**POWER BI FUNCTIONS:**

# Date and time functions:

These functions help you create calculations based on dates and time. Many of the functions in DAX are similar to the Excel date and time functions. However, DAX functions use a **datetime** data type, and can take values from a column as an argument.

**CALENDAR:** Returns a table with a single column named "Date" that contains a contiguous set of dates. The range of dates is from the specified start date to the specified end date, inclusive of those two dates.

Syntax: CALENDAR(<start\_date>, <end\_date>)

calendar = CALENDAR(MIN('P6-SuperStoreUS-2015'[Order Date]),MAX('P6-SuperStoreUS-2015'[Ship Date]))

add new table and write above calculation so that it displays all dates between mentioned days.

**CALENDARAUTO**

Syntax: CALENDARAUTO([fiscal\_year\_end\_month])

Table 3 = CALENDARAUTO(3) --display dates from april 1 --coz fiscal year end month is 3.

Table 3 = CALENDARAUTO() --display min date to max date in your data.

Note: Dates beginning with March 1, 1900 are supported.

**DATE:** Returns the specified date in datetime format.

**Syntax:** DATE(<year>, <month>, <day>)

The following formula returns the date July 8, 2009:

= DATE(2009,7,8)

If **day** is greater than the number of days in the month specified, **day** adds that number of days to the first day in the month. The following formula returns the date February 4, 2008:

o/p:= DATE(2008,1,35)

If you enter an integer larger than 12, the following computation occurs:

The date is calculated by adding the value of month to the year. For example,

if you have DATE( 2008, 18, 1), the function returns a datetime value equivalent to June 1st of 2009,

because 18 months are added to the beginning of 2008 yielding a value of June 2009.

**DATEDIFF:**

Returns the count of interval boundaries crossed between two dates.

**Syntax:** DATEDIFF(<start\_date>, <end\_date>, <interval>)

The interval to use when comparing dates. The value can be one of the following:

- SECOND

- MINUTE

- HOUR

- DAY

- WEEK

- MONTH

- QUARTER

- YEAR

->Measure 4 = DATEDIFF(MIN('P6-SuperStoreUS-2015'[Order Date]),MAX('P6-SuperStoreUS-2015'[Order Date]),day)

[DATEVALUE](https://docs.microsoft.com/en-us/dax/datevalue-function-dax)   
Converts a date in the form of text to a date in datetime format.

DATEVALUE(date\_text)

datevalue---Column = DATEVALUE('P6-SuperStoreUS-2015'[Order Date].[Date])

# DAY

# DAY(<date>) -Returns day of the given date.

=day(today())

= DAY("3-4-1007") O/p:4

= DAY("March 4 2007") o/p: 4

**EDATE:** Returns the date that is the indicated number of months before or after the start date.

Syntax: EDATE(<start\_date>, <months>)

Column 3 = EDATE('P6-SuperStoreUS-2015'[Order Date],-1) gives the same date on before month.

ex: EDATE(2020/sep/10, 1) gives 2020/oct/10

EX: EDATE(2020/sep/10, -1) gives 2020/aug/10

**EOMONTH**

Returns the date in datetime format of the last day of the month, before or after a specified number of months.

Syntax:EOMONTH(<start\_date>, <months>)

Use EOMONTH to calculate maturity dates or due dates that fall on the last day of the month.

Column 4 = EOMONTH('P6-SuperStoreUS-2015'[Order Date],1) gives the last day on nextmonth

EX: EOMONTH(2020/sep/10, 1) gives 2020/oct/31

EOMONTH(2020/sep/10,-1) gives 2020/aug/31

**NOW()** -gives the current datetime

NOW()+3 adds the 3 days to the current date

**Today()** Gives the today date

**year(today())**->2021

**month(today())**->3

**day(today())->**16

**quarter of today-> 1**

**UTCNOW()-** gives the coordinated universal Time

**UTCTODAY()**

**to display the weekday name-->**

Column 5 = FORMAT('P6-SuperStoreUS-2015'[Order Date],"dddd") o/p: sunday,monday...

to display only 3 characters ->FORMAT('P6-SuperStoreUS-2015'[Order Date],"ddd") o/p: sun,mon

**TIME:** The following examples both return the time, 3:00 AM:

= TIME(27,0,0)

= TIME(3,0,0)

# Filter functions :

**ALL()-**Removes all filters everywhere. ALL() can only be used to clear filters but not to return a table

ALL( [<table> | <column>[, <column>[, <column>[,…]]]] )

Ex: ALL =SUMX('P6-SuperStoreUS-2015','P6-SuperStoreUS-2015'[Sales])/SUMX('P6-SuperStoreUS-2015','P6-SuperStoreUS-2015'[Sales])

In the above calculation if any filters applied on the table superstore ALL function will remove all filters applied on the table.

[ALLEXCEPT](https://docs.microsoft.com/en-us/dax/allexcept-function-dax) : Removes all context filters in the table except filters that have been applied to the specified columns.

Syntax:ALLEXCEPT(<table>,<column>[,<column>[,…]])

Ex: ALLEXCEPT= CALCULATE(SUM(ResellerSales\_USD[SalesAmount\_USD]), ALLEXCEPT(DateTime, DateTime[CalendarYear]))

ALL(Table)-- Removes all filters from the specified table. In effect, ALL(Table) returns all of the values in the table

ALL(column[,column[,…]])-Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply.

ALLEXCEPT(Table,column1[,column2…..]])--Removes all context filters in the table except filters that are applied to the specified columns.

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

**ALLNOBLANK(table|column)--** From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist

Syntax: ALLNOBLANKROW( {<table> | <column>[, <column>[, <column>[,…]]]} )

Ex: COUNTROWS(ALLNOBLANKROW('DateTime'))

Gives the count of all rows except blank rows.

= COUNTROWS(ALL('DateTime'))

Gives the count of all rows.

# ALLSELECTED:

**Calculate**->Evaluates an expression in a modified filter context.

Blue Revenue = CALCULATE(

SUM(Sales[Sales Amount]),

'Product'[Color] = "Blue"

)

**CALCULATETABLE:**  
The following example uses the CALCULATETABLE function to get the sum of Internet sales for 2006. This value is later used to calculate the ratio of Internet sales compared to all sales for the year 2006.

= SUMX(

CALCULATETABLE(

'InternetSales\_USD',

'DateTime'[CalendarYear] = 2006

),

[SalesAmount\_USD]

)

**EARLIER:** Returns the current value of the specified column in an outer evaluation pass of the mentioned column.

EARLIER is useful for nested calculations where you want to use a certain value as an input and produce calculations based on that input. In Microsoft Excel, you can do such calculations only within the context of the current row; however, in DAX you can store the value of the input and then make calculation using data from the entire table.

EARLIER is mostly used in the context of calculated columns.

Syntax: EARLIER(<column>, <number>)

SUBCATEGORY SALES RANKING= COUNTROWS(FILTER(ProductSubcategory, EARLIER(ProductSubcategory[TotalSubcategorySales])<ProductSubcategory[TotalSubcategorySales]))+1

**EARLIEST:**

**Filter->**Returns a table that represents a subset of another table or expression.

Syntax: FILTER(<table>,<filter>)

SUMX(FILTER('InternetSales\_USD', RELATED('SalesTerritory'[SalesTerritoryCountry])<>"United States")

,'InternetSales\_USD'[SalesAmount\_USD])

**Keepfilters:** You use KEEPFILTERS within the context CALCULATE and CALCULATETABLE functions, to override the standard behavior of those functions.

**Lookupvalue**

**REMOVE FILTERS:**

**Selectedvalue->**SELECTEDVALUE(<columnName>[, <alternateResult>])

**Information functions:**

**Contains:->** Returns true if values for all referred columns exist, or are contained, in those columns; otherwise, the function returns false.

CONTAINS(<table>, <columnName>, <value>[, <columnName>, <value>]…)

= CONTAINS(InternetSales, [ProductKey], 214, [CustomerKey], 11185)

**Containsrow:**

Except syntax, the **IN** operator and CONTAINSROW function are functionally equivalent.

* + NOT IN is not an operator in DAX. To perform the logical negation of the IN operator, put NOT in front of the entire expression. For example, NOT [Color] IN { "Red", "Yellow", "Blue" }.

->EVALUATE FILTER(ALL(DimProduct[Color]), CONTAINSROW({ "Red", "Yellow", "Blue" }, [Color])) ORDER BY [Color]

->EVALUATE FILTER(ALL(DimProduct[Color]), ([Color]) IN { "Red", "Yellow", "Blue" })ORDER BY [Color]

EVALUATE FILTER(ALL(DimProduct[Color]), NOT CONTAINSROW({ "Red", "Yellow", "Blue" }, [Color]))

ORDER BY [Color]

EVALUATE FILTER(ALL(DimProduct[Color]), NOT [Color] IN { "Red", "Yellow", "Blue" })

ORDER BY [Color]

**CONTAINSSTRING:**

CONTAINSSTRING(<within\_text>, <find\_text>)

EVALUATE

ROW(

"Case 1", CONTAINSSTRING("abcd", "bc"),

"Case 2", CONTAINSSTRING("abcd", "BC"),

"Case 3", CONTAINSSTRING("abcd", "a\*d"),

"Case 4", CONTAINSSTRING("abcd", "ef")

)

Case4 false remaining true

* CONTAINSSTRING is not case-sensitive.
* You can use ? and \* wildcard characters. Use ~ to escape wildcard characters.

**CONTAINSSTRINGEXACT:** CONTAINSSTRINGEXACT is case-sensitive.

CONTAINSSTRINGEXACT(<within\_text>, <find\_text>)

EVALUATE

ROW(

"Case 1", CONTAINSSTRINGEXACT("abcd", "bc"),

"Case 2", CONTAINSSTRINGEXACT("abcd", "BC"),

"Case 3", CONTAINSSTRINGEXACT("abcd", "a\*d"),

"Case 4", CONTAINSSTRINGEXACT("abcd", "ef")

)

Case 1 true, case2 false,case3 false ,case4 false

CUSTOMDATA() :

HASONEFILTER(<columnName>)

HASONEVALUE(<columnName>)

ISBLANK(<value>) Checks whether a value is blank, and returns TRUE or FALSE.

ISCROSSFILTERED(<columnName>)

ISEMPTY

ISERROR

ISEVEN

ISFILTERED(<columnName>)

ISINSCOPE(<columnName>)

ISLOGICAL

ISNONTEXT

ISNUMBER

ISODD  
ISONORAFTER

ISSELECTEDMEASURE

ISSUBTOTAL

ISTEXT

NONVISUAL  
SELECTEDMEASURE

SELETEDMEASUREFORMATSTRING

SELECTEDMEASURENAME

USERNAME  
USEROBJECTID  
USERPRINCIPALNAME

# Logical functions

Logical functions act upon an expression to return information about the values or sets in the expression. For example, you can use the IF function to check the result of an expression and create conditional results.

Logical Operators:

**OR**

Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if both arguments are FALSE.

OR(<logical1>,<logical2>)

-Groups=IF(OR(marks[subjects]=”Math”, Marks[subjects]=”physics”),”group-1”,”group-0”)

-Groups=IF (marks[subjects]=”Math”|| Marks[subjects]=”physics”||marks[subjects]=”computer”,”group-1”,”group-0”)

**AND**

Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE.

Syntax: AND(<logical1>,<logical2>)

When there are multiple filters, they can be evaluated by using the AND (&&) [logical operator](https://docs.microsoft.com/en-us/dax/dax-operator-reference#logical-operators), meaning all conditions must be TRUE, or by the OR (||) logical operator, meaning either condition can be true.

Ex: groups= IF(AND(marks[midterm marks]>15, marks[finalterm marks]>50),”group-1”,”group-0”)

-groups= IF(OR(AND(marks[midterm marks]>15, marks[finalterm marks]>50),marks[subjects]=”math”),”group-1”,”group-0”)

[COALESCE](https://docs.microsoft.com/en-us/dax/coalesce-function-dax) :Returns the first expression that does not evaluate to BLANK. If all expressions evaluate to BLANK, BLANK is returned.

Syntax:COALESCE(<expression>, <expression>[, <expression>]…)

Ex:= COALESCE(SUM(FactInternetSales[SalesAmount]), 0)

Returns the sum of all values in the SalesAmount column in the FactInternetSales table, or 0. This can be used to convert BLANK values of total sales to 0.

[FALSE](https://docs.microsoft.com/en-us/dax/false-function-dax): Returns the logical value FALSE.

Syntax:FALSE()

Ex: = IF(SUM('InternetSales\_USD'[SalesAmount\_USD]) >200000, TRUE(), false())

# IF

:Checks a condition, and returns one value when it's TRUE, otherwise it returns a second value.

Syntax: IF(<logical\_test>, <value\_if\_true>[, <value\_if\_false>])

Ex:Price Group =

IF(

'Product'[List Price] < 500,

"Low",

"High"

)

**Nested IF:**

Price Group =

IF(

'Product'[List Price] < 500,

"Low",

IF(

'Product'[List Price] < 1500,

"Medium",

"High"

)

)

# IF.EAGER

Checks a condition, and returns one value when TRUE, otherwise it returns a second value. It uses an eager execution plan which always executes the branch expressions regardless of the condition expression.

## Syntax

IF.EAGER(<logical\_test>, <value\_if\_true>[, <value\_if\_false>])

# IFERROR

Evaluates an expression and returns a specified value if the expression returns an error; otherwise returns the value of the expression itself.

IFERROR(value, value\_if\_error)

= IFERROR(25/0,9999)

# NOT

# Changes FALSE to TRUE, or TRUE to FALSE.

NOT(<logical>)

**Calculated Column1**, and contains the following formula: = IF([Orders]<300,"true","false")

The formula checks the value in the column, [Orders], and returns "true" if the number of orders is under 300.

Now create a new calculated column, **Calculated Column2**, and type the following formula.

= NOT([CalculatedColumn1])

For each row in **Calculated Column1**, the values "true" and "false" are interpreted as the logical values TRUE or FALSE, and the NOT function returns the logical opposite of that value.

**SWITCH**

Evaluates an expression against a list of values and returns one of multiple possible result expressions.

SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])

Ex:Groups=SWITCH(marks[subjects],

“physics”,”py”,

“chemistry”,”che”,

“math”,”mh”,”unknown”)

**IN Operator**

**Groups=IF(marks[subjects] IN {“computer”,”math”,”physics”},”group-1”,”group-0”)**

TRUE:Returns the logical value TRUE.

TRUE()

= IF(SUM('InternetSales\_USD'[SalesAmount\_USD]) >200000, TRUE(), false())

What is a self service business intelligence?

SSBI is an approach to data analytics that enables business users to filter ,segment , and analyse their data.

Ssbi has made it easier for end users to access their data and create various visuals to get better business insights. Anybody who has basic understanding on data can create visuals and reports dashboards.

2.parts of the Microsoft self service business intelligence are

Excel BI Toolkit: it allows users to create interactive report by importing data from different sources and model data according to report requirement.

Power BI: it is the online solution that enables you to share the interactive reports and queries that you have created using excel BI toolkit.

Diff between excel and power bi toolkit?

Power BI components:



Sales % = SUM('P6-SuperStoreUS-2015'[Sales])/CALCULATE(SUM('P6-SuperStoreUS-2015'[Sales]),ALL('P6-SuperStoreUS-2015'))\*100

What is xVelocity in-memory analytics engine used in power pivot?

The main engine behind power pivot is the xVelocity in-memory analytics engine.

It can handle large amount of data because it stores data in columnar databases, and in memory analytics which results in faster processing of data as it loads all data to RAM memory.

What are some common power query/editor transforms?

Changing data types

Filtering rows

Choosing/removing columns

Grouping

splitting a column into multiple columns

adding new columns

**Table manipulation functions:**

**ADDCOLUMNS**: Adds calculated columns to the given table or table expression.

ADDCOLUMNS FUNCTION = ADDCOLUMNS('P6-SuperStoreUS-2015',"New",SUMX(RELATEDTABLE('P6-SuperStoreUS-2015'),'P6-SuperStoreUS-2015'[Profit]))

**ADDMISSINGITEMS**

Adds rows with empty values to a table returned by [SUMMARIZECOLUMNS](https://docs.microsoft.com/en-us/dax/summarizecolumns-function-dax).

CROSSJOIN

Returns a table that contains the Cartesian product of all rows from all tables in the arguments. The columns in the new table are all the columns in all the argument tables.

CROSSJOIN( Colors, Stationery)

CURRENTGROUP:

Returns a set of rows from the table argument of a [GROUPBY](https://docs.microsoft.com/en-us/dax/groupby-function-dax) expression that belong to the current row of the [GROUPBY](https://docs.microsoft.com/en-us/dax/groupby-function-dax) result.

### Nested IF Condition

Nested IF = IF(Orders[Order Priority] ="Critical", 5, IF(Orders[Order Priority] ="High", 4, IF(Orders[Order Priority] = "Medium", 3, IF(Orders[Order Priority] = "Low", 2, IF(Orders[Order Priority] = "Not Specified", 1)))))

**SUMMARIZE:**

Returns a summary table for the requested totals over a set of groups.

summarize = SUMMARIZE('P6-SuperStoreUS-2015','P6-SuperStoreUS-2015'[Product Category],'P6-SuperStoreUS-2015'[Country],"Sales Amount",SUM('P6-SuperStoreUS-2015'[Sales]),"profit amount",SUM('P6-SuperStoreUS-2015'[Profit]))

Note: To Add apply button to slicer or filters below steps:

File-options and settings-Options-Query Reduction- slicers- or filters click on the add apply buttons.

Link for summarize behavior

https://www.sqlbi.com/articles/all-the-secrets-of-summarize/

# Text functions:

Data Analysis Expressions (DAX) includes a set of text functions based on the library of string functions in Excel, but which have been modified to work with tables and columns in tabular models. This section describes text functions available in the DAX language.

# COMBINEVALUES

Joins two or more text strings into one text string. The primary purpose of this function is to support multi-column relationships in DirectQuery models.

COMBINEVALUES(<delimiter>, <expression>, <expression>[, <expression>]…)

Ex: EVALUATE DISTINCT(SELECTCOLUMNS(DimDate, "Month", COMBINEVALUES(",", [MonthName], [CalendarYear])))

o/p: January,2007

# CONCATENATE

Joins two text strings into one text string.

CONCATENATE(<text1>, <text2>)

ex:= CONCATENATE('Products'[Product abbreviation],'Products'[Product number])

ex:= [Product abbreviation] & "-" & [Product number]

## Conditional Concatenation of Strings in Columns

= CONCATENATE( [FirstName]&" ", CONCATENATE( IF( LEN([MiddleName])>1, LEFT([MiddleName],1)&" ", ""), [LastName]))

# CONCATENATEX

Concatenates the result of an expression evaluated for each row in a table.

# CONCATENATEX(<table>, <expression>, [delimiter])

CONCATENATEX(Employees, [FirstName] & " " & [LastName], ",")

# EXACT

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. EXACT is case-sensitive but ignores formatting differences. You can use EXACT to test text being entered into a document.

EXACT(<text1>,<text2>)

The following formula checks the value of Column1 for the current row against the value of Column2 for the current row, and returns TRUE if they are the same, and returns FALSE if they are different.

= EXACT([Column1],[Column2])

# FIND

Returns the starting position of one text string within another text string. FIND is case-sensitive.

FIND(<find\_text>, <within\_text>[, [<start\_num>][, <NotFoundValue>]])

= FIND("BMX","line of BMX racing goods")

# FIXED

Rounds a number to the specified number of decimals and returns the result as text. You can specify that the result be returned with or without commas.

FIXED(<number>, <decimals>, <no\_commas>)

# FORMAT

Converts a value to text according to the specified format.

FORMAT(<value>, <format\_string>)

= FORMAT( 12345.67, "General Number")

= FORMAT( 12345.67, "Currency")

= FORMAT( 12345.67, "Fixed")

= FORMAT( 12345.67, "Standard")

= FORMAT( 12345.67, "Percent")

= FORMAT( 12345.67, "Scientific")

**12345.67** "General Number" displays the number with no formatting.

**$12,345.67** "Currency" displays the number with your currency locale formatting. The sample here shows the default United States currency formatting.

**12345.67** "Fixed" displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator.

**12,345.67** "Standard" displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator, and includes thousand separators. The sample here shows the default United States number formatting.

**1,234,567.00 %** "Percent" displays the number as a percentage (multiplied by 100) with formatting and the percent sign at the right of the number separated by a single space.

**1.23E+04** "Scientific" displays the number in scientific notation with two decimal digits.

## Predefined numeric formats

"General Number"

"Currency"

"Fixed"

"Standard"

"Percent"

"Scientific"

"Yes/No"

"True/False"

"On/Off"

--------------------50% DONE DIG MORE------------------------------------------

# LEFT

Returns the specified number of characters from the start of a text string.

LEFT(<text>, <num\_chars>)

=LEFT(“STRATAPPS”,4)

o/p:STRA

# LEN

Returns the number of characters in a text string.

LEN(<text>)

LEN(“pallavi”)+LEN(“reddy”)

o/p:12

# LOWER

Converts all letters in a text string to lowercase.

LOWER(<text>)

= LOWER("123ABC") returns **123abc**.

# MID

Returns a string of characters from the middle of a text string, given a starting position and length.

MID(<text>, <start\_num>, <num\_chars>)

MID("abcde",2,3))

Returns **"bcd"**.

# REPLACE

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

REPLACE(<old\_text>, <start\_num>, <num\_chars>, <new\_text>)

= REPLACE(“stratapps”1,5,"St")

o/p: Stapps

# REPT

Repeats text a given number of times. Use REPT to fill a cell with a number of instances of a text string.

REPT(<text>, <num\_times>)

= REPT(“pallavi”,2)

o/p:pallavipallavi

# RIGHT

RIGHT returns the last character or characters in a text string, based on the number of characters you specify.

RIGHT(<text>, <num\_chars>)

= RIGHT(“pallavi”,2)

o/p:vi

# SEARCH

Returns the number of the character at which a specific character or text string is first found, reading left to right. Search is case-insensitive and accent sensitive.

SEARCH(<find\_text>, <within\_text>[, [<start\_num>][, <NotFoundValue>]])

= SEARCH("n","printer")

The formula returns 4 because "n" is the fourth character in the word "printer."

The following formula finds the position of the character "-" within the column, and returns -1 if the string is not found.

= IFERROR(SEARCH("-",[PostalCode]),-1)

# SUBSTITUTE

Replaces existing text with new text in a text string.

SUBSTITUTE(<text>, <old\_text>, <new\_text>, <instance\_num>)

Use the SUBSTITUTE function when you want to replace specific text in a text string; use the REPLACE function when you want to replace any text of variable length that occurs in a specific location in a text string.

The following formula creates a copy of the column [Product Code] that substitutes the new product code **NW** for the old product code **PA** wherever it occurs in the column.

= SUBSTITUTE([Product Code], "NW", "PA")

# TRIM

Removes all spaces from text except for single spaces between words.

TRIM(<text>)

= TRIM("A column with trailing spaces. ")

# UNICHAR

Returns the Unicode character referenced by the numeric value.

UNICHAR(number)

= UNICHAR(65) o/p: A

= UNICHAR(9733) o/p: ★ character

# UNICODE

Returns the number (code point) corresponding to the first character of the text.

UNICODE( “A” ) o/p:65

# UPPER

# UPPER

Converts a text string to all uppercase letters.

UPPER (<text>)

UPPER(“pallavi”) o/p:PALLAVI

# VALUE

Converts a text string that represents a number to a number.

VALUE(<text>)

= VALUE("3") o/p: numeric 3

# Other functions:

These functions perform unique actions that cannot be defined by any of the categories.

[BLANK](https://docs.microsoft.com/en-us/dax/blank-function-dax) returns a blank

BLANK()

The following example illustrates how you can work with blanks in formulas. The formula calculates the ratio of sales between the Resellers and the Internet channels. However, before attempting to calculate the ratio the denominator should be checked for zero values. If the denominator is zero then a blank value should be returned; otherwise, the ratio is calculated.

= IF( SUM(InternetSales\_USD[SalesAmount\_USD])= 0 , BLANK() , SUM(ResellerSales\_USD[SalesAmount\_USD])/SUM(InternetSales\_USD[SalesAmount\_USD]) )

[ERROR](https://docs.microsoft.com/en-us/dax/error-function) raises an error with an wrror message.

ERROR(<text>)

Ex:DEFINE

MEASURE DimProduct[Measure] =

IF(

SELECTEDVALUE(DimProduct[Color]) = "Red",

ERROR("red color encountered"),

SELECTEDVALUE(DimProduct[Color])

)

EVALUATE SUMMARIZECOLUMNS(DimProduct[Color], "Measure", [Measure])

ORDER BY [Color]

Fails and raises and error message containing "red color encountered".

# Parent and Child functions:

These functions manage data that is presented as parent/child hierarchies.

[PATH](https://docs.microsoft.com/en-us/dax/path-function-dax): Returns a delimited text string with the identifiers of all the parents of the current identifier.

The following example creates a calculated column that lists all the managers for each employee.

= PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey])

[PATHCONTAINS](https://docs.microsoft.com/en-us/dax/pathcontains-function-dax) : Returns TRUE if the specified item exists within the specified path.

PATHCONTAINS(<path>, <item>)

The following example creates a calculated column that takes a manager ID and checks a set of employees. If the manager ID is among the list of managers returned by the PATH function, the PATHCONTAINS function returns true; otherwise it returns false.

= PATHCONTAINS(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), "23")

[PATHITEM](https://docs.microsoft.com/en-us/dax/pathitem-function-dax) : Returns the item at the specified position from a string resulting from evaluation of a PATH function.

PATHITEM(<path>, <position>[, <type>])

The following example returns the third tier manager of the current employee; it takes the employee and manager IDs as the input to a PATH function that returns a string with the hierarchy of parents to current employee. From that string PATHITEM returns the third entry as an integer.

= PATHITEM(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), 3, 1)

[PATHITEMREVERSE](https://docs.microsoft.com/en-us/dax/pathitemreverse-function-dax): Returns the item at the specified position from a string resulting from evaluation of a PATH function.

PATHITEMREVERSE(<path>, <position>[, <type>])

<type>:

0-Results are returned with the data type text. (default).

1-Results are returned as integers.

The following example takes an employee ID column as the input to a PATH function, and reverses the list of grandparent elements that are returned. The position specified is 3 and the return type is 1; therefore, the PATHITEMREVERSE function returns an integer representing the manager two levels up from the employee.

= PATHITEMREVERSE(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), 3, 1)

[PATHLENGTH](https://docs.microsoft.com/en-us/dax/pathlength-function-dax) : Returns the number of parents to the specified item in a given PATH result, including self.

PATHLENGTH(<path>)

The following example takes an employee ID as input to a PATH function and returns a list of the managers above that employee in the hierarchy, The PATHLENGTH function takes that result and counts the different levels of employees and managers, including the employee you started with.

= PATHLENGTH(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]))

# Relationship functions:

Functions in this category are for managing and utilizing relationships between tables

# CROSSFILTER

Specifies the cross-filtering direction to be used in a calculation for a relationship that exists between two columns.

CROSSFILTER(<columnName1>, <columnName2>, <direction>)

# RELATED

Returns a related value from another table.

RELATED(<column>)

# RELATEDTABLE

Evaluates a table expression in a context modified by the given filters.

RELATEDTABLE(<tableName>)

= SUMX( RELATEDTABLE('InternetSales\_USD')

, [SalesAmount\_USD])

# USERELATIONSHIP

Specifies the relationship to be used in a specific calculation as the one that exists between columnName1 and columnName2.

USERELATIONSHIP(<columnName1>,<columnName2>)

To calculate the sum of internet sales and allow slicing by ShippingDate instead of the traditional OrderDate, create measure, [InternetSales by ShippingDate] using the following expression:

= CALCULATE(SUM(InternetSales[SalesAmount]), USERELATIONSHIP(InternetSales[ShippingDate], DateTime[Date]))

# Math and Trig functions:

The mathematical functions in Data Analysis Expressions (DAX) are very similar to the Excel mathematical and trigonometric functions. This section lists the mathematical functions provided by DAX.

# ABS

Returns the absolute value of a number.

ABS(<number>)

ABS(-34) o/p;34

# ACOS

Returns the arccosine, or inverse cosine, of a number. The arccosine is the angle whose cosine is number. The returned angle is given in radians in the range 0 (zero) to pi.

ACOS(number) - The cosine of the angle you want and must be from -1 to 1.

= ACOS(-0.5) o/p:2.09

# ACOSH

Returns the inverse hyperbolic cosine of a number. The number must be greater than or equal to 1. The inverse hyperbolic cosine is the value whose hyperbolic cosine is *number*, so ACOSH(COSH(number)) equals number.

ACOSH(number)

**Time Intelligence Functions:**

**DATEADD**: Returns a table that contains a column of dates, shifted either forward or backward in time by the specified number of intervals from the dates in the current context.

Dax Syntax; DATEADD(<dates>,<number\_of\_intervals>,<interval>)

Ex: DATEADD(DateTime[DateKey],-1,year)

**DATESBETWEEN**: Returns a table that contains a column of dates that begins with a specified start date and continues until a specified end date.

This function is suited to pass as a filter to the [CALCULATE](https://docs.microsoft.com/en-us/dax/calculate-function-dax) function. Use it to filter an expression by a custom date range.

Dax Syntax: DATESBETWEEN(<dates>, <start\_date>, <end\_date>)

EX: Customers LTD =

CALCULATE(

DISTINCTCOUNT(Sales[CustomerKey]),

DATESBETWEEN(

'Date'[Date],

BLANK(),

MAX('Date'[Date])

)

)

**DATESINPERIOD**: Returns a table that contains a column of dates that begins with a specified start date and continues for the specified number and type of date intervals.

This function is suited to pass as a filter to the [CALCULATE](https://docs.microsoft.com/en-us/dax/calculate-function-dax) function. Use it to filter an expression by standard date intervals such as days, months, quarters, or years.

Dax Syntax: DATESINPERIOD(<dates>, <start\_date>, <number\_of\_intervals>, <interval>)

EX:

Revenue PY =

CALCULATE(

SUM(Sales[Sales Amount]),

DATESINPERIOD(

'Date'[Date],

MAX('Date'[Date]),

-1,

YEAR

)

)

**DATESMTD**: Returns a table that contains a column of the dates for the month to date, in the current context.

Dax Syntax:DATESMTD(<dates>)

Ex:

CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), DATESMTD(DateTime[DateKey]))

**DATESQTD**: Returns a table that contains a column of the dates for the quarter to date, in the current context.

Dax Syntax:DATESQTD(<dates>)

Ex: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), DATESQTD(DateTime[DateKey]))

**DATESYTD:** Returns a table that contains a column of the dates for the year to date, in the current context.

Dax Syntax: DATESYTD(<dates> [,<year\_end\_date>])

Ex: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), DATESYTD(DateTime[DateKey]))

**FIRSTDATE:** Returns the first date in the current context for the specified column of dates.

Dax Syntax: FIRSTDATE(<dates>)

Ex:

FIRSTDATE('InternetSales\_USD'[SaleDateKey])

**LASTDATE**: Returns the last date in the current context for the specified column of dates.

Dax Syntax: LASTDATE(<dates>)

EX:LASTDATE('InternetSales\_USD'[SaleDateKey])

**NEXTDAY**: Returns a table that contains a column of all dates from the next day, based on the first date specified in the **dates** column in the current context.

Dax Syntax: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), NEXTDAY('DateTime'[DateKey]))

**NEXTQUARTER:** Returns a table that contains a column of all dates in the next quarter, based on the first date specified in the **dates** column, in the current context.

Dax Syntax: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), NEXTQUARTER('DateTime'[DateKey]))

**NEXTMONTH:** Returns a table that contains a column of all dates from the next month, based on the first date in the **dates** column in the current context.

Dax Syntax: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), NEXTMONTH('DateTime'[DateKey]))

**NEXTYEAR:** Returns a table that contains a column of all dates in the next year, based on the first date in the **dates** column, in the current context.

Syn: NEXTYEAR(<dates>[,<year\_end\_date>])

**PARALLELPERIOD:** Returns a table that contains a column of dates that represents a period parallel to the dates in the specified **dates** column, in the current context, with the dates shifted a number of intervals either forward in time or back in time.

Syn: PARALLELPERIOD(<dates>,<number\_of\_intervals>,<interval>)

EX: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), PARALLELPERIOD(DateTime[DateKey],-1,year))

**PREVIOUSDAY:** Returns a table that contains a column of all dates representing the day that is previous to the first date in the **dates** column, in the current context.

Syn: PREVIOUSDAY(<dates>)

EX: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), PREVIOUSDAY('DateTime'[DateKey]))

**PREVIOUDQUARTER:** Returns a table that contains a column of all dates from the previous quarter, based on the first date in the **dates** column, in the current context.

Syn: PREVIOUSQUARTER(<dates>)

Ex: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), PREVIOUSQUARTER('DateTime'[DateKey]))

**PREVIOUSMONTH:** Returns a table that contains a column of all dates from the previous month, based on the first date in the **dates** column, in the current context.

**Syn:** PREVIOUSMONTH(<dates>)

EX: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), PREVIOUSMONTH('DateTime'[DateKey]))

**PREVIOUSYEAR:** Returns a table that contains a column of all dates from the previous year, given the last date in the **dates** column, in the current context.

Syn: PREVIOUSYEAR(<dates>[,<year\_end\_date>])

EX: CALCULATE(SUM(InternetSales\_USD[SalesAmount\_USD]), PREVIOUSYEAR('DateTime'[DateKey]))

**STARTOFMONTH:** Returns the first date of the month in the current context for the specified column of dates.

Syn: STARTOFMONTH(<dates>)

Ex: STARTOFMONTH(DateTime[DateKey])

**STARTOFYEAR:** Returns the first date of the year in the current context for the specified column of dates.

Syn: STARTOFYEAR(<dates>)

Ex: STARTOFYEAR(DateTime[DateKey])

**STARTOFQUARTER:** A table containing a single column and single row with a date value.

Syn: STARTOFYEAR(<dates>)

**Ex:** STARTOFYEAR(DateTime[DateKey])

**TOTALMTD:** Evaluates the value of the **expression** for the month to date, in the current context.

Syn: TOTALMTD(<expression>,<dates>[,<filter>])

Ex: TOTALMTD(SUM(InternetSales\_USD[SalesAmount\_USD]),DateTime[DateKey])

**TOTALQTD:** Evaluates the value of the **expression** for the dates in the quarter to date, in the current context.

Syn: TOTALQTD(<expression>,<dates>[,<filter>])

EX: TOTALQTD(SUM(InternetSales\_USD[SalesAmount\_USD]),DateTime[DateKey])

**TOTALYTD:** Evaluates the year-to-date value of the **expression** in the current context.

Syn: TOTALYTD(<expression>,<dates>[,<filter>][,<year\_end\_date>])

**Ex:** TOTALYTD(SUM(InternetSales\_USD[SalesAmount\_USD]),DateTime[DateKey])