Calc Guide

Appendix C
Description of Functions

Copyright

Contributors

This book is adapted and updated from the *LibreOffice 4.1 Calc Guide.*

To this edition

Steve Fanning

To previous editions

Barbara Duprey
Jean Hollis Weber
Simon Brydon
John A Smith

Feedback

Please direct any comments or suggestions about this document to the Documentation Team's mailing list: documentation@global.libreoffice.org.

Note

Everything you send to a mailing list, including your email address and any other personal information that is written in the message, is publicly archived and cannot be deleted.

Publication date and software version

Published December 2019. Based on LibreOffice 6.2.

Using LibreOffice on macOS

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

Windows or Linux	*macOS equivalent*	*Effect*
Tools > Options menu	**LibreOffice > Preferences**	Access setup options
Right-click	*Control* + click or right-click depending on computer setup	Open a context menu
Ctrl (Control)	⌘ (Command)	Used with other keys
F5	*Shift* + ⌘ + *F5*	Open the Navigator
F11	⌘ + *T*	Open the sidebar Styles panel

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 chapter 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.

Over 500 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** or **_EXCEL2003** are provided for compatibility with Microsoft Excel 2003 functions. They return the same results as the corresponding functions in Excel 2003 (without the suffix), which though they may be correct, are not based on international standards. Calc automatically changes the function to **_ADD** or **_EXCEL2003** for relevant functions in imported Excel 2003 spreadsheets.

Terminology: numbers and arguments

Some of the descriptions in this chapter 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 numbers* (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 chapter attempts to clarify this situation by using the term **arguments**, rather than any of the other phrases.

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

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.
AGGREGATE(Function, Option, Ref1, Ref2, …) or AGGREGATE(Function, Option, Array, k)	Returns an aggregate result across the specified cells. The user can select the aggregation function to be used and which data to be ignored during the aggregation. The AGGREGATE function was designed for vertical ranges of data with the AutoFilter function activated. If AutoFilter is not activated, the AGGREGATE function's results will not be automatically recalculated to reflect newly hidden rows. It is possible to apply the AGGREGATE function to horizontal ranges but there are limitations – in particular, the function will not recognize hiding columns. The **Function** argument is a value, or a reference to a cell containing a value, in the range 1 to 19. This enables the user to select the aggregation function (for example, AVERAGE, COUNT, MAX, or MIN) with the mapping of values to functions described in the Help system. The **Option** argument is a value, or a reference to a cell containing a value, in the range 0 to 7. This enables the user to select the type of data that should be ignored during the aggregation, with the meaning of each value described in the Help system. **Ref1**, **Ref2**, …, **Ref253** are numeric arguments or references to cells that contain them. This syntax variant is only intended for use with **Function** values in the range 1 to 13. The **Array** argument can be specified by the boundaries of the range, the name of the named range, or the column label. **k** is a numeric argument for certain functions and must correspond to the second argument of these functions (for example, the LARGE function requires a value for its Rank_c argument). **k** should be omitted for **Function** values 1 to 13, but should be included for those in the range 14 to 19.
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. This is the angle between the x-axis and a line from the origin to the point (**number_x**, **number_y**).

Syntax	*Description*
ATANH(Number)	Returns the inverse hyperbolic tangent of the given **Number** in radians.
BITAND(Number1, Number2)	This is the bitwise “AND” of two positive integers whose values are less than 2^48.
BITLSHIFT(Number, Shift)	The bitwise left shift of a positive integer value. **Number** is an integer less than 2^48. **Shift** is the number of bits to move by.
BITOR(Number1, Number2)	This is the bitwise “OR” of two positive integers whose values are less than 2^48.
BITRSHIFT(Number, Shift)	The bitwise right shift of a positive integer value. **Number** is an integer less than 2^48. **Shift** is the number of bits to move by.
BITXOR(number1, number2)	This is the bitwise “exclusive OR” of two positive integers whose values are less than 2^48.
CEILING(Number, Significance, Mode)	Rounds up the given **Number** to the nearest multiple of the value of **Significance**. **Significance** is an optional value that defaults to -1 or +1, depending on the sign of **Number**. **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** so that negative numbers are rounded away from zero. If the **Mode** value is equal to zero or omitted, negative numbers are rounded towards zero. If the spreadsheet is exported to MS Excel, the CEILING function is exported as the equivalent CEILING.MATH function that has existed since Excel 2013. If you plan to use the spreadsheet with earlier Excel versions, use either CEILING.PRECISE that has existed since Excel 2010, or CEILING.XCL that is exported as the CEILING function compatible with all Excel versions.
CEILING.MATH(Number, Significance, Mode)	Rounds up the given **Number** to the nearest multiple of the value of **Significance**. **Significance** is an optional value that defaults to 1. **Mode** is an optional value. If the **Mode** value is given and not equal to zero, negative numbers are rounded away from zero. If the **Mode** value is equal to zero or is not given, negative numbers are rounded towards zero.
CEILING.PRECISE(Number; Significance)	Rounds the given **Number** up to the nearest multiple of **Significance**, regardless of the sign of **Significance**. **Significance** is an optional value.
CEILING.XCL(Number, Significance)	Rounds the given **Number** away from zero to the nearest multiple of **Significance**.

Syntax	*Description*
COLOR(Red, Green, Blue, Alpha)	Returns a numeric value calculated by a combination of three colors (red, green, and blue) and the alpha channel, in the RGBA color system. The result depends on the color system used by your computer. The values for the **Red**, **Green** and **Blue** components of the color must be between 0 and 255. **Alpha** is an optional integer argument, with a value between 0 and 255. An **Alpha** value of 0 means the color is fully transparent, whereas a value of 255 gives a fully opaque color.
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.
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_OOO(Value, From Unit, To Unit)	Converts a **Value** in Euros to another pre-Euro European currency, and vice versa. Enter the **From Unit** and **To Unit** directly as text in quotation marks or as a reference. If you enter the currencies in cells, they must correspond exactly with the list of allowed currencies, which is case sensitive. The list of allowed currencies and the conversion factors are defined in the configuration file *main.xcd*. The standard *main.xcd* file distributed with LibreOffice 6.2 includes conversion factors between Euros (“EUR”) and 19 other currencies (for example “ATS”, “BEF”, “CYP”, “DEM”, and so on).
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).
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, using conversion rates set by the European Commission. **value** is the value to be converted. **from_currency** is the ISO 4217 code of the currency from which **value** is to be converted. **to_currency** is the ISO 4217 code of the currency to which **value** is to be converted. **from_currency** and **to_currency** are not case sensitive. **full_precision** is optional. If omitted or set to 0, the result is rounded according to the decimals of the **to_currency**. For other values, 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 **to_currency** is "EUR", **triangulation_precision** is used as if triangulation was needed and conversion from Euro to Euro was applied. The following currency codes are recognized: “ATS”, “BEF”, “CYP”, “DEM”, “ESP”, “EUR”, “FIM”, “FRF”, “GRD”, “IEP”, “ITL”, “LUF”, “NLG”, “PTE”, “SIT”.
EVEN(Number)	Rounds a positive **Number** up to the next even integer, and a negative **Number** down to the next even integer.
EXP(Number)	Returns the constant e raised to the power of the given **Number**.
FACT(Number)	Returns the factorial of the given **Number**.
FLOOR(Number, Significance, Mode)	Rounds down the given **Number** to the nearest multiple of the value of **Significance**. **Significance** is an optional value that defaults to -1 or +1, depending on the sign of **Number**. **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** so that negative numbers are rounded towards zero. If the **Mode** value is equal to zero or omitted, negative numbers are rounded away from zero. If the spreadsheet is exported to MS Excel, the FLOOR function is exported as the equivalent FLOOR.MATH function that has existed since Excel 2013. If you plan to use the spreadsheet with earlier Excel versions, use either FLOOR.PRECISE that has existed since Excel 2010, or FLOOR.XCL that is exported as the FLOOR function compatible with all Excel versions.
FLOOR.MATH(Number, Significance, Mode)	Rounds down the given **Number** to the nearest multiple of the value of **Significance**. **Significance** is an optional value that defaults to 1. **Mode** is an optional value. If the **Mode** value is given and not equal to zero, negative numbers are rounded towards zero. If the **Mode** value is equal to zero or is not given, negative numbers are rounded away from zero.

Syntax	*Description*
FLOOR.PRECISE(Number, Significance)	Rounds the given **Number** down to the nearest multiple of the value of **Significance**, regardless of the sign of **Significance**. **Significance** is an optional value.
FLOOR.XCL(Number, Significance)	Rounds the given **Number** towards zero to the nearest multiple of the absolute value of **Significance**.
GCD(Integer1, Integer2, …, Integer30)	Returns the greatest common divisor of one or more positive integers. **IntegersX** 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_EXCEL2003(Number1, Number 2, …, Number 30)	Returns the greatest common divisor of a list of up to 30 numbers. This function returns the same result as the corresponding Microsoft Excel 2003 GCD function. Use Calc's GCD function to get results based on international standards.
INT(Number)	Rounds the given **Number** down to the nearest integer.
ISO.CEILING(Number, Significance)	Rounds a number up to the nearest multiple of the value of **Significance**, regardless of the sign of **Significance**. **Significance** is an optional value.
LCM(Integer1, Integer2, …, Integer30)	Returns the lowest common multiple of one or more integers. **Integer 1**, **Integer 2**, …, **Integer 30** are integers whose lowest common multiple is to be calculated.
LCM_EXCEL2003(Number1, Number2, …, Number30)	Returns the lowest common multiple of a list of up to 30 numbers. This function returns the same result as the corresponding Microsoft Excel 2003 LCM function. Use Calc's LCM function to get results based on international standards.
LN(Number)	Returns the natural logarithm, based on the constant e, of the given **Number** >0.
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 **Dividend** is divided.
MROUND(Number, Multiple)	Returns **Number** rounded to the nearest multiple of **Multiple**.
MULTINOMIAL (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.
ODD(Number)	Rounds **Number** up if positive and down if negative, to the nearest odd integer.
PI()	Returns the value of the constant pi (π) to fourteen decimal places.

Syntax	*Description*
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 **Base** is to be raised.
PRODUCT(Number1, Number2, …, Number30)	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.
QUOTIENT(Numerator, Denominator)	Returns the integer result of a division operation. **Numerator** is the number that will be divided. **Denominator** is the number **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 *F9* is pressed.
RANDBETWEEN (Bottom, Top)	Returns an integer random number between **Bottom** and **Top** (inclusive). This number will recalculate every time *F9* is pressed).
RAWSUBTRACT(Minuend, Subtrahend1, Subtrahend2, ...)	Subtracts the **Subtrahend(s)** from the **Minuend** without eliminating roundoff errors. The function should be called with at least two arguments.
ROUND(number, count)	Rounds the given **number** to **count** (optional) decimal places. If the **count** argument 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 (toward zero), to **count** (optional) decimal places. If **count** is omitted or zero, the function rounds down to an integer. If **count** is negative, the function rounds down to the next 10, 100, 1000 and so on.
ROUNDSIG(Value, Digits)	Rounds a **Value** to a defined number of significant **Digits**.
ROUNDUP(number, count)	Rounds the given **number** up (away from zero), to **count** (optional) decimal places. If **count** is omitted or zero, the function rounds up to an integer. If **count** is negative, the function rounds up to the nearest 10, 100, 1000 and so on.
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 the initial terms in a power series. **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).

Syntax	*Description*
SINH(number)	Returns the hyperbolic sine of the given **number** (angle in radians).
SQRT(number)	Returns the positive square root of the given positive **number**.
SQRTPI(Number)	Returns the square root of the product of the given **Number** and the constant pi (π).
SUBTOTAL(Function, range)	Calculates subtotals. If a range already contains subtotals, these are not used for further calculations. Use this function with AutoFilters activated to take only the filtered records into account. **Function** is a value that selects which function should be used to create the subtotal (for example, Average, Count, Min, Sum, or Var) and also determines whether manually hidden rows should be included. The set of allowed **Function** values is listed in the Help system. **range** is the range of cells 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 (in double quotes). **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, …, range 127, criteria 127)	Totals the values of cells in a range that meet multiple criteria in multiple ranges. **sum_range** is the cell range from which the values are to be totaled. **range 1** is the cell range to be evaluated by **criteria 1**, **range 2** (optional) by **criteria 2** (optional), and so on. All ranges must have the same size and shape. The function can have up to 255 arguments, meaning that you can specify 127 criteria ranges and criteria for them.
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** is the number of decimal places which are retained. If **count** is 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 arguments 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** argument 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 date at which the interest accrued up until then is to be calculated. **Rate** is the annual nominal rate of interest (coupon interest rate). **Par** (optional) is the par value of the security. **Frequency** is the number of interest payments per year (1, 2 or 4). **Basis** (optional) indicates how the year is to be calculated.

Syntax	*Description*
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** (optional) is the par value of the security. **Basis** (optional) 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** (optional) 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** (optional) 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** (optional) 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** (optional) indicates how the year is to be calculated.
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** (optional) 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** (optional) indicates how the year is to be calculated.

Syntax	*Description*
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** (optional) 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** (optional) 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 **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.
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.
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.

Syntax	*Description*
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 (defaults to 12).
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** (optional) 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(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** (optional) indicates how the year is to be calculated.
EFFECT(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 rate refers to the amount of interest due at the end of a calculation period. **NOM** is the nominal interest rate. **P** is the number of interest payment periods per year.

Syntax	*Description*
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 rate refers to the amount of interest due at the end of a calculation period. **Nominal rate** is the nominal interest rate. **Npery** is the number of interest payments per year. This function returns the same result as the corresponding Microsoft Excel 2003 function without the suffix. Use the Calc function without a suffix to get results based on international standards.
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** (optional) 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 a 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.
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. **rate** sets the periodic interest rate. **Period** is the number of installments for calculation of interest. **total_periods** is the total number of installment periods. **invest** is the amount of the investment.

Syntax	*Description*
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** (optional) 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, default 0) is the future value, which is reached at the end of the last period. **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.
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** (optional) indicates how the year is to be calculated.

Syntax	*Description*
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** (optional) 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** (optional) 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** (optional) indicates how the year is to be calculated.

Syntax	*Description*
OPT_BARRIER(Spot, Volatility, Rate, Foreign Rate, Maturity, Strike, Lower Barrier, Upper Barrier, Rebate, Put or Call, In or Out, Barrier Monitoring, Greek)	Returns the pricing for a barrier option, calculated using the Black-Scholes option pricing model. **Spot** is the price / value of the underlying asset and should be greater than 0.0. **Volatility** is the annual percentage volatility of the underlying asset expressed as a decimal. The value should be greater than 0.0. **Rate** is the continuously compounded interest rate. This is a percentage expressed as a decimal. **Foreign Rate** is the continuously compounded foreign interest rate. This is a percentage expressed as a decimal. **Maturity** is the time to maturity of the option, in years, and should be non-negative. **Strike** is the strike price of the option and should be non-negative. **Lower Barrier** is the predetermined lower barrier price; set to zero for no lower barrier. **Upper Barrier** is the predetermined upper barrier price; set to zero for no upper barrier. **Rebate** is the amount of money to be paid at maturity if the barrier is hit. **Put or Call** is a string that defines whether the option is a put (“p”) or a call (“c”). **In or Out** is a string that defines whether the option is knock-in (“i”) or knock-out (“o”). **Barrier Monitoring** is a string that defines whether the barrier is monitored continuously (“c”) or only at the end / maturity (“e”). **Greek** (optional) is a string argument. If omitted or set to “value”, “v”, “price”, or “p”, then the function simply returns the option price. If another valid string is entered (see Help files), the function returns price sensitivities (Greeks) to one of the input arguments.
OPT_PROB_HIT(Spot, Volatility, Drift, Maturity, Lower Barrier, Upper Barrier)	Returns the probability that an asset hits a predetermined barrier price, assuming that the stock price can be modeled as a process S that follows the stochastic differential equation, as follows. $$\frac{dS}{S} = \mu\, dt + vol\, dW$$ See the Help files for an explanation of the parameters appearing in this equation. **Spot** is the price / value of the underlying asset and should be greater than 0.0. **Volatility** is the annual percentage volatility of the underlying asset expressed as a decimal. The value should be greater than 0.0. **Drift** is the annual stock price percentage drift rate, expressed as a decimal. **Maturity** is the time to maturity of the option, in years, and should be non-negative. **Lower Barrier** is the predetermined lower barrier price; set to zero for no lower barrier. **Upper Barrier** is the predetermined upper barrier price; set to zero for no upper barrier.

Syntax	*Description*
OPT_PROB_INMONEY(Spot, Volatility, Drift, Maturity, Lower Barrier, Upper Barrier, Strike, Put or Call)	Returns the probability that an asset will end up between two barrier levels at maturity, assuming that the stock price can be modeled as a process S that follows the stochastic differential equation, as follows. $$\frac{dS}{S} = \mu\, dt + vol\, dW$$ See the Help files for an explanation of the parameters appearing in this equation. If the optional **Strike** and **Put or Call** arguments are included, then • For a call option, the function returns the probability that the asset will end up between **Strike** and **Upper Barrier**. • For a put option, the function returns the probability that the asset will end up between **Lower Barrier** and **Strike**. The function ignores the possibility of knock-out before maturity. **Spot** is the price / value of the underlying asset and should be greater than 0.0. **Volatility** is the annual percentage volatility of the underlying asset expressed as a decimal. The value should be greater than 0.0. **Drift** is the annual stock price percentage drift rate, expressed as a decimal. **Maturity** is the time to maturity of the option, in years, and should be non-negative. **Lower Barrier** is the predetermined lower barrier price; set to zero for no lower barrier. **Upper Barrier** is the predetermined upper barrier price; set to zero for no upper barrier. **Strike** (optional) is the strike price of the option and should be non-negative. The default is -1.0 (to indicate that a value has not been set). **Put or Call** (optional) is a string that defines whether the option is a put (“p”) or a call (“c”). The default is “c”.

Syntax	*Description*
OPT_TOUCH(Spot, Volatility, Rate, Foreign Rate, Maturity, Lower Barrier, Upper Barrier, Domestic or Foreign, In or Out, Barrier Monitoring, Greek)	Returns the pricing of a touch / no-touch option, calculated using the Black-Scholes option pricing model. **Spot** is the price / value of the underlying asset and should be greater than 0.0. **Volatility** is the annual percentage volatility of the underlying asset expressed as a decimal. The value should be greater than 0.0. **Rate** is the continuously compounded interest rate. This is a percentage expressed as a decimal. **Foreign Rate** is the continuously compounded foreign interest rate. This is a percentage expressed as a decimal. **Maturity** is the time to maturity of the option, in years, and should be non-negative. **Lower Barrier** is the predetermined lower barrier price; set to zero for no lower barrier. **Upper Barrier** is the predetermined upper barrier price; set to zero for no upper barrier. **Domestic or Foreign** is a string that defines whether the option pays domestic (“d”) or foreign (“f”) currency. **In or Out** is a string that defines whether the option is knock-in (“i”) or knock-out (“o”). **Barrier Monitoring** is a string that defines whether the barrier is monitored continuously (“c”) or only at the end / maturity (“e”). **Greek** (optional) is a string argument. If omitted or set to “value”, “v”, “price”, or “p”, then the function simply returns the option price. If another valid string is entered (see Help files), the function returns price sensitivities (Greeks) to one of the input arguments.
PDURATION(Rate, PV, FV)	Calculates the number of periods required by an investment to attain a desired value. **Rate** is the constant rate of interest. **PV** is the present value of the investment. **FV** is the desired future value of the investment.
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 the 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 the annuity is paid. **Period** = 1 for the first period and **Period** = **NPER** for the last period. **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.

Syntax	*Description*
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** (optional) 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** (optional) 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** (optional) 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.
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** (optional) 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.

Syntax	*Description*
SLN(Cost, Salvage, Life)	Returns the straight-line depreciation of an asset for one 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.
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 argument. Type = 0 means a switch to linear depreciation. In Type = 1, no switch is made.

Syntax	*Description*
XIRR(Values, Dates, 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 a 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 a 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** (optional) 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** (optional) indicates how the year is to be calculated.
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** (optional) indicates how the year is to be calculated.

Statistical analysis functions

Calc includes over 150 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 chapter.

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.
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, criterion, average_range)	Returns the arithmetic mean of all cells in a range that satisfy a given condition. Sums up all the results that match a logical test and divides this sum by the number of selected values. **range** is an array, a named range, or the label of a column or a row. It should contain numbers for averaging, or numbers or text for the condition. **criterion** is a condition in the form of an expression, or a cell reference with expression, that defines which cells should be used to calculate the mean. The expression can contain text, numbers, regular expressions, or wildcards (if enabled). **criterion** should be a string expression and should be enclosed in quotation marks, except for the names of functions, cell references, and string concatenation (&). **average_range** is an optional argument that gives a range of values for calculating the mean. If **average_range** is not specified, **range** is used for both the calculation of the mean and the search according to the condition. If **average_range** is specified, **range** is used only for the condition test, while **average_range** is used for the mean calculation. If a cell in a range of values for calculating the mean is empty or contains text, the cell is ignored. If the whole range is empty, contains only text, or includes no values that satisfy the condition, the function returns an error.

Syntax	*Description*
AVERAGEIFS(average_range, range 1, criterion 1, range 2, criterion 2, …, range 127, criterion 127)	Returns the arithmetic mean of all cells in a range that satisfy given multiple criteria. Sums up all the results that match the logical tests and divides this sum by the number of selected values. **average_range** is a range of cells, a name of a named range, or a label of a column or a row containing values for calculating the mean. **range1** is an array, a named range, or the label of a column or a row. It should contain numbers or text for the condition. **range2**, **range3**, …, and so on are optional but have the same meaning as **range1**. **average_range**, **range1**, **range2**, …, and so on must have the same size, otherwise the function returns an invalid argument error. **criterion 1** is a string expression representing a logical condition, or a cell reference to such a string expression. The expression can contain text, numbers, regular expressions, or wildcards (if enabled). **criterion 1**, **criterion 2**, …, and so on are optional but have the same meaning as **criterion 1**. The logical relation between criteria can be defined as logical AND (conjunction). In other words, a value from the corresponding cell of the given **average_range** is taken into the calculation if and only if all given criteria are met. The function can have up to 255 arguments, meaning that you can specify up to 127 range and criterion combinations.
B(trials, SP, T_1, T_2)	Returns the probability of a specified number of successes in a binomial experiment. **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 successful trials. **T_2** (optional) defines the upper limit for the number of successful trials.
BETA.DIST(Number, Alpha, Beta, Cumulative, Start, End)	Calculates either the cumulative distribution function or the probability density function for the given arguments, based on a Beta distribution. **Number** is the value at which to evaluate the function and should lie between **Start** and **End**. **Alpha** and **Beta** are the positive shape parameters of the Beta distribution. Set **Cumulative** to 0 or False to calculate the probability density function; set to any other value to calculate the cumulative distribution function. **Start** is the lower bound for the value interval of the distribution and **End** is the upper bound. **Start** and **End** are optional, defaulting to 0 and 1 respectively.
BETA.INV(Number, Alpha, Beta, Start, End)	Returns the inverse of the cumulative distribution function, based on a Beta distribution. **Number** is the probability value for which to evaluate the function and should lie between **Start** and **End**. **Alpha** and **Beta** are the positive shape parameters of the Beta distribution. **Start** is the lower bound for the value interval of the distribution and **End** is the upper bound. **Start** and **End** are optional, defaulting to 0 and 1 respectively.

Syntax	*Description*
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** and **beta** are the positive shape parameters of the Beta distribution. **Start** is the lower bound for **number**. **End** is the upper bound for **number**. **Start** and **End** are optional, defaulting to 0 and 1 respectively. **Cumulative** can be 0 or False to calculate the probability density function, and can be any other value or omitted to calculate the cumulative distribution function.
BETAINV(number; alpha, beta, Start, End)	Returns the inverse of the cumulative distribution function, based on a Beta distribution. **number** is the value between **Start** and **End** at which to evaluate the function. **alpha** and **beta** are the positive shape parameters of the Beta distribution. **Start** is the lower bound for **number**. **End** is the upper bound for **number**. **Start** and **End** are optional, defaulting to 0 and 1 respectively.
BINOM.DIST(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.
BINOM.INV(Trials, SP, Alpha)	Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value. **Trials** is the total number of trials. **SP** is the probability of success on each trial. **Alpha** is the border probability that is to be attained or exceeded.
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.

Syntax	*Description*
CHISQ.DIST(Number, Degrees of Freedom, Cumulative)	Returns values of the cumulative distribution function, or the probability density function, for the chi-square distribution. **Number** is the value for which the probability density function or cumulative distribution function is to be calculated. **Degrees of Freedom** defines the degrees of freedom of the experiment. **Cumulative** can be 0 or False to calculate the probability density function. It can be any other value to calculate the cumulative distribution function.
CHISQ.DIST.RT(Number, Degrees of Freedom)	Returns the probability value from the indicated chi-square that a hypothesis is confirmed. **Number** is the chi-square value of the random sample used to determine the error probability. **Degrees of Freedom** defines the degrees of freedom of the experiment. The probability determined by CHISQ.DIST.RT can also be determined by CHITEST.
CHISQ.INV(Probability, Degrees of Freedom)	Returns the inverse of the left-tailed probability of the chi-square distribution. **Probability** is the probability value for which the inverse of the chi-square distribution is to be calculated. **Degrees of Freedom** defines the degrees of freedom of the experiment.
CHISQ.INV.RT(Number, Degrees of Freedom)	Returns the inverse of the one-tailed probability of the chi-squared distribution. **Number** is the probability value for which the inverse chi-square distribution is to be calculated. **Degrees of Freedom** defines the degrees of freedom of the experiment.
CHISQ.TEST(Data_B, Data_E)	Returns the probability of a deviance from a random distribution of two test series based on the chi-squared test for independence. CHISQ.TEST returns the chi-squared distribution of the data. **Data_B** is the array of the observations and **Data_E** is the range of the expected values. The probability determined by this function can also be determined with CHISQ.DIST, in which case the chi-square of the random sample must then be passed as an argument instead of the data row.
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** can be 0 or False to calculate the probability density function. It can be any other value to calculate the cumulative distribution function.
CHISQINV(Probability, Degrees of Freedom)	Returns the inverse of CHISQDIST. **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 for the experiment.

Syntax	*Description*
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 an argument 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.
CONFIDENCE.NORM(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.
CONFIDENCE.T(Alpha, StDev, Size)	Returns the (1-alpha) confidence interval for a Student's t-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.
COUNTBLANK(Range)	Returns the number of empty cells within the specified **Range** of cells.

Syntax	*Description*
COUNTIF(Range, Criteria)	Returns the number of cells that meet the given criteria within a range of cells. **Range** is the cell range to which the criteria are to be applied. **Criteria** indicates the criteria in the form of a number, an expression, or a character string. These criteria determine which cells are counted. If regular expressions are enabled, you may also enter a search text in the form of a regular expression. If wildcards are enabled, you may enter a search text with wildcards. You may also indicate a cell address that contains the search criterion. If you search for literal text, enclose the text in double quotes.
COUNTIFS(Range1, Criteria1, Range2, Criteria2, ...)	Returns the count of cells that meet given criteria in multiple ranges. The function can take up to 255 arguments, meaning that you can specify up to 127 range / criteria pairs. **Range1** is a range of cells, a name of a named range, or a label of a column or a row containing values for counting and finding the corresponding criteria. **Range2** … **Range127** are optional and mean the same as **Range1**. If **Range1** … **Range127** are not the same height and width, the function returns an invalid argument error. **Criteria1** is a string expression representing a logical condition or a cell reference to such a string expression. The expression can contain text, numbers, regular expressions or wildcards (if enabled). **Criteria2** … **Criteria127** are optional and mean the same as **Criteria1**. If a cell contains TRUE, it is treated as 1. If a cell contains FALSE, it is treated as 0 (zero). The logical relation between criteria can be defined as logical AND (conjunction). The count is incremented if a cell in **Range1** meets **Criteria** 1, and the corresponding cell in **Range2** meets **Criteria2**, and so on for all specified range / criteria pairs. Note that in this context the top left cell in **Range1** *corresponds* to the top left cell in all other ranges – the top left cell in each range need not be the same spreadsheet cell.
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. Both data sets should be the same size and shape. Any empty element or non-numeric value in an element will cause the corresponding element to be ignored.
COVARIANCE.P(Data1, Data2)	Returns the covariance of the product of paired deviations, for the entire population. **Data1** is the first data set and **Data2** is the second data set. **Data1** and **Data2** should contain the same number of data points.
COVARIANCE.S(Data1, Data2)	Returns the covariance of the product of paired deviations, for a sample of the population. **Data1** is the first data set and **Data2** is the second data set. **Data1** and **Data2** should contain the same number of data points.

Syntax	*Description*
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.
ERF.PRECISE(LowerLimit)	Returns values of the Gaussian error integral between 0 and the specified **LowerLimit.**
ERFC.PRECISE(LowerLimit)	Returns complementary values of the Gaussian error integral between the specified **LowerLimit** and infinity.
EXPON.DIST(Number, Lambda, C)	Returns values of either the probability density function or the cumulative distribution function for the exponential distribution. **Number** is the value for which the exponential distribution is to be calculated. **Lambda** is the rate parameter of the exponential distribution. The argument **C** determines the form of the function calculated – a value of 0 requests the probability density function and a value of 1 requests the cumulative distribution function.
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 for which the exponential distribution is to be calculated. **lambda** is the rate parameter value. **C** is a logical value that determines the form of the function. **C** = 0 calculates the probability density function, and **C** = 1 calculates the cumulative distribution function.
F.DIST(Number, Degrees of Freedom1, Degrees of Freedom2, Cumulative)	Calculates values of either the probability density function or the cumulative distribution function, for the left tail of the F probability distribution. **Number** is the value for which the function is to be calculated. **Degrees of Freedom1** specifies the degrees of freedom in the numerator in the F distribution. **Degrees of Freedom2** specifies the degrees of freedom in the denominator in the F distribution. **Cumulative** is set to 0 or False to calculate the probability density function. **Cumulative** is set to 1 or True to calculate the cumulative distribution function.
F.DIST.RT(Number, Degrees of Freedom1, Degrees of Freedom2)	Calculates F probability distribution values for the right tail. **Number** is the value for which the function is to be calculated. **Degrees of Freedom1** specifies the degrees of freedom in the numerator in the F distribution. **Degrees of Freedom2** specifies the degrees of freedom in the denominator in the F distribution.

Syntax	*Description*
F.INV(Number, Degrees of Freedom1, Degrees of Freedom2)	Returns the inverse of the cumulative F distribution. **Number** is the probability value for which the inverse F distribution is to be calculated. **Degrees of Freedom1** specifies the degrees of freedom in the numerator of the F distribution. **Degrees of Freedom2** specifies the number of degrees of freedom in the denominator of the F distribution.
F.INV.RT(Number, Degrees of Freedom1, Degrees of Freedom2)	Returns the inverse of the F distribution, for the right tail. **Number** is the probability value for which the inverse F distribution is to be calculated. **Degrees of Freedom1** specifies the degrees of freedom in the numerator of the F distribution. **Degrees of Freedom2** specifies the number of degrees of freedom in the denominator of the F distribution.
F.TEST(Data1, Data2)	Returns the result of an F test for the two specified data sets. **Data1** is the first data array. **Data2** is the second data array.
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.
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.
FISHER(Number)	Returns the Fisher transformation for the given **Number**, which must be greater than -1 and less than +1. 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, using a linear regression. **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.

Syntax	*Description*
FORECAST.ETS.ADD(Targets, Values, Timeline, Period Length, Data Completion, Aggregation)	Calculates the additive forecast(s) (future values) based on the historical data using Exponential Triple Smoothing (ETS) or Exponential Double Smoothing (EDS) algorithms. **Targets** is a date, time or numeric single value or range. It gives the data point/range for which to calculate a forecast. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.

Syntax	*Description*
FORECAST.ETS.MULT(Targets, Values, Timeline, Period Length, Data Completion, Aggregation)	Calculates the multiplicative forecast(s) (future values) based on the historical data using Exponential Triple Smoothing (ETS) or Exponential Double Smoothing (EDS) algorithms. **Targets** is a date, time or numeric single value or range. It gives the data point/range for which to calculate a forecast. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.

Syntax	*Description*
FORECAST.ETS.PI.ADD(Target, Values, Timeline, Confidence Level, Period Length, Data Completion, Aggregation)	Calculates the prediction interval(s) for additive forecast based on the historical data using Exponential Triple Smoothing (ETS) or Exponential Double Smoothing (EDS) algorithms. **Target** is a date, time or numeric single value or range. It gives the data point/range for which to calculate a forecast. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Confidence Level** is a numeric value between 0 and 1 (exclusive), indicating a confidence level for the calculated prediction interval (default 0.95). **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.

Syntax	*Description*
FORECAST.ETS.PI.MULT(Target, Values, Timeline, Confidence Level, Period Length, Data Completion, Aggregation)	Calculates the prediction interval(s) for multiplicative forecast based on the historical data using Exponential Triple Smoothing (ETS) or Exponential Double Smoothing (EDS) algorithms. **Target** is a date, time or numeric single value or range. It gives the data point/range for which to calculate a forecast. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Confidence Level** is a numeric value between 0 and 1 (exclusive), indicating a confidence level for the calculated prediction interval (default 0.95). **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.
FORECAST.ETS.SEASONALITY (Values, Timeline, Data Completion, Aggregation)	Returns the number of samples in period as calculated by Calc in the case of FORECAST.ETS functions when the Period Length argument equals 1. The same result is returned with FORECAST.ETS.STAT functions when argument Stat Type equals 9 and Period Length equals 1. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.

Syntax	*Description*
FORECAST.ETS.STAT.ADD (Values, Timeline, Stat Type, Period Length, Data Completion, Aggregation)	Returns statistical value(s) that are results of the Exponential Triple Smoothing (ETS) / Exponential Double Smoothing (EDS) algorithms. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and Values must be the same size. **Stat Type** is a value from 1 to 9 that indicates which statistic will be returned for the given values and x-range. The mapping of values to statistics is given in the Help system. **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.

Syntax	*Description*
FORECAST.ETS.STAT.MULT (Values, Timeline, Stat Type, Period Length, Data Completion, Aggregation)	Returns statistical value(s) that are results of the Exponential Triple Smoothing (ETS) / Exponential Double Smoothing (EDS) algorithms. **Values** is a numeric array or range that gives the historical values for which you want to forecast the next points. **Timeline** is a numeric array or range that gives the timeline (x-value) range for the historical values. The **Timeline** values must have a consistent step between them. **Timeline** must contain at least two periods of data and **Values** must be the same size. **Stat Type** is a value from 1 to 9 that indicates which statistic will be returned for the given values and x-range. The mapping of values to statistics is given in the Help system. **Period Length** is an optional positive integer indicating the number of samples in a period (default 1). A value of 1 indicates that Calc is to determine the number of samples in a period automatically. A value of 0 indicates no periodic effects, a forecast is calculated with EDS algorithms. For all other positive values, forecasts are calculated with ETS algorithms. **Data Completion** is an optional logical value (default 1). If set to 0, the algorithm will add missing data points with zero as a historical value. If set to 1, the algorithm will add missing data points by interpolating between the neighboring data points. **Aggregation** is an optional value from 1 to 7 (default 1) indicating which method to use to aggregate identical time values. The mapping of values to aggregate functions is given in the Help system.
FORECAST.LINEAR(Value, DataY, DataX)	Extrapolates future values based on existing x and y values. **Value** is the x value for which the y value on the linear regression is to be returned. **DataY** is the array or range of known y values. **DataX** is the array or range of known x values.
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.
GAMMA.DIST(Number, Alpha, Beta, C)	Returns values of the probability density function or the cumulative distribution function for a Gamma distribution. The inverse function is GAMMAINV or GAMMA.INV. This function is identical to GAMMADIST and was introduced for interoperability with other office suites. **Number** is the value for which the Gamma distribution is to be calculated. **Alpha** is the shape parameter of the Gamma distribution. **Beta** is the rate parameter of the Gamma distribution. **C** determines the type of function required. If **C** = 0 or False, Calc calculates the probability density function. If **C** = 1 or True, Calc calculates the cumulative distribution function.

Syntax	*Description*
GAMMA.INV(Number, Alpha, Beta)	Returns the inverse of the Gamma cumulative distribution. This function allows you to search for variables with different distribution. This function is identical to GAMMAINV and was introduced for interoperability with other office suites. **Number** is the probability value for which the inverse Gamma distribution is to be calculated. **Alpha** is the shape parameter of the Gamma distribution. **Beta** is the rate parameter of the Gamma distribution.
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 shape parameter of the Gamma distribution. **beta** is the rate parameter of the Gamma distribution. **Cumulative** = 0 calculates the probability density function, and **Cumulative** = 1 calculates the cumulative distribution function.
GAMMAINV(Number, alpha, beta)	Returns the inverse of the GAMMADIST. 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 shape parameter of the Gamma distribution. **beta** is the rate parameter of the Gamma distribution.
GAMMALN(Number)	Returns the natural logarithm of the Gamma function for the given **Number**.
GAMMALN.PRECISE(Number)	Returns the natural logarithm of the Gamma function. **Number** is the value for which the natural logarithm of the Gamma function is to be calculated.
GAUSS(Number)	Returns the standard normal cumulative distribution for the given **Number**.
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. **number 1**, **number 2**, …, **number 30** are values or ranges for which you want to calculate the harmonic mean.
HYPGEOM.DIST(X, NSample, Successes, NPopulation, Cumulative)	Returns probability density function or cumulative distribution function values from the hypergeometric distribution. **X** is the number of results achieved in the random sample. **NSample** is the size of the random sample. **Successes** is the number of possible results in the total population. **NPopulation** is the size of the total population. **Cumulative** defines whether to calculate the probability density function (0 or FALSE) or the cumulative distribution function (other values).

Syntax	*Description*
HYPGEOMDIST(X, n_sample, successes, n_population)	Returns the hypergeometric distribution. **X** is the number of results achieved in the random sample. **n_sample** is the size of the random sample. **successes** is the number of possible results 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** argument.
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.
LOGNORM.DIST(Number, Mean, StDev, Cumulative)	Returns values of the probability density function or the cumulative distribution function for a lognormal distribution. **Number** is the probability value for which the standard logarithmic distribution is to be calculated. **Mean** is the mean value of the standard logarithmic distribution. **StDev** is the standard deviation of the standard logarithmic distribution. **Cumulative** defines whether to calculate the probability density function (0) or the cumulative distribution function (1).
LOGNORM.INV(Number, Mean, StDev)	Returns values for the inverse of the lognormal distribution. This function is identical to LOGINV and was introduced for interoperability with other office suites. **Number** is the probability value for which the inverse standard logarithmic distribution is to be calculated. **Mean** is the arithmetic mean of the standard logarithmic distribution. **StDev** is the standard deviation of the standard logarithmic distribution.

Syntax	*Description*
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 probability density function, **Cumulative** = 1 calculates the cumulative distribution function.
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. Non-numbers are ignored.
MAXA(value 1, value 2, …, value 30)	Returns the maximum value in a list of arguments. Unlike MAX, text values can be entered. Text is evaluated as 0. **value 1**, **value 2**, …, **value 30** are values or ranges.
MAXIFS(Func_Range, Range1, Criterion1, Range2, Criterion2, ...)	Returns the maximum of the values of cells in a range that meet multiple criteria in multiple ranges. **Func_Range** is a range of cells, a name of a named range or a label of a column or a row containing values for calculating the maximum. **Range1** is a range of cells, a named range, or a row / column label, containing the data to be evaluated against **Criterion1**. **Criterion1** is a string expression representing a logical condition or a cell reference to such string expression. The expression can contain text, numbers, regular expressions, or wildcards (if enabled). **Range2** … **Range127** are optional arguments and all mean the same as **Range1**. **Criterion2** … **Criterion127** are optional arguments and all mean the same as **Criterion1**. The function can have up to 255 arguments, meaning that you can specify 127 range / criterion pairs. **Func_Range**, **Range1**, and **Range2** ... **Range127** (where present) must have the same dimensions. The logical relation between criteria can be defined as logical AND (conjunction). In other words, if and only if all given criteria are met, a value from the corresponding cell of the given **Func_Range** is taken into calculation.
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. Text cells are ignored.
MINA(value 1, value 2, …, value 30)	Returns the minimum value in a list of arguments. Text is evaluated as 0. **value 1**, **value 2**, …, **value 30** are values or ranges.

Syntax	*Description*
MINIFS(Func_Range, Range1, Criterion1, Range2, Criterion2, ...)	Returns the minimum of the values of cells in a range that meet multiple criteria in multiple ranges. **Func_Range** is a range of cells, a name of a named range or a label of a column or a row containing values for calculating the minimum. **Range1** is a range of cells, a named range, or a row / column label, containing the data to be evaluated against **Criterion1**. **Criterion1** is a string expression representing a logical condition or a cell reference to such string expression. The expression can contain text, numbers, regular expressions, or wildcards (if enabled). **Range2** … **Range127** are optional arguments and all mean the same as **Range1**. **Criterion2** … **Criterion127** are optional arguments and all mean the same as **Criterion1**. The function can have up to 255 arguments, meaning that you can specify 127 range / criterion pairs. **Func_Range**, **Range1**, and **Range2** ... **Range127** (where present) must have the same dimensions. The logical relation between criteria can be defined as logical AND (conjunction). In other words, if and only if all given criteria are met, a value from the corresponding cell of the given **Func_Range** is taken into calculation.
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 no value appears more than once.
MODE.MULT(Number1, Number2, …, Number30)	Returns a vertical array of the statistical modes (the most frequently occurring values) within a list of supplied numbers. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges. As the MODE.MULT function returns an array of values, it must be entered as an array formula. If the function is not entered as an array formula, only the first mode is returned, which is the same as using the MODE.SNGL function.
MODE.SNGL(Number1, Number2, …, Number30)	Returns the most frequently occurring, or repetitive, value in an array or range of data. If there are several values with the same frequency, it returns the smallest value. An error occurs when a value doesn't appear twice. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges.
NEGBINOM.DIST(X, R, SP, Cumulative)	Returns values of the probability density function or cumulative distribution function for 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. **Cumulative** = 0 calculates the density probability density function, **Cumulative** = 1 calculates the cumulative distribution function.

Syntax	*Description*
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**.
NORM.DIST(Number, Mean, StDev, C)	Returns values of the probability density function or the cumulative distribution function for the normal distribution. **Number** is the value for which the normal distribution is to be calculated. **Mean** is the mean value of the distribution. **StDev** is the standard deviation of the distribution. **C** = 0 calculates the probability density function, **C** = 1 calculates the cumulative distribution function.
NORM.INV(Number, Mean, StDev)	Returns values of the inverse of the normal cumulative distribution. **Number** represents the probability value used to determine the inverse normal distribution. **Mean** represents the mean value of the normal distribution. **StDev** represents the standard deviation of the normal distribution.
NORM.S.DIST(Number; Cumulative)	Returns values of the probability density function or the cumulative distribution function for the standard normal distribution. This distribution has a mean of 0 and a standard deviation of 1. **Number** is the value to which the standard normal cumulative distribution is calculated. **Cumulative** 0 or FALSE calculates the probability density function. Any other value calculates the cumulative distribution function.
NORM.S.INV(Number)	Returns values of the inverse of the standard normal distribution. **Number** is the probability for which the inverse standard normal distribution is calculated.
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. If **C** = 0 or FALSE it calculates the probability density function, and if **C** = any other value or is 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 of the normal distribution. **STDEV** is the standard deviation of the normal distribution.

Syntax	*Description*
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()).
NORMSINV(number)	Returns the inverse of the standard normal distribution for the given probability value, **number**. **number** must be in the range 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.
PERCENTILE.EXC(Data, Alpha)	Returns a percentile value for a set of data based on the given value of alpha, which lies within the range 0 to 1 (*exclusive*). A percentile returns the scale value for a data series which goes from the smallest (alpha=0) to the largest (alpha=1) value. For alpha = 25%, the percentile means the first quartile; alpha = 50% is the median. **Data** is the array of data. **Alpha** represents the percentage of the scale between 0 and 1. If **Alpha** is not a multiple of 1/(n+1), (where n is the number of values in the supplied array), the function interpolates between the values in the supplied array, to calculate the percentile value. However, if **Alpha** is less than 1/(n+1) or **Alpha** is greater than n/(n+1), the function is unable to interpolate, and so returns an error.
PERCENTILE.INC(Data, Alpha)	Returns a percentile value for a set of data based on the given value of alpha, which lies within the range 0 to 1 (*inclusive*). A percentile returns the scale value for a data series which goes from the smallest (alpha=0) to the largest (alpha=1) value. For alpha = 25%, the percentile means the first quartile; alpha = 50% is the median. **Data** is the array of data. **Alpha** represents the percentage of the scale between 0 and 1.
PERCENTRANK(data, value, Significance)	Returns the percentage rank (percentile) of the given **value** in a sample. **data** is the array of data in the sample. **Significance** is an optional argument that specifies the number of significant digits for rounding the returned percentage value.

Syntax	*Description*
PERCENTRANK.EXC(Data, Value, Significance)	Returns the relative position, between 0 and 1 (*exclusive*), of a specified value within a supplied array. **Data** is the array of data in the sample. **Value** represents the value whose percentile rank must be determined. **Significance** is an optional argument that specifies the number of significant digits for rounding the returned percentage value.
PERCENTRANK.INC(Data, Value, Significance)	Returns the relative position, between 0 and 1 (*inclusive*), of a specified value within a supplied array. **Data** is the array of data in the sample. **Value** represents the value whose percentile rank must be determined. **Significance** is an optional argument that specifies the number of significant digits for rounding the returned percentage value.
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 density function or the cumulative distribution function for the Poisson distribution. **Number** is the value for which the distribution is to be calculated. **mean** is the middle value of the Poisson distribution. **Cumulative** = 0 or False calculates the probability density function, and **Cumulative** = 1 or True calculates the cumulative distribution function. When **C** is omitted, the default value True is inserted when you save the document, for best compatibility with other programs and older versions of LibreOffice.
POISSON.DIST(Number, Mean, C)	Returns a probability for the Poisson distribution, using either the probability density function or the cumulative distribution function. **Number** represents the value for which the Poisson distribution is calculated. **Mean** represents the middle value of the Poisson distribution. **C** is an optional argument. When **C** is set to 0 or False, the function calculates the probability density function. When **C** is set to 1 or True, the function calculates the cumulative distribution function. When **C** is omitted, the default value True is inserted when you save the document, for best compatibility with other programs and older versions of LibreOffice.

Syntax	*Description*
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 argument 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.
QUARTILE.EXC(Data, Type)	Returns a requested quartile of a supplied range of values, based on a percentile range of 0 to 1 *exclusive*. **Data** represents the range of data values for which you want to calculate the specified quartile. **Type** is an integer between 1 and 3, representing the required quartile. If **Type** = 1 or 3, the supplied array must contain more than two values.
QUARTILE.INC(Data, Type)	Returns a requested quartile of a supplied range of values, based on a percentile range of 0 to 1 *inclusive*. **Data** is the array of data in the sample. **Type** represents the quartile required, that is: • 0 – minimum value • 1 – first quartile • 2 – second quartile (median) • 3 – third quartile • 4 – maximum value
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. **Type** = 0 means descending from the last item of the array to the first (this is the default). **Type** = 1 means ascending from the first item of the range to the last.
RANK.AVG(Value, Data, Type)	Returns the statistical rank of a given value, within a supplied array of values. If there are duplicate values in the list, *the average rank is returned*. **Value** is the value whose rank is to be determined. **Data** is the array or range of data in the sample. **Type** is an optional argument defining the sequence order. A **Type** value of 0 means descending from the last item of the array to the first and this is the default. A **Type** value of 1 means ascending from the first item of the range to the last.

Syntax	*Description*
RANK.EQ(Value, Data, Type)	Returns the statistical rank of a given value, within a supplied array of values. If there are duplicate values in the list, *these are given the same rank*. **Value** is the value whose rank is to be determined. **Data** is the array or range of data in the sample. **Type** is an optional argument defining the sequence order. A **Type** value of 0 means descending from the last item of the array to the first and this is the default. A **Type** value of 1 means ascending from the first item of the range to the last.
RSQ(data_Y, data_X)	Returns the square of the Pearson correlation coefficient based on the given values. RSQ (also called the coefficient of determination) is a measure for the accuracy of an adjustment and can be used to produce a regression analysis. **data_Y** is an array or range of data points. **data_X** is an array or range of data points.
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.
STDEV.P(Number1, Number2, ... , Number30)	Calculates the standard deviation based on the entire population. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges representing an entire population. **Number2**, ..., **Number30** are optional.

Syntax	*Description*
STDEV.S(Number1, Number2, ... , Number30)	Calculates the standard deviation based on sample of the population. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges representing a sample of the population. **Number2**, ..., **Number30** are optional.
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 an entire population.
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.
T.DIST(Number, Degrees of Freedom, Cumulative)	Returns values from the one-tailed Student's t-distribution, using either the probability density function or the cumulative distribution function. **Number** is the value for which the t-distribution is calculated. **Degrees of Freedom** is the number of degrees of freedom for the t-distribution. Set **Cumulative** to 0 or FALSE to return values of the probability density function. Set **Cumulative** to 1 or TRUE to return values of the cumulative distribution function.
T.DIST.2T(Number, Degrees of Freedom)	Calculates values of the two-tailed Student's t-distribution, which is a continuous probability distribution that is frequently used for testing hypotheses on small sample data sets. **Number** is the value for which the t-distribution is calculated. **Degrees of Freedom** is the number of degrees of freedom for the t-distribution.
T.DIST.RT(Number, Degrees of Freedom)	Calculates the right-tailed Student's t-distribution, which is a continuous probability distribution that is frequently used for testing hypotheses on small sample data sets. **Number** is the value for which the t-distribution is calculated. **Degrees of Freedom** is the number of degrees of freedom for the t-distribution.
T.INV(Number, Degrees of Freedom)	Returns the one tailed inverse of the Student's t-distribution. **Number** is the probability associated with the one-tailed t-distribution. **Degrees of Freedom** is the number of degrees of freedom for the t-distribution.

Syntax	*Description*
T.INV.2T(Number, Degrees of Freedom)	Calculates the inverse of the two-tailed Student's t-distribution, which is a continuous probability distribution that is frequently used for testing hypotheses on small sample data sets. **Number** is the probability associated with the two-tailed t-distribution. **Degrees of Freedom** is the number of degrees of freedom for the t-distribution.
T.TEST(Data1, Data2, Mode, Type)	Returns the probability associated with a Student's t-Test. **Data1** is the dependent array or range of data for the first record. **Data2** is the dependent array or range of data for the second record. The value of **Mode** determines whether a one-tailed test (set to 1) or a two-tailed test (set to 2) is performed. **Type** specifies the type of t-test to perform, as follows: • 1 means paired. • 2 means two samples, equal variance (homoscedastic). • 3 means two samples, unequal variance (heteroscedastic).
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** is the array of data in the sample. **Alpha** 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.
VAR.P(Number1, Number2, ..., Number30)	Calculates a variance based on the entire population. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges representing an entire population. **Number2**, ..., **Number30** are optional.

Syntax	*Description*
VAR.S(Number1, Number2, ..., Number30)	Estimates the variance based on a sample. **Number1**, **Number2**, ..., **Number30** are numeric values or ranges representing a sample based on an entire population. **Number2**, ..., **Number30** are optional.
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.
WEIBULL(Number, Alpha, beta, C)	Returns values of the Weibull distribution, from either the probability density function or the cumulative distribution function. **Number** is the value at which to calculate the Weibull distribution. **Alpha** is the shape parameter of the Weibull distribution. **beta** is the scale parameter of the Weibull distribution. **C** indicates the type of function: If **C**= 0 the probability density function is calculated, if **C**=1 the cumulative distribution function is calculated.
WEIBULL.DIST(Number, Alpha, Beta, C)	Returns values of the Weibull distribution, from either the probability density function or the cumulative distribution function. **Number** is the value at which to calculate the Weibull distribution. **Alpha** is the shape parameter of the Weibull distribution. **Beta** is the scale parameter of the Weibull distribution. **C** indicates the type of function. If **C** is set to 0, the probability density function is calculated. If **C** is set to 1, the cumulative distribution is calculated.
Z.TEST(Data, mu, Sigma)	Calculates the probability of observing a z-statistic greater than the one computed based on a sample. **Data** is the given sample, drawn from a normally distributed population. **mu** is the known mean of the population. **Sigma** is an optional argument which specifies the known standard deviation of the population. If omitted, the standard deviation of the given sample is used.
ZTEST(data, mu, sigma)	Calculates the probability of observing a z-statistic greater than the one computed based on a sample. **data** is the given sample, drawn from a normally distributed population. **mu** is the known mean of the population. **sigma** (optional) is the known standard deviation of the population. If omitted, the standard deviation of the given sample is used.

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 select the start date for the internal conversion from days to numbers used by Calc, go to **Tools > Options > LibreOffice Calc > Calculate**. The initial default start date is 1899-12-30.

Calc stores time internally as a decimal fraction in the range 0 to 1. A value of 0.0 represents 00:00:00 (midnight) and a value of 0.5 represents 12:00:00 (noon).

Caution

When entering dates as part of formulas, slashes or dashes used as date separators are interpreted as arithmetic operators. Therefore, dates entered in this format are not recognised as dates and result in erroneous calculations. To keep dates from being interpreted as parts of formulas use the DATE function, for example, DATE(1954;7;20), or place the date in quotation marks and use the ISO 8601 notation, for example, "1954-07-20". Avoid using locale dependent date formats such as "07/20/54"; the calculation may produce errors if the document is loaded under different locale settings. See also A note about dates on page 12.

Note

Unambiguous conversion is possible for ISO 8601 dates and times in their extended formats with separators. If a #VALUE! error occurs, then select **Tools > Options > LibreOffice Calc > Formula> Custom > Details** and deselect the **Generate #VALUE! error** option in the *Conversion from text to number* menu on the Detailed Calculation Settings dialog.

Table 5: Data and time functions

Syntax	***Description***
DATE(year, month, day)	Converts a date written as year, month, day to an internal date number and displays it in the cell’s formatting. **year** is an integer between 1583 and 9956, or 0 and 99. An option accessed through **Tools > Options > LibreOffice > General** enables the user to control how Calc handles two-digit year values. **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 dates, **Start date** and **End date**. **Interval** is a string, accepted values are "d", "m", "y", "ym", "md" or "yd". The meanings associated with these strings are described in the Help system. **Start date** and **End date** must be entered using double quotes.
DATEVALUE(text)	Returns the internal date number for text in double quotes using the current locale. **text** is a valid date expression.

Syntax	*Description*
DAY(Number)	Returns the day, as an integer in the range 1 to 31, of the given date value. **Number** is the internal date number (a negative date/time value can be entered) or a date value entered in double quotes.
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 NASD method (0) or the European method (otherwise). 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. An option accessed through **Tools > Options > LibreOffice > General** enables the user to control how Calc handles two-digit year values.
EDATE(Start date, Months)	Returns the internal date 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 internal date number 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 in the range 0 to 23, for the given time value. **Number** is a time value and can be either text or an internal time 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 an internal date number.
ISOWEEKNUM(Number)	Calculates the ISO 8601 week number of the year for the internal date number. ISO 8601 defines that Monday shall be the first day of the week. A week that lies partly in one year and partly in another is assigned a number in the year in which most of its days lie. That means that week number 1 of any year is the week that contains January 4th. **Number** is text or an internal date number.

Syntax	*Description*
MINUTE(Number)	Returns the minute, as an integer in the range 0 to 59, for the given time value. **Number** is text or an internal date number.
MONTH(Number)	Returns the month, as an integer in the range 1 to 12, for the given date value. **Number** is text or an internal date number.
MONTHS(Start date, End date, Type)	Calculates the difference, in months, between two date values. **Start date** is the start 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.
NETWORKDAYS(Start date, End date, Holidays, Workdays)	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 / non-workdays. Enter a cell range in which the holidays are listed individually. **Workdays** is an optional list of number values defining the standard work week. This list starts with Sunday, workdays are indicated by zero and non-working days by non-zero value.
NETWORKDAYS.INTL(StartDate, EndDate, Weekend, Holidays)	Returns the number of workdays between a start date and an end date. There are options to define weekend days and holidays. The optional weekend argument (or a string) can be used to define the weekend days (or the non-working days in each week). Also, optionally, the user can define a holiday list. The weekend days and user-defined holidays are not counted as working days. **StartDate** is the date from when the calculation is carried out. If the start date is a workday, the day is included in the calculation. **EndDate** is the date up until when the calculation is carried out. If the end date is a workday, the day is included in the calculation. **Weekend** is an optional argument – a number or a string used to specify the days of the week that are weekend days and are not considered working days. **Weekend** number values are in the range 1 … 17, with the meaning of each code defined in the Help system. The **Weekend** string provides another way to define the weekly non-working days. It must have seven characters – zeroes (0) for working day and ones (1) for non-working day. Each character represents a day of the week, starting with Monday. Only 1 and 0 are valid. “1111111” is an invalid string and should not be used. For example, the weekend string “0000011” defines Saturday and Sunday as non-working days. **Holidays** is an optional list of dates that must be counted as non-working days. The list can be given in a cell range.

Syntax	*Description*
NETWORKDAYS_EXCEL2003 (Start date, End date, Holidays)	Returns the number of workdays between a start date and an 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** is an optional 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 or each time a cell value is modified.
SECOND(Number)	Returns the second, as an integer in the range 0 to 59, for the given time value. **Number** is text or an internal date number.
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 correctly formatted time string. **hour**, **minute** and **second** must all be integers.
TIMEVALUE(text)	Returns the internal time number value from **text** enclosed by quotes in a time entry format.
TODAY()	Returns the current computer system date. The value is updated when your document recalculates or each time a cell value is modified.
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**, which determines the type of calculation. The possible values of **Type** and their associated meanings are listed in the Help system.
WEEKNUM(Number, mode)	Calculates the number of the calendar week of the year for the given internal date **Number**. **mode** sets the start of the week and the calculation type. The possible values of **mode** and their associated meanings are listed in the Help system.
WEEKNUM_EXCEL2003(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.
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).
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 an internal date 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.
WORKDAY.INTL(Start Date, Days, Weekend, Holidays)	Returns an internal date number that can be formatted as a date. The user can see the date of a day that is a certain number of workdays away from the start date (before or after). **Start Date** is the date from when the calculation is carried out. If the start date is a workday, the day is included in the calculation. **Days** is the number of workdays. Enter a positive value for a result after the **Start date**, a negative value for a result before the **Start date**. **Weekend** is an optional argument – a number or a string used to specify the days of the week that are weekend days and are not considered working days. **Weekend** is a weekend number or string that specifies when weekends occur. **Weekend** number values are in the range 1 … 17, with the meaning of each code defined in the Help system. The **Weekend** string provides another way to define the weekly non-working days. It must have seven characters – zeroes (0) for working day and ones (1) for non-working day. Each character represents a day of the week, starting with Monday. Only 1 and 0 are valid. “1111111” is an invalid string and should not be used. For example, the weekend string “0000011” defines Saturday and Sunday as non-working days. **Holidays** is an optional list of dates that must be counted as non-working days. The list can be given in a cell range.
YEAR(Number)	Returns the calendar year as an integer according to the internal calculation rules. **Number** is the date value in internal date 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 chosen from a list of options and indicates how the year is to be calculated (see Help system for more details).
YEARS(Start date, End date, Type)	Calculates the difference in years between 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 and, if any element is FALSE, returns FALSE. **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 an argument, the function uses all values in the range. The result is TRUE if the logical value in all cells within the cell range is TRUE.
FALSE()	Returns the logical value FALSE.
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**.
IFNA(value, alternative value)	Evaluates **value**; if it is not the #N/A error value, it returns the result for **value**, or else it returns the **alternative value**.

IFS(expression1, result1, expression2, result2, …)	IFS is a multiple IF-function.
	expression1, **expression2**, ... are any boolean values or expressions that can be TRUE or FALSE. **result1**, **result2**, ... are the values that are returned if the logical test is TRUE.
	IFS(expression1, result1, expression2, result2, expression3, result3) is executed as
	`IF expression1 is TRUE`
	`THEN result1`
	`ELSE IF expression2 is TRUE`
	`THEN result2`
	`ELSE IF expression3 is TRUE`
	`THEN result3`
	To get a default result should no expression be TRUE, add a last expression that is always TRUE, like TRUE or 1=1 followed by the default result.
	If there is a result missing for an expression or if no expression is TRUE, a #N/A error is returned.
	If expression is neither TRUE nor FALSE, a #VALUE error is returned.

NOT(Logical value)	Reverses the logical value. **Logical value** is the TRUE or FALSE value to be reversed (complemented).
OR(Logical value 1, Logical value 2, …, Logical value 30)	Returns TRUE if at least one argument is TRUE or returns 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. If a range is entered as an argument, the function uses all values of the range.

Syntax	Description
SWITCH(Expression, Value1, Result1, Value2, Result2, … , Default Result)	Compares an expression with a number of values and returns the result corresponding to the first value that equals the expression. If there is no match and a default result is given, that will be returned. **Expression** is a text, numeric, logical or date input or reference to a cell. **Value1**, **Value2**, ... are any values or references to cells. Each value must have a result given. **Result1**, **Result2**, ... are any values or references to cells. **Value2**, … and **Result2**, … are optional arguments. **Default Result** is optional and is any value or reference to a cell that is returned when there is no match. If no value equals **Expression** and no default result is given, a #N/A error is returned.
TRUE()	Returns the logical value TRUE.
XOR(Logical value 1, Logical value 2, …, Logical value 30)	Computes the logical XOR of the arguments. If an even number of arguments are TRUE it returns FALSE, if an odd number of arguments are 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: Information functions

Syntax	Description
CELL(info_type, Reference)	Returns information about a cell such as its address, formatting or contents, depending on the value of the **info_type** argument. **info_type** is a text string that specifies the type of information to be returned and comes from a list of available options which are listed in the Help files. **info_type** is not case sensitive, but it must be enclosed within quotes. **Reference** (optional) 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()	Returns the result to date of evaluating the formula of which it is a part (in other words the result as far as that evaluation has got). Its main use is together with the STYLE() function to apply selected styles to a cell depending on the cell content.
FORMULA(Reference)	Returns the formula of a formula cell as a text string. **Reference** is a reference to a cell containing a formula. An invalid reference or a reference to a cell with no formula results in the error value #N/A.
INFO(Text)	Returns information about the current working environment. **Text** is a string constant entered in double quotes taken from a list of available options which are listed in the Help files. These strings are case insensitive.

Syntax	*Description*
ISBLANK(value)	Returns TRUE if the reference to a cell is blank, FALSE otherwise. This function is used to determine if the cell is empty. A cell containing a formula is not empty. If an error occurs, the function returns a logical or numerical value. **value** is the content to be tested.
ISERR(value)	Returns TRUE if the value refers to any error value except #N/A, FALSE otherwise. If an error occurs, the function returns a logical or numerical value. **value** is any value or expression which is tested to determine whether an error value not equal to #N/A is present.
ISERROR(value)	Returns TRUE if the value refers to any error value (including #N/A), FALSE otherwise. If an error occurs, the function returns a logical or numerical value. **value** is, or refers to, the value to be tested to determine whether it is an error value.
ISEVEN(value)	Returns TRUE if the given **value** is even, 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 1 if **Number** divided by 2 gives a whole number, 0 otherwise.
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 value. The function is used in order to check for both TRUE and FALSE values in certain cells. If an error occurs, the function returns FALSE. **value** is the cell reference to be tested for logical number format.
ISNA(value)	Returns TRUE if **value** contains the #N/A error value, FALSE otherwise. If an error occurs, the function returns FALSE. **value** is the cell, value or expression to be tested.
ISNONTEXT(value)	Return TRUE if the value is non-text, else returns FALSE. If an error occurs, the function returns TRUE. **value** is any value or expression where a test is performed to determine whether it is a text string, or numbers, or a Boolean value. Empty cells are considered non-text and will return TRUE.
ISNUMBER(value)	Returns TRUE if the cell content is, or refers 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 and FALSE 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.

Syntax	*Description*
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 but is truncated to an integer before evaluation. Does not return logical type TRUE/FALSE like ISODD; it returns a number.
ISREF(value)	Returns TRUE if **value** is of type reference (including a reference list), else returns 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, or refers to, a text string, else FALSE. If an error occurs, the function returns FALSE. **value** is a value, number, Boolean value, or error value to be tested.
N(value)	Returns the numeric value of the given argument. Returns 0 if the argument is text or FALSE. If an error occurs the function returns the error value. **value** is the argument to be converted into a number. N() returns the numeric value if it can. It returns the logical values TRUE and FALSE as 1 and 0 respectively. It returns text as 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 a logical or numeric value. The numerical value from which the data type is determined is; 1 = number, 2 = text, 4 = Boolean value, 8 = formula, 16 = error value, 64 = array.

Database functions

This section deals with functions used with data organized as one row of data for one record. The *Database* category in Calc 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 Base 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 where the function operates after the search criteria is applied and the data rows are selected. It is not related to the search criteria itself. For the **Database field** argument you can enter a reference to a header cell or a number to specify the column within the **Database** area, starting with 1. To reference a column by means of the literal column header name, place quotation marks around the header name.
- **Search criteria** is the cell range containing search criteria. If you write several criteria in one row they are connected by AND. If you write the criteria in different rows they are connected by OR. 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 functions

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 row found, or Err502 for more than one cell found.
DMAX(Database, Database field, Search criteria)	Returns the maximum value of a field in a database that matches the specified **Search criteria**.
DMIN(Database, Database field, Search criteria)	Returns the minimum value of a field in a database that matches the specified **Search criteria**.
DPRODUCT(Database, Database field, Search criteria)	Multiplies all cells of a data range where the cell content matches the **Search criteria**.
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.
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** = 0 then y = m^x functions are calculated, otherwise functions of the form y = b*m^x.
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** = TRUE full statistics are returned, otherwise just the regression coefficient. See the Help files 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** = TRUE the entire table is returned, otherwise just the regression coefficient. See the Help files 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 for which the determinant is required. The **Array** check-box is not automatically selected in this case.
MINVERSE(array)	Returns the inverse array. **array** is a square array that is to be inverted.
MMULT(array1, array2)	Calculates the array product of two arrays. **array1** is the first array used in the array product. **array2** is the second array with the same number of rows as the first array has columns.

Syntax	*Description*
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 in this case.
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 in this case.
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 in this case.
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 in this case.
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** = 0, y = mx functions are calculated. Otherwise functions of the form y = mx + c 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 a text string, according to the specified row and column numbers. **row** is the row number for the cell reference. **column** 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 details of the available options. 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 argument will shift that sheet name to become the fifth argument. A new fourth argument 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 argument, that argument 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. The function expects a single argument. If you state multiple ranges, you must enclose them within additional parentheses (round brackets). Multiple ranges can be entered using the semicolon (;) as divider, but this gets automatically converted to the tilde (~) operator. The tilde is used to join ranges. **reference** is the reference to a cell or cell range.
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 a reference to a cell or as individual values. Only the selected value from the list is evaluated, any other formulas in the list are not checked for validity.

Syntax	*Description*
COLUMN(reference)	Returns the column number of a **reference** (optional). If the reference is a single cell, the column number of the cell is returned; if the argument 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)	Returns the result of a Dynamic Data Exchange (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 a 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 argument that controls the method by which the DDE server converts its data into numbers. See the Help files for more information on the available options.
ERROR.TYPE(Error Value)	Returns a number representing a specific error type, or the error value #N/A, if there is no error. **Error Value** is the error value or a reference to a cell, whose value needs to be processed. For each possible **Error Value**, the function returns a value in the range 1 to 7 or #N/A. The meanings of these numeric values are given in the Help system.
ERRORTYPE(reference)	Evaluates the cell value at **reference** location. If the cell contains an error then a logical or numerical value is returned. The numerical value is the error number (see Help for full listing of possible error numbers).

Syntax	*Description*
GETPIVOTDATA(Data Field, Pivot Table, Field Name1/Item1, Field Name2/Item2, …, Field Name30/Item30) or GETPIVOTDATA(Pivot Table, Constraints)	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. For the first syntax (which is used in the Function Wizard); **Data Field** is a string that selects one of the pivot table's data fields. The string can be the name of the 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 NameX/ItemX** pairs are given, the grand total is returned. Otherwise, each pair adds a constraint that the result must satisfy. **Field NameX** 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 NameX/ItemX** pairs can be entered. The second syntax is assumed if exactly two arguments 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 system for more detailed information.
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 ascending order. Searching is faster for sorted columns.
HYPERLINK(URL, CellText)	When you press *Ctrl* +click over the text in a cell that contains the HYPERLINK function (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** argument is not specified, the **URL** text is displayed.

Syntax	*Description*
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. The **Array** checkbox must be selected in this function unless **row** and **column** are both included.
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. Otherwise, the A1 notation is used. If you open an Excel spreadsheet that uses indirect addresses calculated from string functions, the sheet addresses will not be translated automatically. For example, the Excel address in INDIRECT("[filename]sheetname!"&B1) is not converted into the Calc address in INDIRECT("filename#sheetname."&B1). The INDIRECT function is saved without conversion to ODF 1.0/1.1 format. If the second argument was present, an older version of Calc will return an error for that function.

Syntax	*Description*
LOOKUP(Search criterion, Search vector, result_vector)	Returns the contents of a cell either from a one-row or one-column range. 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. **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.
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** = 1 or if this optional argument is omitted, it is assumed that the first column of the search array is sorted in ascending order. If **Type** = -1 it is assumed that the column is sorted in descending order. This corresponds to the same function in Microsoft Excel. If **Type** = 0, only exact matches are found. If the search criterion is found more than once, the function returns the index of the first matching value. Only if **Type** = 0 can you search for regular expressions (if enabled in calculation options) or wildcards (if enabled in calculation options). If **Type** = 1 or the third argument is omitted, the index of the last value that is smaller than or equal to the search criterion is returned. This applies even when the search array is not sorted. For **Type** = -1, the index of the first value that is larger than or equal is returned.
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 rows 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. If the width or height is included, the OFFSET function returns a range and thus must be entered as an array formula. If both the width and height are missing, a cell reference is returned.

Syntax	*Description*
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** (optional) 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 arguments 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 arguments 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 text entries entered in double quotes. The initial style is applied for **Time** seconds after the cell itself is recalculated. **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.
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**. **Search criterion** is the value searched for in the first column of the array. **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** argument 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** argument 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.

The following text functions are provided in two related forms:

- FIND / FINDB
- LEFT / LEFTB
- LEN / LENB
- MID / MIDB
- REPLACE / REPLACEB
- RIGHT / RIGHTB
- SEARCH / SEARCHB

In each case the first named function is intended for use with languages that use the single-byte character set (SBCS), whereas the second named function (name ending "B") is intended for use with languages that use the double-byte character set (DBCS).

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. This is the complementary function to JIS.
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 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 in the range 2 to 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, zeroes 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 in the range 1 to 255 representing the code value for the character. Codes greater than 127 may depend on your system's character mapping (for example iso-8859-1, iso-8859-2, Windows-1252, Windows-1250) and hence may not be portable.
CLEAN(text)	Removes all non-printing characters from the string entered into **text**. Text is entered using double quotes.

Syntax	*Description*
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. The code used here does not refer to ASCII, but to the code table currently loaded. Codes greater than 127 may depend on your system's character mapping (for example iso-8859-1, iso-8859-2, Windows-1252, Windows-1250) and hence may not be portable.
CONCAT(String1, String2, …)	Concatenates one or more strings. The CONCAT function is an enhancement of CONCATENATE, as CONCAT also accepts ranges as arguments, such as B2:E5, K:K or K:M. When ranges are used, the cells are traversed row by row (from top to bottom) to concatenate. **String1**, **String2**, … are strings or references to cells or ranges that contain strings to concatenate. **String2** onward are optional.
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. **text 2** onward are optional.
DECIMAL(text, radix)	Converts a text string with characters from a number system to a positive integer in the base radix given. Spaces and tabs are ignored. The text field is not case-sensitive. If the radix is 16, a leading x or X or 0x or 0X, and an appended h or H, is disregarded. If the radix is 2, an appended b or B is disregarded. Other characters that do not belong to the number system generate an error. **text** is the text string to be converted. To differentiate between a hexadecimal number, such as A1 and the reference to cell A1, you must place the number in quotation marks, for example, "A1" or "FACE". **radix** indicates the base of the number system. It may be any positive integer in the range 2 to 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.
ENCODEURL(Text)	Returns a URL-encoded string. Use this function to transform text with symbols of national alphabets (for example accented characters, non-ASCII alphabets, or Asian words) to a string of URL-standard symbols. **Text** is a string to encode to a sequence of URL-standard symbols.

Syntax	*Description*
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.
FILTERXML(XML Document, XPath Expression)	Applies an XPath expression to an XML document and returns the relevant XML content. **XML Document** is a string containing a valid XML stream. **XPath Expression** is a string containing a valid XPath expression.
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.
FINDB(Find Text, Search Text, Position)	Returns the starting position of a given text string within another text string, using byte positions. Where to begin the search can also be defined. The search is case sensitive. **Find Text** is the text to be found. **Search Text** is the text to be searched. **Position** is optional and specifies the position in **Start Text** from which the search should start. **Position** defaults to the first character of the string to be searched. Text strings must be entered in double quotation marks.
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 argument is equal to 0 or omitted, the thousands separators of the current locale setting are displayed, else the separators are suppressed.
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 argument is omitted, one character is returned. If **number** is greater than the length of the string, the whole string is returned.

Syntax	*Description*
LEFTB(Text, Number of Bytes)	Returns the first character(s) of a DBCS text string. **Text** is the text string from which the initial partial words are to be determined. **Number of Bytes** is optional and specifies the number of characters to be extracted, expressed as bytes (two bytes constitute one complete DBCS character). If this argument is not defined, one character is returned.
LEN(text)	Returns the length of a string including spaces. **text** is the text whose length is to be determined.
LENB(Text)	Returns the number of bytes used to represent the characters in a DBCS text string. **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 arguments 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.
MIDB(Text, Start, Number of Bytes)	Returns a sub-string from a DBCS text string. The arguments specify the starting position and the number of characters. **Text** is the text containing the characters to extract. **Start** is the position of the first character in the text to extract. **Number of Bytes** specifies the number of characters to be returned, in bytes.
NUMBERVALUE(text, decimal_separator, group_separator)	Converts text to number, in a locale-independent way. **text** is a valid number expression and must be entered with quotation marks. **decimal_separator** (optional) defines the character used as the decimal separator. **group_separator** (optional) defines the character(s) used as the group separator. The length of the **decimal_separator** must be 1 and the decimal_separator should not appear in **group_separator**.
PROPER(text)	Capitalizes the first letter in all words of a text string. **text** is the text to be converted.

Syntax	*Description*
REGEX(Text, Expression, Replacement, Flags or Occurrence)	Matches and extracts or optionally replaces text using regular expressions. **Text** is a text string or reference to a cell where the regular expression is to be applied. **Expression** is text representing the regular expression, using International Components for Unicode (ICU) regular expressions. If there is no match and **Replacement** is not given, #N/A is returned. **Replacement** is optional and gives the replacement text and references to capture groups. If there is no match, **Text** is returned unmodified. **Flags** is optional. "g" replaces all matches of **Expression** in **Text**, not extracted. If there is no match, **Text** is returned unmodified. **Occurrence** is optional and gives a number to indicate which match of **Expression** in **Text** is to be extracted or replaced. If there is no match and **Replacement** is not given, #N/A is returned. If there is no match and **Replacement** is given, **Text** is returned unmodified. If **Occurrence** is 0, **Text** is returned unmodified.
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**.
REPLACEB(Text, Position, Length, New Text)	Replaces part of a text string, based on the number of bytes you specify, with a different text string. **Text** is a text string, a part of which will be replaced. **Position** is the position of the character within the text where the replacement will begin. **Length** is the number of bytes in **Text** to be replaced. **New Text** is replacement 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.
RIGHTB(Text, Number of Bytes)	Returns the last character(s) of a DBCS text string. **Text** is the text string of which the right part is to be determined. **Number of Bytes** is optional and specifies the number of characters you want to extract, expressed as bytes (two bytes constitute one complete DBCS character).

Syntax	*Description*
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 previously encrypted by this 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.
SEARCHB(Find Text, Search Text, Position)	Returns the start position of a text string within a larger DBCS text 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. **Find Text** is the text to be searched for. **Search Text** is the text where the search will take place. **Position** is optional and defines 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 which occurrence of the search text is to be replaced. If this argument is missing, the search text is replaced throughout.
T(value)	Returns the target text, or a blank text string if the target is not a text string. **value** is the value to be evaluated. A reference can be used as an argument. If the dereferenced value is not of type text, the result will be an empty string.
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. Use decimal and thousands separators according to the language set in the cell format. More information on number format codes is available in the Help system.

Syntax	Description
TEXTJOIN(Delimiter, Skip Empty, String1, String2, …)	Concatenates one or more strings, and inserts delimiters between them. **Delimiter** is a text string and can be a range. **Skip Empty** is a logical (TRUE or FALSE, 1 or 0) argument. When TRUE, empty strings will be ignored. **String1**, **String2**, … are strings or references to cells or ranges that contain text to join. **String2** onward are optional. Ranges are traversed row by row (from top to bottom). If **Delimiter** is a range, the range need not be of the same size as the number of strings to be joined. If there are more delimiters than strings to be joined, not all delimiters will be used. If there are fewer delimiters than strings to be joined, the delimiters will be used again from the start.
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 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** argument to uppercase characters.
VALUE(text)	Converts a text string into a number. **text** is the text to be converted to a number.
WEBSERVICE(URI)	Get some web content from a Uniform Resource Identifier (URI). **URI** is the URI text of the web service.

Add-in functions

Table 12: Add-in functions

Syntax	Description
BESSELI(X, N)	Calculates the modified Bessel function of the first kind In(x). **X** is the value on which the function will be calculated. **N** is a positive integer giving the order of the Bessel function.
BESSELJ(X, N)	Calculates the Bessel function of the first kind Jn(x) (cylinder function). **X** is the value on which the function will be calculated. **N** is a positive integer giving the order of the Bessel function.

Syntax	*Description*
BESSELK(X, N)	Calculates the modified Bessel function of the second kind Kn(x). **X** (>0) is the value on which the function will be calculated. **N** is a positive integer giving the order of the Bessel function.
BESSELY(X, N)	Calculates the modified Bessel function of the second kind Yn(x), also known as the Weber or Neumann function. **X** (>0) is the value on which the function will be calculated. **N** is a positive integer giving 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. The number can have a maximum of 10 places (bits). The most significant bit is the sign bit. Negative numbers are entered as two's complement.
BIN2HEX(Number, Places)	Returns the hexadecimal number for the binary number entered. **Number** is the binary value entered as a number or text. The number can have a maximum of 10 places (bits). The most significant bit is the sign bit. Negative numbers are entered as two's complement. **Places** (optional) is the number of places to be output.
BIN2OCT(Number, Places)	Returns the octal number for the binary number entered. **Number** is the binary value entered as a number or text. The number can have a maximum of 10 places (bits). The most significant bit is the sign bit. Negative numbers are entered as two's complement. **Places** (optional) 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(Number, From Unit, To Unit)	Converts a value from one unit of measure to the corresponding value in another unit of measure. Enter the units of measure directly as text in quotation marks or as a reference. **Number** is the number 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. Both units must be of the same type. If you enter the units of measure in cells, they must correspond exactly with the list of allowed units, which is case-sensitive. For example, in order to enter a lower-case "l" (for litre) in a cell, enter the apostrophe (') immediately followed by l. Some units of measure can be preceded by a prefix character. For example, use prefix M (mega) for 10^6. The information units "bit" and "byte" may also be prefixed by one of the IEC 60027-2 / IEEE 1541 prefixes. More information on all these aspects can be found in the Help system.

Syntax	*Description*
DEC2BIN(Number, Places)	Returns the binary number for the decimal number entered between –512 and 511. **Number** is the decimal number. If **Number** is negative, the function returns a binary number with 10 characters. The most significant bit is the sign bit, the other 9 bits return the value. **Places** (optional) 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. If **Number** is negative, the function returns a hexadecimal number with 10 characters (40 bits). The most significant bit is the sign bit, the other 39 bits return the value. **Places** (optional) 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. If **Number** is negative, the function returns an octal number with 10 characters (30 bits). The most significant bit is the sign bit, the other 29 bits return the value. **Places** (optional) 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)	Returns values of the Gaussian error integral. **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)	Returns complementary values of the Gaussian error integral between a given lower limit and infinity. **Lower limit** is the lower limit of the integral.
FACTDOUBLE(Number)	Returns the double factorial of a number. **Number** is an integer greater than or equal to zero. For even numbers FACTDOUBLE(n) returns: 2*4*6*8* ... *(n-2)*n For odd numbers FACTDOUBLE(n) returns: 1*3*5*7* ... *(n-2)*n FACTDOUBLE(0) returns 1 by definition.
GESTEP(Number, Step)	Returns 1 if **Number** is greater than or equal to **Step** (optional, default 0), 0 otherwise.
HEX2BIN(Number, Places)	Returns the binary number for the hexadecimal number entered. **Number** is a hexadecimal number or a string that represents a hexadecimal number. It can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement. **Places** (optional) is the number of places to be output.

Syntax	*Description*
HEX2DEC(Number)	Returns the decimal number for the hexadecimal number entered. **Number** is is a hexadecimal number or a string that represents a hexadecimal number. It can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement.
HEX2OCT(Number, Places)	Returns the octal number for the hexadecimal number entered. **Number** is a hexadecimal number or a string that represents a hexadecimal number. It can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement. **Places** (optional) is the number of places to be output.
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** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMCOS(Complex number)	Returns the cosine of the entered **Complex number** as a text string. 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** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMCOT(Complex number)	Returns the cotangent of the entered **Complex number** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMCSC(Complex number)	Returns the cosecant of the entered **Complex number** as a text string. 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** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMDIV(Numerator, Denominator)	Returns the division of two complex numbers as a text string. **Numerator** and **Denominator** are entered in the form "x + yi" or "x + yj".
IMEXP(Complex number)	Returns the power of e (the Eulerian number) to the complex number as a text string. **Complex number** is entered in the form "x + yi" or "x + yj".

Syntax	*Description*
IMLN(Complex number)	Returns the natural logarithm of the entered **Complex number** as a text string. 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** as a text string. 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** as a text string. 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** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMPRODUCT(Complex number1, Complex number2, …, Complex number29)	Returns the product of up to 29 entered **Complex numbers** as a text string. The complex numbers are entered in the form "x + yi" or "x + yj".
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".
IMSEC(Complex Number)	Returns the secant of the entered **Complex Number** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMSECH(Complex Number)	Returns the hyperbolic secant of the entered **Complex Number** as a text string. The complex number is entered in the form "x + yi" or "x + yj".
IMSIN(Complex number)	Returns the sine of the entered **Complex number** as a text string. 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** as a text string. 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** as a text string. 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 as a text string. The complex numbers are entered in the form "x + yi" or "x + yj".
IMSUM(Complex number1, Complex number 2, …, Complex number29)	Returns the sum of up to 29 entered **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** as a text string. The complex number is entered in the form "x + yi" or "x + yj".

Syntax	*Description*
OCT2BIN(Number, Places)	Returns the binary number for the octal value entered. **Number** is the octal number. The number can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement. **Places** (optional) is the number of places to be output.
OCT2DEC(Number)	Returns the decimal number for the octal value entered. **Number** is the octal number. The number can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement.
OCT2HEX(Number, Places)	Returns the hexadecimal number for the octal value entered. **Number** is the octal number. The number can have a maximum of 10 places. The most significant bit is the sign bit, the following bits return the value. Negative numbers are entered as two's complement. **Places** (optional) is the number of places to be output.