Clearfuncs** |  clearfuncs |  Clean functions |
|  **Operators** |   |   |
|  **Addition** |  summand1 + summand2 |  Add |
|  **Subtraction** |  minuend - subtrahend |  Subtract |
|  **Multiplication** |  factor1 \* factor2 |  Multiplication |
|  **Division** |  dividend / divisor |  Division |
|  **circumflex** |  base ^ exponent |  Power |
|  **parenthesis** |  ( expression ) |  Parenthesis |
|  **Precedence** |   |  Operator precedence |
|  **constants** |   |   |
|  **e** |  e |  Euler number. |
|  **pi** |  pi |  Pi constant |
|  **ln2** |  ln2 |  Napierian logarithm of 2 |
|  **other\_commands** |   |   |
|  **exit** |  exit |  Exit |
|  **functions** |   |   |
|  **abs** |  abs( argument ) |  Absolute value |
|  **absolute** |  absolute( argumento ) |  Absolute value |
|  **acos** |  acos( argument ) |  Arc cosine.  |
|  **acosh** |  acosh( argument ) |  Hyperbolic arc cosine |
|  **add** |  add( summand1, summand2 ) |  Add |
|  **arccos** |  arccos( argument ) |  Arc cosine.  |
|  **arccosh** |  arccosh( argument ) |  Hyperbolic arc cosine |
|  **arcsin** |  arcsin( argument ) |  Arcsine |
|  **arcsinh** |  arcsinh( argument ) |  Hyperbolic arc cosine |
|  **arctan** |  arctan( argument ) |  Arc tangent |
|  **arctanh** |  arctanh( argument ) |  Hyperbolic arc tangent |
|  **asin** |  asin( argument ) |  Arcsine |
|  **asinh** |  asinh( argument ) |  Hyperbolic arc cosine |
|  **atan** |  atan( argument ) |  Arc tangent |
|  **atanh** |  atanh( argument ) |  Hyperbolic arc tangent |
|  **ceil** |  ceil( n, argument ) |  Ceiling |
|  **ceiling** |  ceiling( n, argument ) |  Ceiling |
|  **cos** |  cos( argument ) |  Cosine |
|  **cosh** |  cosh( argument ) |  Hyperbolic cosine |
|  **divide** |  divide( dividend, divisor ) |  Division |
|  **down** |  down( n, argument ) |  down |
|  **exp** |  exp( argument ) |  Exponential. |
|  **floor** |  floor( n, argument ) |  Floor |
|  **gcd** |  gcd( argument1, ..., argumentN ) |  Greatest common divisor (greatest common divisor) |
|  **ln** |  ln( argument ) |  Napierian logarithm |
|  **log** |  log( base, argument ) |  Logarithm |
|  **max** |  max( argument1, ..., argumentN ) |  Maximum |
|  **min** |  min( argument1, ..., argumentN ) |  Minimum |
|  **multiply** |  multiply( factor1, factor2 ) |  Multiplication |
|  **power** |  power( base, exponent ) |  Power |
|  **quotient** |  quotient(( dividend, divisor ) |  Division |
|  **root** |  root( n, argument ) |  Root |
|  **round** |  round( n, argument ) |  Round |
|  **sgn** |  sgn( argument ) |  Sign |
|  **sin** |  sin( argument ) |  Sine |
|  **sinh** |  sinh( argument ) |  Hyperbolic sine |
|  **sqrt** |  sqrt( argument ) |  Square root |
|  **subtract** |  subtract( minuend, subtrahend ) |  Subtract |
|  **sum** |  sum( summand1, summand2 ) |  Add |
|  **tan** |  tan( argument ) |  Tangent |
|  **tanh** |  tanh( argument ) |  Hyperbolic tangent |
|  **up** |  up( n, argument ) |  Up |
|  **value** |  value( argument ) |  Value |

# Invoking from the command interface

In this version, a minimum .jar binary has been created that can be invoked from the command interface although it doesn´t allow interactive execution.

The command interface application only executes a single command that is passed by parameter in the invocation.

This application accepts three entry parameters.

* -precision (-precision=value). Precision to be used
* -threads (-threads=value). Threads to be used in the execution
* -command (“-command=comand”). Command to be executed in the calculator.

Note that this parameter will normally have to go between double quotation marks, since it is normal that spaces are included, and without quotation marks, those spaces could not form part of the same parameter.

Example:

java -jar multicalcu-commandline-v1.2-SNAPSHOT-all.jar -precision=1000 -threads=7 "-command=print( pi )"

Result:

3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128475648233786783165271201909145648566923460348610454326648213393607260249141273724587006606315588174881520920962829254091715364367892590360011330530548820466521384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943702770539217176293176752384674818467669405132000568127145263560827785771342757789609173637178721468440901224953430146549585371050792279689258923542019956112129021960864034418159813629774771309960518707211349999998372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137838752886587533208381420617177669147303598253490428755468731159562863882353787593751957781857780532171226806613001927876611195909216420199