Supported functions

The following is information on the supported functions.

Date/Time functions

To use a date/time function, the value **must** be stored as a datetime data type. Datetime data types can be identified by the datetime data type ( ) icon in the header row. If the value is **not** stored as a datetime data type, convert the value to a datetime data type using the **DATEVALUE** function. See the DATEVALUE () section of this article.

DATE

Takes three separate arguments and combines them to form a date in a new DateTime column.

Syntax: `DATE(YEAR, MONTH, DATE)`

- **YEAR** is four-digit value
- **MONTH** is two-digit value
- **DATE** is two-digit value

Example: `DATEADD(@year@, @month@, @day@)`

year	month	day	New Column
1999	5	8	1999-05-08T00:00:00.000Z
1999	6	8	1999-06-08T00:00:00.000Z
1999	7	8	1999-07-08T00:00:00.000Z
1999	8	8	1999-08-08T00:00:00.000Z

Notes on use: Leading zeros for MONTH and DATE are not supported, for example: `DATE(1999,05,08)` should be expressed as `DATE(1999,5,8)`

DATEADD

Calculates the date that is so many days, weeks, months from a given date.

Syntax: `DATEADD(DATETIME, INCREMENT, INTERVAL)`

- **DATETIME** is the date you want to start with.
- **INCREMENT** is the number you provide to be added to the to the **DATETIME**.
- **INTERVAL** is the interval (minutes, days, years, etc.) to add by.
  The following is a list of the recognized values for **INTERVAL**:
  - Years
  - Months
  - Weeks
  - Days
  - Hours
  - Minutes
  - Seconds
  - Millis

Example: `DATEADD(@Date Received@, 6, "months")`

Date Received	New Column
2015-08-24T06:36:33.000Z	2016-02-24T06:36:33.000Z
2011-09-08T07:38:59.000Z	2012-03-08T07:38:59.000Z
2012-09-03T07:13:18.000Z	2013-03-03T07:13:18.000Z

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object. The **INCREMENT** provided **must** be an integer. Millis accepts a maximum of **+/- 2147483647**.

DATEDIFF

Calculates the days, weeks, months between two dates.

Syntax: `DATEDIFF(DATETIME_1, DATETIME_2, INTERVAL)`

- **DATETIME_1** is the date you want to start with.
- **DATETIME_2** is the date you want to end with.
- **INTERVAL** is the interval type (minutes, days, years, etc.) you want returned. The following is a list of the recognized values for the **INTERVAL** value:
    - Years
    - Months
    - Weeks
    - Days
    - Hours
    - Minutes
    - Seconds
    - Millis

Example: `DATEDIFF(@Date Received@, @Date Shipped@, “months”)`

Date Received	Date Shipped	New Column
2015-08-24T06:36:33.000Z	2016-02-24T06:36:33.000Z	6
2011-09-08T07:38:59.000Z	2012-04-08T07:38:59.000Z	7
2012-09-03T07:13:18.000Z	2013-04-03T07:13:18.000Z	7

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object. The **INCREMENT** provided **must** be an integer. Millis accepts a maximum of **+/- 2147483647**.

It is recommended you use the latest datetime value for the **DATETIME_2**. If you enter the earliest date as the **DATETIME_2** value, the **DATEDIFF** function will return a negative number.

DATEDIFF always rounds the result down to the nearest whole number. For example, if the difference between two dates is 3 years and 11 months, the **DATEDIFF** function returns the difference as 3 years.

DATEFORMAT

Converts a value stored as a datetime data type to a text sting in a given format.

Syntax: `DATEFORMAT(DATETIME, FORMAT)`

- **DATETIME** is the date you want to convert.
- **FORMAT** is format you want the **DATETIME** converted to.

Example: `DATEFORMAT(@Date Received@, “dd-MMM-yyyy HH:mm”)`

Date Received	A-Z New Column
2015-08-24T06:36:33.000Z	August 24, 2015
2011-09-08T07:38:59.000Z	September 08, 2011
2012-09-03T07:13:18.000Z	September 03, 2012

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

See the Date and Time Syntax article for more information on the syntax to use for the **DATE_STRING**.

DATETRUNC

Removes the unwanted detail of a timestamp and rounds it to the interval you want. This provides the same output as the SQL DATE_TRUNC() function. Use case: you want to explore trends in your Community user signups and you need to aggregate signup event data by the time each event occurred. You’re only interested in signups by year, month, or day but not the hour, minute, and millisecond. Use DateTrunc to remove the portion of the timestamp that you don't need.

Syntax: `DATETRUNC(x)` where x can be any one of the following arguments.

- **minutes**
- **month**
- **weeks**
- **days**
- **hours**
- **seconds**

Example: `DATETRUNC(@DATE@, "months")`

DATE	New Column
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-01T00:00:00.000Z

DATEVALUE

Converts a datetime text string to a datetime object so you can use it for calculations.

Syntax: `DATEVALUE(DATETIME, FORMAT, TIME_ZONE)`

- **DATETIME** is the datetime as a text string.
- **FORMAT** is format of the **DATETIME**.
- **TIME_ZONE** is the time zone you want associated with the datetime object.

Example: `DATEVALUE(@Date@, "yyyy-MMM-dd hh:mm a", "GMT-05:00")`

A-Z Date	New Column
2015-Aug-24 06:36 AM	2015-08-24T06:36:00.000 -05:00
2011-Sep-08 07:38 AM	2011-09-08T07:38:00.000 -05:00
2012-Sep-03 07:13 AM	2012-09-03T07:13:00.000 -05:00

Notes on use: See the Date and Time Syntax article for more information on the syntax to use for **FORMAT** and **TIME_ZONE**.

Use the **DATEVALUE** function to convert a text column into a date column, or a date that you type into a date object. With the resulting data object, you can use Paxata date functions, for example return the number of days or years between two dates (see next example below for **Date**

Manipulation).
Date objects can store a date, time, or a combination of date and time.

To convert text to a Paxata date object, review specify its format in **Paxata date format syntax**. Repeated characters indicate the length of the field, such as **yyyy** means a 4-digit year. For column "DateCol" with input text that specifies February 28, 2012 as:
`2012/28/02`
Convert to date object:
`DATEVALUE(@DateCol@, "yyyy/dd/MM")`
The date format must match your input data.

If February 28, 2012 looks like:
`2012-15-02`
Use date format:
`"yyyy-dd-MM"`

If February 28, 2012 looks like:
`2-28-12`
Use date format:
`"dd-MM-yy"`

If the time 1:29 pm looks like:
`13:29`
Use time format:
`"HH:mm"`

If the time 1:29 pm looks like:
`01:29PM`
Use time format:
`"hh:mmaa"`

Advanced example: if the input text is a date and time, separated by the letter T, then a time zone:
`2012-02-28T09:29:00-05:00`

For letter characters that literally appear in input text, surround the letter with single straight quotes. Use the following date format:
`"yyyy-MM-dd'T'HH:mm:ssZZ"`

Date manipulation

Use the **DATEDIFF** function to calculate differences in date-time values between two Paxata date objects. Calculate days between August 1, 1998 and a date column:

```
  DATEDIFF(DATEVALUE("01-AUG-1998" ,
"dd-MMM-yyyy"), @MyDate@ , "days")
```

DAY

Extracts the day from a date.

Syntax: `DAY(DATETIME)`

DATETIME is the date you want to extract the day from.

Example: `DAY(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	15
2011-01-23T07:09:58.000Z	23
2011-01-30T07:27:56.000Z	30

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 1 to 31.

DAYOFWEEK

Returns the day of the week from a date.

Syntax: `DAYOFWEEK(DATETIME)`

DATETIME is the date you want to evaluate.

Example: `DAYOFWEEK(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	6
2011-01-23T07:09:58.000Z	7
2011-01-30T07:27:56.000Z	7

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 1 (Monday) to 7 (Sunday).

DAYOFYEAR

Returns the day of the year from a date.

Syntax: `DAYOFYEAR(DATETIME)`

DATETIME is the date you want to evaluate.

Example: `DAYOFYEAR(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	15
2011-01-23T07:09:58.000Z	23
2011-01-30T07:27:56.000Z	30

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range 1 to 365 (366 on a leap year).

ENDOFMONTH

Returns the datetime for the last day of the month in a new DateTime column. This provides the same output as Excel's EOMONTH function.

Syntax: `ENDOFMONTH(DATE_TIME)`

DATE_TIME is a DateTime object.

Example: `ENDOFMONTH(@Date@)`

DATE	New Column
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000
2019-01-10T00:00:00.000Z	2019-01-31T00:00:00.000

FROMUNIXTIME

Returns a date-time object from a Unix timestamp. This provides the same output as the MySQL FROM_UNIXTIME() function.

Syntax: `FROMUNIXTIME(MILLISECONDS)`

MILLISECONDS is the int value represented as milliseconds

Example: `FROMUNIXTIMESTAMP(@UNIX TIME STAMP@)`

UNIX TIME STAMP	New Column
831877766	1970-01-10T15:04:37.766Z
834556166	1970-01-10T15:49:16.166Z
837148166	1970-01-10T16:32:28.166Z

HOUR

Extracts the hour from a time.

Syntax: `HOUR(DATETIME)`

DATETIME is the time you want to extract the hour from.

Example: `HOUR(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	6
2011-01-23T07:09:58.000Z	7
2011-01-30T07:27:56.000Z	7

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 0 (12:00 am) to 23 (11:00 pm).

MAXDATE

Compares two or more dates and returns the latest date in the comparison.

Syntax: `MAXDATE(DATETIME_1, [DATETIME_2, ...])`

- **DATETIME_1** is the first date.
- **DATETIME_2, ... [optional]** are the additional dates.

Example: `MAXDATE(@Target Ship Date@ ,@Date Shipped@)`

Target Ship Date	Date Shipped	New Column
2015-09-23T06:36:33.000Z	2015-09-29T06:36:33.000Z	2015-09-29T06:36...
2011-10-08T07:38:59.000Z	2011-10-16T07:38:59.000Z	2011-10-16T07:38...
2012-10-03T07:13:18.000Z	2012-09-16T07:13:18.000Z	2012-10-03T07:13...

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Here's how the **MAXDATE** function will respond to some common scenarios:

- If only one date is provided, the provided date is returned.
- The time zone of all the dates are temporarily converted to the same time zone to determine the latest date. The conversion is neither a permanent nor a visual transformation.
- Cells with text strings are ignored.
- Blank cells are ignored.
- Cells with errors are ignored.
- If no datetime objects are found, a blank cell is returned.

MIDNIGHT

Resets the given time to midnight (00:00).

Syntax: `MIDNIGHT(DATETIME)`

DATETIME is the time you want to reset.

Example: `MIDNIGHT(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	2011-01-15T00:00:00.000Z
2011-01-23T07:09:58.000Z	2011-01-23T00:00:00.000Z
2011-01-30T07:27:56.000Z	2011-01-30T00:00:00.000Z

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

The time zone isn't affected.

MINDATE

Compares two or more dates and returns the earliest date in the comparison.

Syntax: `MINDATE(DATETIME_1, [DATETIME_2, ...])`

- **DATETIME_1** is the first date.
- **DATETIME_2, ... [optional]** are the additional dates.

Example: `MINDATE(@Target Ship Date@ ,@Date Shipped@)`

Target Ship Date	Ship Date	New Column
2015-09-23T06:36:33.000Z	2015-09-29T06:36:33.000Z	2015-09-23T06:36...
2011-10-08T07:38:59.000Z	2011-10-16T07:38:59.000Z	2011-10-08T07:38...
2012-10-03T07:13:18.000Z	2012-09-16T07:13:18.000Z	2012-09-16T07:13...

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Here's how the **MINDATE** function will respond to some common scenarios:

- If only one date is provided, the provided date is returned.
- The time zone of all the dates are temporarily converted to the same time zone to determine the latest date. The conversion is neither a permanent nor a visual transformation.
- Cells with text strings are ignored.
- Blank cells are ignored.
- Cells with errors are ignored.
- If no datetime objects are found, a blank cell is returned.

MINUTE

Extracts the minute from a time.

Syntax: `MINUTE(DATETIME)`

DATETIME is the time you want to extract the minute from.

Example: `MINUTE(@Date@)`

Date	New Column
2011-01-15T06:37:40.000Z	37
2011-01-23T07:09:58.000Z	9
2011-01-30T07:27:56.000Z	27

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 0 to 59.

MONTH

Extracts the month from a date.

Syntax: `MONTH(DATETIME)`

DATETIME is the date you want to extract the month from.

Example: `MONTH(@Date@)`

Date	123 New Column
2011-03-20T06:03:57.000Z	3
2011-06-25T07:32:34.000Z	6
2012-08-06T08:23:39.000Z	8

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 1 (January) to 12 (December).

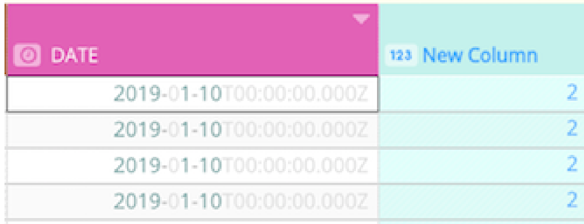
NETWORKDAYS

Returns the number of working days between two date-time objects. This provides the same output as Excel's NETWORKDAYS function.

Syntax: `NETWORKDAYS(DATE_TIME_START, DATE_TIME_END)`

DATE_TIME_START is a date-time object for start date.
DATE_TIME is a date-time object for end date.

Example: `NETWORKDAYS(@DATE@, DATE(2019,1,12))`



DATE	New Column
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2

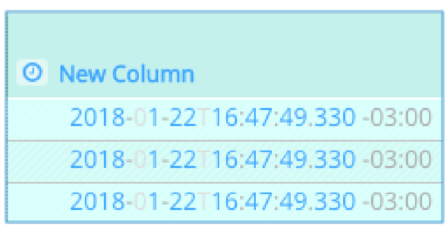
NOW

Returns the current date and time.

Syntax: `NOW(TIME_ZONE)` returns the current date and time.

TIME_ZONE, optional, sets the time zone.

Example: `NOW("GMT-03:00")`



New Column
2018-01-22T16:47:49.330 -03:00
2018-01-22T16:47:49.330 -03:00
2018-01-22T16:47:49.330 -03:00

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

If a time zone is not specified in the function, the returned datetime object will default to Greenwich Mean Time (GMT). See the Date and Time Syntax article for a list of time zones and their appropriate syntax.

QUARTER

Returns the quarter as an integer from a given date-time object.

Syntax: `QUARTER(DDATE_TIME)`

- **DATE_TIME** is a date-time object.

Example: `QUARTER(@DATE@)`

DATE	New Column
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1
2019-01-10T00:00:00.000Z	1

SECOND

Extracts the seconds from a time.

Syntax: `SECOND(DATETIME)`

DATETIME is the time you want to extract the seconds from.

Example: `SECOND(@Date@)`

Date	New Column
2011-03-20T06:03:57.000Z	57
2011-06-25T07:32:34.000Z	34
2012-08-06T08:23:39.000Z	39

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Returned values range from 0 to 59.

SETTIMEZONE

Changes the time zone of a time to the time zone you specify.

Syntax: `SETTIMEZONE(DATETIME, TIME_ZONE)`

- **DATETIME** is the time you want to set the time zone of.
- **TIME_ZONE** is the time zone you want associated with the datetime object.

Example: `SETTIMEZONE(@Date Received@, "GMT-3:00")`

Date Received	New Column
2015-08-24T06:36:33.000Z	2015-08-24T06:36:33.000 -03:00
2011-09-08T07:38:59.000Z	2011-09-08T07:38:59.000 -03:00
2012-09-03T07:13:18.000Z	2012-09-03T07:13:18.000 -03:00

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

The transformation doesn't change the time, it simply assigns a new time zone to the existing time. See the Date and Time Syntax article for a list of time zones and their appropriate syntax.

TODAY

Returns the current date, doesn't include the time.

Syntax: `TODAY()`

Example: `TODAY()`

New Column
2018-01-22T00:00:00.000 -05:00
2018-01-22T00:00:00.000 -05:00
2018-01-22T00:00:00.000 -05:00

WEEKOFYEAR

Returns the week number as an integer from a given date-time object. This provides the same output as Excel's WEEKNUM function.

Syntax: `WEEKOFYEAR(DATE_TIME)`

- **DATE_TIME** is a date-time object.

Example: `WEEKOFYEAR<@DATE@>`

DATE	New Column
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2
2019-01-10T00:00:00.000Z	2

WORKDAY

Returns a number that represents a date that is the indicated number of working days before or after a date (the starting date). Working days exclude weekends and any dates identified as holidays. This provides the same output as Excel's WORKDAY function. Use WORKDAY to exclude weekends or holidays when you calculate invoice due dates, expected delivery times, or the number of days of work performed.

Syntax: `WORKDAY(STARTDATE, DAYS)`

- **STARTDATE** is a date that represents the start date.
- **DAYS** is the number of nonweekend and nonholiday days before or after start date. A positive value for days yields a future date; a negative value yields a past date.

Example: `WORKDAY(@DATE@ ,12)`

DATE	New Column
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z
2019-01-10T00:00:00.000Z	2019-01-28T00:00:00.000Z

YEAR

Extracts the year from a date.

Syntax: `YEAR(DATETIME)`

DATETIME is the date you want to extract the year from.

Example: `Year()`

Date	New Column
2012-07-20T07:15:24.000Z	2012
2016-03-25T07:06:44.000Z	2016
2014-01-30T07:16:25.000Z	2014

Notes on use: The **DATETIME** you provide **must** be a datetime object, a column that contains a datetime object, or a function that returns a datetime object.

Back to contents ()

Informational functions

FIRSTNONBLANK

Compares the values of two or more columns and returns the first non-blank value. This function provides the same output as Excel's FIRSTNONBLANK function.

Syntax: `FIRSTNONBLANK(ARGUMENT_1, [ARGUMENT_2, ...])`

- **ARGUMENT_1** is the first column.
- **ARGUMENT_2, ... [optional]** are the additional columns.

Example: `FIRSTNONBLANK(@Current Employer@, @Previous Employer@, @School@)`

Current Employer	Previous Employer	School	New Column
Banana Inc	Mermaidhut	Camden College	Banana Inc
CloudCo			CloudCo
		Greendale Community College	Greendale Community College
	Bansheelectronis	University of New York	Bansheelectronis
	Pyramidustries		Pyramidustries

Notes on use: If only one column is specified, the value of the provided column is returned.

If no non-blank values are found, the FIRSTNONBLANK function will return an empty cell (unless you include a final argument for what value to display in the output when no non-blank values are found).

ISBLANK

Checks for blank or null values within a specified column. If a blank or null value is found, the value **TRUE** is returned.

Syntax: `ISBLANK(ARGUMENT)`

ARGUMENT is the column to check.

Example: `ISBLANK(@Column@)`

123 Column	New Column
Rufus Daniel	false
	true
	true
1789	false
Bryant Carr	false

ISDATE

Checks for date-time values within a specified column. If a date-time value is found, the value **TRUE** is returned.

Syntax: `ISDATE(ARGUMENT)`

ARGUMENT is the column to check.

Example: `ISDATE(@Column@)`

A-Z Column	New Column
2013-01-02T00:01:...	true
	false
3/6/2014	false

Notes on use: The values **must** be datetime objects, not datetime text strings. Datasets imported from an Excel spreadsheet will automatically import dates as datetime objects. Dates from all other sources need to be converted to a datetime object using the **DATEVALUE** function. See the DATEVALUE () section of this article.

ISNULL

Checks for blanks or null values within a specified column. If a blank or null value is found, the value **TRUE** is returned.

Syntax: `ISNULL(ARGUMENT)`

ARGUMENT is the column to check.

Example: `ISNULL(@Column@)`

Column	New Column
Bryant Carr	false
	true
2013-01-02T00:01:00.000Z	false
Kelli Martinez	false
Rufus Daniel	false
	true

ISNUMBER

Checks for numeric values within a specified column. If a numeric value is found, the value **TRUE** is returned.

Syntax: `ISNUMBER(ARGUMENT)`

ARGUMENT is the column to check.

Example: `ISNUMBER(@Column@)`

Column	New Column
3/6/2014	false
2013-01-02T00:01:...	false
6	true
7	true

ISTEXT

Checks for text within a specified column. If a blank or null is found, the value **TRUE** is returned.

Syntax: `ISTEXT(ARGUMENT)`

ARGUMENT is the column to check.

Example: `ISTEXT(@Column@)`

Column	New Column
2013-01-02T00:01:...	false
Rufus Daniel	true
Kelli Martinez	true
	false
3/6/2014	true

Back to contents ()

Logical functions

AND

Evaluates whether all arguments within an expression evaluate to **TRUE**. If the arguments do evaluate to **TRUE**, the value **TRUE** is returned.

Syntax: `AND(ARGUMENT_1, [ARGUMENT_2, ...])`

- **ARGUMENT_1** is the argument to evaluate.
- **ARGUMENT_2, ... [optional]** are the additional arguments.

Example: `AND(@Column_A@, @Column_B@, @Column_C@)`

Column_A	Column_B	Column_C	New Column
false	true	true	false
true	true	true	true
false	true	false	false
false	false	false	false

Notes on use: The **ARGUMENT**s you provide **must** be either a **TRUE** or **FALSE** value, a column that contains either value, or a function that returns either value.

The **AND** function is case insensitive, so it treats True, TRUE, true the same way. Similarly, False, FALSE, false are treated the same.

IF

Allows you to specify a different output depending on whether or not a given statement is true.

Syntax: `IF(CONDITION, TRUE_VALUE, FALSE_VALUE)`

- **CONDITION** is the expression you want to evaluate.
- **TRUE_VALUE** is the value the function returns if the **CONDITION** is true.
- **FALSE_VALUE** is the value that is returned if the **CONDITION** is **not** true.

Example: `IF(@Current Employer@ = 0, "N/A", @Current Employer@)`

A-Z Current Employer	A-Z New Column
0	N/A
Banana Computers Inc	Banana Computers Inc
0	N/A
Banana	Banana
CloudCo inc	CloudCo inc
Microstuff	Microstuff
0	N/A
0	N/A
Acme	Acme
0	N/A

Notes on use: The **IF** function is ideal in cases where a set of values need to be created based on information in one or more other columns

The **CONDITION must** provide either a **TRUE** or **FALSE** value. Other functions can be incorporated as part of the **CONDITION**. Another **IF** function can be used as one or both of the values. This allows for very fine-grained control over the returned value. In most cases, the **CONDITION** will include an operator, see the Comparison operators () section of this article.

IFERROR

Checks the cell values for errors within a specified column. If an error is found, a value you specify is returned.

Syntax: `IFERROR(ARGUMENT, VALUE)`

- **ARGUMENT** is the column you want to check.
- **VALUE** is the value to return if the column cell contains an error.

Example: `IFERROR(@New Column@, "N/A")`

New Column	New Column (1)
	N/A
Banana Computers Inc	Banana Computers Inc
	N/A
Banana	Banana
CloudCo inc	CloudCo inc
Microstuff	Microstuff
	N/A

Notes on use: The **VALUE**s you provide can be a text string or numeric value, a column that contains a text string or numeric value, or a function that returns a text string or numeric value.

For cells where no error is found, the cells original value is returned.

NOT

Reverses the result of an expression that results in a TRUE or FALSE value.

Syntax: `NOT(ARGUMENT)`

ARGUMENT is the **TRUE** or **FALSE** value you want to reverse.

Example: `NOT(@Column@)`

Column	New Column
false	true
true	false

Notes on use: The **ARGUMENT** you provide **must** be either a **TRUE** or **FALSE** value, a column that contains either value, or a function that returns either value.

The **NOT** function is case insensitive, so it treats True, TRUE, true the same way. Similarly, False, FALSE, false are treated the same.

OR

Determines if at least one value within an expression is **TRUE**. If one value is **TRUE**, the value **TRUE** is returned.

Syntax: `OR(ARGUMENT_1, [ARGUMENT_2, ...])`

- **ARGUMENT_1** is the first argument you want to evaluate.
- **ARGUMENT_2, ... [optional]** are the additional columns.

Example: `OR(@Column_A@, @Column_B@, @Column_C@)`

Column_A	Column_B	Column_C	New Column
false	true	true	true
true	true	true	true
false	true	false	true
false	false	false	false

Notes on use: The **ARGUMENT**s you provide **must** be either a **TRUE** or **FALSE** value, a column that contains either value, or a function that returns either value.

The **OR** function is case insensitive, so it treats True, TRUE, true the same way. Similarly, False, FALSE, false are treated the same.

Back to contents ()

Mathematical functions

To use a math function, the value **must** be stored as a numeric data type. Numeric data types can be identified by the numeric data type (123) icon in the header row. If the value is **not** stored as a numeric data type, convert the value to numeric format using the **VALUE** function. See the VALUE () section of this article.

In addition to the mathematical functions listed in this section, the following standard mathematical operations are supported:

- Multiply, divide a column by any number.
- Add to, subtract from a column by any number
  Examples using column name "Revenue"
  `@Revenue@ * 100`
  `@Revenue@ / 100`
  `@Revenue@ + 100`
  `@Revenue@ - 100`

ABS

Returns the absolute value (ABS) of a real number.

In mathematical notation, absolute values is indicated with a bar on either side. For example, the absolute value of *x* would be written as **|x|**.

Syntax: `ABS(VALUE)`

VALUE is the value you want to find the absolute value of.

Example: `ABS(@Column@)`

123 Column	123 New Column
6	6
14	14
-10	10
-11	11

Notes on use: The **VALUE** you provide **must** be a real number, a column that contains a real number, or a function that returns a real number.

ABS can be thought of as the distance from zero a given number has on a number line. For ABS, positive and negative do not matter. The number's distance from zero is the same, or absolute, regardless of whether the number is to the right of zero (positive) or to the left of zero (negative). In mathematical notation, absolute values is indicated with a bar on either side. For example, the absolute value of *x* would be written as **|x|**.

CEILING

Returns a given number rounded up to whole number.

Syntax: `CEILING(VALUE)`

- **VALUE** is the value you want to round.

Example: `CEILING(@Column A value@)`

123 Column A value	123 New Column
6.588	7
9.43	10
11.345	12
14.796	15
20	20

EXP

Returns the exponential for the specified value.

Syntax: `EXP(NUMBER)`

NUMBER is any real number.

Example: `EXP(@Column A value@)`

Column A value	New Column
6.588	726.3267627508812
9.43	12456.526731608414
11.345	84541.68455061226
14.796	2665760.6580085587
20	485165195.4097903

FACTORIAL

Returns the product of an integer and all the integers below it.

Syntax: `FACTORIAL(NUMBER)`

NUMBER is any real number.

Example: `FACTORIAL(@Column A value@)`

Column A value	New Column
6.588	1464.461395243295301844992
9.43	468217.34343705300007777749
11.345	36056362.17229731793933645445682699
14.796	640499653925.3259018035792303107524
20	2.43290200817664E+18

FLOOR

Returns a given number rounded down to whole number.

Syntax: `FLOOR(VALUE)`

- **VALUE** is the value you want to round.

Example: `FLOOR(@Column A value@)`

Column A value	New Column
6.588	6
9.43	9
11.345	11
14.796	14
20	20

INT

Rounds a real number down to the next integer that is less than or equal to it.

Syntax: `INT(VALUE)`

VALUE is the real number you want to round down.

Example: `INT(@Column@)`

Column	New Column
6.1	6
14.11	14
-9.88	-10
-10.88	-11

Notes on use: The **VALUE** you provide **must** be a real number, a column that contains a real number, or a function that returns a real number.

LN

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904). This provides same output as Excel's LN function.

Syntax: `LN(NUMBER)`

NUMBER is positive real number for which you want the natural logarithm.

Example: `LN(@Column A value@)`

Column A value	New Column
6.588	1.8852498123153938
9.43	2.2438960966453663
11.345	2.428777118231805
14.796	2.6943568739702077
20	2.995732273553991

LOG

Returns the logarithm of a number to the base you specify. This function provides the same output as Excel's Log function.

Syntax: `LOG(NUMBER,BASE)`

NUMBER is positive real number for which you want the natural logarithm.

BASE is the base of the logarithm.

Example: `LOG(@Column A value@,2)`

Column A value	New Column
6.588	2.719840555064268
9.43	3.237257770900372
11.345	3.5039847038976126
14.796	3.8871353004619085
20	4.321928094887363

LOG10

Returns the base-10 logarithm of a number. This function provides the same output as Excel's Log10 function.

Syntax: `LOG10(NUMBER)`

NUMBER is positive real number for which you want the natural logarithm.

Example: `LOG10(@Column A value@)`

Column A value	New Column
6.588	0.8187535904977168
9.43	0.9745116927373284
11.345	1.0548045002209547
14.796	1.1701443226433565
20	1.3010299956639813

MOD

Returns the remainder after number is divided by divisor. The result has the same sign as divisor. This provides same output as Excel's MOD function.

Syntax: `MOD(MUMBER,DIVISOR)`

- **NUMBER** is any real number.
- **DIVISOR** is any real number.

Example: `MOD(@Column A value@,3)`

Column A value	New Column
6.588	0.588
9.43	0.43
11.345	2.345
14.796	2.796
20	2

POWER

A Math/Trigonometric function that computes and returns the result of a number raised to a power. This function provides the same output as Excel's POWER function.

Syntax: `POWER(NUMBER, POWER)`

NUMBER is the base number that is any real number.

POWER is the exponent, any real number, to which the base number is raised.

Example: `POWER(@Column A value@,3)`

Column A value	New Column
6.588	285.930689472
9.43	838.5618069999999
11.345	1460.2038886250002
14.796	3239.1642303359995
20	8000

ROUND

Rounds a number to the number of decimal places you specify.

Syntax: `ROUND(VALUE, PLACES)`

- **VALUE** is the real number you want to round.
- **PLACES** is the number of decimal places to round to.

Example: `ROUND(@Column@, 2)`

Column	New Column
6.31415926535897932384626433	6.31
14.34557519189487725623089073	14.35
-9.62300888156922481138448284	-9.62
-10.62300888156922481138448284	-10.62

Notes on use: The **VALUE** you provide **must** be a real number, a column that contains a real number, or a function that returns a real number.

The **PLACES** value must be a positive integer (not negative or a contain decimal). **VALUE**s with less decimal places than what you specified won't be effected by the **ROUND** function.

ROUNDDOWN

Rounds a number down to the number of decimal places you specify.

Syntax: `ROUNDDOWN(VALUE, PLACES)`

- **VALUE** is the real number you want to round down.
- **PLACES** is the number of decimal places to round down to.

Example: `ROUNDDOWN(@Column@, 3)`

Column	New Column
6.3141592653589793238462643	6.314
14.34557519189487725623089073	14.345
-9.62300888156922481138448284	-9.623
-10.62300888156922481138448284	-10.623

Notes on use: **ROUNDDOWN** is similar to **ROUND** except that it **always** rounds down.

The **VALUE** you provide **must** be a real number, a column that contains a real number, or a function that returns a real number.

The **PLACES** value **must** be a positive integer (**not** negative or contain a decimal). **VALUE**s with less decimal places than what you specified won't be effected by the **ROUNDDOWN** function. If zero decimal places are specified, **ROUNDDOWN** operates like the **INT** function.

ROUNDPERC

Rounds a real percentage value (between -100 and 100) to the number of decimal places you specify, ensuring that values near 0% and ±100% have at least the number of decimal places you specify and as many, but no more, decimal places to distinguish the value from true 0% or ±100% values.

Syntax: `ROUNDPERC(VALUE, PLACES)`

- **VALUE** is the real percentage value you want to round.
- **PLACES** is the minimum number of decimal places to round to.

Example: `ROUNDPERC(@Column@, 2)`

Column	New Column
6.3141592653589793238462643	6.31
14.34557519189487725623089073	14.35
-9.62300888156922481138448284	-9.62
-10.62300888156922481138448284	-10.62

Notes on use: The **VALUE** you provide **must** be a real percentage value between -100 and 100, a column that contains a real percentage value, or a function that returns a real percentage value.

The **PLACES** value must be an integer (no decimals). **VALUE**s with less decimal places than what you specified won't be effected by the **ROUNDPERC** function.

ROUNDUP

Rounds a number up to the number of decimal places you specify.

Syntax: `ROUNDUP(VALUE, PLACES)`

- **VALUE** is the real number you want to round up.
- **PLACES** is the number of decimal places to round up to.

Example: `ROUNDUP(@Column@, 3)`

123 Column	123 New Column
6.3141592653589793238462643	6.315
14.34557519189487725623089073	14.346
-9.62300888156922481138448284	-9.624
-10.62300888156922481138448284	-10.624

Notes on use: **ROUNDUP** is similar to **ROUND** except that it **always** rounds up.

The **VALUE** you provide **must** be a real number, a column that contains a real number, or a function that returns a real number.

The **PLACES** value **must** be a positive integer (**not** negative or a contain decimal).**VALUE**s with less decimal places than what you specified won't be effected by the **ROUNDUP** function.

SIGN

Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative. This provides same output as Excel's SIGN function.

Syntax: `SIGN(NUMBER)`

NUMBER is any real number.

Example: `SIGN(@Column A value@)`

Column A value	New Column
-4.44	-1
9.43	1
-11.345	-1
14.796	1
-20	-1

SQRT

Returns a positive square root. This function provides the same output as Excel's SQRT function.

Syntax: `SQRT(NUMBER)`

NUMBER is any positive number for which you want to calculate the square root.

Example: `SQRT(@Column A value@)`

Column A value	New Column
6.588	2.5667099563448925
9.43	3.0708305065568173
11.345	3.368233958619858
14.796	3.8465569019579053
20	4.47213595499958

SUM

Adds the given numeric values together.

Syntax: `SUM(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `SUM(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
6	14	36	56
6	17	36	59
7	11	24	42
6	13	24	43

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

Back to contents ()

Statistical functions

To use a math function, the value **must** be stored as a numeric data type. Numeric data types can be identified by the numeric data type (123) icon in the header row. If the value is **not** stored as a numeric data type, convert the value to numeric format using the **VALUE** function. See the VALUE () section of this article.

AVERAGE

Calculates the value equal to the sum of a list of numbers divided by the number of items in the list.

Syntax: `AVERAGE(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `AVERAGE(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
5	15	10	10
7	14		10.5
7	21	5	11
5	10	11	8.666666666666...

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

MAX

Returns the greatest (maximum) value from a set of values.

Syntax: `MAX(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `MAX(@Column_A@, @Column_B@, @Column_C@)`

Column_A	Column_B	Column_C	New Column
5	17	7	17
6	17		17
6	10	4	10
7	12	13	13

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

MEDIAN

Returns the number that exists in the middle of a range of numbers ordered from lowest to highest value.

Syntax: `MEDIAN(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `MEDIAN(@colum_A@, @colum_B@, @colum_C@, @colum_C@, @colum_E@)`

Colum A	Column B	Column C	Column D	Column E	New Column
3	4	5	2	10	4
10	4	1	0	0	1
4	2	0	7	0	2
6	1	0	4	4	4
9	3	0	4	3	3
4	4	7	4	6	4
2	3	8	9	10	8
2	3	10	3	1	3

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

In a range with an even set of numbers, the median is the center number, half the numbers are to the right of the value returned and half of the numbers are to the left of the value returned. Where there is no single number in the middle of the range), **MEDIAN** calculates the average on the two numbers on either side of the mid-point.

Note that **MEDIAN** is different than **AVERAGE**. **AVERAGE** is an arithmetic mean—calculated by adding up a set of numbers and then dividing by the number of values in the set. **MEDIAN** simply takes the value at the center of the range. In those number ranges that exhibit a balance in the distribution of values within the collection, the **MEDIAN** and **AVERAGE** calculations may coincide; in skewed distributions, the values will be different.

MIN

Returns the smallest (minimum) value from a set of values.

Syntax: `MIN(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `MIN(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
5	15	10	5
7	14		7
7	21	5	5
5	10	11	5

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

MODE

Returns the value that occurs most frequently in a set of numbers.

Syntax: `MODE(VALUE_1, VALUE_2, [VALUE_3, ...])`

- **VALUE_1** is the first value.
- **VALUE_2** is the second value.
- **VALUE_3, ... [optional]** are the additional values.

Example: `MODE(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
3	9	9	9
8	13	12	8
7	11	11	11
6	19	11	6

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

If multiple numbers have an equal occurrence count greater than one, the value returned is the number (of those that are equal in frequency) that appears first in the set (reading from left to right). If no number appears more than once, the function returns an error.

The most common problem related to **MODE** is when the provided set of numbers has no duplicates. At least one number must appear two or more times for the function to successfully evaluate. If the minimum number of arguments (two) are used, then each argument must evaluate to the same number or an error will occur. As you might expect, larger sets of numbers with a more limited variation will reduce the chance of **MODE** returning an error.

STDEV

Estimates the standard deviation, how much variation from the average, that exists within a sample set of data.

Syntax: `STDEV(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `STDEV(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
0.66	0.28	0.12	0.2773685875028...
0.66	0.34	0.12	0.2715388247255...
0.77	0.22	0.08	0.3647373484212...
0.66	0.26	0.08	0.2968725877094...

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

The standard deviation for data is the square root of its variance. If the set under analysis represents **all** data points (referred to as a population), use **STDEVP** instead.

STDEVP

Estimates the standard deviation, how much variation from the average, that exists within the entire set (population) of data.

Syntax: `STDEVP(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `STDEVP(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
0.66	0.28	0.12	0.2264705033528...
0.66	0.34	0.12	0.2217105219775...
0.77	0.22	0.08	0.2978067979225...
0.66	0.26	0.08	0.2423954528359...

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

If the set under analysis represents only a sample of the data points, use **STDEV** instead.

VAR

Estimates how much dispersion exists (how much the values are spread out) within a samples set of data.

Syntax: `VAR(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `VAR(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
0.55	0.3	0.1	0.0508333333333...
0.55	0.34	0.07	0.0579
0.66	0.34	0	0.1089333333333...
0.66	0.2	0.04	0.1036

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

If the set under analysis represents **all** data points (referred to as a population), use **VARP** instead.

VARP

Estimates how much dispersion exists (how much the values are spread out) within the entire set (population) of data.

Syntax: `VARP(VALUE_1, [VALUE_2, ...])`

- **VALUE_1** is the first value.
- **VALUE_2, ... [optional]** are the additional values.

Example: `VARP(@Column_A@, @Column_B@, @Column_C@)`

123 Column_A	123 Column_B	123 Column_C	123 New Column
0.55	0.3	0.1	0.0338888888888...
0.55	0.34	0.07	0.0386
0.66	0.34	0	0.0726222222222...
0.66	0.2	0.04	0.0690666666666...

Notes on use: The **VALUE** you provide **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

If the set under analysis represents only a sample of data points, use **VAR** instead.

Back to contents ()

Text functions

CHAR

Returns the character for the specified ASCII value.

Syntax: `CHAR(INT)`

- **INT** is the ASCII value for the character to return.

Example: `CHAR(ASCII)`

123 ASCII	A-Z New Column
90	Z
85	U
70	F

CONCATENATE

Use the + operator to combine text. To combine two columns with a hyphen between them:
`@Last@ +"-" + @First@` Combines a series of text strings into a single text string.

Or use the **CONCATENATE** function:
`CONCATENATE(@Last@ , "-" , @First@ )`

Syntax: `CONCATENATE(STRING_1, [STRING_2, ...])`

- **STRING_1** is the first value.
- **STRING_2, ... [optional]** are the additional strings.

Example: `CONCATENATE(@Applicant Last@ ,", ",@Applicant First@ , " of ",@City@)`

A-Z Applicant First	A-Z Applicant Last	A-Z City	A-Z New Column
Maximo	Ehmann	Wolfdale	Ehmann, Maximo of Wolfdale
Velia	Goldman	Wolfdale	Goldman, Velia of Wolfdale
Nick	Walters	Glennville	Walters, Nick of Glennville
Rachel	Ray	Hull	Ray, Rachel of Hull

Notes on use: The **STRING** you provide can be a text string or numeric value, a column that contains a text string or numeric value, or a function that returns a text string or numeric value.

FIND

Returns the numeric position of a text string that is found within another text string.

Syntax: `FIND(STRING_1, STRING_2, [VALUE])`

- **STRING_1** is the string you want to find.
- **STRING_2** is the string you want to search in.
- **VALUE**, optional, is the numeric position in **STRING_2** you want to start your search.

Example: `FIND("Tech",@School@)`

School	New Column
UC Sunnydale	0
Pacific Tech	9
Blue Mountain State	0
South Harmon Institute of Technology	27

Notes on use: The **STRING** you provide **must** be a text string, a column that contains a text string, or a function that returns a text string. Likewise, the **VALUE** must be a numeric value, a column that contains a numeric value, or a function that returns a numeric value.

If **STRING_1** occurs multiple times in **STRING_2**, **FIND** only indicates the position of the first match — **not** any successive matches in the pair.

The **FIND** function is case sensitive, so it treats True, TRUE, true separately.

HASHVALUE

Transforms a text string to make fuzzy matching easier.

Syntax: `HASHVALUE(STRING, OPTION, [VALUE])`

- **STRING** is the string you want to transform.
- **OPTION** is the algorithm to use for the transformation. Available options are:
  - METAPHONE
  - NGRAM
  - FINGERPRINT
- **VALUE**, used with **NGRAM**, specifies the number of ngrams to use.

Example: `HASHVALUE(@Current Employer@, "metaphone")`

Current Employer	New Column
Boogle	PKL
CloudCo inc	KLTKNK
Self	SLF
Banana Inc	PNNNK
Acme inc.	AKMNK

Notes on use: The **STRING** you provide **must** be a text string, a column that contains text strings, or a function that returns a text string. Both **OPTION** and **VALUE** are treated as strings and **must** be surrounded by quotation marks, i.e. `"metaphone"` .

HASHVALUE uses algorithms to generate hashes based on provided string values. The algorithms used are also used by the **Cluster + Edit** column operation to find close matches between values within a column. See the Cluster + Edit article for more information on **METAPHONE**, **NGRAM**, and **FINGERPRINT**.

LEFT

Returns a given number of characters starting from the left-most (beginning) position of a text string.

Syntax: `LEFT(STRING, VALUE)`

- **STRING** is the string you want to search.
- **VALUE** is how many characters to return. The default is **1**.

Example: `LEFT(@School@,4)`

School	New Column
UC Sunnydale	UC S
Pacific Tech	Paci
Blue Mountain State	Blue
South Harmon Institute of Technology	Sout

Notes on use: The **STRING** you provide **must** be a text string, a column that contains text strings, or a function that returns a text string.

LEN

Counts the number of characters in a text string.

Syntax: `LEN(STRING)`

STRING is the text string you want to evaluate.

Example: `LEN(@School@)`

A-Z School	123 New Column
Coolidge College	16
The University of Los Angeles	29
Camden College	14
South Harmon Institute of Technology	36

Notes on use: The **STRING** you provide **must** be a text string, a column that contains text strings, or a function that returns a text string.

LOWER

Converts text in column to all lowercase. Optional argument: locale.

Syntax: `LOWER(STRING, LOCALE)`

- **STRING** is the string or column that you want to convert to lowercase.
- **LOCALE** is the locale, which may need to be specified in order to output required characters for the lowercase.
  Refer to https://www.oracle.com/technetwork/java/javase/java8locales-2095355.html (https://www.oracle.com/technetwork/java/javase/java8locales-2095355.html) for the supported locale values.

Example: `LOWER(@Values@, "tr")`

A-Z Values	A-Z New Column
IAŞLIK	ıaşlık
IAŞLIK	ıaşlık
IAŞLIK	ıaşlık

MID

Returns a given number of characters from the middle of a text sting.

Syntax: `MID(STRING, VALUE_1, VALUE_2)`

- **STRING** is the text string you want to evaluate.
- **VALUE_1** is the start position.
- **VALUE_2** is the number of characters to return.

Example: `MID(@School@,4, 5)`

A-Z School	A-Z New Column
Blue Mountain State	e Mou
Pennbrook University	nbroo
Hillman College	lman

Notes on use: The **STRING** you provide **must** be a text string, a column that contains text strings, or a function that returns a text string. The **VALUE**s provided **must** be a numeric value, a column that contains numeric values, or a function that returns a numeric value.

PADLEFT

Pads a string with a specified character, for the specified number of times. This provides same output as MySQL LPAD.

Syntax: `PADLEFT(STRING, NUMBER, VALUE)`

- **STRING** or column is the value to pad.
- **NUMBER** is the number of times to replace with the VALUE.
- **VALUE** is the literal replacement value.

Example: `PADLEFT(@set@, 10, "-")`

A-Z set4	A-Z New Column
test1	-----test1
test2	-----test2
test3	-----test3

PADRIGHT

Pads a string with a specified character, for the specified number of times. This provides same output as MySQL LPAD and RPAD.

Syntax: `PADRIGHT(STRING, NUMBER, VALUE)`

- **STRING** or column is the value to pad.
- **NUMBER** is the number of times to replace with the VALUE.
- **VALUE** is the literal replacement value.

Example: `PADRIGHT(@set@, 10, "-")`

A-Z set4	A-Z New Column
test1	test1-----
test2	test2-----
test3	test3-----

REGEXP

Executes a search and replace on a text string using regular expression (regex). Note this function is based on Java Regex.

Syntax: `REGEXP(STRING_1, STRING_2, STRING_3)`

- **STRING_1** is the text string you want to search.
- **STRING_2** is the text you are searching for.
- **STRING_3** is the text you want to replace **STRING_2**.

Example: `REGEXP(@School@," ", "_")`

A-Z School	A-Z New Column
Adams College	Adams_College
California University	California_University
Adams College	Adams_College
University of New York	University_of_New_York

Notes on search and replace

The **REGEXP** function takes 3 arguments: a text column or expression, a regular expression (https://en.wikipedia.org/wiki/Regular_expression) to search for, and a value to replace for found items. **STRING_1** must be a text string, a column that contains text strings, or a function that returns a text string. **STRING_2** and **STRING_3** are composed of character combinations that define the search and replace activity.

Warning: In regular expressions, there are 12 characters with special meanings:
\ ^ $. | ? * + ( ) [and open curly brace.
If you want to search for those actual characters and not their special meanings, add a double backslash (not a single backslash) before it. For example, to search for asterisk characters with a regular expression, type "*****" not "*****". To search for a backslash character with a regular expression, type four backslash characters.

For more guidance on Regex pattern matching, refer to
https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

(https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html)

Examples

To replace text **ABC** with **DEF**:
```
REGEXP(@ProductID@ , "ABC", "DEF")
```

To convert space character to hyphen:
```
REGEXP(@ProductID@ , " ", "-")
```

To convert asterisk (special character) to hyphen:
```
REGEXP(@ProductID@ , "\\*", "-")
```

To convert a backlash (special character) to hyphen:
```
REGEXP(@ProductID@ , "\\\\", "-")
```

Supported Extract and Replace patterns:
```
RegexpExtract("replace me", "e m") should be "e m"
RegexpExtract("replace me", "e.?m") should be "e m"
RegexpExtract("replace me", "r.*c") should be "replac"
RegexpExtract("123123456789", "(123)+456(.*)") should be "123123456789"
RegexpExtract("123123456789", "(123)+456(.*)", 0) should be "123123456789"
RegexpExtract("123123456789", "(123)+456(.*)", 1) should be "123"
RegexpExtract("123123456789", "(123)+456(.*)", 2) should be "789"
RegexpExtract("456789", "(123)*456(.*)", 2) should be "789"

RegexpReplace("replace me", "e m", "---") should be "replac---e"
RegexpReplace("replace me", "e.?m", "---") should be "replac---e"
RegexpReplace("replace me", "r.*c", "--") should be "--e me"
RegexpReplace("123123456789", "(123)+456(.*)", "---") should be "---"
RegexpReplace("123123456789", "abc", "---") should be "123123456789"
```

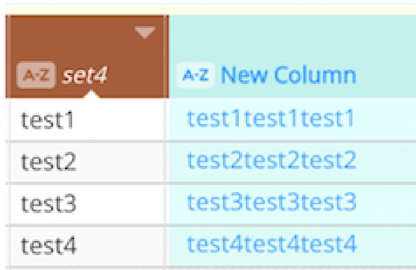
REPEAT

Repeats a specified string N number of times.

Syntax: `REPEAT(VALUE,REPEAT)`

- **VALUE** is the string or column to locate and repeat.
- **REPEAT** is the number of times to repeat the VALUE.

Example: `REPEAT(@set4@, 3)`



set4	New Column
test1	test1test1test1
test2	test2test2test2
test3	test3test3test3
test4	test4test4test4

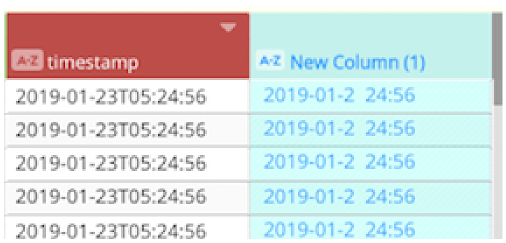
REPLACE

Replaces part of a text string, based on the number of characters you specify, with a different text string.

Syntax: `REPLACE(VALUE, START NUM,NUM CHARS, NEW VALUE)`

- **VALUE** is the text or the column in which you want to replace characters
- **START NUM** is the start position of the character in the VALUE that you want to replace
- **NUM CHARS** is the number of characters in the text that you want to replace with the new string
- **NEW VALUE** is the replacement value. Note this is case-sensitive.

Example: `REPLACE(@timestamp@,10,5," ")`



timestamp	New Column (1)
2019-01-23T05:24:56	2019-01-2 24:56
2019-01-23T05:24:56	2019-01-2 24:56
2019-01-23T05:24:56	2019-01-2 24:56
2019-01-23T05:24:56	2019-01-2 24:56
2019-01-23T05:24:56	2019-01-2 24:56

Notes on use: Use REPLACE when you want to replace any text that occurs in a specific location in a text string; use SUBSTITUTE () when you want to replace specific text in a text string. Example: `REPLACE(@Hospital Name@, Search(@Hospital Name@,"ADVOCATE"), 8, "ALPHA")`

REVERSE

Reverses the specified string.

Syntax: `REVERSE(STRING)`

- **STRING** is the column's value or string to reverse.

Example: `REVERSE(@set4@)`

set4	New Column
test1	1tset
test2	2tset
test3	3tset
test4	4tset

RIGHT

Returns a give number of characters starting from the right-most (end) position of a text string.

Syntax: `RIGHT(STRING, VALUE)`

- **STRING** is the string to search.
- **VALUE** is how many characters to return. The default is **1**.

Example: `RIGHT(@School@,4)`

School	New Column
Pacific Tech	Tech
Grand Lakes University	sity
Coolidge College	lege
South Central Louisiana State University	sity

Notes on use: The **STRING** you provide must be a text string, a column that contains text strings, or a function that returns a text string.

SEARCH

Searches for a specified string and returns the index of the string. If not found, returns value of `-1`.

Syntax: `SEARCH(VALUE, STRING)`

- **VALUE** is the text or the column in which you want to substitute characters
- **STRING** is the string to search.

Example: `SEARCH(@Hospital Name@, "ADVENTIST")`

Hospital Name	New Column (1)
ADVANCED SURGICAL HOSPITAL	-1
ADVENTIST BOLINGBROOK HOSPITAL	1
ADVENTIST GLENOAKS	1
ADVENTIST LA GRANGE MEMORIAL HOSPITAL	1
ADVENTIST MEDICAL CENTER	1
ADVENTIST MEDICAL CENTER	1
ADVENTIST MEDICAL CENTER - REEDLEY	1
ADVOCATE BROMENN MEDICAL CENTER	-1
ADVOCATE CHRIST HOSPITAL & MEDICAL CEN...	-1

Notes on use: **SEARCH** can be combined with REPLACE ().
Example: `REPLACE(@Hospital Name@, Search(@Hospital Name@,"ADVOCATE"), 8, "ALPHA")`

STR

Converts the data in the argument into a text string.

Syntax: `STR(VALUE)`

VALUE is the value you want to convert to a text string.

Example: `STR(@Date@)`

Date	New Column
2016-03-19T00:00:00.000Z	2016-03-19T00:00:00.000Z
2012-06-30T00:00:00.000Z	2012-06-30T00:00:00.000Z
2013-12-28T00:00:00.000Z	2013-12-28T00:00:00.000Z

Notes on use: The **Value** you provide must be a numeric value, a column that contains a numeric value, or a function that returns a numeric value.

This is useful for converting a numeric value into text or for ensuring that a column of mixed text and number values is treated entirely as a column of text so that other text functions can successfully be executed against it.

SUBSTITUE

Substitutes new text for old text in a text string.

Syntax: `SUBSTITUTE(VALUE, OLD TEXT, NEW TEXT)`

- **VALUE** is the text or the column in which you want to substitute characters
- **OLD TEXT** is the text you want to replace. Note this is case-sensitive
- **NEW TEXT** is the text you want to use to replace OLD TEXT. Note this is case-sensitive

Example: `SUBSTITUTE(@Hospital Name@ ,"CREIGHTON","Merton")`

A-Z Hospital Name	A-Z New Column
ALBANY MEMORIAL HOSPITAL	ALBANY MEMORIAL HOSPITAL
ALBANY VA MEDICAL CENTER	ALBANY VA MEDICAL CENTER
ALBEMARLE HOSPITAL AUTHORITY	ALBEMARLE HOSPITAL AUTHORITY
ALBERT EINSTEIN MEDICAL CENTER	ALBERT EINSTEIN MEDICAL CENTER
ALEGENT CREIGHTON HEALTH BERGAN MERCY MEDICAL CTR	ALEGENT MERTON HEALTH BERGAN MERCY MEDICAL ...
ALEGENT CREIGHTON HEALTH CREIGHTON UNIVERSITY MED	ALEGENT MERTON HEALTH MERTON UNIVERSITY MED
ALEGENT CREIGHTON HEALTH IMMANUEL MEDICAL CENTER	ALEGENT MERTON HEALTH IMMANUEL MEDICAL CENT...
ALEGENT CREIGHTON HEALTH LAKESIDE HOSPITAL	ALEGENT MERTON HEALTH LAKESIDE HOSPITAL
ALEGENT CREIGHTON HEALTH MEMORIAL HOSPITAL, SCHUYL	ALEGENT MERTON HEALTH MEMORIAL HOSPITAL, SCH...

Notes on use: Use **SUBSTITUTE** when you want to replace specific text in a text string; use REPLACE () when you want to replace any text that occurs in a specific location in a text string.

TRIM

Removes all leading and trailing spaces for the specified string.
Important note: the TRIM function was designed to trim the 7-bit ASCII space character (value 32) from text. In the Unicode character set, there is an additional space character called the nonbreaking space character that has a decimal value of 160. This character is commonly used in Web pages as the HTML entity, ** **. By itself, the TRIM function does not remove this nonbreaking space character.

Syntax: `TRIM(STRING)`

STRING is the value you want to you want trimmed. Note: column can be specified as the STRING value.

Example: `TRIM(@Company@)`

A-Z Company	A-Z New Column
Apple corp	Apple corp
Apple corporation	Apple corporation
Apple computers	Apple computers

TRIMLEFT

Returns the string stripped of whitespace from the left end of the string.

Syntax: `TRIMLEFT(STRING)`

STRING is the column's values you want to you want trimmed.

Example: `TRIMLEFT(@Company@)`

Company	New Column
Apple corp	Apple corp
Apple corporation	Apple corporation
Apple computers	Apple computers

TRIMRIGHT

Returns the string stripped of whitespace from the right end of the string.

Syntax: `TRIMRIGHT(STRING)`

STRING is the column's values you want to you want trimmed.

Example: `TRIMRIGHT(@Company@)`

Company	New Column
Apple corp	Apple corp
Apple corporation	Apple corporation
Apple computers	Apple computers

UPPER

Converts text in column to all uppercase. Optional argument: locale

Syntax: `UPPER(STRING,LOCALE)`

- **STRING** is the string or column that you want to convert to uppercase.
- **LOCALE** is the locale, which may need to be specified in order to output required characters for the uppercase.
  Refer to https://www.oracle.com/technetwork/java/javase/java8locales-2095355.html (https://www.oracle.com/technetwork/java/javase/java8locales-2095355.html) for the supported locale values.

Example: `UPPER(@Values@, "tr")`

A-Z Values	A-Z New Column
ıaşlık	IAŞLIK
ıaşlık	IAŞLIK
ıaşlık	IAŞLIK

VALUE

Converts numbers stored as a string value into a numeric value.

Syntax: `VALUE(STRING)`

STRING is the numbers, stored as a text string, you want to convert to a numeric value.

Example: `VALUE(@COLUMN@)`

123 Column A value	123 New Column
6.588	1464.461395243295301844992
9.43	468217.34343705300007777749
11.345	36056362.17229731793933645445682699
14.796	640499653925.3259018035792303107524
20	2.43290200817664E+18

Notes on use: The **STRING** you provide must be a number stored as text string, a column that contains a number stored as text string, or a function that returns a number stored as text string.

If **STRING** contains characters other than numbers, the functions returns an error. A single period (decimal point) is allowed within the argument in order to create a real number.

This is useful for converting a text value into numbers to ensure that a column of number values is treated as a column of numbers so that number based functions can successfully be executed against it.

Back to contents ()

Comparison operators

Comparison operators are used to test logical conditions. They are most commonly used within the first argument of the **IF** function in order to generate a **TRUE** or **False** value.

Operators

The following is information on the operators:

Operator	Definition	Example that return true
=	Equal to	1 + 2 = 3
>	Greater than	3 > 2
>=	Greater than or equal to	11 >= 10 11 >= 11
<	Less than	2 < 3
<=	Less than or equal to	10 <= 11 10 <= 10
<>	Not equal to	2 <> 3

"if" statements

The **IF** function takes 3 arguments: a condition that returns true or false, an expression to return if true, and an expression to return if false:

```
IF (@Age@ < 18, "Minor", "Adult")
```

Use with numeric values

Using comparison operators to conduct comparisons between numeric values is straightforward. Bear in mind, however, that the two values to be compared **must** both be of the same data type. The text value "3" is not the same as the numeric value 3.

To safeguard against mixing data types, use the **VALUE** function to convert numbers stored as text to numeric value. For example, "3" = 3 would evaluate to **FALSE**, but VALUE("3") = 3 would evaluate to **TRUE**. See the VALUE () section of this article.

Use with text values

The most commonly used comparison operator with text is = (equals). It is used to determine if two text strings are the same. Note that like other string functions that perform matching (such as FIND), it is case sensitive. In other words, it treats "The" as a different string than "the". For the comparison to be true, the two pieces of text must match EXACTLY—including capitalization. Use of <> (not equal to) follows the same pattern as use of the = (equals). It is also case sensitive when examining text strings.

It may be surprising to note that even comparisons that include < (less than) and > (greater than)—including <= (less than or equals to) and >= (greater than or equals to)—can be used on text values. Without going into detail, characters are represented by a numeric value. Since no two characters are the same, no two characters share the same numeric value.

Predicting the behavior of text comparisons requires some additional information about how printable characters are encoded by computers.

Back to contents ()