DAX for Power BI

Sample manual - first two chapters

WiseOwl Training

Manual 1159 - 128 pages –

TABLE OF CONTENTS (1 of 4)

TABLE OF CONTENTS (2 of 4)

TABLE OF CONTENTS (3 of 4)

TABLE OF CONTENTS (4 of 4)

CHAPTER 1 - GETTING STARTED

1.1 Introducing DAX

The *DAX* language (standing loosely for *Data Analysis eXpressions*) allows you to create calculated columns, measures and queries (an example of each is shown in this section).

Where is DAX used?

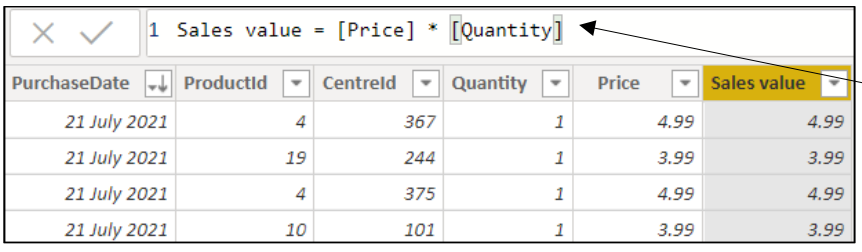
You can write DAX within the following programs:

Program	Notes
Power BI	Power BI is a standalone application which allows you to create business intelligence reports, and publish them to a website or server.
PowerPivot	PowerPivot is an add-in within Excel which allows you to combine data from multiple data sources, and present this in a pivot table.
SQL Server Analysis Services (Tabular)	SSAS Tabular allows you to combine data from lots of different data sources, apply security to it to control who sees what and then allow employees of your organisation to share the resulting data model.

DAX initially looks similar to Excel, but you will quickly realise that it is actually very different!

How DAX is Used 1 - Calculated Columns

A *calculated column* is like a formula in Excel:



This DAX calculated column gives the sales value for each row of a purchases table, by multiplying the quantity of items bought by the price paid.

As we will see in this manual, a calculated column is evaluated for each row of a table; DAX creates a *row context* for each separate calculation.

How DAX is Used 2 – Measures

Much of this courseware will be devoted to creating *measures* like this one, to calculate the value of sales for any region, year or other constraint:

```
Total sales = SUMX(

    // sum the value of all sales for each
    // cell in the current filter context
    Purchase,
    [Price] * [Quantity]
)
```

RegionName	Air	Land	Water	Total
East Anglia	1,268.71	13,640.28	2,050.91	**16,959.90**
East Midlands	2,609.87	26,289.48	5,033.67	**33,933.02**
London	4,899.45	54,018.62	9,136.50	**68,054.57**
North	2,874.62	25,980.55	4,334.49	**33,189.66**
North West	7,950.17	69,751.18	12,614.71	**90,316.06**
South East	10,697.97	102,171.04	17,507.12	**130,376.13**
South West	2,616.48	23,769.80	4,506.69	**30,892.97**
West Midlands	4,702.33	49,116.82	8,878.73	**62,697.88**
Yorkshire & Humberside	6,231.78	47,396.27	8,509.78	**62,137.83**
Total	**43,851.38**	**412,134.04**	**72,572.60**	**528,558.02**

Measures use the same language, but are usually more complicated (and always involve some aggregation).

This measure is showing that total sales for the **East Anglia** region and **Air** environment are £1,268.71. As we will see, each measure is calculated for a particular combination of constraints called the *filter context*.

How DAX is Used 3 – Queries

As well as for creating calculated columns and measures, you can also use DAX to create queries to get data out of a Power BI, PowerPivot or SSAS Tabular data model:

```
// list out the regions
EVALUATE
    Region
ORDER BY
    Region[RegionName]
```

145 %

Results

RegionId	RegionName
1	East Anglia
2	East Midlands
3	London
4	North
5	North West
6	South East
7	South West
8	West Midlands
9	Yorkshire & Humberside

The DAX query language is similar to SQL, but whereas SQL is used to get data out of a relational database, DAX is used to get data out of a data model.

This query lists out all of the regions in a data model, alphabetically by region name.

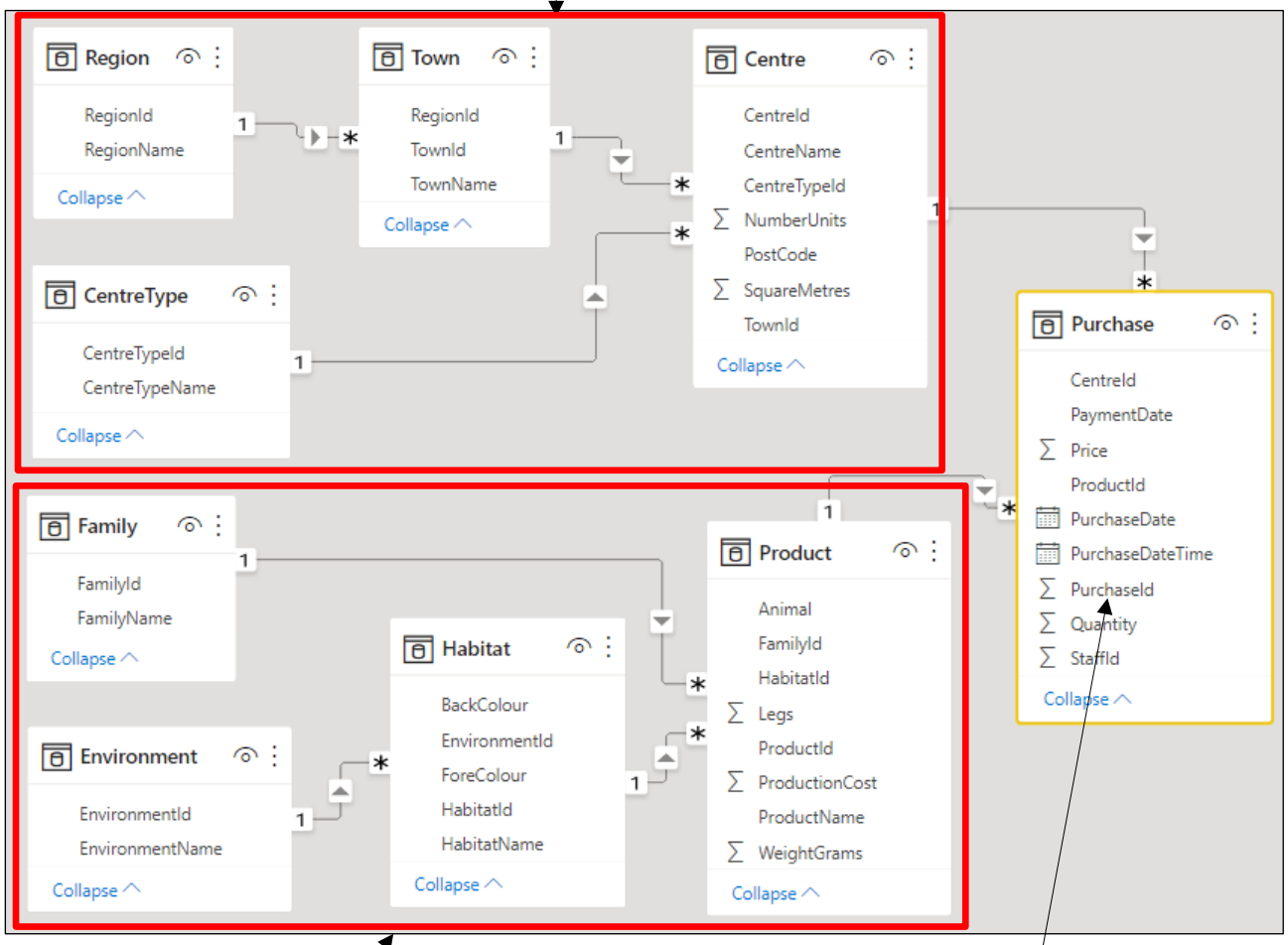
1.2 The Construct-a-Creature Database

This courseware uses data from the (fictitious) Wise Owl subsidiary *Construct-a-Creature* (a retail chain loosely modelled on *Build-a-Bear*, but with a wider range of animals available for purchase).

The Database Tables and Relationships

Here are the tables in the *Construct-a-Creature* (*CAC*) database:

There are four geographical tables to do with where purchases took place, giving the shopping centre, type of shopping centre, town and region.



There are also four tables to do with the biological classification of each animal (for example, a frog is an amphibian which lives in a fresh water habitat in a watery environment).

The purchases table specifies how many of each product were bought in each location in each transaction.

1.3 Column Storage

To understand how DAX works in Power BI, and how to tweak it, it's vital to understand how the underlying engine (called either *VertiPaq* or *xVelocity*, depending on what you read) stores data.

Row versus Column Storage

Most databases (such as SQL Server) use a row-based storage algorithm:

Typically each row in a table is stored as a record, and is accessed by its primary key (unique identifier).

ProductId	ProductName	Animal	HabitatId	Legs	FamilyId
1	Sammy	Snake	1	0	1
2	Pokyo	Penguin	4	2	3
3	Fenella	Frog	3	4	4
4	Layla	Lemur	2	2	5

Power BI models, by contrast, store data by column:

In Power BI columns are stored separately, which makes any calculation summing or otherwise aggregating this column run much more quickly.

ProductId	ProductName	Animal	HabitatId	Legs	FamilyId
1	Sammy	Snake	1	0	1
2	Pokyo	Penguin	4	2	3
3	Fenella	Frog	3	4	4
4	Layla	Lemur	2	2	5
5	Dave	Dachsund	1	4	5
6	Kylie	Camel	5	4	5
7	Jeremy	Jackdaw	7	2	3
8	Faye	Fox	6	4	5

Data Compression

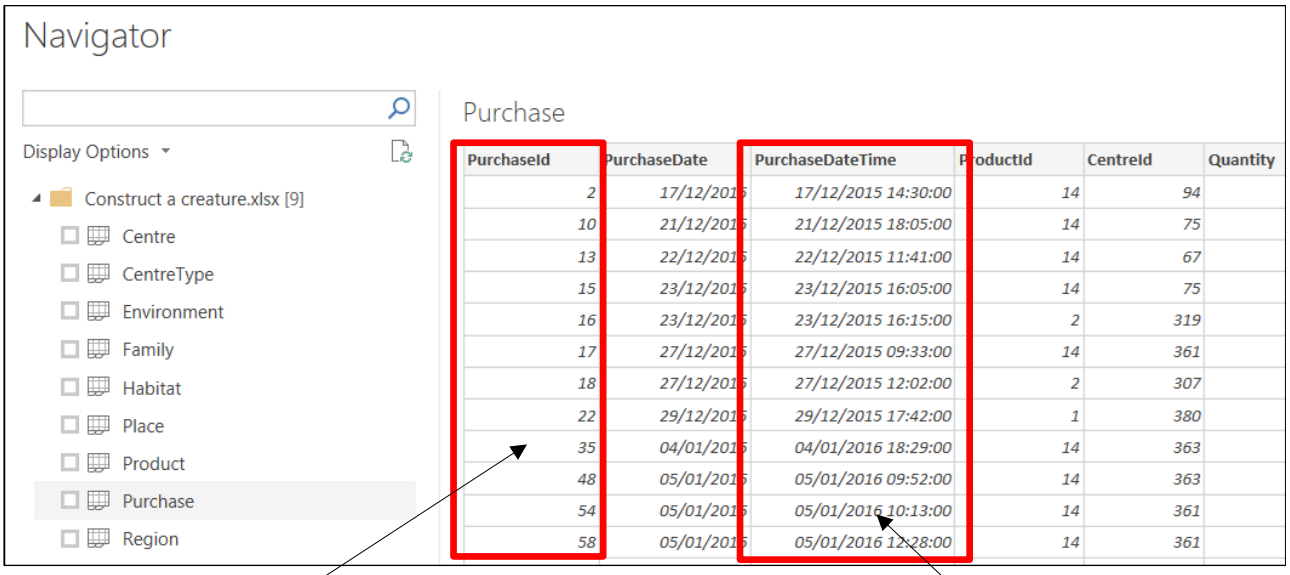
Duplicate column values are only ever stored once. Thus the **FamilyId** and **Legs** columns above might be stored something like this:

Column	Dictionary	Values
FamilyId	1,2,3,4,5,6	0,2,3,4,4,4,2,4,2,1,4,5,4,2,0,3,4,4,4
Legs	0,2,4,6	0,1,2,1,2,2,1,2,1,0,2,3,2,1,0,2,2,2,2

This shows that the lower the cardinality *of a column (ie the smaller the number of distinct values there are, and hence the more duplication there is), the more efficiently the data will be stored.*

Implications for Loading Data

What column storage implies is that you should avoid loading columns with high cardinality (that is, with very little repeated data) unless you need them:



You should avoid loading the **PurchaseId** column. It isn't used to link to any other table, and it has the highest possible cardinality (each number is unique) so will take up a lot of memory.

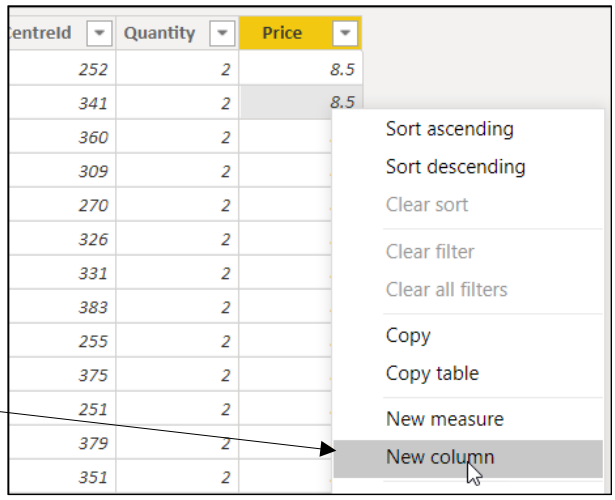
The other column to avoid loading, since each time of day is stored as a separate number internally (unless, of course, you want to analyse purchases by the time of day when they occurred).

*Note that for the example above you have to import the **ProductId** and **CentreId** columns because they are used to link to other tables.*

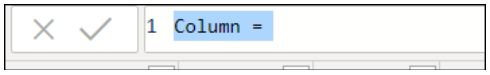
CHAPTER 2 - WRITING DAX

2.1 Calculated Columns

The simplest way to write DAX is as a *calculated column*:

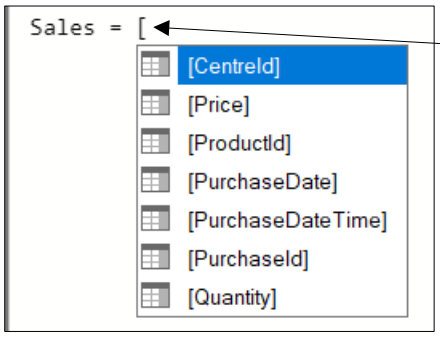


Right-click on a column and choose to insert a new one as here, then start typing into the formula bar which appears:



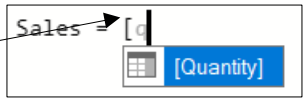
Referring to Columns/Fields

The easiest way to create a formula in DAX is to use the keyboard:

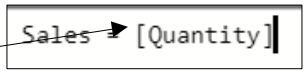


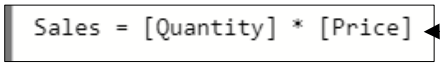
a) After typing a name for your new column, type the opening square bracket symbol **[**.

b) Type in the first letter of the field you want to use (here **q**).

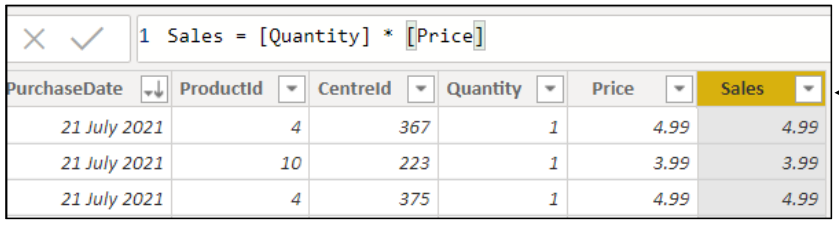


c) Either double-click on the field name to insert it, or press **TAB**.





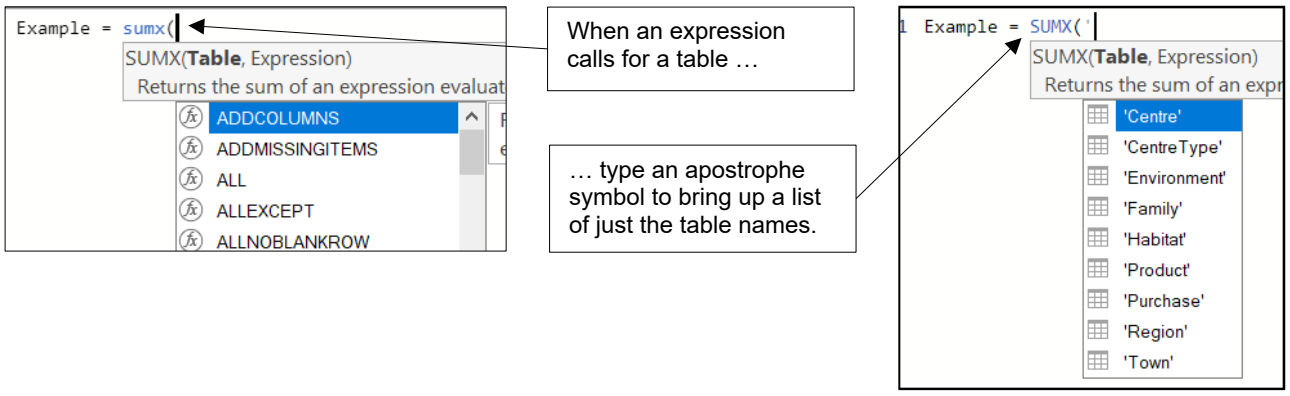
d) Continue in this way to build up the formula you want to create.



e) When you press **ENTER** to create your formula, DAX will calculate it and show the results for every row in the table.

Referring to Tables

If you want to refer to a table, the easiest way to do it is to type in the ' apostrophe character:



*Although the method above makes it easy to insert a table name into a formula, the apostrophe characters are optional, and most people miss them out. The exception to this is when your table name is also a reserved word (for example, **Calendar** is a table name which you would have to type as **'Calendar'**).*

Fully Qualified References

You can always refer to a column using its full reference: **=TableName[ColumnName]**

However, you can often miss out the table name where it is unambiguous from the context. So both of these calculated columns will work:

Here we haven't specified the table name, so DAX assumes that it is the current one.

```
Sales = [Quantity] * [Price]
```

ProductId	CentreId	Quantity	Price	Sales
4	367	1	4.99	4.99
10	223	1	3.99	3.99

Here, by contrast, we have included the table name, even though it wasn't necessary.

```
Sales = Purchase[Quantity] * Purchase[Price]
```

ProductId	CentreId	Quantity	Price	Sales
4	367	1	4.99	4.99
10	223	1	3.99	3.99

It's probably best practice always to fully qualify column references in DAX (although the author confesses to frequently taking the lazy way out and omitting them where they're not needed).

2.2 Writing DAX

The more you get into DAX, the longer and more complicated your formulae will become – and the more important it will be to format and comment them properly!

Laying out your Formulae

DAX formulae can quickly become quite long, and hard to read. You can make formulae easier to interpret by *indenting* arguments to functions:

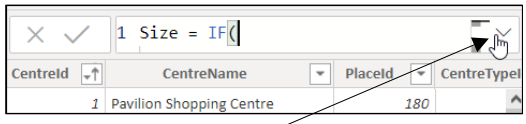
```
Size = IF(
   [SquareMetres] > 10000,
   "Large",
   "Small"
)
```

Using multiple lines and indenting code together make formulae easier to read.

When you copy DAX formulae from Power BI you lose any colours. Because of this all of the formulae in this courseware are copied from DAX Studio, a separate standalone DAX editor. As a result the colours shown will have slightly different shading to those you'll see in Power BI.

Using Multiple Lines

You can use this drop down arrow to give yourself more space to work in:



Click on this drop arrow to give yourself more room for typing.

```
1 Size = IF(
2
3     [SquareMetres] > 10000,
4     "Large",
5     "Small"
6 )
```

Too much room, sometimes! Click again to collapse the space.

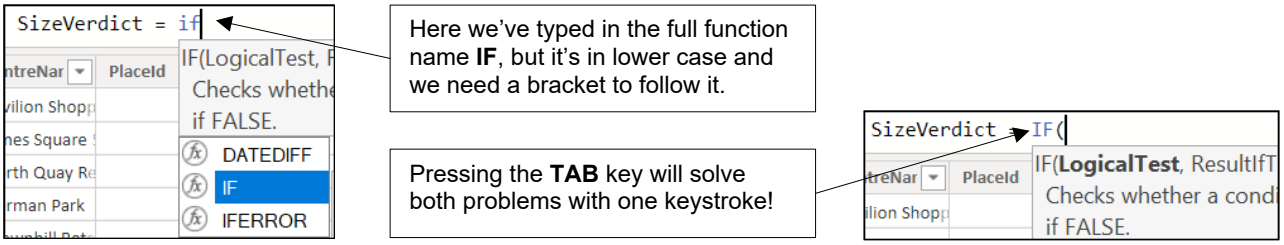
You can also use the following keys to add carriage returns into a formula:

Key	What it does
Shift + ↵	Add a new line, and also an indentation level if appropriate.
Alt + ↵	Add a new line, but don't indent it.

*Irritatingly, the one key which doesn't work is just pressing **Enter**. Instead, this makes Power BI create your formula, even if you haven't finished it. You'll then have to sort out the brackets Power BI has thoughtfully added at the end of the formula to make your parentheses balance out!*

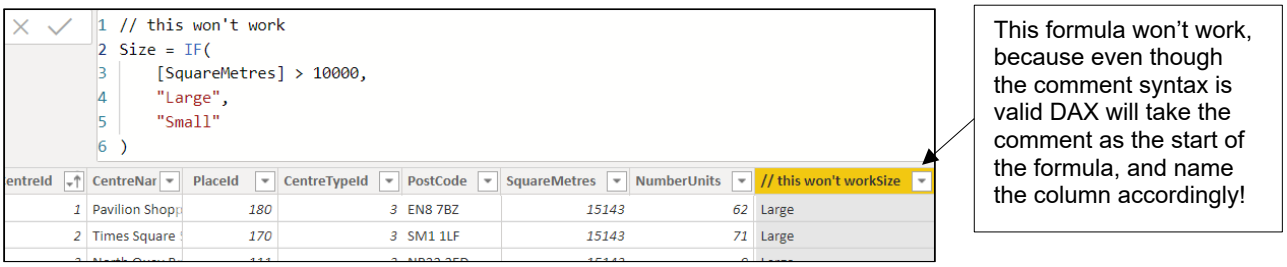
Pressing the TAB Key

When you've typed in a function or field, the best key that you can press is Tab. This is true even if you've typed in the whole name of a function:



Comments

You can't insert comments at the start of DAX formulae:



You can, however, insert them anywhere else, using one of 3 different syntaxes:

You can add comments at the end of formulae using one of these three syntaxes (although until you learn DAX variables it's not obvious why you'd want to do this).

```
Size = IF(
  [SquareMetres] > 20000,
  "Large",
  "Small"
)
// this formula categories
// shopping centres by size

-- you can also write comments
-- like this

/*
or even with long comments
like this
*/
```

```
Size = IF(
  // condition (bigger than
  // 20,000 square metres)
 [SquareMetres] > 20000,

 // return LARGE if a centre is big
 "Large",

 // otherwise return SMALL
 "Small"
)
```

You can also (more usefully) put comments between arguments – any line beginning with // or - - will be ignored.

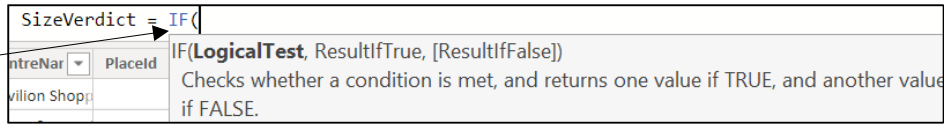
2.3 DAX Syntax

This section explains the rules you have to follow when creating DAX formulae.

Functions and Arguments

When you type any DAX function, Intellisense will tell you the arguments you need to specify. Here's an example:

A simple DAX function, returning different values if something is true or false.



Here are the *arguments* to this function (the bits of information that you need to specify):

Argument name	Status	What it should contain
LogicalTest	Compulsory	A test to perform to see if something is true or not
ResultIfTrue	Compulsory	What to return if the test returns true
ResultIfFalse	Optional	What to return if the test returns false

You can tell whether an argument is compulsory or optional by seeing whether it is enclosed in square brackets [like this].

Mathematical Operators

You can use the following standard mathematical symbols in DAX expressions:

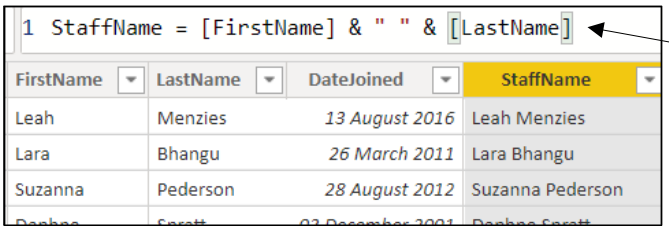
Symbol	What it means	Example
+ / -	Addition / subtraction	= 3 + 5 – 2 would return 6
* / /	Multiplication / division	= 2 * 6 / 3 would return 4
^	Raising to the power of	= 2 ^ 3 would return 8

Standard rules of arithmetic (BODMAS) apply: so 2 + 3 * 5 would return 17, since the multiplication would take precedence over the addition.

*Division and multiplication take equal precedence, and are read from left to right. So 15 / 3 * 2 would return 10, not 2.5.*

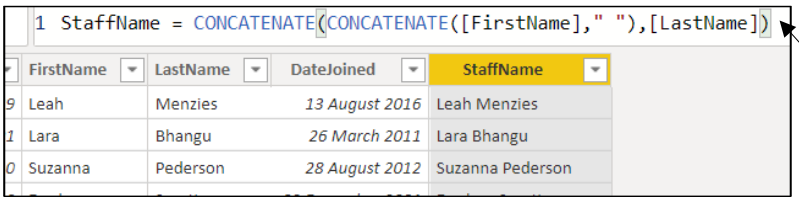
Concatenating Text

There are two ways to join text together. You can either use the & symbol:



This formula would join together the first name and last name fields, with a space in between.

Or alternatively, the **CONCATENATE** function (although this doesn't work well when you have more than two things that you want to join together):



Unlike in Excel, to join more than two things together you have to repeat the function name, making for a messy formula.

What we do!

		Basic training	Advanced training	Systems / consultancy
Office	Microsoft Excel			
	VBA macros			
	Office Scripts			
	Microsoft Access			
Business Intelligence	Power BI			
	Power Apps			
	Power Automate / PAD			
SQL Server	SQL			
	Reporting Services			
	Report Builder			
	Integration Services			
	Analysis Services			
Coding	Visual C# programming			
	VB programming			
	DAX			
	Python			


WiseOwl
Training