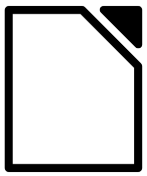

LibreOffice
The Document Foundation

Calc Guide

Appendix B
Description of Functions

Copyright

Contributors

Barbara Duprey
Jean Hollis Weber
Simon Brydon
John A Smith

Feedback

Please direct any comments or suggestions about this document to:
documentation@global.libreoffice.org

Acknowledgments

This appendix is based on Appendix B of the *OpenOffice.org 3.3 Calc Guide*. The contributors to that appendix are:

Magnus Adielsson
Iain Roberts
Richard Barnes
Jean Hollis Weber
Peter Kupfer

Publication date and software version

Published 24 November 2013. Based on LibreOffice 4.1.3.

Note for Mac users

Some keystrokes and menu items are different on a Mac from those used in Windows and Linux. The table below gives some common substitutions for the instructions in this appendix. For a more detailed list, see the application Help.

Windows or Linux	*Mac equivalent*	*Effect*
Tools > Options menu selection	**LibreOffice > Preferences**	Access setup options
Right-click	*Control+click*	Opens a context menu
Ctrl (Control)	⌘ (*Command*)	Used with other keys
F5	*Shift+⌘+F5*	Opens the Navigator
F11	*⌘+T*	Opens the Styles and Formatting window

Contents

Functions available in Calc

Calc provides all of the commonly used functions found in modern spreadsheet applications. Since many of Calc's functions require very specific and carefully calculated input arguments, the descriptions in this appendix should not be considered complete references for each function. Refer to the application Help or the LibreOffice wiki for details and examples of all functions. On the wiki, start with http://help.libreoffice.org/Calc/Functions_by_Category

Over 300 standard functions are available in Calc. More can be added through extensions to Calc (see Chapter 14). The following tables list Calc's functions organized into eleven categories.

Note	Functions whose names end with **_ADD** are provided for compatibility with Microsoft Excel functions. They return the same results as the corresponding functions in Excel (without the suffix), which though they may be correct, are not based on international standards. Calc automatically changes the function to _ADD for relevant functions in imported Excel spreadsheets.

Terminology: numbers and arguments

Some of the descriptions in this appendix define limitations on the number of values or arguments that can be passed to the function. Specifically, functions that refer to the following arguments may lead to confusion:

- **Number_1, number_2, ... number_30**
- Number 1 to 30
- a list of up to 30 numbers

There is a significant difference between a *list of number*s (or integers) and the *number of arguments* a function will accept. For, example the *SUM* function will only accept a maximum of 30 arguments. This limit does NOT mean that you can only sum 30 numbers, but that you can only pass 30 separate arguments to the function.

Arguments are values separated by commas, and can include ranges which often refer to multiple values. Therefore one argument can refer to several values, and a function that limits input to 30 arguments may in fact accept more then 30 separate numerical values.

This appendix attempts to clarify this situation by using the term **arguments**, rather than any of the other phrases.

In the LibreOffice Calc functions, parameters marked as "optional" can be left out only when no parameter follows. For example, in a function with four parameters, where the last two parameters are marked as "optional", you can leave out parameter 4 or parameters 3 and 4, but you cannot leave out parameter 3 alone.

Note	In the tables of functions in this Appendix, several bugs are listed; if you wish to check on the progress of fixing those bugs, you can visit http://bugs.libreoffice.org/ and enter the bug number.

Mathematical functions

Table 1: Mathematical functions

Syntax	***Description***
ABS(Number)	Returns the absolute value of the given **Number**.
ACOS(Number)	Returns the inverse cosine of the given **Number** in radians.
ACOSH(Number)	Returns the inverse hyperbolic cosine of the given **Number** in radians.
ACOT(Number)	Returns the inverse cotangent of the given **Number** in radians.
ACOTH(Number)	Returns the inverse hyperbolic cotangent of the given **Number** in radians.
ASIN(Number)	Returns the inverse sine of the given **Number** in radians.
ASINH(Number)	Returns the inverse hyperbolic sine of the given **Number** in radians.
ATAN(Number)	Returns the inverse tangent of the given **Number** in radians.
ATAN2(number_x, number_y)	Returns the inverse tangent of the specified x and y coordinates in radians. **number_x** is the value for the x coordinate. **number_y** is the value for the y coordinate.
ATANH(Number)	Returns the inverse hyperbolic tangent of the given **Number**. (Angle is returned in radians.)
BITAND(Number, Number)	This is the bitwise “AND” of two positive integers whose values are less than 2^48. Both parameters are required. Bug 71810, concerning parameter names in BITAND, BITOR, and BITXOR.
BITLSHIFT(Number, Shift)	The bitwise left shift of an integer value. Both parameters are required. **Number** is an integer less than 2^48. **Shift** is the number of bits to move by.
BITOR(Number, Number)	This is the bitwise “OR” of two positive integers whose values are less than 2^48. Both parameters are required.
BITRSHIFT(Number, Shift)	The bitwise right shift of an integer value. Both parameters are required. **Number** is an integer less than 2^48. **Shift** is the number of bits to move by.
BITXOR(number, number)	This is the bitwise “exclusive OR” of two positive integers whose values are less than 2^48. Both parameters are required.
CEILING(Number, Significance, Mode)	Rounds up the given **Number** to the nearest multiple of the value of **Significance**. **Mode** is an optional value. If the mode value is given and not equal to zero, and if number and significance are negative, then rounding is done based on the absolute value of number. Omit this value for Excel compatibility.
COMBIN(count_1, count_2)	Returns the number of combinations for elements without repetition. **count_1** is the total number of elements. **count_2** is the number to be combined from the elements. This is the same as the nCr function on a calculator.

Syntax	*Description*
COMBINA(count_1, count_2)	Returns the number of combinations for a given number of objects (repetition included). **count_1** is the total number of elements. **count_2** is the number to choose from the elements.
CONVERT(value, text, text)	Converts a value from one unit of measurement to another. **value** is the quantity to be converted. The first **text** is the official abbreviation for the measurement in question (for example, "mi" for miles). The second **text** parameter gives the unit to which it is to be converted. Both text arguments must be within quotes and are case sensitive. The conversion is done according to a table in the configuration (main.xcd). Bug 69539: This function does not work.
COS(Number)	Returns the cosine of the **Number** (the angle in radians).
COSH(Number)	Returns the hyperbolic cosine of the **Number** (the angle in radians).
COT(Number)	Returns the cotangent of the **Number** (the angle in radians).
COTH(Number)	Returns the hyperbolic cotangent of the **Number** (the angle in radians).
COUNTBLANK(range)	Returns the number of empty cells. **range** is the cell range in which the empty cells are counted.
COUNTIF(range, criteria)	Returns the number of cells that meet the criteria within a cell range. **range** is the range to which the criteria are to be applied. **criteria** indicates the criteria in the form of a number, a regular expression, or a character string by which the cells are counted.
COUNTIFS(range 1, criteria 1, range 2, criteria 2, …,)	Returns the number of cells that meet multiple criteria in multiple cell ranges. **range 1** (required), **range 2, …,** are the ranges to which the criteria are to be applied. **criteria 1** (required), **criteria 2, …,** indicate the criteria in the form of a number, a regular expression, or a character string by which the cells are evaluated. All ranges must have the same dimension and size.
CSC(Angle)	Returns the cosecant of an angle given in radians (1/SIN(X)).
CSCH(Angle)	Returns the hyperbolic cosecant of a hyperbolic angle (1/SINH(X)).
DEGREES(Number)	Converts the given **Number** in radians to degrees.

Syntax	*Description*
EUROCONVERT(value, from_currency, to_currency, full_precision, triangulation_precision)	Converts from one pre-Euro currency to another. **value** is the value to be converted. The **from_currency** is the ISO 4217 code of the currency from which **value** is to be converted. The **to_currency** is the ISO 4217 code of the currency to which **value** is to be converted. The entries are not case sensitive. The above parameters are required. The optional **full_precision** parameter, if omitted, is 0 or FALSE rounds the result to the decimals of the **to_currency**. If full_precision is TRUE, the result is not rounded. **triangulation_precision** is optional. If **triangulation_precision** is given and >=3, the intermediate result of a triangular conversion (currency1, EURO, currency2) is rounded to that precision. If **triangulation_precision** is omitted, the intermediate result is not rounded. Also if t**o_currency** is "EUR", **triangulation_precision** is used as if triangulation was needed and conversion from Euro to Euro was applied. Conversion rates and currency codes can be found here: http://ec.europa.eu/economy_finance/euro/adoption/conversion/index_en.htm The Cyprus pound has been omitted from this list but is “CYP”. Bug 71850: These are NOT case sensitive as stated in the Function Wizard.
EVEN(Number)	Rounds the given **Number** up to the nearest even integer, and a negative number down to the next even number.
EXP(Number)	Returns *e* raised to the power of the given **Number**.
FACT(Number)	Returns the factorial of the given **Number**.
FLOOR(Number, Significance, Mode)	Rounds the given **Number** down to the nearest multiple of **Significance**. **Significance** is the value to whose multiple the number is to be rounded down. **Mode** is an optional value. If it is indicated and non-zero and if the number and significance are negative, rounding is done based on the absolute value of the number. Note: Many application user interfaces have a FLOOR function with only two parameters, and somewhat different semantics than given here (e.g., they operate as if there was a non-zero mode value). These FLOOR functions are inconsistent with the standard mathematical definition of FLOOR.
GCD(Integer 1, Integer 2, …, Integer 30))	Returns the greatest common divisor of one or more positive integers. **Integers x** is a list of up to 30 integers, at least one of which must be greater than zero, whose greatest common divisor is to be calculated. This gives a result based on international standards.
GCD_ADD(Number(s), Number(s)1, …, Number(s)30)	Returns the greatest common divisor of a list of numbers. **Number(s) X** is a list of up to 30 numbers, additional to **Number(s)** separated by commas. This gives the same results as MS Excel.
INT(Number)	Rounds the given **Number** down to the nearest integer.
LCM(Integer 1, Integer 2, …, Integer 30)	Returns the least common multiple of one or more integers. **Integer 1, Integer 2, …, Integer 30** are integers whose lowest common multiple is to be calculated.

Syntax	*Description*
LCM_ADD(Number(s), Number(s)1, …, Number(s)30)	**Number(s) X** is a list of up to 30 numbers, additional to **Number(s)**, separated by commas. The result is the lowest common multiple of a list of numbers.
LN(Number)	Returns the natural logarithm, based on the constant *e,* of the given **Number**.
LOG(Number, Base)	Returns the logarithm of the given **Number** (value >0) to the specified base. **Base** is the base for the logarithm calculation. If omitted, 10 is assumed.
LOG10(Number)	Returns the base-10 logarithm of a **Number** >0.
MOD(Dividend, Divisor)	Returns the remainder after a number is divided by a divisor. **Dividend** is the number to be divided. **Divisor** is the number by which the dividend is divided.
MROUND(Number, Multiple)	Returns **Number** rounded to the nearest multiple of **Multiple**.
MULTINOMIAL (Number(s), Number(s)1, …, Number(s)30)	Returns the factorial of the sum of the arguments divided by the product of the factorials of the arguments. **Number(s) X** is a list of up to 30 numbers, additional to **Number(s)**, separated by commas.
ODD(Number)	Rounds **Number** up if positive and down if negative, to the nearest odd integer.
PI()	Returns the value of PI to fourteen decimal places.
POWER(Base, Exponent)	Returns the result of a number raised to a power. **Base** is the number that is to be raised to the given power. **Exponent** is the exponent by which the base is to be raised.
PRODUCT(Number 1, Number 2, …, Number 30)	Multiplies all the numbers given as arguments and returns the product. **Number 1** to **Number 30** are up to *30 arguments* whose product is to be calculated, separated by commas.
QUOTIENT(Numerator, Denominator)	Returns the integer result of a division operation. **Numerator** is the number that will be divided. **Denominator** is the number the numerator will be divided by.
RADIANS(Number)	Converts the given **Number** in degrees to radians.
RAND()	Returns a random number between 0 and 1. This number will recalculate every time data is entered, *Ctrl+Shift+F9* or *F9* is pressed.
RANDBETWEEN (Bottom, Top)	Returns an integer random number between **Bottom** and **Top** (inclusive). This number will recalculate when the *Ctrl+Shift+F9* key combination is pressed (not F9 alone).
ROUND(number, count)	Rounds the given **number** to **count** (optional) decimal places. If the **count** parameter is omitted or zero, **number** rounds to the nearest integer. If **count** is negative, the function rounds to the nearest 10, 100, 1000 and so on.
ROUNDDOWN(number, count)	Rounds the given **number** down, to **count** (optional) decimal places. If the **count** parameter is omitted or zero, **number** rounds down to the nearest integer. If **count** is negative, the function rounds down to the nearest 10, 100, 1000 and so on. Number rounds toward zero.

Syntax	*Description*
ROUNDUP(number, count)	Rounds the given **number** up to **count** (optional) decimal places. If the **count** parameter is omitted or zero, **number** rounds up to the nearest integer. If **count** is negative, the function rounds up to the nearest 10, 100, 1000 and so on. Number rounds away from zero.
SEC(Angle)	Returns the secant of an **Angle** given in radians. SEC(x)=1/COS(x).
SECH(Angle)	Returns the hyperbolic secant of an **Angle** given in radians. SECH(x)=1/COSH(x).
SERIESSUM(X, N, M, Coefficients)	Returns the sum of a powers series. SERIESSUM(X, N, M, Coefficients) = coefficient_1*x^n + coefficient_2*x^(n+m) + coefficient_3*x^(n+2m) +...+ coefficient_i*x^(n+(i-1)m). **X** is the number as an independent variable. **N** is the starting power. **M** is the increment. **Coefficients** is a series of coefficients. For each coefficient the series sum is extended by one section. You can only enter coefficients using a cell range.
SIGN(Number)	Returns the sign of the given **Number**. The function returns the result 1 for a positive sign, –1 for a negative sign, and 0 for zero.
SIN(number)	Returns the sine of the given **number** (angle in radians).
SINH(number)	Returns the hyperbolic sine of the given **number** (angle in radians).
SQRT(number)	Returns the positive square root of the given **number**. The value of the **number** must be positive.
SQRTPI(Number)	Returns the square root of the product of the given **Number** and PI.
SUBTOTAL(Function, range)	Calculates subtotals. If a range already contains subtotals, these are not used for further calculations. **Function** is a value that stands for another function such as Average, Count, Min, Sum, Var. **range** is the range whose cells are included.
SUM(number 1, number 2, …, number 30)	Adds all the numbers in a range of cells. **Number 1, number 2, …, number 30** are up to 30 arguments whose sum is to be calculated. You can also enter a range using cell references.
SUMIF(range, criteria, sum_range)	Adds the cells specified by the given criteria. The search supports regular expressions. **range** is the range to which the criteria are to be applied. **criteria** is the cell in which the search criterion is shown, or the search criterion itself. **sum_range** (optional) is the range from which values are summed; if it has not been entered, the values found in the **range** are summed. If supplied, **sum_range** must be the same size and shape as **range**.
SUMIFS(sum_range, range 1, criteria 1, range 2, criteria 2, …,)	Totals the values of cells in a range that meet multiple criteria in multiple ranges. **sum_range** (required) is the cell range from which the values are to be totaled. **range 1** (required) is the cell range to be evaluated by **criteria 1** (required), **range 2** by **criteria 2** and so on. All ranges must have the same size and shape.

Syntax	Description
SUMSQ(number 1, number 2, …, number 30)	Calculates the sum of the squares of numbers (totaling up of the squares of the arguments) **number 1, number 2, …, number 30** are up to 30 arguments, the sum of whose squares is to be calculated.
TAN(number)	Returns the tangent of the given **number** (angle in radians).
TANH(number)	Returns the hyperbolic tangent of the given **number** (angle in radians).
TRUNC(number, count)	Truncates a number by removing decimal places. **number** is the number whose decimal places are to be trimmed. **count** (optional) is the number of decimal places which are retained. If **count** is missing or zero, it effectively truncates to a decimal integer. If **count** is negative, it truncates to the left of the decimal point.

Financial analysis functions

A note about dates

Date values used as parameters for Calc's financial functions must comply with ISO8601 and be entered surrounded by double quotes. For example, a date representing August 6, 2004, must be entered "2004-08-06", single digits are padded with leading zeroes. If you do not enter the date values as required by the function, you will not get the correct results. Date formats are locale specific and will allow other formats to be used. Among others, the en_US locale allows "2004/08/06" and "08/06/2004" for example. Check the Help for the acceptable formatting.

A note about interest rates

You can enter interest rates in either of two ways:

- As a decimal. To enter an interest rate as a decimal, divide it by 100 before entering it into a function. For example, to compute a loan with a 3.25% interest rate, enter *.0325* into the function.
- As a percentage. To enter an interest rate as a percentage, type in the interest rate followed by the *%* key. For example, to compute a loan with a 3.25% interest rate, enter *3.25%* into the function.

If you enter it as *3.25*, the function will treat it as a 325% interest rate.

Accounting systems vary in the number of days in a month or a year used in calculations. The following table gives the integers used for the **basis** parameter used in some of the financial analysis functions.

Table 2: Basis calculation types

Basis	Calculation
0 or missing	US method (NASD), 12 months of 30 days each.
1	Exact number of days in months, exact number of days in year.
2	Exact number of days in month, year has 360 days.
3	Exact number of days in month, year has 365 days.
4	European method, 12 months of 30 days each.

Table 3: Financial analysis functions

Syntax	***Description***
ACCRINT(Issue, First interest, Settlement, Rate, Par, Frequency, Basis)	Calculates the accrued interest of a security in the case of periodic payments. **Issue** is the issue date of the security. **First interest** is the first interest date of the security. **Settlement** is the maturity date. **Rate** is the annual nominal rate of interest (coupon interest rate). **Par** is the par value of the security. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
ACCRINTM(Issue, Settlement, Rate, Par, Basis)	Calculates the accrued interest of a security in the case of one-off payment at the settlement date. **Issue** is the issue date of the security. **Settlement** is the maturity date. **Rate** is the annual nominal rate of interest (coupon interest rate). **Par** is the par value of the security. **Basis** indicates how the year is to be calculated.
AMORDEGRC(Cost, Date purchased, First period, Salvage, Period, Rate, Basis)	Calculates the amount of depreciation for a settlement period as degressive amortization. Unlike AMORLINC, a depreciation coefficient that is independent of the depreciable life is used here. **Cost** is the acquisition cost. **Date purchased** is the date of acquisition. **First period** is the end date of the first settlement period. **Salvage** is the salvage value of the capital asset at the end of the depreciable life. **Period** is the settlement period to be considered. **Rate** is the rate of depreciation. **Basis** indicates how the year is to be calculated.
AMORLINC(Cost, Date purchased, First period, Salvage, Period, Rate, Basis)	Calculates the amount of depreciation for a settlement period as linear amortization. If the capital asset is purchased during the settlement period, the proportional amount of depreciation is considered. **Cost** is the acquisition cost. **Date purchased** is the date of acquisition. **First period** is the end date of the first settlement period. **Salvage** is the salvage value of the capital asset at the end of the depreciable life. **Period** is the settlement period to be considered. **Rate** is the rate of depreciation. **Basis** indicates how the year is to be calculated.
COUPDAYBS(Settlement, Maturity, Frequency, Basis)	Returns the number of days from the first day of interest payment on a security until the settlement date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
COUPDAYS(Settlement, Maturity, Frequency, Basis)	Returns the number of days in the current interest period in which the settlement date falls. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.

Syntax	*Description*
COUPDAYSNC(Settlement, Maturity, Frequency, Basis)	Returns the number of days from the settlement date until the next interest date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
COUPNCD(Settlement, Maturity, Frequency, Basis)	Returns the date of the first interest date after the settlement date, and formats the result as a date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
COUPNUM(Settlement, Maturity, Frequency, Basis)	Returns the number of coupons (interest payments) between the settlement date and the maturity date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
COUPPCD(Settlement, Maturity, Frequency, Basis)	Returns the date of the interest date prior to the settlement date, and formats the result as a date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
CUMIPMT(Rate, NPER, pv, S, E, Type)	Calculates the cumulative interest payments (the total interest) for an investment based on a constant interest rate. **Rate** is the periodic interest rate. **NPER** is the payment period with the total number of periods. NPER can also be a non-integer value. The R**ate** and **NPER** must refer to the same unit, and thus both must be calculated annually or monthly. **pv** is the current value in the sequence of payments. **S** is the first period. **E** is the last period. **Type** is the due date of the payment at the beginning (1) or end (0) of each period.
CUMIPMT_ADD(Rate, Nper, Pv, Start period, End period, Type)	Calculates the accumulated interest for a period. **Rate** is the interest rate for each period. **Nper** is the total number of payment periods. The **Rate** and **Nper** must refer to the same unit, and thus both must be calculated annually or monthly. **Pv** is the current value. **Start period** the first payment period for the calculation. **End period** the last payment period for the calculation. **Type** is the due date of the payment at the beginning (1) or end (0) of each period.
CUMPRINC(Rate, NPER, PV, S, E, Type)	Returns the cumulative interest paid for an investment period with a constant interest rate. **Rate** is the periodic interest rate. **NPER** is the payment period with the total number of periods. **NPER** can also be a non-integer value. The **Rate** and **NPER** must refer to the same unit, and thus both must be calculated annually or monthly. **PV** is the current value in the sequence of payments. **S** is the first period. **E** is the last period. **Type** is the due date of the payment at the beginning (1) or end (0) of each period.

Syntax	*Description*
CUMPRINC_ADD(Rate, Nper, Pv, Start period, End period, Type)	Calculates the cumulative redemption of a loan in a period. **Rate** is the interest rate for each period. **Nper** is the total number of payment periods. The **Rate** and **Nper** must refer to the same unit, and thus both must be calculated annually or monthly. **Pv** is the current value. **Start period** is the first payment period for the calculation. **End period** is the last payment period for the calculation. **Type** is the due date of the payment at the beginning (1) or end (0) of each period.
DB(Cost, Salvage, Life, Period, month)	Returns the depreciation of an asset for a specified period using the fixed-declining balance method. **Cost** is the initial cost of an asset. **Salvage** is the value of an asset at the end of the depreciation. **Life** defines the period over which an asset is depreciated. **Period** is the length of each period. The life must be entered in the same date unit as the depreciation period. **month** (optional) denotes the number of months for the first year of depreciation.
DDB(Cost, Salvage, Life, Period, Factor)	Returns the depreciation of an asset for a specified period using the arithmetic-declining method. Note that the book value will never reach zero under this calculation type. **Cost** fixes the initial cost of an asset. **Salvage** fixes the value of an asset at the end of its life. **Life** is the number of periods defining how long the asset is to be used. **Period** defines the length of the period. The period must be entered in the same time unit as the life. **Factor** (optional) is the factor by which depreciation decreases. If no value is entered, a value of 2 is assumed, making this double declining.
DISC(Settlement, Maturity, Price, Redemption, Basis)	Calculates the allowance (discount) of a security as a percentage. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Price** is the price of the security per 100 currency units of par value. **Redemption** is the redemption value of the security per 100 currency units of par value. **Basis** indicates how the year is to be calculated.
DOLLARDE(Fractional dollar, Fraction)	Converts a quotation that has been given as a decimal fraction into a decimal number. **Fractional dollar** is a number given as a decimal fraction. (In this number, the decimal value is the numerator of the fraction.) **Fraction** is a whole number that is used as the denominator of the decimal fraction.
DOLLARFR(Decimal dollar, Fraction)	Converts a quotation that has been given as a decimal number into a mixed decimal fraction. The decimal of the result is the numerator of the fraction that would have **Fraction** as the denominator. **Decimal dollar** is a decimal number. **Fraction** is a whole number that is used as the denominator of the decimal fraction.
DURATION(RATE, pv, FV)	Calculates the number of periods required by an investment to attain the desired value. **RATE** (a constant) is the interest rate to be calculated for the entire duration. Entering the interest rate divided by the periods per year, can calculate the interest after each period. **pv** is the present value. **FV** is the desired future value of the investment.

Syntax	*Description*
DURATION_ADD (Settlement, Maturity, Coupon, Yield, Frequency, Basis)	Calculates the duration of a fixed interest security in years. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Coupon** is the annual coupon interest rate (nominal rate of interest). **Yield** is the annual yield of the security. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
EFFECT_ADD(Nominal rate, Npery)	Calculates the effective annual rate of interest on the basis of the nominal interest rate and the number of interest payments per annum. Nominal interest refers to the amount of interest due at the end of a calculation period. **Nominal rate** is the annual nominal rate of interest. **Npery** is the number of interest payments per year.
EFFECTIVE(NOM, P)	Calculates the effective annual rate of interest on the basis of the nominal interest rate and the number of interest payments per annum. Nominal interest refers to the amount of interest due at the end of a calculation period. **NOM** is the nominal interest. **P** is the number of interest payment periods per year.
FV(Rate, NPER, PMT, PV, Type)	Returns the future value of an investment based on periodic, constant payments and a constant interest rate. **Rate** is the periodic interest rate. **NPER** is the total number of periods. **PMT** is the annuity paid regularly per period. **PV** (optional) is the present cash value of an investment. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.
FVSCHEDULE(Principal, Schedule)	Calculates the accumulated value of the starting capital for a series of periodically varying interest rates. **Principal** is the starting capital. **Schedule** is a series of interest rates. **Schedule** has to be entered with cell references, or with a list.
INTRATE(Settlement, Maturity, Investment, Redemption, Basis)	Calculates the annual interest rate that results when a security (or other item) is purchased at an investment value and sold at a redemption value with no interest being paid. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security is sold. **Investment** is the purchase price. **Redemption** is the selling price. **Basis** indicates how the year is to be calculated.
IPMT(Rate, Period, NPER, pv, FV, Type)	Calculates the periodic amortization for an investment with regular payments and a constant interest rate. **Rate** is the periodic interest rate. **Period** is the period for which the compound interest is calculated. **NPER** is the total number of periods during which annuity is paid. **Period**=**NPER**, if compound interest for the last period is calculated. **pv** is the present cash value in sequence of payments. **FV** (optional) is the desired value (future value) at the end of the periods. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.

Syntax	*Description*
IRR(Values, Guess)	Calculates the internal rate of return for an investment. The values represent cash flow values at regular intervals; at least one value must be negative (payments), and at least one value must be positive (income). **Values** is an array or cell range containing the values. **Guess** (optional) is the estimated value. If you can provide only a few values, you should provide an initial guess to enable the iteration.
ISPMT(rate, Period, total_periods, invest)	Calculates the level of interest for unchanged amortization installments. r**ate** sets the periodic interest rate. **Period** is the number of installments for calculation of interest. t**otal_periods** is the total number of installment periods. i**nvest** is the amount of the investment.
MDURATION(Settlement, Maturity, Coupon, Yield, Frequency, Basis)	Calculates the modified Macauley duration for a security with an assumed par value of 100 currency units. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Coupon** is the annual nominal rate of interest (coupon interest rate) **Yield** is the annual yield of the security. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
MIRR(Values, investment, reinvest_rate)	Calculates the modified internal rate of return of a series of investments. **Values** corresponds to the array or the cell reference for cells whose content corresponds to the payments. **investment** is the rate of interest of the investments (the negative values of the array) **reinvest_rate** is the rate of interest of the reinvestment (the positive values of the array).
NOMINAL(effect_rate, npery)	Calculates the yearly nominal interest rate, given the effective rate and the number of compounding periods per year. **effect_rate** is the effective interest rate. **npery** is the number of periodic interest payments per year. Returns a percentage.
NOMINAL_ADD(Effective_rate, Npery)	Calculates the yearly nominal rate of interest, given the effective rate and the number of compounding periods per year. **Effective_rate** is the effective annual rate of interest. **Npery** is the number of interest payments per year. Returns a number.
NPER(Rate, PMT, PV, FV, Type)	Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate. **Rate** is the periodic interest rate. **PMT** is the constant annuity paid in each period. **PV** is the present value (cash value) in a sequence of payments. **FV** (optional) is the future value, which is reached at the end of the last period. If FV is omitted it is assumed to be zero. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.
NPV(Rate, value 1, value 2, …, value 30)	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate. **Rate** is the discount rate for a period. **value 1, value 2, ..., value 30** are values representing deposits or withdrawals.

Syntax	*Description*
ODDFPRICE(Settlement, Maturity, Issue, First coupon, Rate, Yield, Redemption, Frequency, Basis)	Calculates the price per 100 currency units par value of a security, having an odd (short or long) first period. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Issue** is the date of issue of the security. **First coupon** is the first interest date of the security. **Rate** is the annual rate of interest. **Yield** is the annual yield of the security. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
ODDFYIELD(Settlement, Maturity, Issue, First coupon, Rate, Price, Redemption, Frequency, Basis)	Calculates the yield of a security that has an odd (short or long) first period. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Issue** is the date of issue of the security. **First coupon** is the first interest date of the security. **Rate** is the annual rate of interest. **Price** is the price of the security. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** is chosen from a list of options and indicates how the year is to be calculated.
ODDLPRICE(Settlement, Maturity, Last interest, Rate, Yield, Redemption, Frequency, Basis)	Calculates the price per 100 currency units par value of a security, that has an odd (short or long) last period. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Last interest** is the last interest date of the security. **Rate** is the annual rate of interest. **Yield** is the annual yield of the security. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
ODDLYIELD(Settlement, Maturity, Last interest, Rate, Price, Redemption, Frequency, Basis)	Calculates the yield of a security that has an odd (short or long) last period. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Last interest** is the last interest date of the security. **Rate** is the annual rate of interest. **Price** is the price of the security. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
OPT_BARRIER(spot, vol, r, rf, T, strike, barrier_low, barrier_up, rebate, put/call, knock in/out, barrier_type, greek)	A function following the Black Scholes formula. It calculates the pricing of a barrier option. **spot** (required) is the price/value of the asset. **vol** (required) is the annual volatility of the asset. **rebate** (required) is the amount of money paid at maturity if the barrier was hit. **put/call** (required) is a string to define if the option is a (p)ut/ or a (c)all. **knock** (required) (i)n/(o)ut is a string to define if the option is of type knock-(i)n or knock-(o)ut. **barrier_type** (required) is a string to define whether the barrier is observed (c)ontinuosly or only at the (e)nd/maturity. **greek** is an optional parameter, which if left out causes the function to return the option price. If included, the function returns price sensitivities (Greeks) to one of the input parameters, such as “vega” for sensitivity to volatility.

Syntax	*Description*
OPT_PROB_HIT(spot, vol, drift, T, barrier_low, barrier_up)	Returns the probability that an asset hits a barrier assuming it follows $\frac{dS}{S} = \mu\, dt + vol\, dW$ **spot** is the price/value **S** of the underlying asset. **vol** is the annual volatility of the underlying asset. **drift** is the **μ** value of the formula. **T** is the time to maturity. **barrier_low** is the lower barrier and set to zero if there is no lower barrier. **barrier_up** is the upper barrier and is set to zero if there is no upper barrier. All parameters are required.
OPT_PROB_INMONEY(spot, vol, drift, T, barrier_low, barrier_up, put/call, strike)	Returns the probability that an asset will at maturity end up between two barrier levels, assuming it follows $\frac{dS}{S} = \mu\, dt + vol\, dW$. (If the last two optional parameters, put/call and strike, are specified, the probability of S_T in [strike, upper barrier] for a call, and S_T in [lower barrier, strike] for a put will be returned). S_T is the spot at maturity and ignores the possibility of knock-out before maturity. **spot** (required) is the price/value of the asset. **vol** (required) is the annual volatility of the asset. **drift** (required) is the parameter μ from the formula above. **T** is the time to maturity in years. **barrier_low** (required) is the lower barrier and set to zero if there is no lower barrier. **barrier_up** (required) is the upper barrier and is set to zero if there is no upper barrier. **put/call** (optional) is the (p)ut/(c)all indicator. **strike** (optional) is the strike level.
OPT_TOUCH(spot, vol, r, rf, T, barrier_low, barrier_up, foreign/domestic, knock in/out, barrier_type, greek)	Returns the pricing of a touch/no-touch option. **spot** (required) is the price/value of the asset. **vol** (required) is the annual volatility of the asset. **r** (required) is the interest rate continuously compounded. **rf** (required) is the foreign interest rate continuously compounded. **T** (required) is the time to maturity entered in years. **strike** (required) is the strike level of the option. **barrier_low** (required) is the lower barrier and set to zero if there is no lower barrier. **barrier_up** (required) is the upper barrier and is set to zero if there is no upper barrier. **foreign/domestic** (required) is a string to define if the option pays one unit of (d)omestic currency (cash or nothing) or (f)oreign currency (asset or nothing). **knock** (required) (i)n/(o)ut is a string to define if the option is of type knock-(i)n (touch) or knock-(o)ut (no touch). **barrier_type** (required) is a string to define whether the barrier is observed (c)ontinuously or only at the (e)nd/maturity. **greek** is an optional parameter, which if left out causes the function to return the option price. If included, the function returns price sensitivities (Greeks) to one of the input parameters, such as “theta” for time sensitivity.

Syntax	*Description*
PMT(Rate, NPER, PV, FV, Type)	Returns the periodic payment for an annuity with constant interest rates. **Rate** is the periodic interest rate. **NPER** is the number of periods in which annuity is paid. **PV** is the present value (cash value) in a sequence of payments. **FV** (optional) is the desired value (future value) to be reached at the end of the periodic payments. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.
PPMT(Rate, Period, NPER, PV, FV, Type)	Returns for a given period the payment on the principal for an investment that is based on periodic and constant payments and a constant interest rate. **Rate** is the periodic interest rate. **Period** is the amortization period. **NPER** is the total number of periods during which annuity is paid. **PV** is the present value in the sequence of payments. **FV** (optional) is the desired (future) value. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.
PRICE(Settlement, Maturity, Rate, Yield, Redemption, Frequency, Basis)	Calculates the price per 100 currency units of par value of an interest-bearing security. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Rate** is the annual nominal rate of interest (coupon interest rate). **Yield** is the annual yield of the security. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
PRICEDISC(Settlement, Maturity, Discount, Redemption, Basis)	Calculates the price per 100 currency units of par value of a discounted security. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Discount** is the discount of a security as a percentage. **Redemption** is the redemption value per 100 currency units of par value. **Basis** indicates how the year is to be calculated.
PRICEMAT(Settlement, Maturity, Issue, Rate, Yield, Basis)	Calculates the price per 100 currency units of par value of a security, that pays interest on the maturity date. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Issue** is the date of issue of the security. **Rate** is the interest rate of the security on the issue date. **Yield** is the annual yield of the security. **Basis** indicates how the year is to be calculated.
PV(Rate, NPER, PMT, FV, Type)	Returns the present value of an investment resulting from a series of regular payments. **Rate** defines the interest rate per period. **NPER** is the total number of payment periods. **PMT** is the regular payment made per period. **FV** (optional) defines the future value remaining after the final installment has been made. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period.

Syntax	*Description*
RATE(NPER, PMT, PV, FV, Type, Guess)	Returns the constant interest rate per period of an annuity. **NPER** is the total number of periods, during which payments are made (payment period). **PMT** is the constant payment (annuity) paid during each period. **PV** is the cash value in the sequence of payments. **FV** (optional) is the future value, which is reached at the end of the periodic payments. **Type** (optional) defines whether the payment is due at the beginning (1) or the end (0) of a period. **Guess** (optional) determines the estimated value of the interest with iterative calculation.
RECEIVED(Settlement, Maturity, Investment, Discount, Basis)	Calculates the amount paid out at maturity for a fully invested security. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures. **Investment** is the purchase sum. **Discount** is the percentage discount on acquisition of the security. **Basis** indicates how the year is to be calculated.
RRI(P, pv, FV)	Calculates the interest rate resulting from the profit (return) of an investment. **P** is the number of periods used for calculating the interest rate. **pv** is the present value (must be >0). **FV** is the final value of the security.
SLN(Cost, Salvage, Life)	Returns the straight-line depreciation of an asset for one period. The amount of the depreciation is constant during the depreciation period. **Cost** is the initial cost of an asset. **Salvage** is the value of an asset at the end of the depreciation. **Life** is the number of periods in the useful life of the asset.
SYD(Cost, Salvage, Life, Period)	Returns the arithmetically declining value of an asset (depreciation) for a specified period. It uses the Sum-of-Years'-Digits method. **Cost** is the initial cost of an asset. **Salvage** is the value of an asset after depreciation. **Life** is the period fixing the time span over which an asset is depreciated. **Period** defines the period for which the depreciation is to be calculated. Must use the same units as **life**.
TBILLEQ(Settlement, Maturity, Discount)	Calculates the bond equivalent yield for a treasury bill. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). The settlement and maturity date must be within a year of each other. **Discount** is the percentage discount on acquisition of the security. Calculated using the 360 days in a year basis (basis 2).
TBILLPRICE(Settlement, Maturity, Discount)	Calculates the price per 100 currency units face value of a treasury bill. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). The settlement and maturity date must be within a year of each other. **Discount** is the percentage discount upon acquisition of the security.

Syntax	*Description*
TBILLYIELD(Settlement, Maturity, Price)	Calculates the yield of a treasury bill. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). The settlement and maturity date must be within a year of each other. **Price** is the price (purchase price) of the treasury bill per 100 currency units of par value.
VDB(Cost, Salvage, Life, S, end, Factor, Type)	Returns the depreciation of an asset for a specified or partial period using a variable declining balance method. **Cost** is the initial value of an asset. **Salvage** is the value of an asset at the end of the depreciation. **Life** is the depreciation duration of the asset. **S** is the start period of the depreciation, entered in the same date unit as **Life**. **end** is the last period of the depreciation, entered in the same date unit as **Life**. **Factor** (optional) is the depreciation factor. If factor is omitted, a factor of two is assumed (the double-declining balance method). **Type** is an optional parameter. Type = 1 means a switch to linear depreciation. In Type = 0, no switch is made.
XIRR(Values, dDates, Guess)	Calculates the internal rate of return for a list of payments which take place on different dates. The calculation is based on a 365 days per year basis, ignoring leap years. If the payments take place at regular intervals, use the IRR function. **Values** and **Dates** are a series of payments and the series of associated date values entered as cell references. Values shall include at least one negative value and one positive value. **Guess** (optional) is a guess for the internal rate of return. If omitted, the value 10% is assumed.
XNPV(Rate, Values, Dates)	Calculates the capital value (net present value) for a list of payments which take place on different dates. The calculation is based on a 365 days per year basis, ignoring leap years. If the payments take place at regular intervals, use the NPV function. **Rate** is the internal rate of return for the payments. **Values** and **Dates** are a series of payments and the series of associated date values entered as cell references. The first value-date pair indicates the start of the payments, other dates can be in any order. Values shall include at least one negative value and one positive value.
YIELD(Settlement, Maturity, Rate, Price, Redemption, Frequency, Basis)	Calculates the yield of a security that pays periodic interest. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Rate** is the annual rate of interest. **Price** is the price (purchase price) of the security per 100 currency units of par value. **Redemption** is the redemption value per 100 currency units of par value. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** indicates how the year is to be calculated.
YIELDDISC(Settlement, Maturity, Price, Redemption, Basis)	Calculates the annual yield of a non-interest-bearing security. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Price** is the price (purchase price) of the security per 100 currency units of par value. **Redemption** is the redemption value per 100 currency units of par value. **Basis** indicates how the year is to be calculated.

Syntax	Description
YIELDMAT(Settlement, Maturity, Issue, Rate, Price, Basis)	Calculates the annual yield of a security, the interest of which is paid on the date of maturity. **Settlement** is the date of purchase of the security. **Maturity** is the date on which the security matures (expires). **Issue** is the date of issue of the security. **Rate** is the interest rate of the security on the issue date. **Price** is the price (purchase price) of the security per 100 currency units of par value. **Basis** indicates how the year is to be calculated.

Statistical analysis functions

Calc includes over 70 statistical functions which enable the evaluation of data from simple arithmetic calculations, such as averaging, to advanced distribution and probability computations. Several other statistics-based functions are available through the Add-ins which are noted at the end of this appendix.

Table 4: Statistical analysis functions

Syntax	Description
AVEDEV(number 1, number 2, …, number 30)	Returns the average of the absolute deviations of data points from their mean. Displays the diffusion in a data set. **number 1, number 2, …, number 30** are values or ranges that represent a sample. Each number can also be replaced by a reference.
AVERAGE(number 1, number 2, …, number 30)	Returns the average of the arguments. **number 1, number 2, …, number 30** are numerical values or ranges. Text is ignored.
AVERAGEA(value 1, value 2, …, value 30)	Returns the average of the arguments. The value of text is taken to be 0. **value 1, value 2, …, value 30** are values or ranges.
AVERAGEIF(range, criteria, average_range)	Averages the arguments that meet the conditions. If the optional **average_range** is omitted, **range**, which is required, is the range of cells that will be averaged. **criteria** is a required value which determines which cells in **range** are averaged. If the optional **average_range** is used, it averages the values of cells of a range that is constructed using the top left cell of **range** and applying the dimensions, shape and size, of **average_range**. If no cell in **range** matches the **criteria** value, an Error is returned. If no numbers are in the range to be averaged, an Error is returned.
AVERAGEIFS(average_range, range 1, criteria 1, range 2, criteria 2, …, range 30, criteria 30)	Averages the values of the cells in a range that meet multiple criteria in multiple ranges. **average_range**, **range 1** and **criteria 1** are required values. Averages the values of cells in **average_range** that meet the **criteria 1** in **range 1** and the **criteria 2** in **range 2**, and so on. All ranges must have the same dimension and size, else an Error is returned. A logical AND is applied between each array result of each selection; a cell of **average_range** is evaluated only if the same position in each array is the result of a criteria match. If no numbers are in the result set to be averaged, an Error is returned.

Syntax	*Description*
B(trials, SP, T_1, T_2)	Returns the probability of a sample with binomial distribution. **trials** is the number of independent trials. **SP** is the probability of success on each trial. **T_1** defines the lower limit for the number of trials. **T_2** (optional) defines the upper limit for the number of trials.
BETADIST(number, alpha, beta, Start, End, Cumulative)	Returns the value of the probability density function or the cumulative distribution function for the beta distribution. **number** is the value between **Start** and **End** at which to evaluate the function. **alpha** is a parameter to the distribution. **beta** is a parameter to the distribution. **Start** (optional) is the lower bound for **number**. **End** (optional) is the upper bound for **number**. **Cumulative** (optional) can be 0 or False to calculate the probability density function. It can be any other value or True or omitted to calculate the cumulative distribution function.
BETAINV(number; alpha, beta, Start, End)	Returns the inverse of the cumulative beta probability density function. **number** is the value between **Start** and **End** at which to evaluate the function. **alpha** is a parameter to the distribution. **beta** is a parameter to the distribution. **Start** (optional) is the lower bound for **number**. **End** (optional) is the upper bound for **number.**
BINOMDIST(X, trials, SP, C)	Returns the individual term binomial distribution probability. **X** is the number of successes in a set of trials. **trials** is the number of independent trials. **SP** is the probability of success on each trial. **C** = 0 calculates the probability of a single event and **C** = 1 calculates the cumulative probability.
CHIDIST(Number, degrees_freedom)	Returns the probability value that a hypothesis will be confirmed from the indicated chi square. The probability determined by CHIDIST can also be determined by CHITEST. **Number** is the chi-square value of the random sample used to determine the error probability. **degrees_freedom** is the degrees of freedom of the experiment. This function is defined by the ODF as LEGACY.CHIDIST. Use CHISQDIST for possible greater accuracy.
CHIINV(number, degrees_freedom)	Returns the inverse of the one-tailed probability of the chi-squared distribution. **number** is the value of the error probability. **degrees_freedom** is the degrees of freedom of the experiment. This function is defined by the ODF as LEGACY.CHIINV. Use CHISQINV for possible greater accuracy.
CHISQDIST(Number, Degrees of Freedom, Cumulative)	Returns the value of the probability density function or the cumulative distribution function for the chi-square distribution. **Number** is the value at which you want to evaluate the distribution. **Degrees of Freedom** is the number of degrees of freedom. **Cumulative** (optional) is a logical value that determines the form of the function. If cumulative is TRUE, CHISQDIST returns the cumulative distribution function; if FALSE, it returns the probability density function. If omitted, it is assumed TRUE.

Syntax	*Description*
CHISQINV(Probability, Degrees of Freedom)	Returns the inverse of CHISQDIST(x, Degrees of Freedom, TRUE()). **Probability** is the probability value for which the inverse of the chi square distribution is to be calculated. **Degrees of Freedom** is the number of degrees of freedom.
CHITEST(Data_B, data_E)	Returns the chi-square distribution from a random distribution of two test series based on the chi-square test for independence. The probability determined by CHITEST can also be determined with CHIDIST, in which case the chi square of the random sample must then be passed as a parameter instead of the data row. **Data_B** is the array of the observations. **data_E** is the range of the expected values. This function is defined by the ODF as LEGACY.CHITEST.
CONFIDENCE(alpha, STDEV, size)	Returns the (1-alpha) confidence interval for a normal distribution. **alpha** is the level of the confidence interval. **STDEV** is the standard deviation for the total population. **size** is the size of the total population.
CORREL(Data_1, Data_2)	Returns the correlation coefficient between two data sets. **Data_1** is the first data set. **Data_2** is the second data set. Both arrays shall be the same size and shape. Any empty element or non-numeric value in an element will cause the corresponding element to be ignored.
COUNT(value 1, value 2, …, value 30)	Counts how many numbers are in the list of arguments. Text entries are ignored. **value 1, value 2, …, value 30** are values or ranges which are to be counted.
COUNTA(value 1, value 2, …, value 30)	Counts how many values are in the list of arguments. Text entries are also counted, even when they contain an empty string of length 0. If an argument is an array or reference, empty cells within the array or reference are ignored. **value 1, value 2, …, value 30** are up to 30 arguments representing the values to be counted.
COVAR(Data_1, Data_2)	Returns the covariance of the product of paired deviations. **Data_1** is the first data set. **Data_2** is the second data set. Any empty element or non-numeric value in an element will cause the corresponding element to be ignored.
CRITBINOM(trials, SP, alpha)	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value. **trials** is the total number of trials. **SP** is the probability of success for one trial. **alpha** is the threshold probability to be reached or exceeded.
DEVSQ(number 1, number 2, …, number 30)	Returns the sum of squares of deviations based on a sample mean. **number 1, number 2, …, number 30** are numerical values or ranges representing a sample.
EXPONDIST(Number; lambda, C)	Returns the value of the probability density function or the cumulative distribution function for the exponential distribution. **Number** is the value of the function. **lambda** is the parameter value. **C** is a logical value that determines the form of the function. **C** = 0 calculates the density function, and **C** = 1 calculates the distribution function.

Syntax	*Description*
FDIST(Number, degrees_freedom_1, degrees_freedom_2)	Calculates the values of an F probability distribution. **Number** is the value for which the F distribution is to be calculated. **degrees_freedom_1** is the degrees of freedom in the numerator in the F distribution. **degrees_freedom_2** is the degrees of freedom in the denominator in the F distribution. In the ODF specification this is named LEGACY.FDIST and a new FDIST has been defined which has yet to be implemented in Calc.
FINV(number, degrees_freedom_1, degrees_freedom_2)	Returns the inverse of the F probability distribution. **number** is the probability value for which the inverse F distribution is to be calculated. **degrees_freedom_1** is the number of degrees of freedom in the numerator of the F distribution. **degrees_freedom_2** is the number of degrees of freedom in the denominator of the F distribution. In the ODF specification this is named LEGACY.FINV and a new FINV has been defined which has yet to be implemented in Calc.
FISHER(Number)	Returns the Fisher transformation for the given **Numbe**r. FISHER is a synonym for ATANH.
FISHERINV(Number)	Returns the inverse of the Fisher transformation for the given **Number.** FISHERINV is a synonym for TANH.
FORECAST(value, data_Y, data_X)	Extrapolates future values based on existing x and y values. **value** is the x value, for which the y value of the linear regression is to be returned. **data_Y** is the array or range of known Y-values. **data_X** is the array or range of known X-values. Does not work for exponential functions. Both arrays must be the same size and shape. A non-numeric value in an element causes the corresponding element to be ignored.
FTEST(data_1, data_2)	Returns the result of an F test. **data_1** is the first record array. **data_2** is the second record array.
GAMMA(Number)	Returns the value of the Gamma function. **Number** is the value for which the Gamma function is to be calculated.
GAMMADIST(Number, alpha, beta, Cumulative)	Returns the value of the probability density function or the cumulative distribution function for the Gamma distribution. **Number** is the value for which the Gamma distribution is to be calculated. **alpha** is the parameter Alpha of the Gamma distribution. **beta** is the parameter Beta of the Gamma distribution. **Cumulative** = 0 calculates the density function, and **Cumulative** = 1 calculates the distribution.
GAMMAINV(Number, alpha, beta)	Returns the inverse of the GAMMADIST(Number, alpha, beta, TRUE()). This function allows you to search for variables with different distribution. **Number** is the probability value for which the inverse Gamma distribution is to be calculated. **alpha** is the parameter Alpha of the Gamma distribution. **beta** is the parameter Beta of the Gamma distribution.
GAMMALN(Number)	Returns the natural logarithm of the Gamma function for the given **Number**.
GAUSS(Number)	Returns 0.5 less than the standard normal cumulative distribution for the given **Number**.

Syntax	*Description*
GEOMEAN(number 1, number 2, …, number 30)	Returns the geometric mean of a sample. **number 1, number 2, …, number 30** are numerical arguments or ranges that represent the sample.
HARMEAN(number 1, number 2, …, number 30)	Returns the harmonic mean of a data set. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals. **number 1, number 2, …, number 30** are values or ranges for which you want to calculate the harmonic mean.
HYPGEOMDIST(X, n_sample, successes, n_population)	Returns the hypergeometric distribution. **X** is the number of successes achieved in the random sample. **n_sample** is the size of the random sample. **successes** is the number of successes in the total population. **n_population** is the size of the total population. This function does not fully comply with the ODF v1.2 specification, having no logical **Cumulative** parameter.
INTERCEPT(data_Y, data_X)	Calculates the y-value at which a line will intersect the y-axis by using known x-values and y-values. **data_Y** is the array of Y-values. **data_X** is the array of X-values Numbers or names, arrays or references containing numbers must be used here.
KURT(number 1, number 2, …, number 30)	Returns the kurtosis of a data set (at least 4 values required). **number 1, number 2, …, number 30** are numerical arguments or ranges representing a random sample of distribution. Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive kurtosis indicates a relatively peaked distribution (compared to the normal distribution), while negative kurtosis indicates a relatively flat distribution.
LARGE(data, Rank_c)	Returns the Rank_c-th largest value in a data set. **data** is the cell range of data. **Rank_c** is the ranking of the value (2nd largest, 3rd largest, etc.) written as an integer.
LOGINV(number; mean, STDEV)	Returns the inverse of the lognormal distribution for the given **number**, a probability value. **mean** is the arithmetic mean of the standard logarithmic distribution. **STDEV** is the standard deviation of the standard logarithmic distribution.
LOGNORMDIST(Number, mean, STDEV, Cumulative)	Returns the value of the probability density function or the cumulative distribution function for the lognormal distribution with the mean and standard deviation given. **Number**, a probability value. **mean** is the mean value of the standard logarithmic distribution. **STDEV** is the standard deviation of the standard logarithmic distribution. **Cumulative** (optional) = 0 calculates the density function, **Cumulative** = 1 calculates the distribution.
MAX(number 1, number 2, …, number 30)	Returns the maximum value in a list of arguments. **number 1, number_2, …, number 30** are numerical values or ranges. None-numbers are ignored.

Syntax	*Description*
MAXA(value 1, value 2, …, value 30)	Returns the maximum value in a list of arguments. Unlike MAX, text and logical values can be entered. Text is evaluated as 0, logical True is treated as 1 and logical False as 0. **value 1, value 2, …, value 30** are values or ranges.
MEDIAN(number 1, number 2, …, number 30)	Returns the median of a set of numbers. **number 1, number 2, …, number 30** are values or ranges, which represent a sample. Each number can also be replaced by a reference. MEDIAN logically ranks the numbers (lowest to highest). If given an odd number of values, MEDIAN returns the middle value. If given an even number of values, MEDIAN returns the arithmetic average of the two middle values.
MIN(number 1, number 2, …, number 30)	Returns the minimum value in a list of arguments. **number 1, number 2, …, number 30** are numerical values or ranges.
MINA(value 1, value 2, …, value 30)	Returns the minimum value in a list of arguments. Text and logical values are evaluated. Text is evaluated as 0, logical True is treated as 1 and logical False as 0. **value 1, value 2, …, value 30** are values or ranges.
MODE(number 1, number 2, …, number 30)	Returns the most common value in a data set. **number 1, number 2, …, number 30** are numerical values or ranges. If several values have the same frequency, it returns the smallest value. An error occurs if a value does not appear more than once.
NEGBINOMDIST(X, R, SP)	Returns the negative binomial distribution. **X** is the value returned for unsuccessful tests. **R** is the value returned for successful tests. **SP** is the probability of the success of an attempt. NEGBINOMDIST returns the probability that there will be x failures before the r-th success, when the constant probability of a success is sp.
NORMDIST(Number, Mean, STDEV, C)	Returns the value of the probability density function or the cumulative distribution function for the normal distribution with the mean and standard deviation given. **Number** is the value for which the normal distribution is to be calculated. **Mean** is the mean value of the normal distribution. **STDEV** is the standard deviation of the normal distribution. **C** = 0 or FALSE it calculates the probability density function, and **C** = 1, TRUE or omitted, it calculates the cumulative distribution function.
NORMINV(number, mean, STDEV)	Returns the inverse of the normal distribution for the given probability value, **number**, in the distribution. **mean** is the mean value in the normal distribution. **STDEV** is the standard deviation of the normal distribution.
NORMSDIST(Number)	Returns the standard normal cumulative distribution for the given **Number**. This function is defined as LEGACY.NORMSDIST in the ODF v1.2 specification. This is exactly NORMDIST(x,0,1,TRUE()).

Syntax	*Description*
NORMSINV(number)	Returns the inverse of the standard normal distribution for the given probability value, **number**. number must be 0 < number < 1. This function is defined as LEGACY.NORMSINV in the ODF v1.2 specification.
PEARSON(Data_1, Data_2)	Returns the Pearson correlation coefficient, **r**, of two data sets. **Data_1** is the array of the first data set. **Data_2** is the array of the second data set. For an empty element or an element of type Text or Boolean in **Data_1** the element at the corresponding position of **Data_2** is ignored, and vice versa. Both arrays must be the same size and shape.
PERCENTILE(data, Alpha)	Returns the alpha-percentile of data values in an array. **data** is the array of data. **Alpha** is the percentile value between 0 and 1. If Alpha is not a multiple of 1/(n - 1), PERCENTILE interpolates to determine the value between two data points.
PERCENTRANK(data, value)	Returns the percentage rank (percentile) of the given **value** in a sample. **data** is the array of data in the sample.
PERMUT(Count_1, Count_2)	Returns the number of permutations for a given number of objects without repetition. **Count_1** is the total number of objects. **Count_2** is the number of objects in each permutation.
PERMUTATIONA(Count_1, Count_2)	Returns the number of permutations for a given number of objects (repetition allowed, meaning an object can combine with itself). **Count_1** is the total number of objects. **Count_2** is the number of objects in each permutation.
PHI(number)	Returns the values of the distribution function for a standard normal distribution for the given **number**. PHI(number) is a synonym for NORMDIST(number,0,1,FALSE()).
POISSON(Number, mean, Cumulative)	Returns the probability, or the cumulative distribution function for the Poisson distribution of **Number**. **mean** is the middle value of the Poisson distribution. **Cumulative** = 0 calculates the probability density function, and **Cumulative** = 1 calculates the distribution.
PROB(data, probability, Start, End)	Returns the probability that values in a range are between two limits. **data** is the array or range of data in the sample. **probability** is the array or range of the corresponding probabilities. **Start** is the start value of the interval whose probabilities are to be summed. **End** (optional) is the end value of the interval whose probabilities are to be summed. If this parameter is missing, then **End** = **Start** value is assumed.
QUARTILE(data, Type)	Returns the quartile of a data set. **data** is the array of data in the sample. **Type** is the number of the quartile to return. (0 = Min, 1 = 25%, 2 = 50% (Median), 3 = 75% and 4 = Max). Based on the statistical rank of the data points in **data**, QUARTILE returns the percentile value indicated by **Type**. The percentile is calculated as **Type** divided by 4. The same algorithm used in PERCENTILE is used here to interpolate between two data points.

Syntax	*Description*
RANK(value, Data, Type)	Returns the rank of the given **value** in a sample. **Data** is the array or range of data in the sample. **Type** (optional) is the ranking order, if omitted or 0 data is ranked in ascending order, if not 0 data is ranked in descending order.
RSQ(data_Y, data_X)	Returns the square of the Pearson product moment correlation coefficient based on the given values. **data_Y** is an array of data points. **data_X** is an array of data points. The arguments shall be either numbers or names, arrays, or references that contain numbers. If an array or reference argument contains Text, Logical values, or empty cells, those values are ignored; however, cells with the value zero are included. Both arrays must have the same size and shape.
SKEW(number 1, number 2, …, number 30)	Returns the skewness of a distribution. **number 1, number 2, …, number 30** are numerical values or ranges. There must be a minimum of three numbers.
SKEWP(number 1, number 2, …, number 30)	Calculates the skewness of a distribution using the population of a random variable. **number 1, number 2, …, number 30** are numerical values or ranges. There must be a minimum of three numbers.
SLOPE(data_Y, data_X)	Returns the slope of the linear regression line. **data_Y** is the array or matrix of Y data. **data_X** is the array or matrix of X data. Both arrays must have the same size and shape. For an empty element or an element of type Text or Boolean in y the element at the corresponding position of x is ignored, and vice versa.
SMALL(data, Rank_c)	Returns the Rank_c-th smallest value in a data set. **data** is the cell range of data. **Rank_c** is the rank of the value (2nd smallest, 3rd smallest, etc.) written as an integer.
STANDARDIZE(Number, mean, STDEV)	Converts a random variable to a normalized value. **Number** is the value to be standardized. **mean** is the arithmetic mean of the distribution. **STDEV** is the standard deviation of the distribution.
STDEV(number 1, number 2, …, number 30)	Computes the sample standard deviation of a set of numbers. **number 1, number 2, …, number 30** are numerical values or ranges representing a sample based on an entire population.
STDEVA(value 1, value 2, …, value 30)	Calculates the standard deviation using a sample set of values, including values of type Text and Logical. **value 1, value 2, …, value 30** are values or ranges representing a sample derived from an entire population. Text has the value 0.
STDEVP(number 1, number 2, …, number 30)	Calculates the standard deviation using the population of a random variable, including values of type Text and Logical. **number 1, number 2, …, number 30** are numerical values or ranges representing a sample based on an entire population.

Syntax	*Description*
STDEVPA(value 1, value 2, …, value 30)	Calculates the standard deviation based on the entire population. **value 1, value 2, …, value 30** are values or ranges representing a sample derived from an entire population. Text has the value 0. Logical FALSE is 0 and logical TRUE is 1.
STEYX(data_Y, data_X)	Returns the standard error of the predicted y value for each x in the regression. **data_Y** is the array or matrix of Y data. **data_X** is the array or matrix of X data. Both arrays must have the same size and shape and contain at least three numbers.
TDIST(Number, degrees_freedom, mode)	Returns the t-distribution for the given **Number**. **degrees_freedom** is the number of degrees of freedom for the t-distribution. **mode** = 1 returns the one-tailed test, **mode** = 2 returns the two-tailed test. This function is named LEGACY.TDIST in the ODF v1.2 specification.
TINV(number, degrees_freedom)	Returns the inverse of the t-distribution, for the given **number** associated with the two-tailed t-distribution. **degrees_freedom** is the number of degrees of freedom for the t-distribution.
TRIMMEAN(data, Alpha)	Returns the mean of a data set, ignoring a proportion of high and low values. **data** (required) is the array of data in the sample. **Alpha** (required) is the fractional number of data points to exclude from the calculation. For example, if **Alpha** = 0.2, 4 points are trimmed from a data set of 20 points (20 x 0.2): 2 from the top and 2 from the bottom of the set.
TTEST(data_1, data_2, mode, Type)	Returns the probability associated with a Student's t-Test. **data_1** is the dependent array or range of data for the first record. **data_2** is the dependent array or range of data for the second record. **mode** = 1 calculates the one-tailed distribution, **mode** = 2 the two- tailed distribution. **Type** of t-test to perform: paired (1), equal variance (homoscedastic) (2), or unequal variance (heteroscedastic) (3).
VAR(number 1, number 2, …, number 30)	Calculates the variance based on a sample. **number 1, number 2, …, number 30** are numerical values or ranges representing a sample based on an entire population. Requires at least two numbers.
VARA(value 1, value 2, …, value 30)	Estimates a variance based on a sample. **value 1, value 2, …, value 30** are values or ranges representing a sample derived from an entire population. Text is evaluated as 0. Logical TRUE is evaluated as 1 and FALSE as 0.
VARP(number 1, number 2, …, number 30)	Calculates a variance based on the entire population. **number 1, number 2, …, number 30** are numerical values or ranges representing an entire population.
VARPA(value 1, value 2, …, value 30)	Calculates the variance based on the entire population. The value of text is 0. **value 1, value 2, …, value 30** are values or ranges representing an entire population. Text is evaluated as 0. Logical TRUE is evaluated as 1 and FALSE as 0.

Syntax	Description
WEIBULL(Number, Alpha, beta, C)	Returns the values of the Weibull distribution at the given **Number**. **Alpha** is the alpha parameter of the Weibull distribution. **beta** is the beta parameter of the Weibull distribution. **C** indicates the type of function: C= 0 the probability density function is calculated, C=1 the cumulative distribution function is calculated.
ZTEST(data, mu, sigma)	Returns the two-tailed P value of a z test with standard distribution. **data** is the array of the data. **mu** is the value to be tested. **sigma** (optional) is the standard deviation of the total population. If this argument is missing, the standard deviation of the sample is processed.

Date and time functions

Use these functions for inserting, editing, and manipulating dates and times. LibreOffice handles and computes a date/time value as a number. When you assign the number format “Number” to a date or time value, it is displayed as a number. For example, 01/01/2000 12:00 PM, converts to 36526.5. This is just a matter of formatting; the actual value is always stored and manipulated as a number. To see the date or time displayed in a standard format, change the number format (date or time) accordingly.

To set the default date format used by Calc, go to **Tools > Options > LibreOffice Calc > Calculate.**

Caution

When entering dates, slashes or dashes used as date separators may be interpreted as arithmetic operators. To keep dates from being interpreted as parts of formulas, and thus returning erroneous results, always place them in quotation marks, for example, "12/08/52". See also A note about dates on page 10.

Table 5: Data and time functions

Syntax	Description
DATE(year, month, day)	Converts a date written as year, month, day to an internal serial number and displays it in the cell’s formatting. **year** is an integer between 1583 and 9956 or 0 and 99. **month** is an integer between 1 and 12. **day** is an integer between 1 and 31.
DATEDIF(Start date, End date, Interval)	Returns the difference in years, months, or days of two date numbers, **Start date** and **End date**. **Interval** is entered as “y”, “m” or “d”, to return the value in years, months or days or as “ym”, “md” or “yd” for months ignoring the years value; days ignoring the months and years values; or days ignoring the months and years values. **Start date** and **End date** must be entered using double quotes.
DATEVALUE(text)	Returns the date serial number for text in double quotes using the current locale. **text** is a valid date expression.
DAY(Number)	Returns the day, as an integer, of the given date value. **Number** is the date serial number (a negative date/time value can be entered) or a date value entered in double quotes.

Syntax	*Description*
DAYS(Date_2, Date_1)	Calculates the difference, in days, between two date values. **Date_1** is the start date. **Date_2** is the end date. If **Date_2** is an earlier date than **Date_1**, the result is a negative number. Dates can be entered as numbers or text.
DAYS360(Date_1, Date_2, Type)	Returns the difference between two dates based on the 360 day year used in interest calculations. If **Date_2** is earlier than **Date_1**, the function will return a negative number. **Type** (optional) determines the type of difference calculation: the US method (0) or the European method (≠0). Dates can be entered as numbers or text.
DAYSINMONTH(Date)	Calculates the number of days in the month of the given **Date**. Date can be entered as a number or text.
DAYSINYEAR(Date)	Calculates the number of days in the year of the given **Date**. Date can be entered as a number or text.
EASTERSUNDAY(year)	Returns the date of Easter Sunday for the entered **year**. **year** is an **integer** between 1583 and 9956 or 0 and 99 (19xx or 20xx depending on the option set)..
EDATE(Start date, Months)	Returns the serial number of the date a number of **Months** away from the given **Start date**. Only months are considered; days are not used for calculation. **Months** is the number of months before (negative) or after (positive) the start date. **Start date** may be entered as text or a number.
EOMONTH(Start date, Months)	Returns the serial number date of the last day of a month which falls **Months** away from the given **Start date**. **Months** is the number of months before (negative) or after (positive) the start date. **Start date** may be entered as text or a number.
HOUR(Number)	Returns the hour, as an integer, for the given time value. **Number** is a time value and can be either text or a number.
ISLEAPYEAR(Date)	Determines whether a given **Date** falls within a leap year. Returns either 1 (TRUE) or 0 (FALSE). **Date** must be a full date for text, a reference to a date value or a serial number.
MINUTE(Number)	Returns the minute, as an integer, for the given time value. **Number** is a time value.
MONTH(Number)	Returns the month, as an integer, for the given date value. **Number** is a time value.
MONTHS(Start date, End date, Type)	Calculates the difference, in months, between two date values. **Start date** is the start (earlier) date. **End date** is the end date. **Type** determines the type of calculation and is one of two possible values; 1 returns the difference between the calendar month values in the two dates, disregarding the day values; 0 returns the number of months that separate the dates taking into account the day values of the two dates. If **End date** is an earlier date than **Start date**, the result is a negative number.

Syntax	*Description*
NETWORKDAYS(Start date, End date, Holidays)	Returns the number of workdays between **Start date** and **End date**. Holidays can be deducted. **Start date** is the date from which the calculation is carried out. **End date** is the date up to which the calculation is carried out. If the start or end date is a workday, the day is included in the calculation. **Holidays** (optional) is a list of holidays. Enter a cell range in which the holidays are listed individually. Saturdays and Sundays are considered non-workdays.
NOW()	Returns the computer system date and time. The value is updated when your document recalculates. NOW() is a function without arguments.
SECOND(Number)	Returns the second, as an integer, for the given time value. **Number** is a time value.
TIME(hour, minute, second)	Returns the time value from values for hours, minutes and seconds. This function can be used to convert a time based on these three elements to a decimal time value. **hour**, **minute** and **second** must all be integers.
TIMEVALUE(text)	Returns the time serial number value from **text** enclosed by quotes in a time entry format. The value of the decimal number returned is the result of the date system used under LibreOffice to calculate date entries.
TODAY()	Returns the current computer system date. The value is updated when your document recalculates. TODAY() is a function without arguments.
WEEKDAY(Number, Type)	Returns the day of the week for the given **Number** (date value). The day is returned as an integer based on the type. **Type** determines the type of calculation: **Type** = 1 (assumed if **Type** is omitted), the weekdays are counted (1-7) starting from Sunday (Monday = 2); **Type** = 2, the weekdays are counted (1-7) starting from Monday (Monday = 1); **Type** = 3, the weekdays are counted (0-6) starting from Monday (Monday = 0).
WEEKNUM(Number, mode)	Calculates the number of the calendar week of the year for the given date **Number**. **mode** sets the start of the week and the calculation type: 1 = Sunday, any other value = Monday.
WEEKNUM_ADD(Date, Return type)	Calculates the calendar week of the year for a **Date**. **Date** is the date within the calendar week. **Return type** sets the start of the week and the calculation type: 1 = Sunday, 2 = Monday. This function returns the same results as the WEEKNUM function in Excel.
WEEKS(Start date, End date, Type)	Calculates the difference in weeks between two dates, **Start date** and **End date**. **Type** is one of two possible values, 0 (number of whole weeks in the interval) or 1 (returns the number of different weeks in which the two dates appear). This function uses the ISO weeknumber.
WEEKSINYEAR(Date)	Calculates the number of weeks in a year for a given **Date**. A week that spans two years is added to the year in which most days of that week occur (so any week containing four or more days in the calendar year of **Date** is counted).

Syntax	Description
WORKDAY(Start date, Days, Holidays)	Returns a date serial number which is a specified number of work days (**Days**) before or after an input date, **Start date**. **Holidays** (optional) is a list of holidays. Enter a cell range in which the holidays are listed individually. Work days exclude Saturdays and Sundays. This function does not fully implement the ODFv1.2 specification which allows you to set the non-work days.
YEAR(Number)	Returns the calendar year as an integer according to the internal calculation rules. **Number** is the date value in date serial number format or as a text date, for which the year is to be returned.
YEARFRAC(Start date, End date, Basis)	Extracts the number of years (including fractional part) between two date values, **Start date** and **End date**. **Basis** is a value either omitted or between 0 and 4, chosen from a list of options and indicates how the year is to be calculated (see Help files). If omitted it is evaluated as 0.
YEARS(Start date, End date, Type)	Calculates the difference in years between two dates: the **Start date** and the **End date**. **Type** calculates the type of difference. Possible values are 0 (interval) and 1 (in calendar years).

Logical functions

Use the logical functions to test values and produce results based on the result of the test. These functions are conditional and provide the ability to write longer formulas based on input or output.

Table 6: Logical functions

Syntax	Description
AND(Logical value 1, Logical value 2, …, Logical value 30)	Returns TRUE if all arguments are TRUE. If any element is FALSE, this function returns the FALSE value. **Logical value 1, Logical value 2, …, Logical value 30** are conditions to be checked. All conditions can be either TRUE or FALSE. If a range is entered as a parameter, only logical values in the range are evaluated. The result is TRUE if the logical value in all cells within the cell range is TRUE. Bug 70632: Concerning range statement given in Help.
FALSE()	Set the logical value to FALSE. The FALSE() function does not require any arguments.
IF(Test; Then_value, Otherwise_value)	Specifies a logical test to be performed. **Test** is any value or expression that can be TRUE or FALSE. **Then_value** (optional) is the value that is returned if the logical test is TRUE. **Otherwise_value** (optional) is the value that is returned if the logical test is FALSE.
IFERROR(value, alternative value)	Evaluates **value;** if it is not an error it returns the result for **value**, or else it returns the **alternative value**. If **value** evaluates to a logical value, then either 1 (for TRUE), or 0 (for FALSE) is returned.

Syntax	Description
IFNA(value, alternative value)	Evaluates **value;** if it is not a #N/A error it returns the result for **value**, or else it returns the **alternative value**. If **value** evaluates to a logical value, then either 1 (for TRUE), or 0 (for FALSE) is returned.
NOT(Logical value)	Reverses the logical value. **Logical value** is the TRUE or FALSE value to be reversed.
OR(Logical value 1, Logical value 2, …, Logical value 30)	Returns TRUE if at least one argument is TRUE. Returns the value FALSE if all the arguments have the logical value FALSE. **Logical value 1, Logical value 2, …, Logical value 30** are conditions to be checked. All conditions can be either TRUE or FALSE.
TRUE()	Sets the logical value to TRUE. The TRUE() function does not require any arguments.
XOR(Logical value 1, Logical value 2, …, Logical value 30)	Computes the logical XOR of the parameters. If an even number of parameters is TRUE it returns FALSE, if an odd number of parameters is TRUE it returns TRUE.

Information functions

These functions provide information (or feedback) regarding the results of a test for a specific condition, or a test for the type of data or content a cell contains.

Table 7: Informational functions

Syntax	Description
CELL(info_type, Reference)	Returns information on a cell such as its address, formatting or contents of a cell based on the value of the info_type argument. **info_type** specifies the type of information to be returned and comes from a predefined list of arguments. See Help files for complete listing. **info_type** is not case sensitive, but it must be enclosed within quotes. **Reference** is the address of the cell to be examined. If **Reference** is a range, the cell reference moves to the top left of the range. If **Reference** is missing, Calc uses the position of the cell in which this formula is located.
CURRENT()	Calculates the current value of a formula at the actual position.
FORMULA(Reference)	Displays in the current location, the formula contained in a cell cell at **Reference** position. If no formula at **Reference** can be found, or if the presented argument is not a reference, returns the error value #N/A.
INFO(Text)	Returns information about the working environment. **Text** is a string constant entered in double quotes taken from a list of arguments. See the Help files for the listing.
ISBLANK(value)	Returns TRUE if the referenced cell is blank, else returns FALSE. If **value** is of type Number, Text, or Logical, return FALSE. If **value** is a reference to a cell, examine the cell; if it is blank (has no value), return TRUE, but if it has a value, return FALSE. A cell with the empty string is not considered blank.

Syntax	*Description*
ISERR(value)	Returns TRUE if the value refers to any error value except #N/A. You can use this function to control error values in certain cells. If an error occurs, the function returns a logical or numerical value. **value** is any value or expression in which a test is performed to determine whether an error value not equal to #N/A is present.
ISERROR(value)	The ISERROR tests if the cells contain general error values. ISERROR recognizes the #N/A error value. If an error occurs, the function returns a logical or numerical value. **value** is any value where a test is performed to determine whether it is an error value.
ISEVEN(value)	Returns TRUE if the given **value** is an even integer, or FALSE if the **value** is odd. If the **value** is not an integer, the function evaluates only the integer part of the value.
ISEVEN_ADD(Number)	Tests for even numbers. Returns TRUE (1) if the integer part of **Number** returns a whole number when divided by 2.
ISFORMULA(reference)	Returns TRUE if a cell is a formula cell. If an error occurs, the function returns a logical or numerical value. **reference** indicates the reference to a cell in which the test will be performed.
ISLOGICAL(value)	Returns TRUE if the cell contains a logical number format. The function is used in order to check for both TRUE and FALSE values in certain cells. If an error occurs, the function returns a logical or numerical value. **value** is the cell reference to be tested for logical number format.
ISNA(value)	Returns TRUE if **value** contains the #N/A (value not available) error value. If an error occurs, the function returns a logical or numerical value. **value** is the cell, value or expression to be tested.
ISNONTEXT(value)	Return TRUE if the parameter does not have type Text, else return FALSE. If an error occurs, the function returns a logical or numerical value. **value** is any value or expression where a test is performed to determine whether it is a text or numbers or a Boolean value. Empty cells are considered non-text and will return TRUE.
ISNUMBER(value)	Returns TRUE if **value** evaluates to a number. If an error occurs, the function returns a logical or numerical value. **value** is any expression to be tested to determine whether it is a number or text. TRUE (1) and FALSE (0) are evaluated as numbers.
ISODD(value)	Returns TRUE if **value** evaluates as an odd integer, else FALSE. **value** is truncated to an integer before evaluation. TRUE (1) and FALSE (0) are evaluated as numbers. Text returns an error. Zero is evaluated FALSE.
ISODD_ADD(Number)	Returns 1 if **Number** does not return a whole number when divided by 2, else 0. **Number** is the number to be tested. Does not return logical type TRUE/FALSE like ISODD; returns number.

Syntax	*Description*
ISREF(value)	Returns TRUE if **value** is of type reference (including a reference list), else return FALSE. If an error occurs, the function returns a logical or numerical value. It does not evaluate the content of the reference.
ISTEXT(value)	Returns TRUE if **value** is of type text, else FALSE. If an error occurs, the function returns a logical or numerical value. **Value** is a value, number, Boolean value, or error value to be tested. If **value** is a reference, the content of the reference is evaluated.
N(value)	Return the number of **value**. If **value** is a reference the reference content is evaluated. If **value** is a logical value, 1 is returned for TRUE, else 0. If **value** is an error it is returned. Text returns a 0.
NA()	Returns the error value #N/A.
TYPE(value)	Evaluates **value** and returns a number indicating its type. If an error occurs, the function returns the error. The numerical value from which the data type is determined is; 1 = number, 2 = text, 4 = Boolean value, 8 = formula, 16 = error value. If **value** references an empty cell, an error is returned. The results of a formula in a reference are not evaluated.

Database functions

This section deals with functions used with data organized as one row of data for one record. The *Database* category should not be confused with the Base database component in LibreOffice. A Calc database is simply a range of cells that comprises a block of related data where each row contains a separate record. There is no connection between a database in LibreOffice and the *Database* category in LibreOffice Calc.

The database functions use the following common arguments:

- **Database** is a range of cells which define the database.
- **Database field** specifies the column which the function operates on after the search criteria of the first parameter is applied and the data rows are selected. It is not related to the search criteria itself. The number 0 specifies the whole data range. To reference a column by using the column header name, place quotation marks around the header name.
- **Search criteria** is a cell range containing the search criteria.. Empty cells in the search criteria range will be ignored.

Note	All of the **search criteria** arguments for the database functions support regular expressions. For example, “all.*” can be entered to find the first location of “all” followed by any characters. To search for text that is also a regular expression, precede every character with a \ character. You can switch the automatic evaluation of regular expressions on and off in **Tools > Options > LibreOffice Calc > Calculate**.

Table 8: Database average

Syntax	***Description***
DAVERAGE(Database, Database field, Search criteria)	Returns the average of the values in a given database field from the records (rows) in a database that match the search criteria. **Database field** cannot be 0 or empty.
DCOUNT(Database, Database field, Search criteria)	Counts the number of records (rows) in a database that match the search criteria and contain numerical values. **Database field** can be empty or 0.
DCOUNTA(Database, Database field, Search criteria)	Counts the number of rows (records) in a database that match the specified search criteria and contain numeric or alphanumeric values. **Database field** can be empty or 0.
DGET(Database, Database field, Search criteria)	Returns the field value from a record in a database, which matches the search criteria. The search criteria must return a single value. In case of an error, the function returns either #VALUE! for no record or field values found, or Err502 for more than one cell in the search criteria.
DMAX(Database, Database field, Search criteria)	Returns the maximum value of a field in a database (all records) that matches the specified **Search criteria**. The search supports regular expressions.
DMIN(Database, Database field, Search criteria)	Returns the minimum value of a field in a database that matches the specified **Search criteria**. The search supports regular expressions.
DPRODUCT(Database, Database field, Search criteria)	Multiplies all cells of a data range where the cell contents match the **Search criteria**. The search supports regular expressions.
DSTDEV(Database, Database field, Search criteria)	Finds the sample standard deviation in a given field from the records (rows) in a database that match a search criteria.
DSTDEVP(Database, Database field, Search criteria)	Finds the population standard deviation in a given field from the records (rows) in a database that match a search criteria.
DSUM(Database, Database field, Search criteria)	Finds the sum of values in a given field from the records (rows) in a database that match a search criteria. The search supports regular expressions.
DVAR(Database, Database field, Search criteria)	Finds the sample variance in a given field from the records (rows) in a database that match a search criteria.
DVARP(Database, Database field, Search criteria)	Finds the population variance in a given field from the records (rows) in a database that match a search criteria.

Array functions

When using the Function Wizard for Array functions, those returning an array result have the Array check-box automatically selected.

Table 9: Array functions

Syntax	***Description***
FREQUENCY(data, classes)	Categorizes values into intervals and counts the number of values in each interval. Returns the results as a vertical array containing one more result than the number of classes. **data** is the data that should be categorized and counted according to the given intervals. **classes** is the array containing the upper boundaries determining the intervals the values in data should be grouped by.
GROWTH(data_Y, data_X, new_data_X, Function_type)	Calculates predicted exponential growth by using existing data. **data_Y** is the Y data array. **data_X** (optional) is the X data array. **new_data_X** (optional) is the X data array, for which the values are to be calculated. If **new_data_X** is omitted it is assumed to be the same size as **data_X**. If both arrays are omitted, they are assumed to be the array {1,2,3,...} that is the same size as the Y data array. **Function_type** is optional. If **Function_type** = 1 or omitted, functions in the form y = b*m^x are calculated, else y = m^x functions are calculated.
LINEST(data_Y, data_X, Linear_type, stats)	Returns the parameters of the (simple or multiple) linear regression equation for the given data and, optionally, statistics on this regression. The equation for the line is y = mx + c, or y = m1x1 + m2x2 + … + c for multiple ranges of x-values, where the dependent y-values are a function of the independent x-values. The m-values are coefficients corresponding to each x-value, and c is a constant value. **data_Y** is a single row or column range specifying the y coordinates in a set of data points. **data_X** (optional) is a corresponding single row or column range specifying the x coordinates. If **data_X** is omitted it defaults to {1,2,3,..., n}. If there is more than one set of variables **data_X** may be a range with corresponding multiple rows or columns. **Linear_type** (optional): if FALSE the straight line found is forced to pass through the origin (the constant c is zero; y = mx). If omitted, **Linear_type** defaults to TRUE (the line is not forced through the origin). **stats** (optional): If **stats** = 0, only the regression coefficient is calculated. Otherwise, other statistics will be returned, see the Help file for full information.
LOGEST(data_Y, data_X, Function_type, stats)	Calculates the adjustment of the entered data as an exponential regression curve (y=b*m^x). **data_Y** is the Y Data array. **data_X** (optional) is the X data array. **Function_type** (optional): If **Function_type** = 0, functions in the form y = m^x are calculated. Otherwise, y = b*m^x functions are calculated. **stats** (optional). If **stats** = 0, only the regression coefficient is calculated; if **stats** = 1 other statistics will be returned, see the Help file for full information.
MDETERM(array)	Returns the determinant of a square array. This function returns a value in the current cell; it is not necessary to define a range for the results. **array** is an array in which the determinants are defined. The Array check-box is not automatically selected.

Syntax	*Description*
MINVERSE(array)	Returns the inverse array. **array** is a square array that is to be inverted.
MMULT(array, array)	Calculates the array product of two arrays. The number of columns for array 1 must equal the number of rows for array 2. **array** at first place is the first array used in the array product. **array** at second place is the second array with the same number of rows as the first array has columns. Bug 71128: Same name for two variables.
MUNIT(Dimensions)	Returns the unitary square array of a certain size. The unitary array is a square array where the main diagonal (top left to bottom right) elements are set to 1 and all other array elements are set to 0. **Dimensions** refers to the column and row size of the array.
SUMPRODUCT(Array 1, Array 2, …, Array 30)	Multiplies corresponding elements in the given arrays, and returns the sum of those products. **Array 1, Array 2, …, Array 30** are arrays whose corresponding elements are to be multiplied. At least one array must be part of the argument list. If only one array is given, the array elements are summed. Arrays must have the same size and shape. Non numeric elements are treated as 0. The Array check-box is not automatically selected.
SUMX2MY2(array_x, array_y)	Returns the sum of the difference of the squares of corresponding values in two arrays. **array_x** is the first array whose elements are to be squared and added. **array_y** is the second array whose elements are to be squared and subtracted. Arrays must have the same size and shape. The Array check-box is not automatically selected.
SUMX2PY2(array_x, array_y)	Returns the sum of the sum of the squares of the individual values in each array. **array_x** is the first array whose arguments are to be squared and summed. **array_y** is the second array, whose arguments are to be squared and summed and then summed with the result from the first array. Arrays must have the same size and shape. The Array check-box is not automatically selected.
SUMXMY2(array_x, array_y)	Adds the squares of the difference between corresponding values in two arrays. **array_x** is the first array from whose elements the corresponding elements of **array_y** are to be subtracted. The results of each subtraction are summed and the results squared. Arrays must have the same size and shape. The Array check-box is not automatically selected.
TRANSPOSE(array)	Transposes the rows and columns of an array. **array** is the array in the spreadsheet that is to be transposed.
TREND(data_Y, data_X, new_data_X, Linear_type)	Returns values along a linear trend. **data_Y** is the Y data array. **data_X** (optional) is the X data array. **new_data_X** (optional) is the array of the X data, which are used for recalculating values. If **new_data_X** is omitted it is assumed to be the same size as **data_X**. If both arrays are omitted, they are assumed to be the array {1,2,3,...} that is the same size as the Y data array. **Linear_type is** optional. If **Linear_type** = 1 or omitted, functions in the form y = mx + c are calculated, else y = mx functions are calculated.

Spreadsheet functions

Use spreadsheet functions to search and address cell ranges and provide feedback regarding the contents of a cell or range of cells. You can use functions such as HYPERLINK() and DDE() to connect to other documents or data sources.

Table 10: Spreadsheet functions

Syntax	***Description***
ADDRESS(row, column, ABS, A1, sheet)	Returns a cell address (reference) as text, according to the specified row and column numbers. Optionally, whether the address is interpreted as an absolute address (for example, A1) or as a relative address (as A1) or in a mixed form (A$1 or $A1) can be determined. The name of the sheet can also be specified. **row** (required) is the row number for the cell reference. **column** (required) is the column number for the cell reference (the number, not the letter). **ABS** (optional) determines the type of reference and is a value between 1 and 4. See the Help files for explanation of list numbers. Optional **A1** if set to 0 uses the R1C1 notation, else it uses the A1 notation. Optional **sheet** is the name of the sheet entered in double quotes. If using R1C1 notation, ADDRESS returns address strings using the exclamation mark '!' as the sheet name separator. The function still uses the dot '.' sheet name separator with A1 notation. When opening documents from ODF 1.0/1.1 format, the ADDRESS functions that show a sheet name as the fourth parameter will shift that sheet name to become the fifth parameter. A new fourth parameter with the value 1 will be inserted. When saving a document in ODF 1.0/1.1 format, if the ADDRESS function has a fourth parameter, that parameter will be removed. A spreadsheet should not be saved in the old ODF 1.0/1.1 format if **A1** is set to 0.
AREAS(reference)	Returns the number of individual ranges that belong to a multiple range. A range can consist of contiguous cells or a single cell. **reference** is a reference list of the ranges. The function expects a single argument. Multiple ranges can be entered using the tilde (~) (Union) operator or a semicolon (;) as the divider, but the semicolon gets automatically converted to the tilde operator after the function is entered into the spreadsheet. If you state multiple ranges and you use the semicolon separator, you must enclose them in additional parentheses. The tilde is the union range operator. See Chapter 7 for range operators. Multiple ranges can be entered into the **reference** input box in two ways. Firstly they can be be typed directly into the argument's input box, noting the parentheses constraint mentioned above for the semicolon. Secondly, by clicking the **Shrink** button to the right of the input box and then clicking and dragging in the sheet to select cell ranges. Add the range operator between selections. Note the parentheses constraint above for use of the semicolon. Bug 71225 concerning problems inputting data.

Syntax	*Description*
CHOOSE(Index, value1, …, value30)	Returns a value from a list of up to 30 values. **Index** is a reference or number between 1 and 30 indicating which value is to be taken from the list. **value1, …, value30** is the list of values entered as any number type, reference, or formula expression. Only the selected value from the list is evaluated, any other formulas in the list are not checked for validity.
COLUMN(reference)	Returns the column number of a **reference**. If the reference is a single cell, the column number of the cell is returned; if the parameter is a cell range containing more than one column, the corresponding column numbers are returned in a single-row array, if the formula is entered as an array formula. If the cell range is not entered as an array formula, only the column number of the first cell within the range is determined. If no reference is entered, the column number of the cell in which the formula is entered is returned as Calc automatically sets the reference to the current cell.
COLUMNS(array)	Returns the number of columns in the given reference. **array** is the reference to a cell range whose total number of columns is to be found. The argument can also be a single cell.
DDE(server, File, range, mode)	Dynamic Data Exchange. Returns the result of a DDE request. If the contents of the linked range or section changes, the returned value will also change. The spreadsheet can be reloaded, or **Edit > Links** selected, to see the updated links. Cross-platform links, for example from an LibreOffice installation running on a Windows machine to a document created on a Linux machine, are not supported. **server** is the name of a server application. LibreOffice applications have the server name “Soffice”. **File** is the complete file name, including path. **range** is the area containing the data to be evaluated. **mode** is an optional parameter that controls the method by which the DDE server converts its data into numbers. See the Help files for information on choices. An earlier bug that caused this function to crash LibreOffice has been fixed in v4.1.4 and later releases.
ERRORTYPE(reference)	Evaluates the cell value at **reference** location. If the cell contains an error then a logical or numerical value is returned else it returns #N/A. The numerical value is the error number (see Help for full listing). For a cell containing the #N/A error, a value of 32767 is returned.
GETPIVOTDATA(Data Field, Pivot Table, Field Name/Item1, Field Name/Item2, …, Field Name/Item30) This is the syntax used in the Function Wizard.	The GETPIVOTDATA function returns a calculated result value from a pivot table. The value is addressed using field and item names, so it remains valid if the layout of the pivot table changes. Two different syntax definitions can be used: the syntax shown on the left and GETPIVOTDATA(Pivot Table, Constraints) For syntax 1; **Data Field** is a string that selects one of the pivot table's data fields. The string can be the name of the

Syntax	*Description*
	source column, or the data field name as shown in the table (like "Sum – Sales"). **Pivot Table** is a reference to a cell or cell range that is positioned within a pivot table or contains a pivot table. If the cell range contains several pivot tables, the table that was created last is used. If no **Field Name /ItemX** pairs are given, the grand total is returned. Otherwise, each pair adds a constraint that the result must satisfy. **Field Name** is the name of a field from the pivot table. **ItemX** is the name of an item from that field. A maximum of 30 **Field Name/ItemX** pairs can be entered. The second syntax is assumed if exactly two parameters are given, **Pivot Table** has the same meaning as in the first syntax. **Constraints** is a space-separated list. Entries can be quoted (single quotes). The whole string must be enclosed in quotes (double quotes), unless you reference the string from another cell. See the Help file for detailed information. In some versions of LibreOffice, the second syntax variation returns a #REF error. See Bug 71234.
HLOOKUP(search_criteria, array, Index, sorted)	Searches for a value given in **search_criteria** in the first row of the given **array**, and returns the value from the row given in **Index** for the column in which the search item was found. If **sorted** is 0 or FALSE the first row of array need not be sorted, else the first row of **array** must be sorted in alpha-numerical and logical order. The search supports regular expressions.
HYPERLINK(URL, CellText)	When the text in a cell that contains the HYPERLINK function is Ctrl-clicked (the cursor becomes a pointing hand when correctly positioned), the hyperlink opens. **URL** specifies the link target. The optional **CellText** argument is the text displayed in the cell. If either argument is a text string, it must be entered in double quotes. If the **CellText** parameter is not specified, the **URL** text is displayed.
INDEX(reference, row, column, range)	Given a **reference**, returns the value at the given **row** and **column** intersection (starting numbering at 1, relative to the top left of the **reference**) of the given area **range**. If **range** is not given, it is assumed to be 1 (the first and possibly only area). If **row** is omitted or empty or 0, an entire column of the given area **range** in **reference** is returned. If **column** is omitted or empty or 0, an entire row of the given area **range** in **reference** is returned. If both, **row** and **column**, are omitted or empty or 0, the entire given area **range** is returned. If **reference** is a one-dimensional column vector, **column** is optional or can be omitted. If **reference** is a one-dimensional row vector, **row** is optional, which effectively makes **row** act as the column offset into the vector, or can be omitted. If **row** or **column** have a value greater than the dimension of the corresponding given area **range**, an Error is returned.

Syntax	*Description*
	The **Array** checkbox must be selected in this function unless **row** and **column** are both included. Bug 71325: Returns #VALUE error when optional arguments are omitted.
INDIRECT(ref, A1)	Returns a reference given a string representation of a reference as **ref**. This function can also be used to return the area of a corresponding string. **ref** is a reference to a cell or an area (in text form) from which to return the contents. Unless **ref** refers to a cell containing a reference, **ref** must be entered in double quotes. **A1** (optional) - if set to 0, the R1C1 notation is used. If this parameter is absent or set to another value than 0, the A1 notation is used.
LOOKUP(Search criterion, Search vector, result_vector)	Returns the contents of a cell either from a one-row or one-column range or from an array. Optionally, the assigned value (of the same index) is returned in a different column and row. As opposed to VLOOKUP and HLOOKUP, search and result vectors may be at different positions; they do not have to be adjacent. Additionally, the search vector for the LOOKUP must be sorted ascending, otherwise the search will not return any usable results. The search supports regular expressions. **Search criterion** is the value to be searched for; entered either directly or as a reference. **Search vector** is the single-row or single-column area to be searched. **result_vector** is another single-row or single-column range from which the result of the function is taken. The result is the cell of the result vector with the same index as the instance found in the search vector. When given two parameters, **Search vector** is first examined: If **Search vector** is square or is taller than it is wide (more rows than columns), LOOKUP searches in the first column (similar to VLOOKUP), and returns the corresponding value in the last column. If **Search vector** covers an area that is wider than it is tall (more columns than rows), LOOKUP searches in the first row (similar to HLOOKUP), and returns the corresponding value in the last row. Bug 71589: This fails if an alphabetic character is used for the search criterion.
MATCH(Search criterion, lookup_array, Type)	Returns the relative position of an item in an array that matches a specified value. The function returns the position of the value found in **lookup_array** as a number. **Search criterion** is the value which is to be searched for. **lookup_array** is the vector to be searched. A lookup array can be a single row or column, or part of a single row or column. **Type** may take the values 1, 0, –1 or be omitted. If **Type** is value 1 or omitted, **lookup_array** must be sorted ascending and the function finds the largest value that is less than or equal to **Search criterion**. If **Type** is of value 0 the function finds the largest value that is less than or equal to **Search criterion**. Values in **lookup_array** do not need to be sorted. If **Type** is of value -1, the function returns the smallest value that is greater than or equal to **Search criterion** in a

Syntax	*Description*
	lookup_array where values are sorted in descending order. The search supports regular expressions.
OFFSET(reference, rows, columns, height, width)	Returns the value of a cell offset by a certain number of rows and columns from a given reference point. **reference** is the cell from which the function searches for the new reference. **rows** is the number of cells by which the reference was corrected up (negative value) or down. **columns** is the number of columns by which the reference was corrected to the left (negative value) or to the right. **height** is the optional vertical height for an area that starts at the new reference position. **width** is the optional horizontal width for an area that starts at the new reference position.
ROW(reference)	Returns the row number of a cell reference. If the reference is a cell, it returns the row number of the cell. If the reference is a cell range, it returns the corresponding row numbers in a one-column array if the formula is entered as an array formula. If the ROW function with a range reference is not used in an array formula, only the row number of the first range cell will be returned. **reference** is a cell, an area, or the name of an area. If a reference is not indicated, Calc automatically sets the reference to the current cell.
ROWS(array)	Returns the number of rows in a reference or array. **array** is the reference or named area whose total number of rows is to be determined.
SHEET(reference)	Returns the sheet number of a reference or a string representing a sheet name. If no parameters are entered, the result is the sheet number of the spreadsheet containing the formula. **reference** (optional) is the reference to a cell, an area, or a sheet name string.
SHEETS(reference)	Determines the number of sheets in a reference. If no parameters are entered, the result is the number of sheets in the current document. **reference** (optional) is the reference to a sheet or an area.
STYLE(Style, Time, Style2)	Applies a style **Style** to the cell containing the formula for a length of time **Time**, after which the final style **Style2** is applied. Styles are listed (and may be created) in the **Format > Styles and Formatting** (**F11**) menu and are text entries entered in double quotes. The initial style is applied for **Time** seconds after the cell itself is recalculated. Please note that a manual recalculation (**F9** key or **Tools > Cell Contents > Recalculate**) will not trigger the initial style. **Time** and **Style2** may together be omitted; **Style** is then applied permanently. This function always returns the value 0, allowing it to be added to another function without changing the value.

Syntax	Description
VLOOKUP(Search criterion, array, Index, sort order)	Searches the first column of an **array** for the value given by **Search criterion** and if found returns the cell value at the intersection of the row in which it is found and the column index given by **Index**. The search supports regular expressions. **Search criterion** is the value searched for in the first column of the array. If text, it must be entered in double quotes. **array** is the reference, which must include at least two columns. **Index** is the number of the column in the array that contains the value to be returned. The first column has the number 1. If the **sort order** parameter is omitted or set to TRUE or not 0, it is assumed that the data is sorted in ascending order. If the exact **Search criterion** is not found, the last value that is smaller than the criterion will be returned. If the **sort order** parameter is set to FALSE or zero, an exact match must be found, otherwise the error Error: Value Not Available will be the result. Thus with a value of zero the data does not need to be sorted in ascending order.

Text functions

Use Calc's text functions to search and manipulate text strings or character codes.

Table 11: Text functions

Syntax	Description
ARABIC(Text)	Calculates the value of a Roman numeral. The value range must be between 0 and 3999 ("MMMIM"). **Text** is the text that represents a Roman numeral. It is not case sensitive and is entered in double quotes.
ASC(text)	The ASC function converts full-width to half-width ASCII and katakana characters. Returns a text string. **text** is the text that contains characters to be converted.
BAHTTEXT(Number)	Converts a number to Thai text, including the Thai currency names. **Number** is any number. "Baht" is appended to the integral part of the number, and "Satang" is appended to the decimal part of the number.
BASE(number, radix, Minimum length)	Converts a positive integer to a specified base then into text using the characters from the base's numbering system (decimal, binary, hexadecimal, etc.). Only the digits 0-9 and the letters A-Z are used. **number** is the positive integer to be converted. **radix** is the base of the number system. It may be any positive integer between 2 and 36. **Minimum length** (optional) is the minimum length of the character sequence that has been created. If the text is shorter than the indicated minimum length, zeros are added to the left of the string.
CHAR(number)	Converts a number into a character according to the current code table. The number can be a two-digit or three-digit integer number. **number** is a number between 1 and 255 representing the code value for the character.

Syntax	*Description*
CLEAN(text)	Removes all non-printing characters from the string entered into **text.** Text is entered using double quotes.
CODE(text)	Returns a numeric code for the first character in a text string. **text** is the text for which the code of the first character is to be found and is entered in double quotes.
CONCATENATE(text 1, text 2, …, text 30)	Combines several text strings into one string. **text 1, text 2, …, text 30** are text passages that are to be combined into one string.
DECIMAL(text, radix)	Converts text with characters from a number system to a positive integer in the decimal system. The radix value defines the number system to which the text belongs. Any characters not in the number system defined are ignored. **text** is the text to be converted and must be entered using double quotes. The **text** field is not case-sensitive. **radix** is the base of the number system from which the conversion is to take place. It may be any positive integer between 2 and 36.
DOLLAR(value, decimals)	Converts a number to text in the locale currency format, rounded to a specified decimal place. **value** is the number to be converted; it can be a number, a reference to a cell containing a number, or a formula which returns a number. **decimals** (optional) is the number of decimal places to be used. If no decimals value is specified, all numbers in currency format will be displayed with two decimal places. The currency format is set in the system settings.
EXACT(text_1, text_2)	Compares two text strings and returns TRUE if they are identical. This function is case-sensitive. **text_1** is the first text to compare. **text_2** is the second text to compare. Both arguments if entered directly must be in double quotes.
FIND(find_text, text, position)	Looks for a string of text within another string and returns the position in the searched text where the searched-for text begins. Where to begin the search can also be defined. The search term can be a number or any string of characters. The search is case-sensitive. **find_text** is the text to be found. **text** is the text which is being searched. **position** (optional) is the position in the text from which the search starts. Text must be entered in double quotes.
FIXED(number, Decimals, No thousands separator)	Returns a number, displayed as text, with a fixed number of decimal places and with or without a thousands separator. This function can be used to apply a uniform format to a column of numbers. **number** is the number to be formatted. **Decimals** is the number of decimal places to be displayed. If **Decimals** is negative, the number is rounded to ABS(**number**) **Decimals** places to the left from the decimal point. **No thousands separator** (optional) determines whether the thousands separator is used or not. If the parameter is equal to 0 or omitted, the thousands separators of the current locale setting are displayed, else the separators are suppressed.

Syntax	*Description*
JIS(text)	The JIS function converts half-width to full-width ASCII and katakana characters. Returns a text string. **text** is the text that contains characters to be converted. This is the complementary function to ASC.
LEFT(text, number)	Returns the number of characters from the left of a text string **text** determined by **number**. If this parameter is omitted, one character is returned. If **number** is greater than the length of the string, the whole string is returned.
LEN(text)	Returns the length of a string including spaces. **text** is the text whose length is to be determined.
LOWER(text)	Converts all uppercase letters in a text string to lowercase. **text** is the text to be converted.
MID(text, start, number)	Returns a text segment of a character string. The parameters specify the starting position and the number of characters to return. **text** is the text containing the characters from which to extract. **start** is the position marking the beginning of the text to extract. **number** is the number of characters from that point on to be returned. If **number** is greater than LEN(text) minus **start**, then the text from **start** to the end of **text** is returned.
NUMBERVALUE(text, decimal_separator, group_separator)	Convert text to number, in a locale-independent way. Converts given text value **text** into a number. If **text** is a reference, it is first dereferenced. **decimal_separator** and **group_separator** are optional parameters. If **text** contains a separator, then that separator must be entered into the relevant optional parameter. All parameters are entered in double quotes. Text is transformed according to the following rules: 1) Starting from the beginning, remove all occurrences of the **group_separator** before any **decimal_separator**. 2) Starting from the beginning, replace the first occurrence in the text of the **decimal_separator** character with the FULL STOP (U+002E) character. 3) Remove all whitespace characters (5.14). 4) If the first character of the resulting string is a period FULL STOP (U+002E) then prepend a zero. 5) If the string ends in one or more instances of PERCENT SIGN (U+0025), remove the percent sign(s). If percent signs were removed in step 5, divide the value of the returned number by 100 for each percent sign removed.
PROPER(text)	Capitalizes the first letter in all words of a text string. **text** is the text to be converted.

Syntax	*Description*
REPLACE(Text, position, length, new text)	Replaces part of a text string with a different text string. This function can be used to replace both characters and numbers (which are automatically converted to text). The result of the function is always displayed as text. To perform further calculations with a number which has been replaced by text, convert it back to a number using the VALUE function. Any text containing numbers must be enclosed in quotation marks so it is not interpreted as a number and automatically converted to text. **Text** is text, a part of which will be replaced. **position** is the position within the text where the replacement will begin. **length** is the number of characters in **text** to be replaced. **new text** is the text which replaces **text**..
REPT(text, number)	Repeats a character string by the given **number** of copies. **text** is the text to be repeated. **number** is the number of repetitions. The result can be a maximum of 255 characters.
RIGHT(text, number)	Returns the right-most **number** of characters of a **text** string. If optional **number** is omitted, 1 is assumed and the right-most character is returned. If **number** is greater than the length of **text**, the whole text is returned.
ROMAN(Number, Mode)	Converts a number into a Roman numeral. The value range must be between 0 and 3999; the modes can be integers from 0 to 4. **Number** is the number that is to be converted into a Roman numeral. **Mode** (optional) indicates the degree of simplification. The higher the value, the greater is the simplification of the Roman numeral.
ROT13(Text)	Encrypts a character string by moving the characters 13 positions in the alphabet. After the letter Z, the alphabet begins again (Rotation). Entering text encrypted by this method, into the function decrypts the text. **Text** is the character string to be encrypted/decrypted.
SEARCH(find_text, text, position)	Returns the start position of a text string within a larger string. The start position for the search can be set as an option. The search text can be a number or any sequence of characters. The search is not case-sensitive. The search supports regular expressions. **find_text** is the text to be searched for. **text** is the text where the search will take place. **position** (optional) is the position in the text where the search is to start.
SUBSTITUTE(text, search_text, new text, occurrence)	Substitutes new text for old text in a string. **text** is the text in which text segments are to be exchanged. **search_text** is the text segment that is to be replaced (a number of times). **new text** is the text that is to replace the text segment. **occurrence** (optional) indicates how many occurrences of the search text are to be replaced. If this parameter is missing, the search text is replaced throughout.
T(value)	Returns **value** if text, else returns a blank text string. **value** is the value to be evaluated. A reference can be used as a parameter. If the dereferenced value is not of type text, the result will be an empty string.

Syntax	Description
TEXT(number, Format)	Converts a number into text according to a given format. **number** is the numerical value to be converted. **Format** is the text which defines the format and can be found on the *Numbers tab* in the *Format Cells* dialog. Use decimal and thousands separators according to the language set in the cell format.
TRIM(text)	Returns a text string from which leading and trailing spaces have been removed, and replaces all internal multiple spaces with a single space. **text** is the text from which spaces are to be removed.
UNICHAR(number)	Returns the character represented by the given number according to the [UNICODE] Standard. **number** is a decimal integer value between 0 and 1114111.
UNICODE(text)	Returns the [UNICODE] code point corresponding to the first character of the text value. **text** is a string from which the code number is returned.
UPPER(text)	Converts the string specified in the **text** parameter to uppercase characters.
VALUE(text)	Converts a text string into a number. **text** is the text to be converted to a number.

Add-in functions

Table 12: Add-in functions

Syntax	Description
BESSELI(X, N)	Calculates the modified Bessel function In(x). **X** is the value on which the function will be calculated. **N** is the order of the Bessel function.
BESSELJ(X, N)	Calculates the Bessel function Jn(x) (cylinder function). **X** is the value on which the function will be calculated. **N** is the order of the Bessel function.
BESSELK(X, N)	Calculates the modified Bessel function Kn(x). **X** is the value on which the function will be calculated. **N** is the order of the Bessel function.
BESSELY(X, N)	Calculates the modified Bessel function Yn(x), also known as the Weber or Neumann function. **X** is the value on which the function will be calculated. **N** is the order of the Bessel function.
BIN2DEC(Number)	Returns the decimal number for the binary number entered. **Number** is the binary value entered as a number or as text.
BIN2HEX(Number, Places)	Returns a string representing the hexadecimal number for the binary number entered. **Number** is the binary value entered as a number or text. **Places** (optional) is the number of places to be output.

Syntax	*Description*
BIN2OCT(Number, Places)	Returns the octal number for the binary number entered. **Number** is the binary value entered as a number or text.. **Places** is the number of places to be output.
COMPLEX(Real num, I num, Suffix)	Returns a complex number from a real coefficient and an imaginary coefficient. **Real num** is the real coefficient of the complex number. **I num** is the imaginary coefficient of the complex number. **Suffix** is optional, and may be "i" or "j". If omitted “i” is assumed. **Suffix** must be lowercase.
CONVERT_ADD(Number, From unit, To unit)	Converts a value from one unit of measure to the corresponding value in another unit of measure. **Number** is the value to be converted. **From unit** is the unit from which conversion is taking place. **To unit** is the unit to which conversion is taking place. A list of abbreviations for units can be found in the Help files. Do not enter the full-stop.
DEC2BIN(Number, Places)	Returns the binary number for the decimal number entered between –512 and 511. **Number** is the decimal number. **Places** is the number of places to be output.
DEC2HEX(Number, Places)	Returns the hexadecimal number for the decimal number entered. **Number** is the decimal number. **Places** is the number of places to be output.
DEC2OCT(Number, Places)	Returns the octal number for the decimal number entered. **Number** is the decimal number. **Places** is the number of places to be output.
DELTA(Number 1, Number 2)	Returns TRUE (1) if both numbers are equal, otherwise returns FALSE (0). **Number 2** is optional and assumes a value of 0 if omitted.
ERF(Lower limit, Upper limit)	Calculates the error function. **Lower limit** is the lower limit of integral. **Upper limit** (optional) is the upper limit of the integral. If this value is missing, the calculation takes places between 0 and the lower limit.
ERFC(Lower limit)	Calculates the complementary error function between x and infinity. **Lower limit** is the lower limit of integral (x).
FACTDOUBLE(Number)	Returns the factorial of **Number** with increments of 2. If **Number** is even, the following factorial is calculated: n*(N-2)*(n-4)*...*4*2. If **Number** is odd, the following factorial is calculated: n*(N-2)*(n-4)*...*3*1.
GESTEP(Number, Step)	Returns 1 if **Number** is greater than or equal to **Step**.
HEX2BIN(Number, Places)	Returns the binary number for the hexadecimal number entered. **Number** is the hexadecimal number. **Places** is the number of places to be output.
HEX2DEC(Number)	Returns the decimal number for the hexadecimal number entered. **Number** is the hexadecimal number.
HEX2OCT(Number, Places)	Returns the octal number for the hexadecimal number entered. **Number** is the hexadecimal number. **Places** is the number of places to be output.

Syntax	*Description*
IMABS(Complex number)	Returns the absolute value (modulus) of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMAGINARY(Complex number)	Returns the imaginary coefficient of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMARGUMENT(Complex number)	Returns the argument (the phi angle) of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCONJUGATE(Complex number)	Returns the conjugated complex complement to the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCOS(Complex number)	Returns the cosine of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCOSH(Complex number)	Returns the hyperbolic cosine of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCOT(Complex number)	Returns the cotangent of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCSC(Complex number)	Returns the cosecant of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMCSCH(Complex number)	Returns the hyperbolic cosecant of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMDIV(Numerator, Denominator)	Returns the division of two complex numbers. **Numerator** and **Denominator** are entered in the form "x + yi" or "x + yj".
IMEXP(Complex number)	Returns the power of e (the Eulerian number) and the complex number. **Complex number** is entered in the form "x + yi" or "x + yj".
IMLN(Complex number)	Returns the natural logarithm of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMLOG10(Complex number)	Returns the common logarithm of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMLOG2(Complex number)	Returns the binary logarithm of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMPOWER(Complex number, Number)	Returns the entered **Complex number** raised to the power **Number**. The complex number is entered in the form "x + yi" or "x + yj".
IMPRODUCT(Complex number, Complex number1, …, Complex number30)	Returns the product of the entered **Complex number** with up to 30 other **Complex numbers**. The complex numbers are entered in the form "x + yi" or "x + yj".

Syntax	*Description*
IMREAL(Complex number)	Returns the real coefficient of the entered **Complex number.** The complex number is entered in the form "x + yi" or "x + yj".
IMSIN(Complex number)	Returns the sine of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMSINH(Complex number)	Returns the hyperbolic sine of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
IMSQRT(Complex number)	Returns the square root of the entered **Complex number**. The complex numbers are entered in the form "x + yi" or "x + yj".
IMSUB(Complex number 1, Complex number 2)	Returns the difference of two complex numbers. The complex numbers are entered in the form "x + yi" or "x + yj".
IMSUM(Complex number, Complex number 1, …, Complex number30)	Returns the sum of the entered **Complex number** with up to 30 other complex numbers. The complex_numbers are entered in the form "x + yi" or "x + yj".
IMTAN(Complex number)	Returns the tangent of the entered **Complex number**. The complex number is entered in the form "x + yi" or "x + yj".
OCT2BIN(Number, Places)	Returns the binary number for the octal value entered. **Number** is the octal number. **Places** is the number of places to be output. **Number** may be entered as text or a number.
OCT2DEC(Number)	Returns the decimal number for the octal value entered. **Number** is the octal number. **Number** may be entered as text or a number.
OCT2HEX(Number, Places)	Returns the hexadecimal number for the octal value entered. **Number** is the octal number. **Places** is the number of places to be output. **Number** may be entered as text or a number.