SumProduct

NEWSLETTER #100 - March 2021

www.sumproduct.com | www.sumproduct.com/thought

We made it!

And they said it couldn't be done. Well, actually, they didn't say anything because few cared! But no matter: welcome to newsletter number **100**!! The monthly newsletter on all things Excel, financial modelling and Power BI comes of age.

100 Newsletters: that's 964 articles, 695 keyboard shortcuts, 47 readers' questions answered and 2,040 pages, this newsletter has truly evolved from its humble beginnings to something for everyone territory. And yes, we know only a crazy person would read it from cover to cover. But that's why it's so badly edited…

We look back at the top three most popular articles over that time and also the top five Excel tips (regular readers might know what's coming!). There is even a teaser for some big news coming to a theatre near you later in the year (well, not really, it's a book…). Most of the regulars are here too, save for Power BI Updates, which were not announced before our printing deadline. We have another Beat the Boredom Challenge, Visual Basics, Power Pivot Principles, Power Query Pointers, and we even put the "fix" in on this month's A to Z of Excel Function(s).

Whether you have been here from the outset or this is your first edition, welcome! Happy reading and remember: stay safe, stay happy, stay healthy.

Here's to the next 100, if it doesn't kill me…

Liam Bastick, Managing Director, SumProduct

Who'd Have Thunk?

When I was a little boy, I dreamed of this moment. I looked up to the stars and gazed at the brightest one. There was a calling. I stood transfixed, alive with every nerve and fibre in my being. It felt like this celestial body's rays were beaming down on me – just for me. It was like a voice from beyond told me to heed my calling: to strive for Excel modelling perfection, work in corporate finance, commit many, many long hours to the cause and write a newsletter.

I never thought it would happen. It was an impossible dream. It took years of training. Who would have thought you could write a monthly newsletter on Excel / financial modelling that would be on average *20 pages for 100 issues?* Imagine the glamour, the fame, the fortune of it all…

If all this sounds like BS (and I am not talking Balance Sheets), you'd be right. What the hell was I thinking? How many years of my life have I wasted..? Aaaaaaaaaaaaaaaaaaaaaaagh.


SumProduct

NEWSLETTER #1 - Dec 2012

Welcome to the inaugural edition of SumProduct's monthly newsletter. Starting right here – on the eve of our third birthday – our aim, in conjunction with our web site's *Thought* articles, is to help our readers boost their Excel and financial modelling skills through a series of short articles, tricks and tips in an informal manner.

I hope you find these newsletters useful – if not, tell us; if you do, tell the world!!

Liam Bastick, Managing Director, SumProduct

Eyes on the Prize!

SumProduct, in conjunction with the Chartered Institute of Management Accountants (CIMA), is pleased to announce the annual Xmas quiz. It's open to everyone, not just accountants. Check out the Excel challenge – with prizes – in the December issue of Insight at http://tinyurl.com/cimaglobal Best of luck!!

Formulae not to be mentioned in polite conversation #1: EVALUATE

Not all functions are documented in Excel. Some seem to have been swept under the rug - proverbial black sheep? - ne'er to be discussed for fear of Microsoft retribution. Try looking up the EVALUATE function in Excel's Help, for instance. In fact, try using it ("That function is not valid!")! EVALUATE is possibly the most bizarre function in Excel - yet it is a useful one if you can work out how to use it...

Consider the following complex spreadsheet:

	A
1	1
2	+
3	2

EVALUATE essentially converts text strings into formulae that can be, er, evaluated. Theoretically,

=EVALUATE(A1&A2&A3)

would be EVALUATE(1+2) which is 3. All good, except it doesn't work unless you use it via a range name definition.

Range names have been discussed on our web site previously (see http://www.sumproduct.com/thought/names for more information).

Assuming the worksheet is imaginatively named 'Sheet1', the trick here is to define a range name (CTRL + F3, then click on 'New...' button) as follows:

Note that you must use the sheet names for this to work. Now, if you type :

=Evaluation

into a cell in Excel, suddenly evaluate comes to life and provides the expected result (here, 3). As you begin to understand the syntax, you will see that the Evaluation range name can be used to evaluate different cells in the same spreadsheet simply by removing the anchoring (i.e. remove some or all of the $ signs).

It gets cleverer. Check out the Thought section of our website at http://www.sumproduct.com/thought/evaluating-evaluate to see how you can plot charts in Excel without creating data table using an extension of this idea.

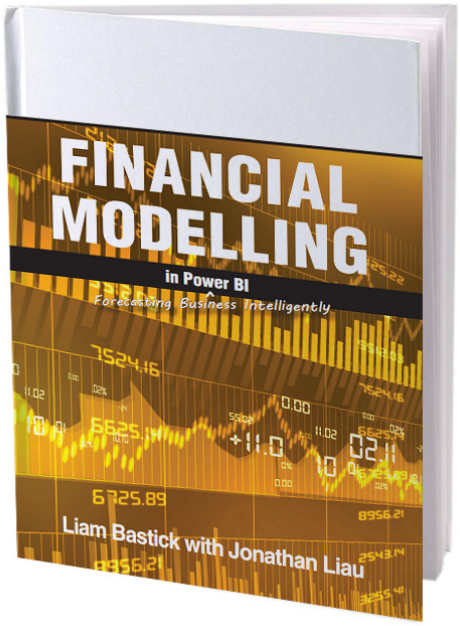
Key Strokes

Each newsletter, we'd like to introduce you to three useful keystrokes you may or may not be aware of. Whether you're an avid mouse user or a cool keyboard cat, these are suggested as useful Excel shortcuts:

Keystroke	What it does
ALT + =	Sums numbers either directly above or to the left of the cell.
CTRL + HOME	'Resets' the sheet so that the top of the worksheet is displayed once more. Precise cell selected depends upon whether frozen panes have been used.
CTRL + END	Selects the cell that represents both the final column and the final row used in the spreadsheet. This is useful for specifying print settings.

Special Announcement

We have saved this one up for our 100th. Coming soon:



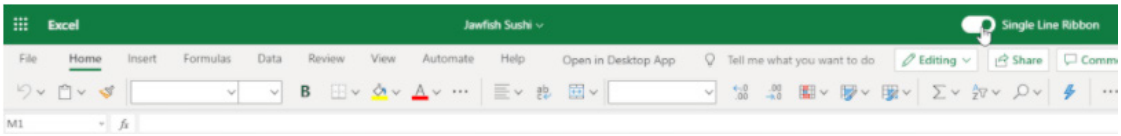
No Excel; just Power BI. Others say they have already done this. We actually have. More anon.

Microsoft Announces Performance Improvements in Excel for the Web

Excel for the web has had some work done under the hood. Microsoft's programmers have addressed and improved several scenarios, including opening workbooks, navigating within a workbook and other interactions:

- **Loading:** the time it takes to load a workbook has now been reduced significantly, making it faster to open your file online
- **Scrolling:** a fundamental part of working in Excel, this has been a little sluggish as a browser experience. However, now, even in incredibly complex sheets, scrolling is smoother and faster
- **Selection:** even more fundamental than scrolling is the need to interact with content in your workbook. Microsoft claim to have optimised the speed of cell selection, so theer will be less of a lag time when choosing ranges
- **Navigating:** several navigation actions, such as Find / Search, Go To, page-up and page-down are all now faster
- **Modifying:** cell editing and formatting experiences are now faster than before.

This, together with the new simplified Ribbon, should make for a better online spreadsheeting experience.



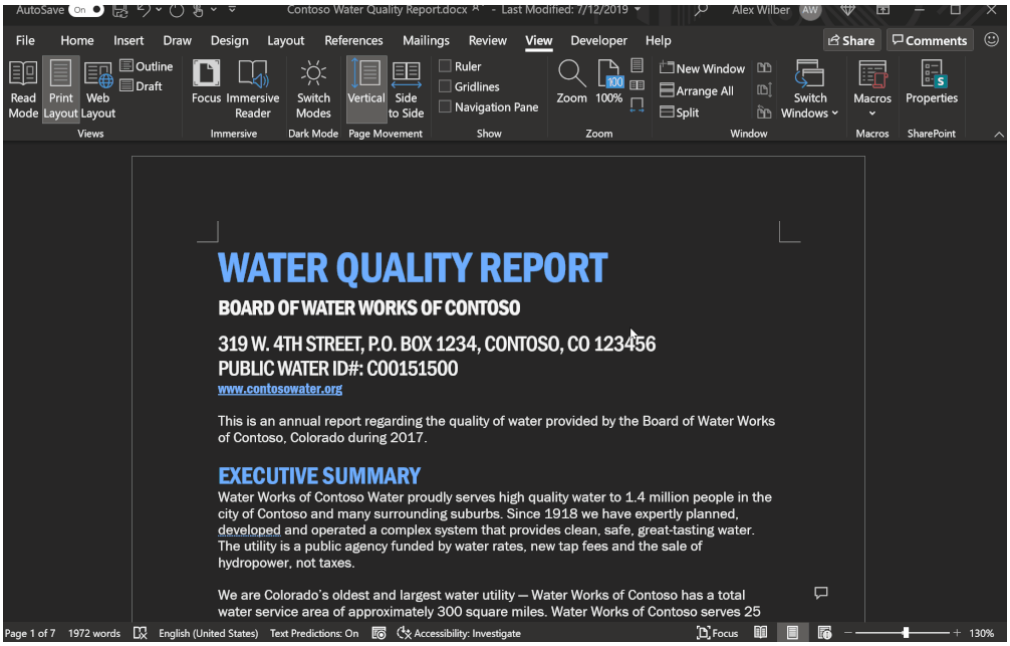
Office 2021 Coming Soon

Microsoft has recently announced two new versions of Office 2021 (not Office ***2022***, as previously thought): a consumer Office 2021 version and Office Long-Term Servicing Channel (LTSC) for commercial customers.

Office 2021 will be available later this year for both Windows and macOS, and as before, it has in mind those that want a "perpetual" licence, who do not wish to subscribe to the cloud / Microsoft 365 versions. Both of the new Office variants will ship with OneNote and include 32-bit and 64-bit versions.

Little information has been divulged so far, but it is known that the Office LTSC variant will include things like dark mode support, accessibility improvements, as well as features like Dynamic Arrays and **XLOOKUP** in Excel. It's probably safe to say that Office 2021 should include similar features.

Dark mode is probably the biggest “new” thing here:



it’s clear that Microsoft strategy pivots on the 365 subscription / cloud-based model and not this product.

Office LTSC will apparently only be supported for five (5) years instead of the usual seven (7) that Microsoft has previously provided for Office. Internet sources state that pricing for Office Professional Plus, Office Standard and individual apps will be increasing 10% for commercial customers, whereas the Office 2021 consumer and small business pricing structures shall remain the same.

The change in Office LTSC support duration will now better align with how Windows is supported, and Microsoft is also aligning its release schedules for both Office and Windows more closely as a result. Both of the next versions of Office LTSC and Windows 10 LTSC will be released in the second half of 2021.

A Preview of Office LTSC is due in April, with a full release later this year. Apparently, the consumer Office 2021 variant won’t be available in Preview, though.

Top 3 Articles: #3 Debt Modelling

For our 100th newsletter, we’ve decided to reproduce the three articles that have produced the most feedback in the past 99 issues. We are counting them down in reverse order. This is Number 3 – exciting, isn’t it? Don’t answer that…

Over the years, we have seen various forms of business and project financing, including equity, shareholder loans, senior debt, mezzanine finance, hire purchase, bonds, convertibles, warrants and so on. Prima facie, this myriad of financial instruments can obfuscate the uninitiated, but like this last phrase, the jargon can be simplified.

No matter what the financial instrument, the mechanics essentially boil down to two key elements:

- **Return on finance:** the yield to investors or the costs of capital to the recipient of capital (*e.g.* interest, dividends); *and*
- **Return of finance:** repayments (or conversion) of original capital issued / drawn down.

And it really is as simple as that. The logic behind how the calculations may vary, such as when capital and returns are paid or rolled up, what order it is paid in and so on, but the computations may be summarised by two control accounts (*i.e.* summaries that show / reconcile how the Balance Sheet varies from one period to the next):

Returns of Finance

Opening Balance (e.g. Debt / Equity) b/f	XX	Previous period Balance Sheet item
Additions (e.g. drawdowns / issuances / conversions)	X	Typically in Cash Flow Statement
Returns on finance rolled up (e.g. “interest capitalised”)	X	Usually a Balance Sheet movement
Deductions (e.g. repayments / buybacks / conversions)	(X)	Typically in Cash Flow Statement
Closing Balance (e.g. Debt / Equity) c/f	**XX**	Current period Balance Sheet item

Returns of Finance

Opening Return Payable (e.g. Interest Payable) b/f	XX	Previous period Balance Sheet item
Return Accrued (e.g. Interest Expense)	X	Income Statement or Balance Sheet movement
Return Paid (e.g. Interest Paid)	(X)	Cash Flow Statement
Closing Return Payable (e.g. Interest Payable) c/f	**XX**	Current period Balance Sheet item

When both businesses and lenders consider debt, they look at two key aspects: risk and return. These are important for credit risk modelling / portfolio analysis, etc. However, when undertaking financial modelling, it is the third 'R' that is often the most important.

In a financial model, risk and return are usually modelled via simple inputs and occasional what-if analysis. Ranking, on the other hand, affects the entire financial structure of the model:

1. Debt Cascade

	Date 1	Date 2	Date 3	Date 4	Date 5	Date 6	Date 7	Date 8	Date 9	Date 10	Date 11	Date 12
Cashflow Before Funding	(16.0)	(0.2)	(0.5)	(0.5)	(0.5)	4.3	6.7	6.8	6.8	7.1	7.3	7.4
Funding	16.0	-	-	-	-	-	-	-	-	-	-	-
Cashflow After Funding	-	(0.2)	(0.5)	(0.5)	(0.5)	4.3	6.7	6.8	6.8	7.1	7.3	7.4
Tax	-	-	-	-	-	-	-	-	-	-	-	-
Cashflow Available before WC Funding	-	(0.2)	(0.5)	(0.5)	(0.5)	4.3	6.7	6.8	6.8	7.1	7.3	7.4
Working Capital Facility Funding	-	0.2	0.5	0.5	0.5	-	-	-	-	-	-	-
Cash Flow Available for Debt Service (CFADS)	-	-	-	-	-	4.3	6.7	6.8	6.8	7.1	7.3	7.4
Senior Debt Service	-	(0.4)	(0.4)	(0.4)	(0.4)	(1.7)	(1.7)	(1.7)	(1.7)	(1.7)	(1.7)	(1.7)
Cashflow Available for Debt Service Reserve Account	-	(0.4)	(0.4)	(0.4)	(0.4)	2.6	5.0	5.1	5.1	5.4	5.6	5.7
Debt Service Reserve Account	-	4.0	0.0	-	-	(2.6)	(0.8)	0.0	0.0	(0.0)	(0.0)	0.0
Cashflow Available for Mezzanine	-	3.6	(0.4)	(0.4)	(0.4)	-	4.2	5.1	5.1	5.4	5.6	5.7
Mezzanine Debt Service	-	(2.7)	-	-	-	-	(3.1)	(3.8)	(3.8)	(4.1)	(4.2)	(4.3)
Cashflow Available for WC Facility	-	0.9	(0.4)	(0.4)	(0.4)	-	1.0	1.3	1.3	1.4	1.4	1.4
Working Capital Facility	-	(0.2)	(0.0)	(0.0)	(0.0)	-	(1.0)	(0.5)	-	-	-	-
Cashflow Available for Equity	-	0.7	(0.4)	(0.4)	(0.5)	-	-	0.7	1.3	1.4	1.4	1.4
Dividends	-	5.3	5.2	5.2	5.3	(2.0)	(2.0)	(2.2)	(2.3)	(3.0)	(3.1)	(3.4)
Net Cashflow	-	5.9	4.7	4.8	4.8	(2.0)	(2.0)	(1.4)	(1.0)	(1.6)	(1.7)	(1.9)
Cash Balance B/f	-	-	5.9	10.7	15.4	20.2	18.3	16.2	14.8	13.8	12.1	10.4
Cash Balance C/f	-	5.9	10.7	15.4	20.2	18.3	16.2	14.8	13.8	12.1	10.4	8.4

As the above graphic shows, if the order of service repaying capital changes, the entire logic will change. This may affect interest / debt service cover ratios (see below). It is important in scoping any such model that the order is understood and how it will be affected by such factors as:

- Breach of covenants
- Conversion of financial instruments
- Breach of covenants or other ratios
- Liquidation / insolvency.

It is not correct to assume that the order of financing will never change.

Further, there is confusion between the jargon used by the banking industry and accountants when considering debt mechanics:

Scenario	Banking term	Accounting term
Interest is not paid (either by agreement or due to insufficient funds) and is added to the outstanding principal for future interest calculations	Interest capitalised	Interest rolled-up
Interest is not added to the balance but is paid (although there may be a slight timing issue)	Interest amortised (principal is amortised similarly)	When accrued: interest expense When paid: interest paid
Regardless of whether paid or not in reality, interest meets the criteria specified in the relevant accounting standards to be held in the Balance Sheet	n/a	Interest capitalised
When capitalised under accounting rules, the interest charge is released to the P&L over the life of a project on some agreed equitable basis	n/a	Interest amortised

When holding conversations with financiers, be sure you are on the same page before building interest into a financial model!

Top 3 Articles: #2 Dynamic Arrays

For our 100th newsletter, we've decided to reproduce the three articles that have produced the most feedback in the past 99 issues. We are counting them down in reverse order. Here is one that caused a massive reaction in the community – either excitement because of what you could do with them, or else disappointment because their version of Excel didn't have them…

September 24, 2018 is the day Excel moved on. Yes, we've had Power Pivot, Power Query / Get & Transform and Power BI, but Microsoft's "Calc" team has been busy behind the scenes rearranging the furniture.

By "furniture" I mean the "calculation engine" – it's had a complete re-write, and there are benefits general Excel users will reap for years to come. The first wave sees a new array calculation ("Dynamic Array"), seven new functions and two new error messages. And that's just the start. There's going to be plenty more coming in the next few years. But it's in Office 365…

So, what's the big deal?

Spilling the Beans

Let me begin by just looking at what a Dynamic Array is. Consider the following data:

Original Data

Shape	Colour	Sides
Triangle	Red	3
Rectangle	Amber	4
Circle	Green	1
Triangle	Red	3
Square	Blue	4
Rectangle	Blue	4
Rectangle	Amber	4
Circle	Amber	1
Triangle	Red	3
Square	Green	4
Circle	Blue	1
Square	Amber	4
Triangle	Blue	3
Circle	Green	1
Rectangle	Blue	4

If I were to type **=F12:H27** into another cell, Excel in the past would have thought I had gone mad. I'd need to wrap it in an aggregation function such as **SUM**, **COUNT** or **MAX**, to name but a few. Otherwise, I would have to wrap it in braces using **CTRL + SHIFT + ENTER** and use it as an array formula.

But no more.

Look at what happens when I type **=F12:H27** into cell **F33**:

F33 =F12:H27

Dynamic Array Result

Shape	Colour	Sides
Triangle	Red	3
Rectangle	Amber	4
Circle	Green	1
Triangle	Red	3
Square	Blue	4
Rectangle	Blue	4
Rectangle	Amber	4
Circle	Amber	1
Triangle	Red	3
Square	Green	4
Circle	Blue	1
Square	Amber	4
Triangle	Blue	3
Circle	Green	1
Rectangle	Blue	4

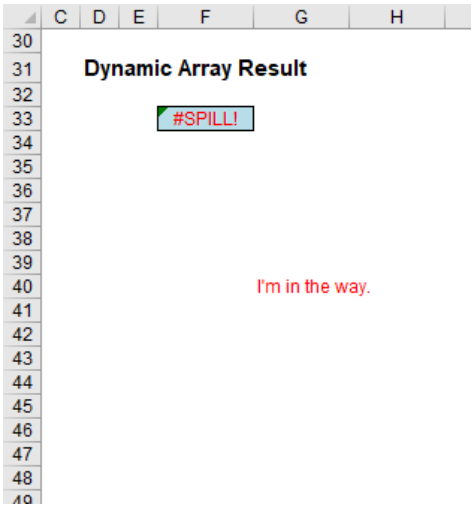
The formula *automatically extends* to three columns by 16 rows! It has *spilled*. Get used to the vernacular. There's a reason this article got the name it did!

Any formula that has the potential to return multiple results can be referred to as a **Dynamic Array** formula. Formulae that are currently returning multiple results, and are successfully spilling, can be referred to as **Spilled Array Formulae**.

Notice I did not have to highlight all of the cells **F33:H48**. It *spilled*. Also take note I formatted cell **F33** – er, that didn't spill, because presently formatting isn't propagated. This is why this is not yet Generally Available. Microsoft is still trying to work out what should and shouldn't be allowed to happen in this first release. But don't let that put you off.

And don't let this basic example put you off either. If you feel a general sense of underwhelm coming over you, it's because I haven't yet communicated how powerful this all is as my example was too basic.

However, before I carry on there is a question I do need to cover with my far too simple example: what happens if something gets in the way?

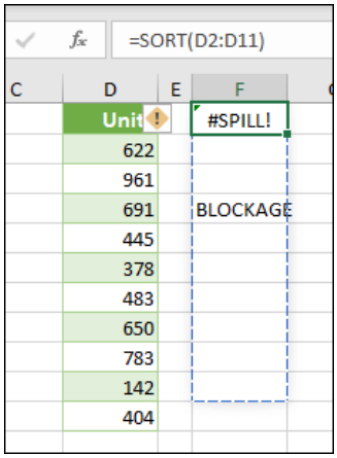


In this example, in cell **G40**, I have typed in the obtrusive text, "I'm in the way". And it quite literally is. Consequently, I have generated the brand new #SPILL! error. The formula cannot spill, so the error message is generated accordingly.

#SPILL! Errors

#SPILL! errors are returned when a formula returns multiple results, and Excel cannot return the results to the spreadsheet. There are various reasons an *#SPILL!* error could occur:

- **spill range is not blank:** as in my example *(above)*, this error occurs when one or more cells in the designated spill range are not blank and thus may not be populated.



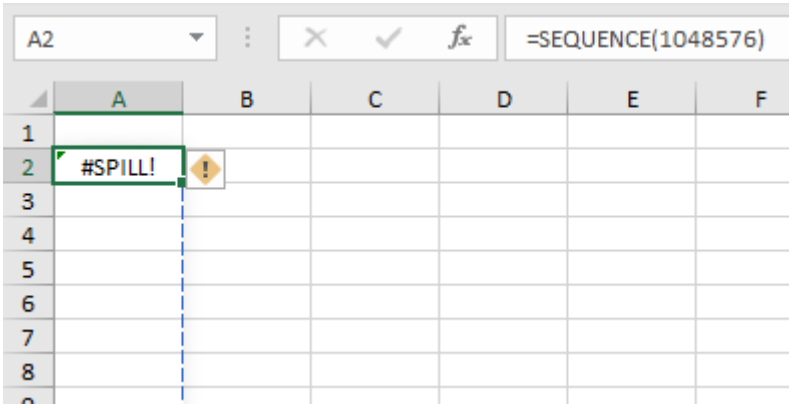
When the formula is selected, a dashed border will indicate the intended spill range. You may select the error "floatie" (believe it or not, this is what Microsoft call these things!), and choose the 'Select Obstructing Cell' option to immediately go the obstructing cell. You can then clear the error by either deleting or moving the obstructing cell's entry. As soon as the obstruction is cleared, the array formula will spill as intended

- **the range is volatile in size:** this means the size is not "set" and can vary. Excel was unable to determine the size of the spilled array because it's volatile and resizes between calculation passes. For example, the new function **SEQUENCE(x)** *(explained in detail below)* generates a list of **x** numbers increasing by 1 from 1 to **x**. That's fine, but the following formula will trigger this *#SPILL!* error:

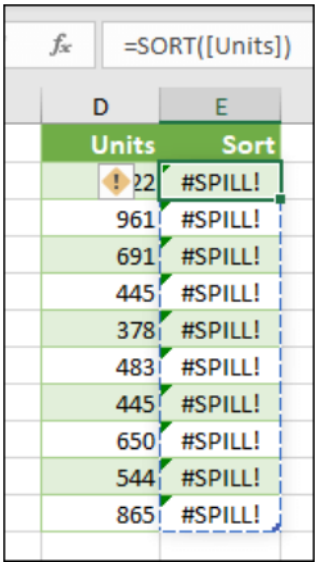
=SEQUENCE(RANDBETWEEN(1,1000)).

Dynamic array resizes may trigger additional calculation passes to ensure the spreadsheet is fully calculated. If the size of the array continues to change during these additional passes and does not stabilise, Excel will resolve the dynamic array as *#SPILL!* This error type is generally associated with the use of **RAND**, **RANDARRAY** and **RANDBETWEEN** functions. Other volatile functions such as **OFFSET**, **INDIRECT** and **TODAY** do not return different values on every calculation pass so tend not to generate this error

- **extends beyond the worksheet's edge:** in this situation, the spilled array formula you are attempting to enter will extend beyond the worksheet's range. You should try again with a smaller range or array. For example, moving the following formula to cell **A1** will resolve the error, and the formula will spill correctly

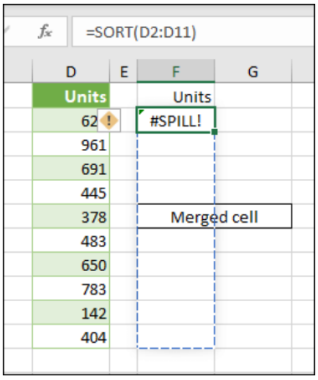


- **Table formula:** as I will explain shortly, Tables and Dynamic Arrays are not yet best friends. Spilled array formulae aren't supported in Excel Tables (generated by **CTRL + T**). Try moving your formula out of the Table, or go to **Table Tools -> Convert to range**



- **out of memory:** I have forgotten what this one means. Sorry, I couldn't resist that. The spilled array formula you are attempting to enter has caused Excel to run out of memory. You should try referencing a smaller array or range

- **spill into merged cells:** spilled array formulae cannot spill into merged cells. You will need to un-merge the cells in question or else move the formula to another range that doesn't intersect with merged cells.

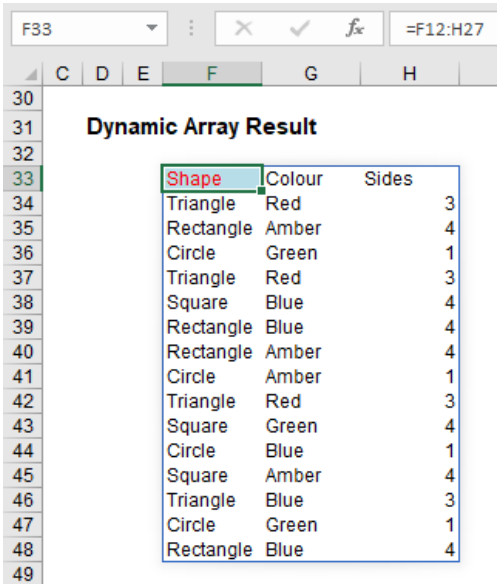


When the formula is selected, a dashed border will indicate the intended spill range. You can again select that wonderfully named error floatie and choose the 'Select Obstructing Cell' option to immediately go the obstructing cell. As soon as the merged cells are cleared, the array formula will spill as intended

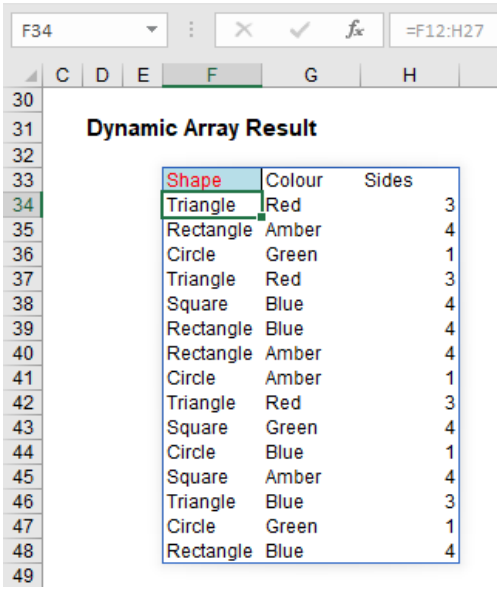
- **unrecognised / fallback error:** the "catch all" variant. Excel doesn't recognise, or cannot reconcile, the cause of this error. Here, you should make sure your formula contains all of the required arguments for your scenario.

Returning to Dynamic Arrays

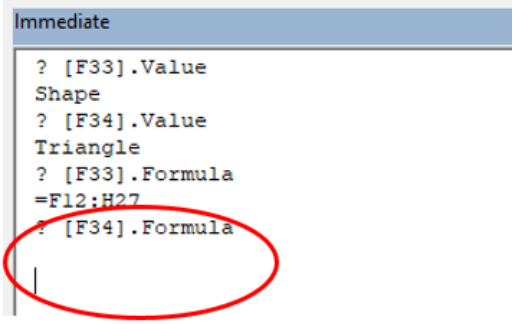
Now that we have considered what happens if you block a Dynamic Array, let me now turn my attention to what happens if you *don't*. You get the following:



Do you see I am not having to anchor cells (*i.e.* use dollar [**$**] signs)? The formula just *spills*. Let me be clear. If I select cell **F34**, I get the following:

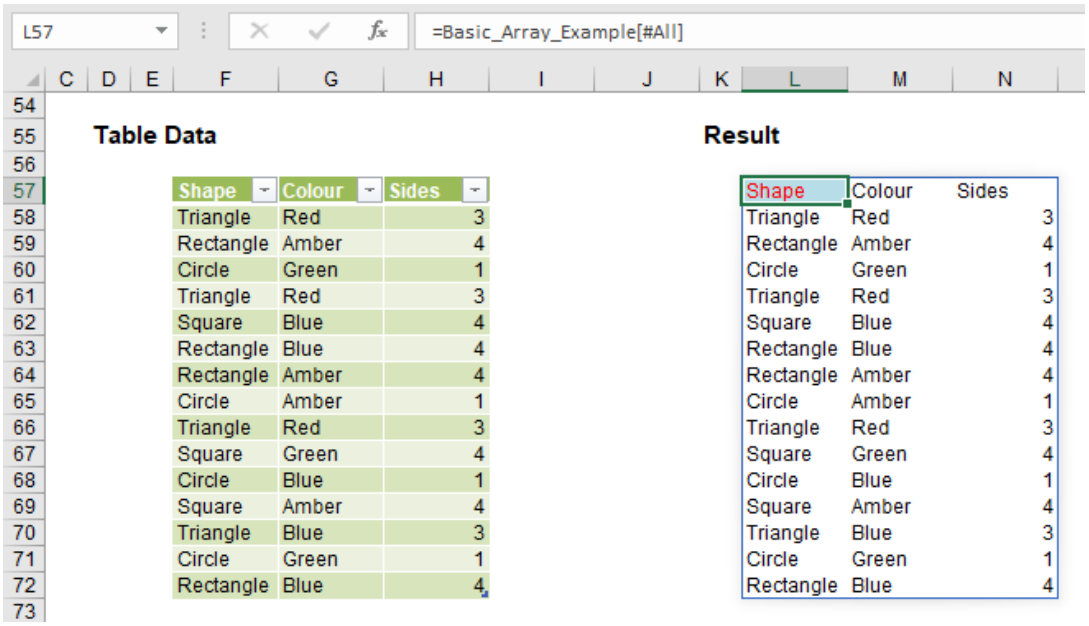


Here's a first. Check out the formula in the formula bar. It's *greyed out*. Ever seen that before? Effectively, cell **F34** contains the value 'Triangle' but it does not actually contain an "Excel" formula in the usual sense. To demonstrate this, let me show you the VBA Immediate Window:



But, to quote Bill Jelen, similar to Schrodinger's Cat, if you select cells **F33:H48** and use 'Go To Special' (**F5 -> Special**), and then select 'Formulas', cells **F33:H48** are shown as formula cells. You can even copy and paste them as values. Ladies and gentlemen, welcome to The Twilight Zone (cue eerie music).

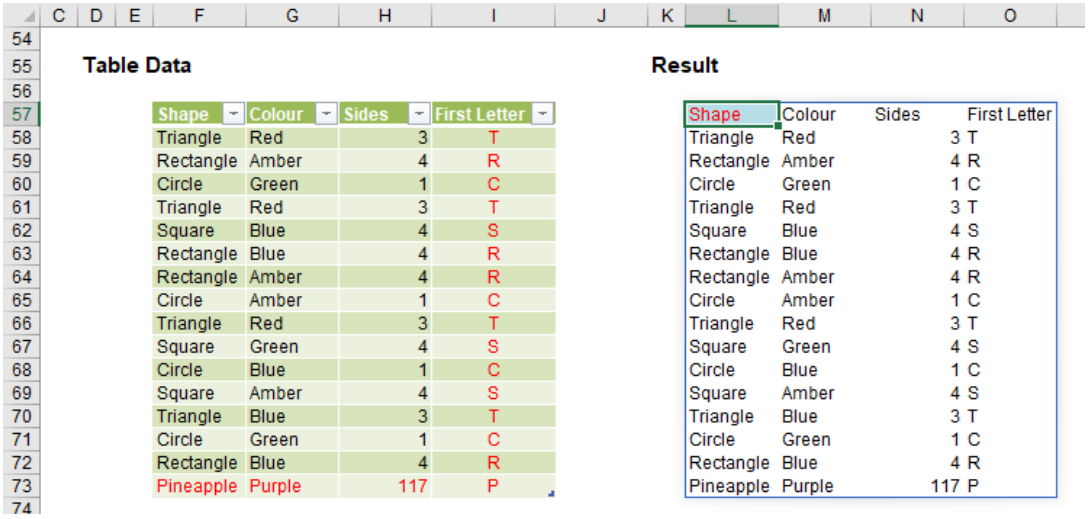
I mentioned in the *#SPILL!* errors section that you cannot use Dynamic Arrays in a Table, but Dynamic Arrays may refer to a Table, *viz*.



Shape	Colour	Sides
Triangle	Red	3
Rectangle	Amber	4
Circle	Green	1
Triangle	Red	3
Square	Blue	4
Rectangle	Blue	4
Rectangle	Amber	4
Circle	Amber	1
Triangle	Red	3
Square	Green	4
Circle	Blue	1
Square	Amber	4
Triangle	Blue	3
Circle	Green	1
Rectangle	Blue	4

In this above illustration, cells **F57:H72** have been converted into a Table (**CTRL + T**), with the Table named **Basic_Array_Example**. In cell **L57**, I have simply typed '=' and then highlighted the entire Table. It was all replicated.

The advantage of linking a Dynamic Array to a Table is clear:

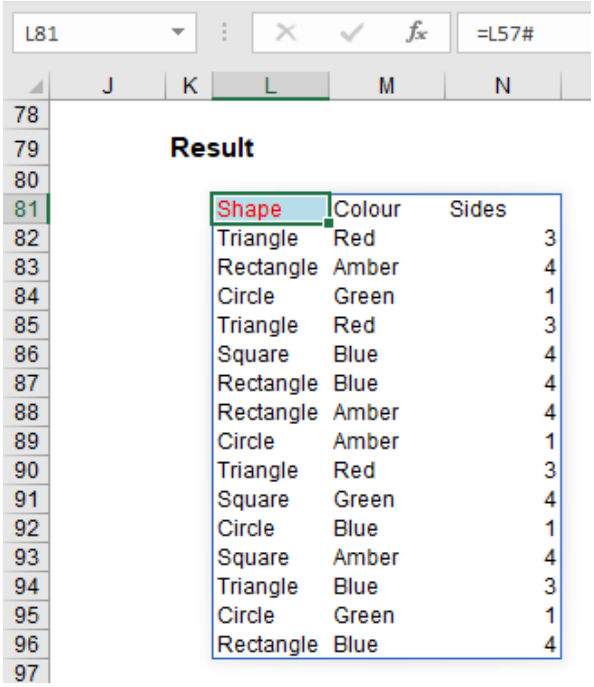


Shape	Colour	Sides	First Letter
Triangle	Red	3	T
Rectangle	Amber	4	R
Circle	Green	1	C
Triangle	Red	3	T
Square	Blue	4	S
Rectangle	Blue	4	R
Rectangle	Amber	4	R
Circle	Amber	1	C
Triangle	Red	3	T
Square	Green	4	S
Circle	Blue	1	C
Square	Amber	4	S
Triangle	Blue	3	T
Circle	Green	1	C
Rectangle	Blue	4	R
Pineapple	Purple	117	P

I can add rows and / or columns and the Dynamic Array will update automatically. Do note that this does not breach the *#SPILL!* range is volatile in size error. This is because the range size will not vary on every calculation pass.

Talking of varying sizes, it's clear to see one potential issue with Dynamic Arrays. If we are not referring to a Table, what happens if the source data changes dimensions? This may be why you should refer to a Table for safety.

However, once you have a Dynamic Array, referring to it is simple using what is known as the **Spilled Range Operator**. For example, if I want to refer to the Dynamic Array in the previous examples, it initially had a range of **L57:N72**. However, once I had added a row and column to the Table, this resized to **L57:O73**. I can easily refer to this array, whatever its size as follows. In its initial state:



Shape	Colour	Sides
Triangle	Red	3
Rectangle	Amber	4
Circle	Green	1
Triangle	Red	3
Square	Blue	4
Rectangle	Blue	4
Rectangle	Amber	4
Circle	Amber	1
Triangle	Red	3
Square	Green	4
Circle	Blue	1
Square	Amber	4
Triangle	Blue	3
Circle	Green	1
Rectangle	Blue	4

The formula **=L57#** allows for variations – you simply type in the top left-hand cell reference (*i.e.* the cell with the non-greyed out formula) and add '**#**", known as the Spilled Range Operator. Simple!

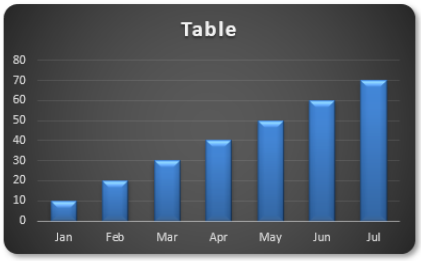
It's not all peaches and cream though. Whilst Dynamic Arrays and Tables share some similarities, they are very different beasts. This couldn't be clearer than when you create charts:

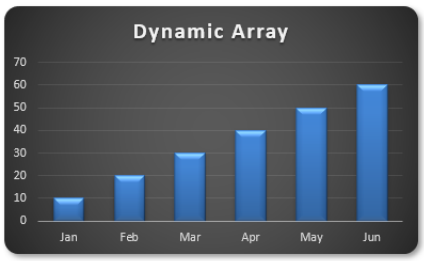
Table

Month	Sales
Jan	10
Feb	20
Mar	30
Apr	40
May	50
Jun	60
Jul	70

Dynamic Array

Month	Sales
Jan	10
Feb	20
Mar	30
Apr	40
May	50
Jun	60
Jul	70





Here, I created two charts when I only had the data up to June. Then, I added the data for July. The chart on the left referencing the Table source data updated instantly. However, the chart on the right still only displayed up to June even though the Dynamic Array had updated. It is true that with clever use of range names this may be overcome, but it doesn't get around the fact that Tables remain a simpler way to retain dynamic chart data (for the time being anyway!).

Conclusion: use Tables, not Dynamic Arrays, as your references for dynamic charts.

Implicit Intersection Implications

It may be an alliteration and sound like something you can get arrested for, but Dynamic Arrays do come at a price. There aren't many users out there who used them, but there are some – and hence there will be some legacy calculations affected.

In the past, if you entered **=A$1:A$10** anywhere in rows 1 through 10, the formula would return only the value from that row. In fact, a spreadsheet our company is presently auditing relies on this behaviour. However, in the brave new world of Office 365 (albeit selected Insider recipients for the time being), typing this formula would create a Spilled Array Formula. To protect existing formulae, we need a new – if not instantly breathtaking – function…

SINGLE Function / @ Operator

Don't judge the remaining functions on our first new feature, originally a function, now an operator. This one is essential to keep Excel running smoothly, but it's probably safe to say it won't set the world alight. It's like toilet roll – imagine your situation without it…

When Dynamic Arrays first came out, the **SINGLE** function returns a single value using logic known as implicit intersection. **SINGLE** could return a value, single cell range or an error.

The function had the following syntax:

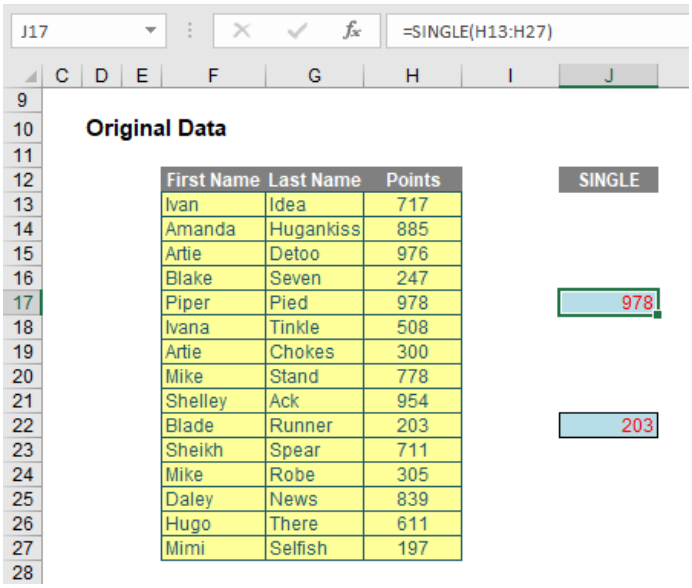
=SINGLE(value).

The function has just one argument:

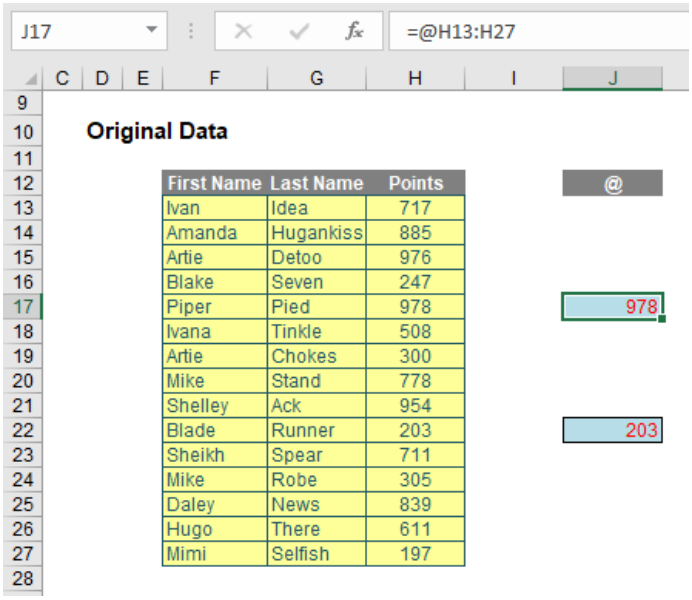
- **value:** this argument is required and represents the array to be selected.

When the supplied argument is a range, **SINGLE** would return the cell at the intersection of the row or column of the formula cell. Where there is no intersection, or more than one cell falls in the intersection, then **SINGLE** would return a *#VALUE!* error. When the supplied argument is an array, **SINGLE** would return the first item (Row 1, Column 1).

In the example below, the two **SINGLE** formulae are supplied a range, **H13:H27**, and return the values in cells **H17** and **H22** respectively.

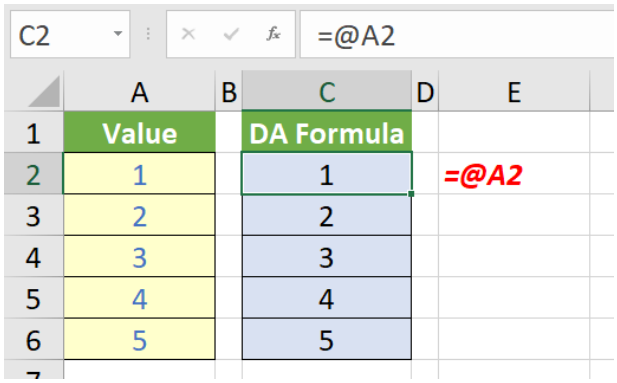


However, more recently, **SINGLE** was replaced with the **@** operator as follows:

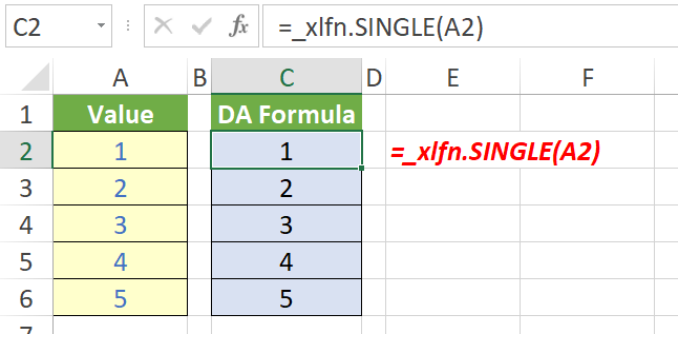


Now, I mention this history with good reason. Excel will only remove **@** from a formula where previous Excel versions would have used implicit intersection (*as described above*) to return a single value from a range, a named range or function parameter.

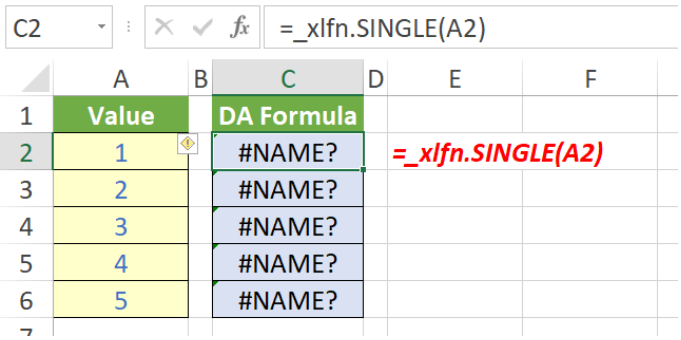
On the positive side, if you attempt to enter such a formula, Excel will warn you and do its utmost to stop you. It is still possible to cause an issue though. For example, in Office 365, you could create the following formula:



In older versions of Excel, this would appear as:



Notice the error message is **=_xlfn.SINGLE(A2)**, not **=_xlfn.@(A2)**. This is confusing if you don't know the history of the **@** operator. Worse comes if you try to evaluate this formula:



It generates an *#NAME?* error, which is far from ideal.

Dynamic Arrays vs. Legacy Array Formulae

Prior to this new functionality, if you wanted to work with ranges in Excel, you used to have to build array formulae, where references would refer to ranges and be entered as **CTRL + SHIFT + ENTER** formulae. The main differences are as follows:

- Dynamic Array formulae may spill outside the cell bounds where the formula is entered. The Dynamic Array formula technically only exists in the cell in the top left-hand corner of the spilled range (*as shown earlier*), whereas with a legacy **CTRL + SHIFT + ENTER** formula, the formula would need to be entered in the entire range
- Dynamic arrays will automatically resize as data is added or removed from the source range. **CTRL + SHIFT + ENTER** array formulae will truncate the return area if it's too small, or return *#N/A* errors if too large
- Dynamic array formulae will calculate in a 1 x 1 context
- Any new formulae that return more than one result will automatically spill. There's simply no need to press **CTRL + SHIFT + ENTER**
- According to Microsoft, **CTRL + SHIFT + ENTER** array formulae are only retained for backwards compatibility reasons. Going forward, you should use Dynamic Array formulae instead
- Dynamic array formulae may be easily modified by changing the source cell, whereas **CTRL + SHIFT + ENTER** array formulae require that the entire range be edited simultaneously
- Column and row insertion / deletion is prohibited in an active **CRL + SHIFT + ENTER** array formula range. You first need to delete any existing array formulae that are in the way.

Everybody clear? I think we are finally good to start introducing the other new functions…

SORT Function

I am not going to do these alphabetically – let me show the new functions then in an order that makes sense (well, to me, anyway).

The **SORT** function sorts the contents of a range or array:

=SORT(array, [sort_index], [sort_order], [by_column]).

It has four arguments:

- **array:** this is required and represents the range that is required to be sorted
- **sort_index:** this is optional and refers to the position of the row or the column in the selected **array** (*e.g.* second row, third column). 99 times out of 98 you will be defining the column, but to select a row you will need to use this argument in conjunction with the fourth argument, **by_column**. And be careful, it's a little counter-intuitive! The default value is 1
- **sort_order:** this is also optional. The choices for **sort_order** are 1 for ascending (default) or -1 for descending. It should be noted that you might not want to hold your breath waiting for 'Sort by Color' *(sic)*, 'Sort by Formula' or 'Sort by Custom List' using this function
- **by_column:** this final argument is also optional. Most people want to sort rows of data, so they will want the value to be FALSE (which is the default value if not specified). Should you be booking your mental health check, you may wish to use TRUE to sort by column in certain instances.

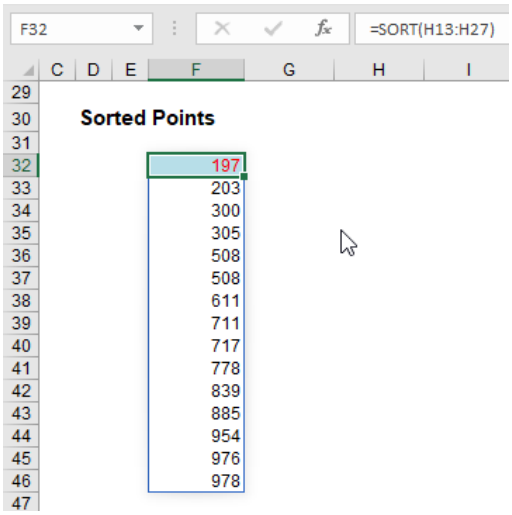
This is a function people have been crying out for, for *years*. Enterprising spreadsheets gurus have developed array formulae and user-defined functions that have replicated this functionality, but you don't need it anymore! **SORT** is coming to a theatre near you very soon.

To show you how devilishly simple it is, consider the following data:

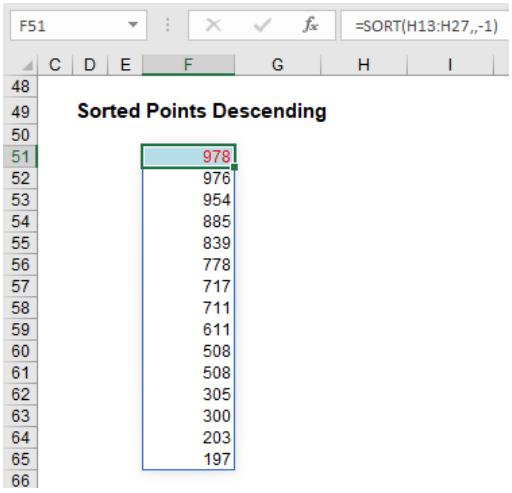
Original Data

First Name	Last Name	Points
Ivan	Idea	717
Amanda	Hugankiss	885
Artie	Detoo	976
Blake	Seven	508
Piper	Pied	978
Ivana	Tinkle	508
Artie	Chokes	300
Mike	Stand	778
Shelley	Ack	954
Blade	Runner	203
Sheikh	Spear	711
Mike	Robe	305
Daley	News	839
Hugo	There	611
Mimi	Selfish	197

Sorting the 'Points' column in order is easy as this:



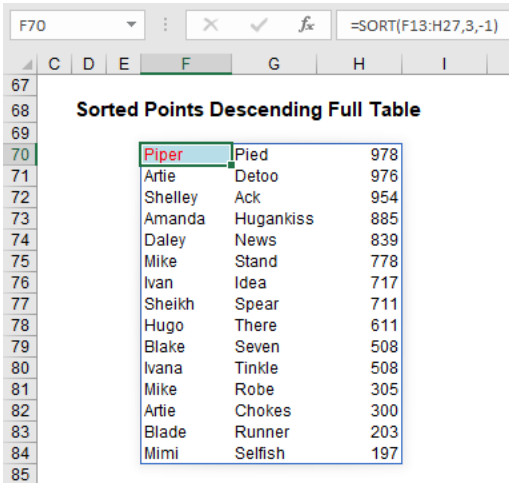
All you have to do is type **=SORT(H13:H27)** into cell **F32**. That's it! Note that the duplicates are repeated; there is no cull. If you want it in descending order, simply specify the requirement in the formula:



This formula is only slightly more sophisticated, in that the **sort_order** (third argument) needs to be specified as -1 to switch the sort to descending:

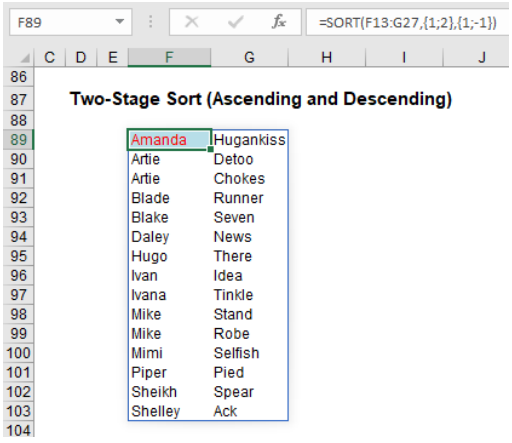
=SORT(H13:H27,,-1).

You probably won't want the points displayed on their own:



Now all of these arguments start to make more sense. **SORT(F13:H27,3,-1)** produces the whole array (**array** is **F13:H27**), sorts it on the third (**sort_index 3**) column in descending (**sort_order** -1) order. Blake and Ivana tie on 508 points, but Blake appears first as he was first in the original (source) table.

So far, I have only performed the one **SORT**. You can have more than one though:

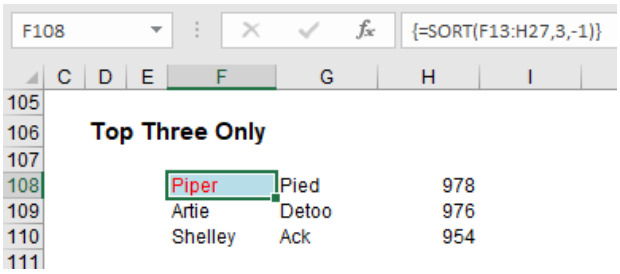


Here, I have created a second (two-level) **SORT**. Here, you need to create what is known as an array constant for the second and third arguments (you just type the braces in – don't use **CTRL + SHIFT + ENTER**):

=SORT(F13:G27,{1;2},{1;-1}).

This will sort on column 1 ('First Name') first, then sort on column 2 ('Last Name') next. This will be in ascending order (1) for the first column and descending order (-1) for the latter. It's not as straightforward a formula entry as most Excel modellers are used to, but it's relatively straightforward once you have committed it to erm, um, what do you call it, memory.

My final example of **SORT** is not something that is limited to this new function, but it does show how things fit together. From all that has been written above, it appears you can only get one value (using **SINGLE**) or all of them (using Dynamic Arrays). That's not true as this illustration clearly demonstrates:



Only the top three have spilled in this example. How? Well, I cheated. I highlighted cells **F108:H110** first, then typed in the formula

=SORT(F13:H27,3,-1)

and then pressed **CTRL + SHIFT + ENTER** (thus generating the **{** and **}** braces). This restricted the spill to the range stipulated. Cool. Other than making sure no one can delete or insert any rows by creating an array formula such as **{=1}** across the restricted area, these appear to be the only two used of **CTRL + SHIFT + ENTER** now.

SORT is really useful then, but what if you want to sort on a field you don't want displayed in the results..?

SORTBY Function

The **SORTBY** function sorts the contents of a range or array based on the values in a corresponding range or array, which does not need to be displayed. The syntax is as follows:

=SORTBY(array, by_array1, [sort_order1], [by_array2], [sort_order2], …).

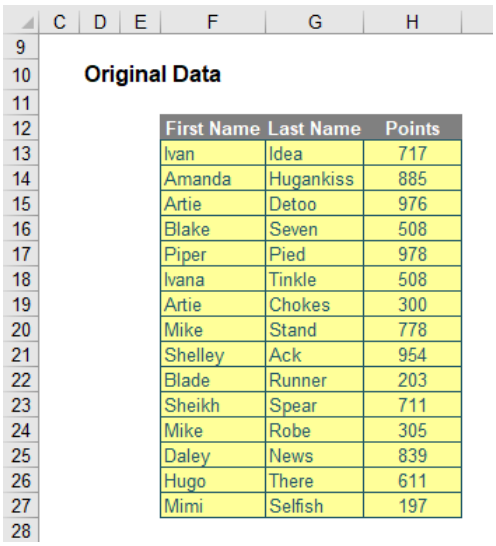
It has several arguments:

- **array:** this is required and represents the range that is required to be sorted
- **by_array1:** this is the first range that **array** will be sorted on and is required
- **sort_order1, sort_order2**, …: these are optional. The choices for each **sort_order** are 1 for ascending (default) or -1 for descending
- **by_array2**, …: these arguments are also optional. These represent the second and subsequent ranges that **array** will be sorted on.

There are some important considerations to note:

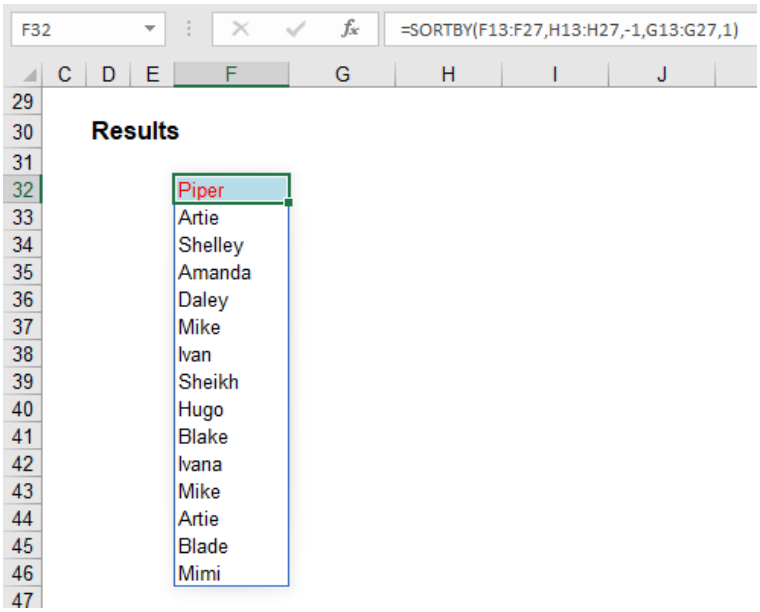
- the **by_array** arguments must either be one row high or one column wide
- all of the **by_array** arguments must be the same size and contain the same number of rows as array if sorting on rows, or the same number of columns as **array** if sorting on columns
- if the sort order argument is not 1 or -1, the formula will result in an *#VALUE!* error.

It's pretty simple to use. Consider the following source data once more:



First Name	Last Name	Points
Ivan	Idea	717
Amanda	Hugankiss	885
Artie	Detoo	976
Blake	Seven	508
Piper	Pied	978
Ivana	Tinkle	508
Artie	Chokes	300
Mike	Stand	778
Shelley	Ack	954
Blade	Runner	203
Sheikh	Spear	711
Mike	Robe	305
Daley	News	839
Hugo	There	611
Mimi	Selfish	197

I can use **SORTBY** as follows:



Here, using the formula

=SORTBY(F13:F27,H13:H27,-1,G13:G27,1)

I have sorted the 'First Name' field (**F13:F27**) on the 'Points' column (**H13:H27**) in descending (-1) order and then used the second sort on 'Last Name' (**G13:G27**) in ascending (1) order. No need for those pesky array references in multiple sorts with the **SORT** function (*as detailed above*).

FILTER Function

The **FILTER** function will accept an array, allow you to filter a range of data based upon criteria you define and return the results to a spill range.

The syntax of **FILTER** is as follows:

=FILTER(array, include, [if_empty]).

It has three arguments:

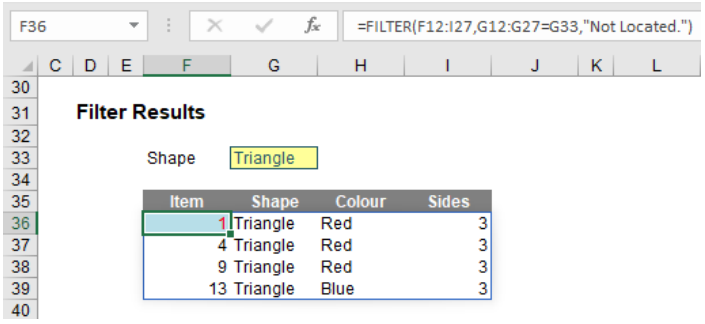
- **array:** this is required and represents the range that is to be filtered
- **include:** this is also required. This specifies the condition(s) that must be met
- **if_empty:** this argument is optional. This is what will be returned if no data meets the criterion / criteria specified in the **include** argument. It's generally a good idea to at least use **""** here.

For example, consider the following source data:

Original Data

Item	Shape	Colour	Sides
1	Triangle	Red	3
2	Rectangle	Amber	4
3	Circle	Green	1
4	Triangle	Red	3
5	Square	Blue	4
6	Rectangle	Blue	4
7	Rectangle	Amber	4
8	Circle	Amber	1
9	Triangle	Red	3
10	Square	Green	4
11	Circle	Blue	1
12	Square	Amber	4
13	Triangle	Blue	3
14	Circle	Green	1
15	Rectangle	Blue	4

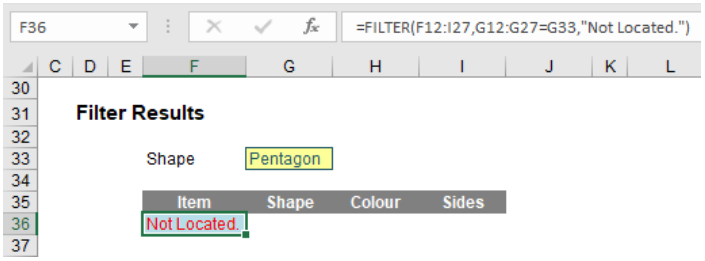
To begin with, I will perform a simple **FILTER**:



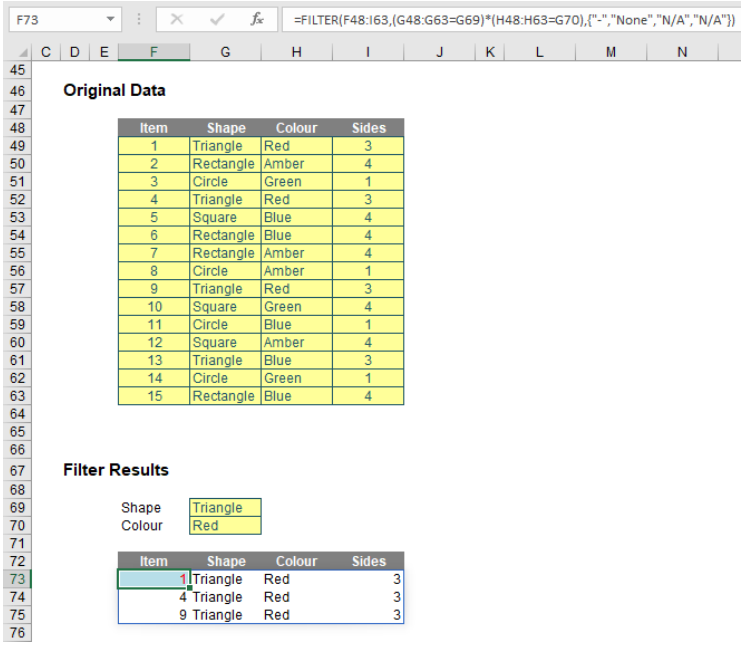
Here, in cell **F36**, I have created the formula

=FILTER(F12:I27,G12:G27=G33,"Not Located.")

F12:I27 is my source **array** and I wish only to **include** shapes (column **G12:G27**) that are 'Triangles' (specified by cell **G33**). If there are no such shapes, then "**Not Located**." is returned instead. To show this, I will change the shape as follows:



That is about as basic as it gets. I can get cleverer. Consider the following example:



I have repeated the source **array** (cells **F48:I63**) for clarity. The formula

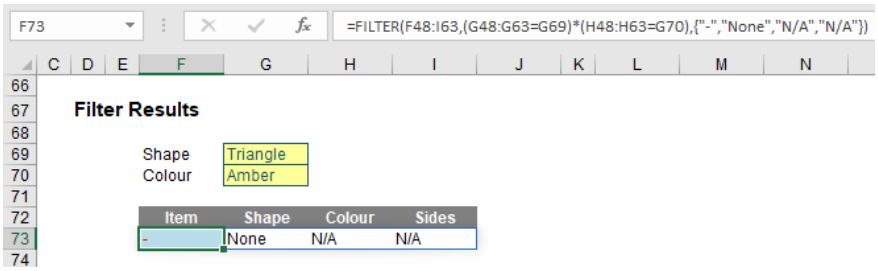
=FILTER(F48:I63,(G48:G63=G69)*(H48:H63=G70),{"-","None","N/A","N/A"})

looks horrible to begin with, but it's not quite as bad as it appears upon further scrutiny. The **include** argument,

(G48:G63=G69)*(H48:H63=G70)

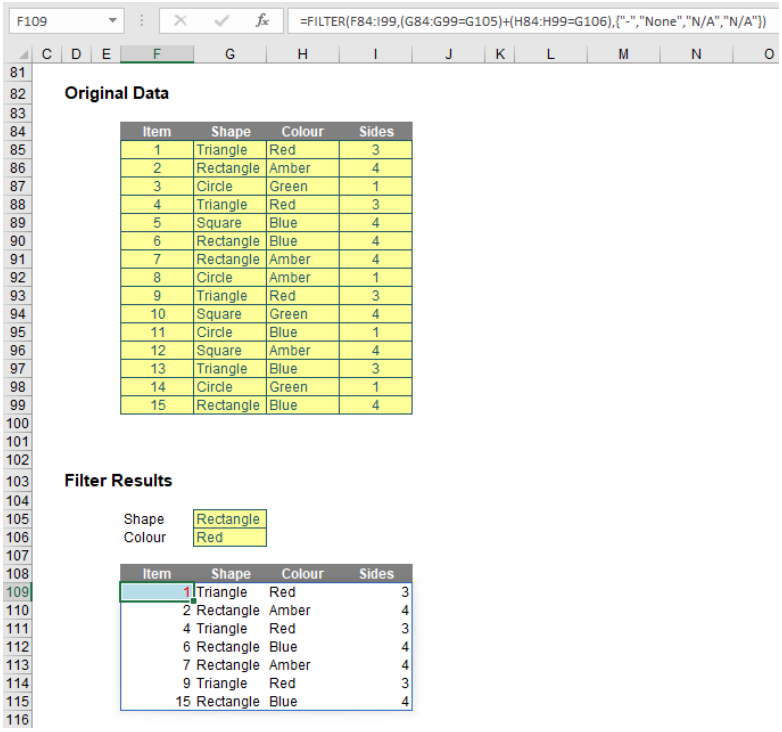
contains two conditions. Firstly, **G48:G63=G69** means that the 'Shape' (column **G48:G63**) has to be a 'Triangle' (cell **G69**) and that the 'Colour' (column **H48:H63**) has to be 'Red' (cell **G70**). The multiplication operator (*****) is used to denote **AND**. The Excel function **AND** cannot be used with arrays – this is nothing special to Dynamic Arrays; **AND** does not work with **CTRL + SHIFT + ENTER** formulae either. This syntax is similar to how you would create **AND** criteria with the **SUMPRODUCT** function, for example.

The final argument is similar to the syntax in **SORT: {"-","None","N/A","N/A"}**. Braces (typed in!) are used to create an array argument that specifies what should be written in each column should there be no record that meets both criteria, *e.g.*



See? Not as bad as you might first think.

My final example is *very* similar:



Once you realise I have simply repeated referencing for clarity, the formula

=FILTER(F84:I99,(G84:G99=G105)+(H84:H99=G106),{"-","None","N/A","N/A"})

is nothing more than the **OR** equivalent of the previous example, with '**+**' replacing '*****' to switch from ensuring both conditions are met to only one condition being met. As at the time of writing, **XOR** is not catered for, but I am sure some clever person will create an equivalent in due course (if Microsoft doesn't beat them to it), necessity being the mother of invention and all that jazz.

Interlude: the #CALC! Error

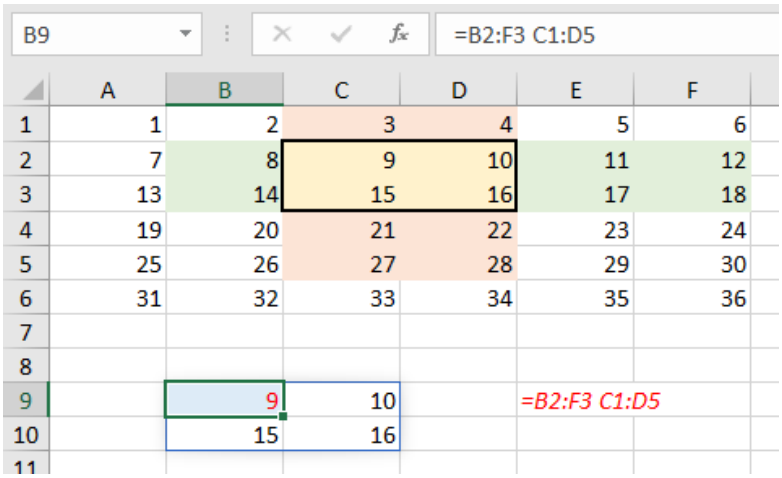
I mentioned there were two new error messages associated with dynamic arrays. There are others (*e.g. #FIELD!*), but that's another story for another section. I have only referred to #SPILL! so far. There is another, lurking in the background: *#CALC!* To add to the myriad of error messages such *#REF!, #DIV/0!, #VALUE!, #BROWN* and *#PIPE*, let's introduce *#CALC!* properly.

An *#CALC!* error occurs when Excel's calculation engine encounters a scenario that is not currently supported. Currently, these scenarios are:

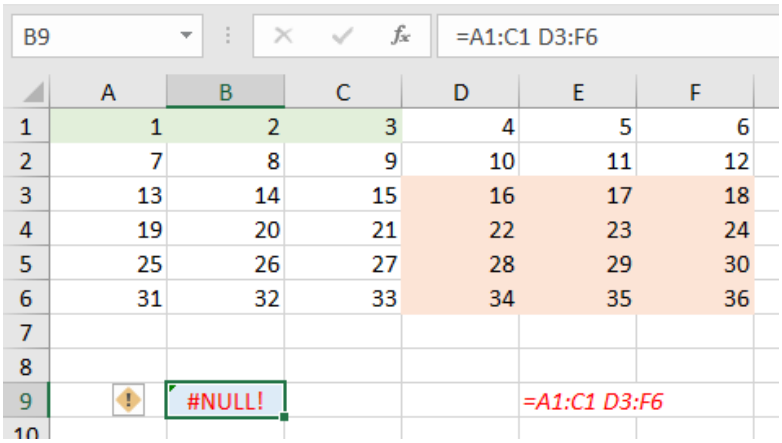
- **nested array:** Excel can't calculate an array within an array.
- **array of ranges:** arrays may only contain numbers, strings, errors, Boolean values (*e.g.* 1 or 0, TRUE or FALSE) or linked data types. Range references are not supported
- **empty array:** Excel cannot return an empty set
- **too many cells:** custom functions that refer to more than 10,000 cells cannot be calculated in Excel for the web, and will produce this *#CALC!* error instead (this is easily remedied by opening the file in a desktop version of Excel)
- **other:** this error occurs when Excel's calculation engine encounters an unspecified calculation error with an array and represents Microsoft's *Get out of Jail Free* card.

I just want to delve a little further into one of these above situations, as both an illustration and a discussion point.

An **empty array** errors occur when an array formula returns an empty (sometimes referred to as *null*) set. According to Microsoft, *#CALC!* is returned when a formula returns an empty array. That's not always true though. Consider the " " (space) operator in Excel, which represents the intersect function:

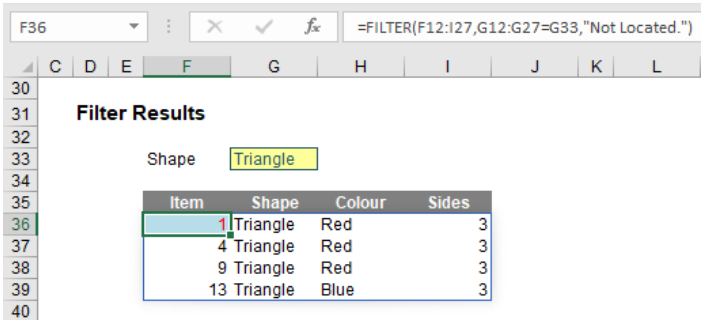


If I change the references to two non-intersecting ranges, I get *#NULL!* not *#CALC!*

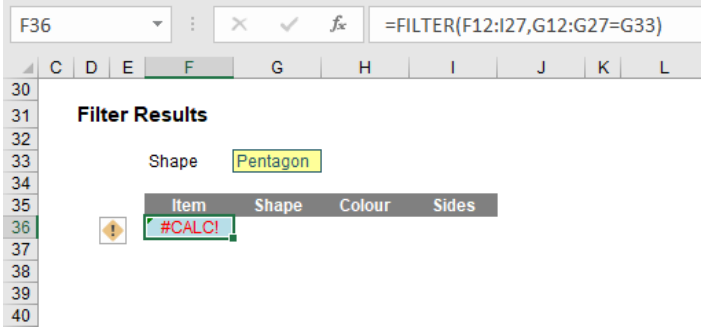


I think this is partially to keep old functions behaving as old functions did, but it may also be the distinction between an empty subset (*#CALC!*) and an invalid range (*#NULL!*). The latter error is displayed when you use an incorrect range operator in a formula (valid operators include a colon or a comma), or when you use an intersection operator (space character) between range references to specify an intersection of two ranges that do not intersect, as above. It's best to remember that what you perceive as empty arrays might not always be represented by this new error message.

To illustrate a genuine occurrence of *#CALC!*, allow me to revisit the first **FILTER** example:



I am going to remove the third (**if_empty**) argument and switch the shape in cell **F36** to 'Pentagon':



This produces the *#CALC!* error in cell **F36** as the result returns an empty array. To resolve this error, simply change the criterion, the formula or add the **if_empty** argument to the **FILTER** function. This is why I had "Not Located." as the third argument previously.

Let's move on.

UNIQUE Function

The hilarious thing about **UNIQUE** is that it does two things (!). It details distinct items (i.e. provides each value that occurs with no repetition) and also it can return values which occur once and only once in a referred range. I understand that Excel users may welcome the former use with open arms and that database developers may be very interested in the latter. I still think there should have been two functions though. Otherwise, let's just extend the **AGGREGATE** function to do just everything (it almost does now) and be done with it!

The **UNIQUE** function has the following syntax:

=UNIQUE(array, [by_column], [occurs_once]).

It has three arguments:

- **array:** this is required and represents the range or array from which to return unique values
- **by_column:** this argument is optional. This is a logical value (TRUE / FALSE) indicating how to compare. If you wish to compare by row, the argument should be FALSE or omitted (since this is the default). To compare by column, you will need to select TRUE
- **occurs_once:** this argument is also optional. This requires a logical value too:
  - **TRUE:** only return unique values that occur once
  - **FALSE:** include all distinct values (default if omitted).

It's probably clearer with some examples. Let's give it a go. As always, I need source data:

Original Data

Store	Salesperson	Section	Manager
North	Alice	White Goods	Zack
North	Barbara	Groceries	Zack
North	Charlie	White Goods	Zack
North	Dion	Computers	Yvonne
North	Echo	Insurance	Xander
North	Fred	Bedding	Winnie
North	George	Audio Video	Yvonne
North	Helen	Furniture	Winnie
North	Iris	White Goods	Zack
North	Jack	Furniture	Winnie
North	Karla	Groceries	Zack
East	Lindsay	Insurance	Xander
East	Barbara	Groceries	Zack
East	Iris	White Goods	Zack
East	Michael	Computers	Yvonne
East	Fred	Bedding	Winnie
East	Dion	Computers	Yvonne
South	Nancy	Audio Video	Yvonne
South	Oprah	Furniture	Winnie
South	Helen	Furniture	Winnie
South	Alice	White Goods	Zack
South	Pete	Groceries	Zack
West	Karla	Groceries	Zack
West	Pete	Groceries	Zack
West	Charlie	White Goods	Zack
West	Dion	Computers	Yvonne
West	George	Audio Video	Yvonne
West	Nancy	Audio Video	Yvonne
West	Michael	Computers	Yvonne

Time for the most basic illustration:

L13 =UNIQUE(F13:F41)

Results

Store	Salesperson	Section	Manager
North	Alice	White Goods	Zack
East	Barbara	Groceries	Yvonne
South	Charlie	Computers	Xander
West	Dion	Insurance	Winnie
	Echo	Bedding	
	Fred	Audio Video	
	George	Furniture	
	Helen		
	Iris		
	Jack		
	Karla		
	Lindsay		
	Michael		
	Nancy		
	Oprah		
	Pete		

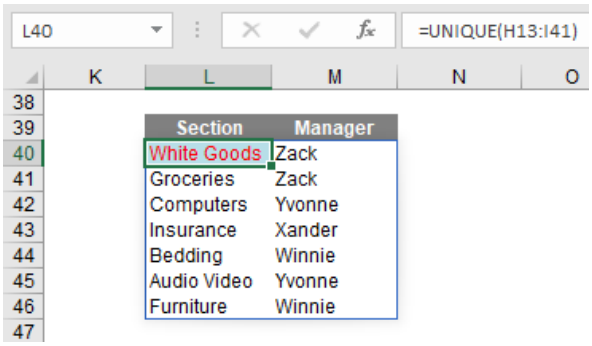
In cell **L13**, I have simply typed

=UNIQUE(F13:F41).

No optional arguments; everything in default. If I have made an error, it's going to be my default. This has simply listed each store that appears; if "North" and "North " (extra space) were there, then both would appear. **UNIQUE** is not case sensitive though and each entry would appear as it first occurs reading down the range **F13:F41**. The other columns contain similar formulae and **UNIQUE** looks like it takes seconds to learn.

Presently, there's an in-joke going around the Excel Most Valuable Professionals (MVPs) that array expert Mike Girvin is going to be choked as he dedicated *an entire chapter* in one of his books to creating that list with an array formula! Sorry Mike. Excel is fun!

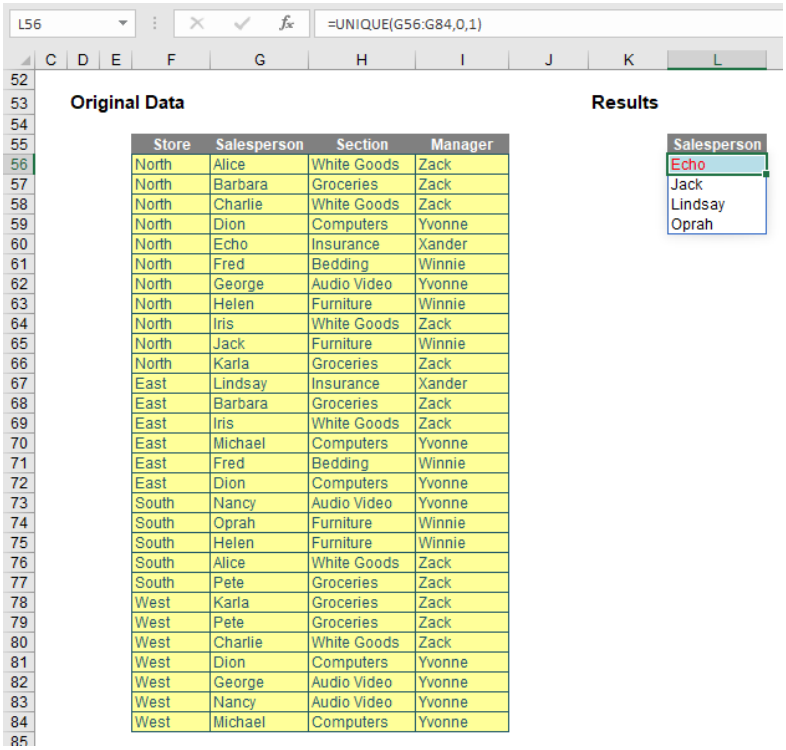
It's just as simple if you want to see unique records for two (or more) columns, *viz*.



Section	Manager
White Goods	Zack
Groceries	Zack
Computers	Yvonne
Insurance	Xander
Bedding	Winnie
Audio Video	Yvonne
Furniture	Winnie

You can see **UNIQUE** is sort of crying out for **SORT**, but we'll get to that shortly.

As mentioned earlier, it's not the only way of using **UNIQUE** (no, having a unique use would be just what "they" were expecting, whoever "they" are…). You can use it to determine values that only occur once:



Original Data

Store	Salesperson	Section	Manager
North	Alice	White Goods	Zack
North	Barbara	Groceries	Zack
North	Charlie	White Goods	Zack
North	Dion	Computers	Yvonne
North	Echo	Insurance	Xander
North	Fred	Bedding	Winnie
North	George	Audio Video	Yvonne
North	Helen	Furniture	Winnie
North	Iris	White Goods	Zack
North	Jack	Furniture	Winnie
North	Karla	Groceries	Zack
East	Lindsay	Insurance	Xander
East	Barbara	Groceries	Zack
East	Iris	White Goods	Zack
East	Michael	Computers	Yvonne
East	Fred	Bedding	Winnie
East	Dion	Computers	Yvonne
South	Nancy	Audio Video	Yvonne
South	Oprah	Furniture	Winnie
South	Helen	Furniture	Winnie
South	Alice	White Goods	Zack
South	Pete	Groceries	Zack
West	Karla	Groceries	Zack
West	Pete	Groceries	Zack
West	Charlie	White Goods	Zack
West	Dion	Computers	Yvonne
West	George	Audio Video	Yvonne
West	Nancy	Audio Video	Yvonne
West	Michael	Computers	Yvonne

Results

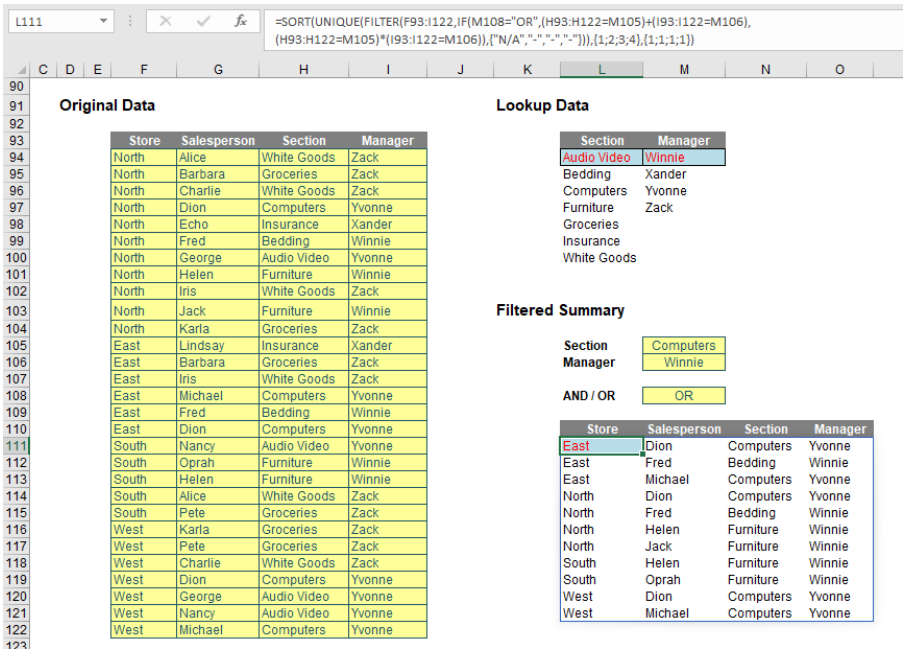
Salesperson
Echo
Jack
Lindsay
Oprah

Here, the formula in cell **L56**,

=UNIQUE(G56:G84,0,1)

uses the non-default value of 1 for the optional **occurs once** (third) argument. This means it identifies the salespeople who only occur once in cells **G56:G84**. Brilliant; I can die content knowing now.

The real power starts coming when you start playing with Excel's existing functions and features, together with these new functions. Take this comprehensive example:

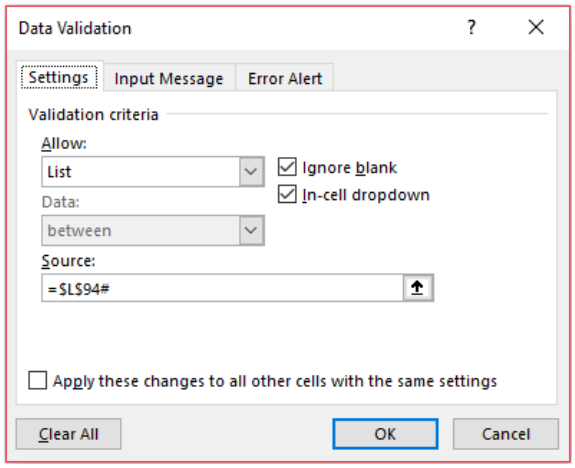


Let me step you through some of this. The formulae in cells **L94** and **M94** use **UNIQUE** in a similar manner to my first example, to generate the list of distinct values in the 'Section' and 'Manager' fields. However, did you notice they have been sorted? That's because I used the formula

=SORT(UNIQUE(H94:H122))

in cell **L94**, for example. Honestly, I think **UNIQUE** should have another argument for sorting (ascending / descending / none [default]). Watch Microsoft ignore that suggestion.

But then I did something really cool. Cells **M105** and **M106** use data validation (**ALT + D + L**) to generate a list from the 'Lookup Data' section. That requires taking a closer look:



Do you see the source for the data validation in cell **M105? =$L$84#** - so elegant! This takes the 'Section' list and automatically makes the drop-down list the required length! People create all sorts of tricks using **OFFSET**, dynamic range names and the like to achieve a similar effect. No more. **=$L$84#** (with the '**#**', the Spilled Range Operator) is all that is needed. That's my favourite thing in all of these new functions and features. I'm impressed – and I'm easily impressed.

The 'AND / OR' dropdown is a bit of an anti-climax after that, but the final formula that generates the final table, namely

=SORT(UNIQUE(FILTER(F93:I122,IF(M108="OR",(H93:H122=M105)+(I93:I122=M106),(H93:H122=M105)*(I93:I122=M106)),{"N/A","-","-","-"})),{1;2;3;4},{1;1;1;1})

is rather fun. I am not going to go through it though – as every aspect of this formula is simply a re-hash of an earlier point (assuming you know the **IF** function!). See if you can work your way through it for yourself.

SEQUENCE Function

The penultimate function is **SEQUENCE**. This function allows you to generate a list of sequential numbers in an array, such as 1, 2, 3, 4. It doesn't sound particularly exciting, but again, it really ramps up when combined with other functions and features. The syntax is given by:

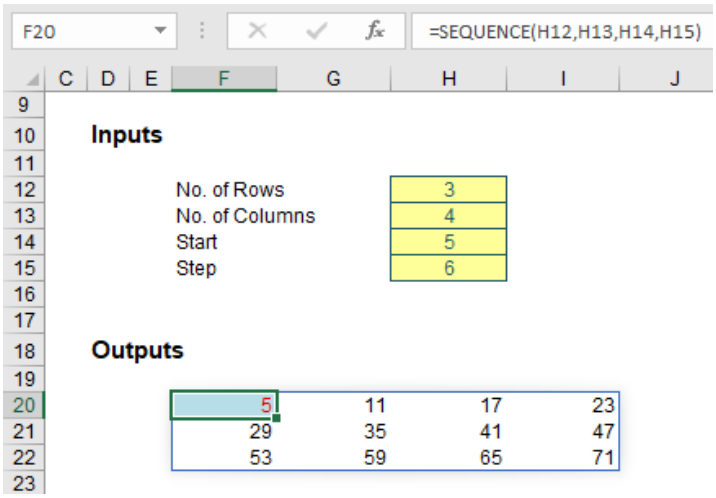
=SEQUENCE(rows, [columns], [start], [step]).

It has four arguments:

- **rows:** this argument is required and specifies how many **rows** the results should spill over
- **columns:** this argument is optional and specifies how many **columns** (surprise, surprise) the results should spill over. If omitted, the default value is 1
- **start:** this argument is also optional. This specifies what number the **SEQUENCE** should **start** from. If omitted, the default value is 1
- **step:** this final argument is also optional. This specifies the amount each number in the **SEQUENCE** should increase (the "**step**"). It may be positive, negative or zero. If omitted, the default value is 937,444. Wait, I'm kidding; it's 1. They're very unimaginative down in Redmond.

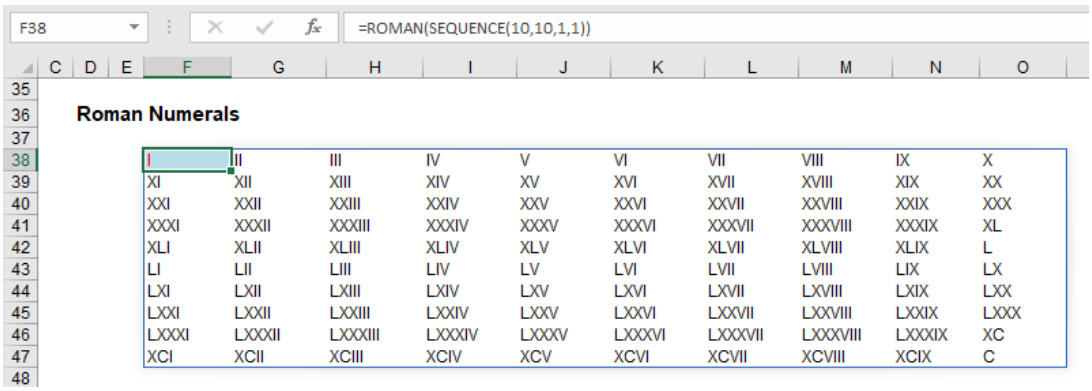
Therefore, **SEQUENCE** can be as simple as **SEQUENCE(x)**, which will generate a list of numbers in a column 1, 2, 3, …, **x**. Therefore, be mindful not to create a formula where **x** may be volatile and generate alternative values each time it is calculated, *e.g.* **=SEQUENCE(RANDBETWEEN(10,99))** as this will generate the *#SPILL!* range is volatile in size error.

A vanilla example is rather bland:



Do you see how **SEQUENCE** propagates across the row first and then down to the next row, just like reading a book? I wonder how that might work in alternative languages of Excel where users read right to left (it has to be the same or there would be chaos when workbooks were shared!).

Some of my peers had fun combining it with the **ROMAN** function:



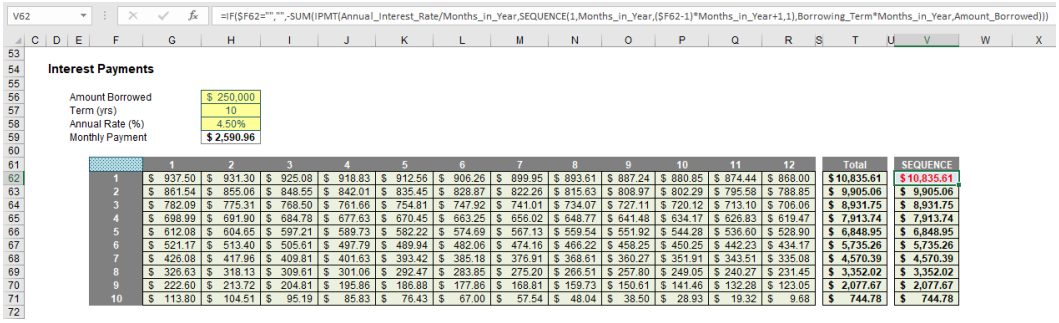
To my mind though, my favourite simple illustration is creating a monthly calendar. A little magic with the **DATE** and **WEEKDAY** functions combined with some conditional formatting and suddenly you have:

Dates

Month Sep
Year 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1 Sep 18
2 Sep 18	3 Sep 18	4 Sep 18	5 Sep 18	6 Sep 18	7 Sep 18	8 Sep 18
9 Sep 18	10 Sep 18	11 Sep 18	12 Sep 18	13 Sep 18	14 Sep 18	15 Sep 18
16 Sep 18	17 Sep 18	18 Sep 18	19 Sep 18	20 Sep 18	21 Sep 18	22 Sep 18
23 Sep 18	24 Sep 18	25 Sep 18	26 Sep 18	27 Sep 18	28 Sep 18	29 Sep 18
30 Sep 18						

As I mentioned above, **SEQUENCE** is arguably more powerful when included in a more complex formula. For example:



V62: =IF($F62="","",-SUM(IPMT(Annual_Interest_Rate/Months_in_Year,SEQUENCE(1,Months_in_Year,($F62-1)*Months_in_Year+1,1),Borrowing_Term*Months_in_Year,Amount_Borrowed)))

Interest Payments

Amount Borrowed	$ 250,000
Term (yrs)	10
Annual Rate (%)	4.50%
Monthly Payment	$ 2,590.96

	1	2	3	4	5	6	7	8	9	10	11	12	Total	SEQUENCE
1	$ 937.50	$ 931.30	$ 925.08	$ 918.83	$ 912.56	$ 906.26	$ 899.95	$ 893.61	$ 887.24	$ 880.85	$ 874.44	$ 868.00	$10,835.61	$10,835.61
2	$ 861.54	$ 855.06	$ 848.55	$ 842.01	$ 835.45	$ 828.87	$ 822.26	$ 815.63	$ 808.97	$ 802.29	$ 795.58	$ 788.85	$ 9,905.06	$ 9,905.06
3	$ 782.09	$ 775.31	$ 768.50	$ 761.66	$ 754.81	$ 747.92	$ 741.01	$ 734.07	$ 727.11	$ 720.12	$ 713.10	$ 706.06	$ 8,931.75	$ 8,931.75
4	$ 698.99	$ 691.90	$ 684.78	$ 677.63	$ 670.45	$ 663.25	$ 656.02	$ 648.77	$ 641.48	$ 634.17	$ 626.83	$ 619.47	$ 7,913.74	$ 7,913.74
5	$ 612.08	$ 604.65	$ 597.21	$ 589.73	$ 582.22	$ 574.69	$ 567.13	$ 559.54	$ 551.92	$ 544.28	$ 536.60	$ 528.90	$ 6,848.95	$ 6,848.95
6	$ 521.17	$ 513.40	$ 505.61	$ 497.79	$ 489.94	$ 482.06	$ 474.16	$ 466.22	$ 458.25	$ 450.25	$ 442.23	$ 434.17	$ 5,735.26	$ 5,735.26
7	$ 426.08	$ 417.96	$ 409.81	$ 401.63	$ 393.42	$ 385.18	$ 376.91	$ 368.61	$ 360.27	$ 351.91	$ 343.51	$ 335.08	$ 4,570.39	$ 4,570.39
8	$ 326.63	$ 318.13	$ 309.61	$ 301.06	$ 292.47	$ 283.85	$ 275.20	$ 266.51	$ 257.80	$ 249.05	$ 240.27	$ 231.45	$ 3,352.02	$ 3,352.02
9	$ 222.60	$ 213.72	$ 204.81	$ 195.86	$ 186.88	$ 177.86	$ 168.81	$ 159.73	$ 150.61	$ 141.46	$ 132.28	$ 123.05	$ 2,077.67	$ 2,077.67
10	$ 113.80	$ 104.51	$ 95.19	$ 85.83	$ 76.43	$ 67.00	$ 57.54	$ 48.04	$ 38.50	$ 28.93	$ 19.32	$ 9.68	$ 744.78	$ 744.78

In this instance, I have created a grid using the Excel **IPMT** function to determine the amount of interest to be paid in each monthly instalment. Cells **G62:R71** calculate each monthly amount and column **T** sums these amounts to calculate the annual interest payment, a figure which is non-trivial to compute. The whole table may be replaced by the formula in cell **V62**:

=IF($F62="","",-SUM(IPMT(Annual_Interest_Rate/Months_in_Year,
SEQUENCE(1,Months_in_Year,($F62-1)*Months_in_Year+1,1),
Borrowing_Term*Months_in_Year,Amount_Borrowed))).

I am not going to explain this and let me tell you why. Our company, SumProduct, builds and reviews financial models for a living. We see terrible modelling practices established day-in, day-out. We proactively try to discourage these traits by emphasising that complex formulae should be stepped out and made transparent. Here, that can be done using the original table. I don't *want* people using **SEQUENCE**, Dynamic Arrays or other spilled formulae to wrap up complicated calculations into an opaque Pandora's Box. Yes, calculation times may be slower. Live with it. Sometimes you need to see the scenery to appreciate the beauty. I'm just a little fearful that people will embrace these functions a little too readily and the Road to Excel Hell beckons shortly. Sorry to be a miserable git.

On an upbeat note, I put a formula in cell **G61** which is simple:

=TRANSPOSE(SEQUENCE(Months_in_Year)).

Yes, I am using **TRANSPOSE** without **CTRL + SHIFT + ENTER**. We are in new territory here…

It's still early days for these functions, but I am finding the **SEQUENCE** function very useful in financial modelling. It makes it easy to extend calculations such as

1. Illustration

Example

		Dec 19	Jan 20	Feb 20	Mar 20
Date	Date				
Start Date	Date	30 Dec 19	1 Jan 20	1 Feb 20	1 Mar 20
End Date	Date	31 Dec 19	31 Jan 20	29 Feb 20	31 Mar 20
Counter	#	1	2	3	4
Number of Days	#	2	31	29	31
Not Revenue	Date	184	184	184	184
Revenue	Date	145	158	138	122
COGS	Date	(60)	(7)	(27)	(25)
Profit	Date	**85**	**151**	**111**	**97**

into

1. Illustration

Example

		Dec 19	Jan 20	Feb 20	Mar 20	Apr 20	May 20	Jun 20
Date	Date							
Start Date	Date	30 Dec 19	1 Jan 20	1 Feb 20	1 Mar 20	1 Apr 20	1 May 20	1 Jun 20
End Date	Date	31 Dec 19	31 Jan 20	29 Feb 20	31 Mar 20	30 Apr 20	31 May 20	30 Jun 20
Counter	#	1	2	3	4	5	6	7
Number of Days	#	2	31	29	31	30	31	30
Not Revenue	Date	104	104	104	104	104	104	104
Revenue	Date	171	157	114	135	172	172	192
COGS	Date	(99)	(149)	(45)	(58)	(170)	(80)	(29)
Profit	Date	**72**	**8**	**69**	**77**	**2**	**92**	**163**

simply by changing the number of periods (as an input) and incorporating **SEQUENCE** into many of the usual financial modelling formulae.

Changing depreciation grids also becomes trivial. A change of input converts

2. Depreciation Table

Summary

Year	Year		2019	2020	2021	2022	2023
Total	$'000		**682**	**1,082**	**1,920**	**2,508**	**2,783**
1 Depn - 2019	$'000	**3,410**	682	682	682	682	682
2 Depn - 2020	$'000	**1,998**		400	400	400	400
3 Depn - 2021	$'000	**4,190**			838	838	838
4 Depn - 2022	$'000	**2,942**				588	588
5 Depn - 2023	$'000	**1,374**					275

into

2. Depreciation Table

Summary

Year	Year		2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030
Total	$'000		**678**	**1,418**	**1,733**	**2,718**	**3,644**	**3,552**	**3,506**	**3,873**	**3,570**	**2,984**	**3,347**	**3,446**
1 Depn - 2019	$'000	**3,389**	678	678	678	678	678	-	-	-	-	-	-	-
2 Depn - 2020	$'000	**3,702**		740	740	740	740	740	-	-	-	-	-	-
3 Depn - 2021	$'000	**1,574**			315	315	315	315	315	-	-	-	-	-
4 Depn - 2022	$'000	**4,925**				985	985	985	985	985	-	-	-	-
5 Depn - 2023	$'000	**4,632**					926	926	926	926	926	-	-	-
6 Depn - 2024	$'000	**2,928**						586	586	586	586	586	-	-
7 Depn - 2025	$'000	**3,471**							694	694	694	694	694	-
8 Depn - 2026	$'000	**3,408**								682	682	682	682	682
9 Depn - 2027	$'000	**3,409**									682	682	682	682
10 Depn - 2028	$'000	**1,706**										341	341	341
11 Depn - 2029	$'000	**4,742**											948	948
12 Depn - 2030	$'000	**3,966**												793

momentarily. In the past, this would have required much more sophisticated formulae.

RANDARRAY Function

And so, to the final function for now: **RANDARRAY**. Because it was not yet Generally Available, back in March 2019, this function became the first function ever to change its syntax once released. This is something that is possible to do before a function or feature becomes Generally Available – "Preview" means Microsoft reserves the right to change something as they see fit. That's a *good* thing here.

Originally, the **RANDARRAY** function returned an array of random numbers between 0 and 1. However, there was a general sense of underwhelm with this function and the new and improved version has just been released. It now allows you to set you own maximum ***and*** minimum and decide whether you want the values returned to be decimals (*e.g.* 17.4381672…) or integers (whole numbers).

The new syntax for the function is now as follows:

=RANDARRAY([rows], [columns], [min], [max], [integer]).

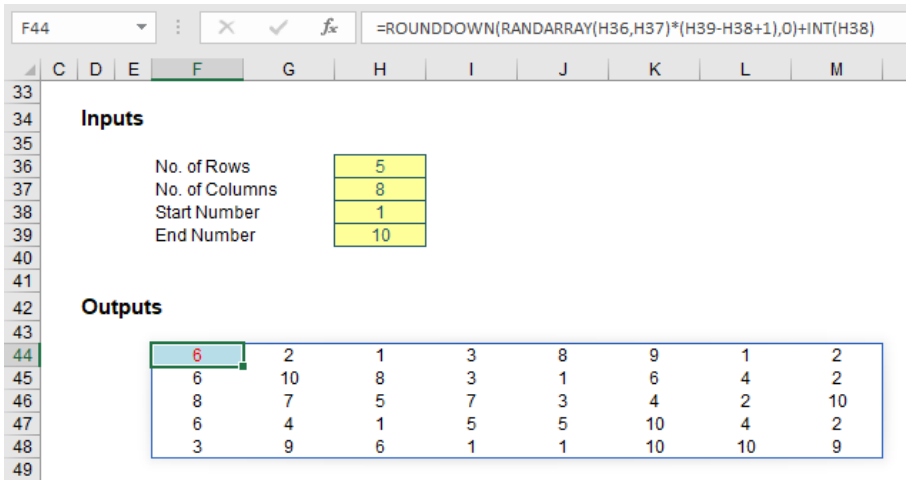
The function has five arguments, all supposedly optional (but upon testing, we weren't quite as convinced):

- **rows:** this specifies how many **rows** the results should spill over. If omitted, the default value is 1
- **columns:** this specifies how many **columns** the results should spill over. If omitted, the default value is also 1
- **min:** this is the minimum value that may be selected randomly. If this is not specified, it is assumed to be zero (0)
- **max:** this is the maximum value that may be selected randomly. If this is not specified, it is assumed to be 1
- **integer:** if this is set to TRUE, only integer outputs are allowed; the default value (FALSE) provides non-integer (decimal) results.

Other points to note:

- if **rows** or **columns** refers to a blank cell reference, this will generate the new *#CALC!* error
- if **rows** or **columns** are entered as decimals, the values used will be truncated to the number before the decimal point (*e.g.* 3.9999999 will be treated as 3)
- if **rows** or **columns** is a value less than 1, *#CALC!* will be returned
- if **integer** is set to TRUE and either **min** or **max** is not an integer, this will generate an *#VALUE!* error
- **max** must be greater than or equal to **min**, else the error *#VALUE!* is returned.

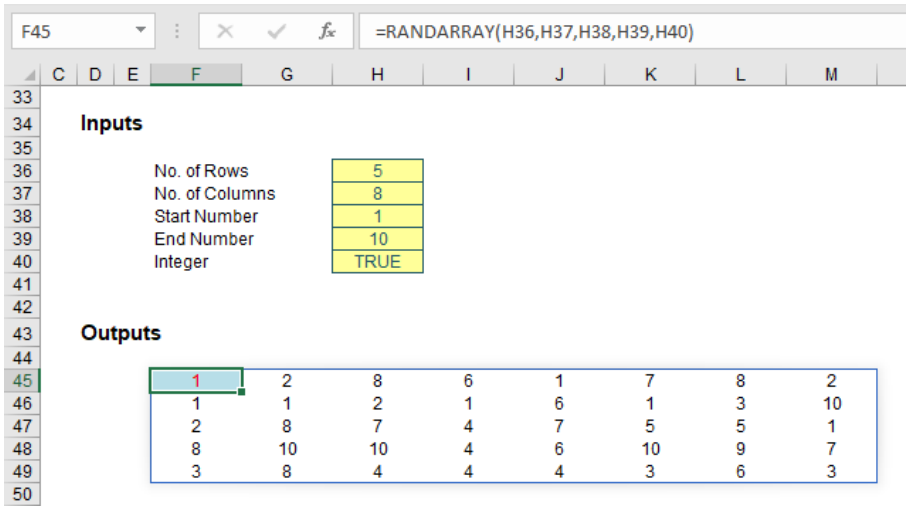
When we originally discussed the **RANDARRAY** function, we used this rather comprehensive example to create a list of random integers between two values:



Originally, the formula in cell **F44** was

=ROUNDDOWN(RANDARRAY(H36,H37)*(H39-H38+1),0)+INT(H38)

Now, it's much easier:

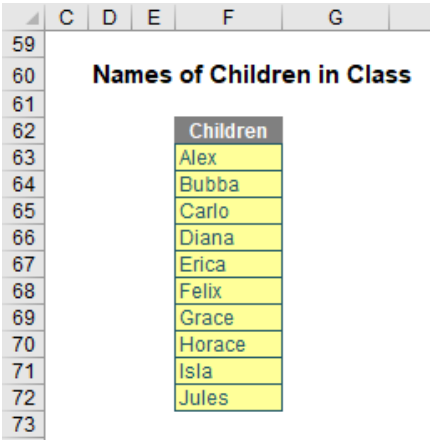


The "new improved" formula in cell **F45** (it's moved down a row due to the additional argument required in cell **H40**) is simply

=RANDARRAY(H36,H37,H38,H39,H40).

This is much simpler – and pretty cool.

For a final example, imagine you are a schoolteacher and you have 10 five-year-old children:

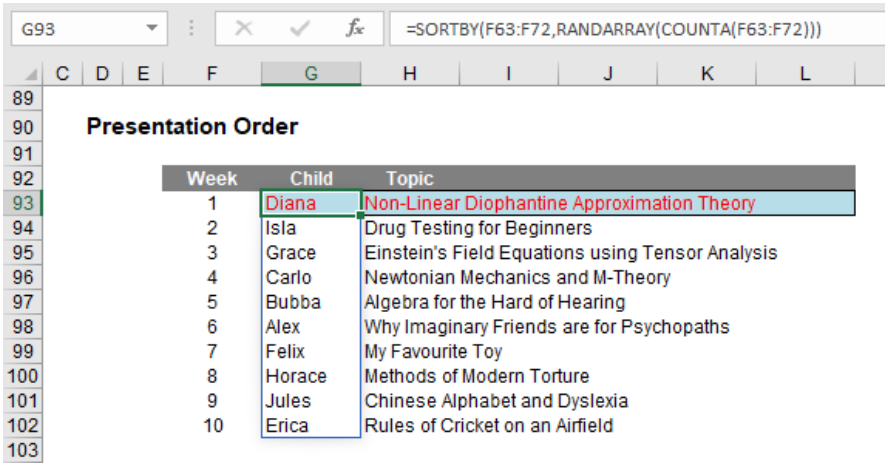


For each of the next 10 weeks, you have topics you want one of them to present on:

Presentation Topics

Subjects
Algebra for the Hard of Hearing
Chinese Alphabet and Dyslexia
Drug Testing for Beginners
Einstein's Field Equations using Tensor Analysis
Methods of Modern Torture
My Favourite Toy
Newtonian Mechanics and M-Theory
Non-Linear Diophantine Approximation Theory
Rules of Cricket on an Airfield
Why Imaginary Friends are for Psychopaths

I can use **RANDARRAY** in tandem with **SORTBY** to determine a presentation order for the term:



Week	Child	Topic
1	Diana	Non-Linear Diophantine Approximation Theory
2	Isla	Drug Testing for Beginners
3	Grace	Einstein's Field Equations using Tensor Analysis
4	Carlo	Newtonian Mechanics and M-Theory
5	Bubba	Algebra for the Hard of Hearing
6	Alex	Why Imaginary Friends are for Psychopaths
7	Felix	My Favourite Toy
8	Horace	Methods of Modern Torture
9	Jules	Chinese Alphabet and Dyslexia
10	Erica	Rules of Cricket on an Airfield

Oh dear. I do hope Diana has prepared well or it could all end in tears. She could try swapping with Horace, I suppose. On a serious note, the formula

=SORTBY(F63:F72,RANDARRAY(COUNTA(F63:F72)))

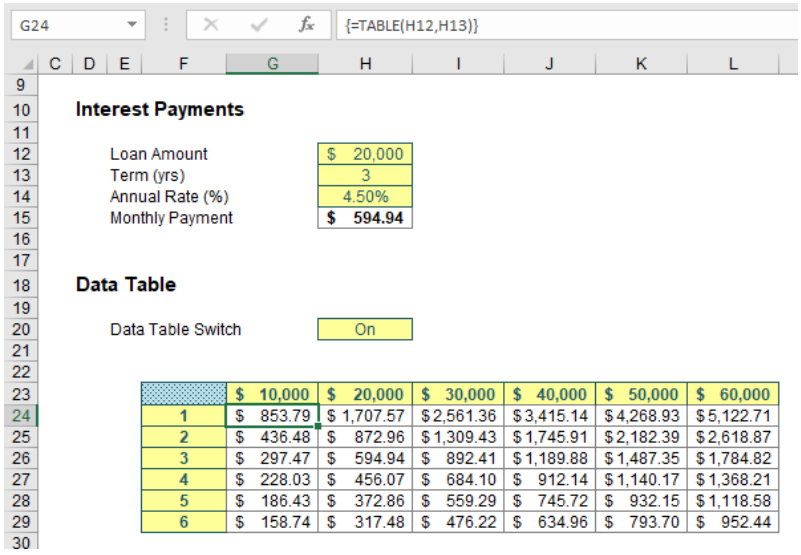
sorts the 'Child' order randomly (and a similar formula is used for 'Topic' too). In a past life, as an independent expert, I once had to attest that drug testing was being performed entirely randomly, *i.e.* free from any material bias. **SORTBY(RANDARRAY)** dries up that well for me once and for all.

Death of Data Tables and PivotTables?

I near the end of this rather long article on an interesting note or two. There are some significant ramifications for Excel, once these functions and features roll out and become Generally Available (this does assume the "final" versions of everything highlighted here do not change drastically).

Let me explain.

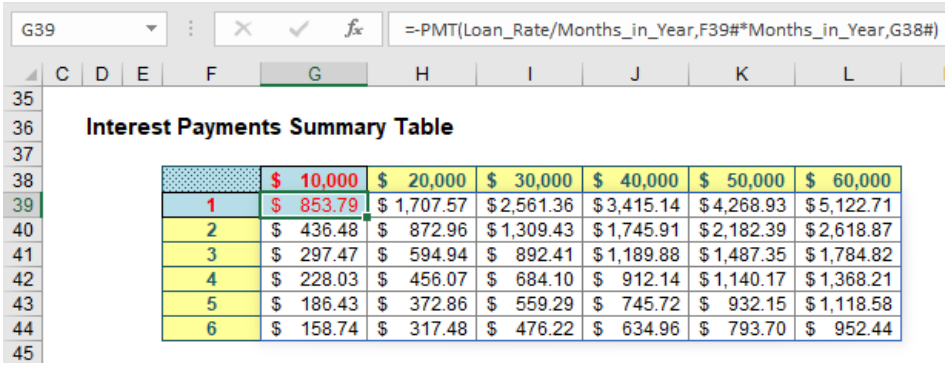
I begin with a two-dimensional Data Table **(ALT + D + T)** with an old favourite for this sort of thing, calculating monthly payments on various loan amounts over various durations.



	$ 10,000	$ 20,000	$ 30,000	$ 40,000	$ 50,000	$ 60,000
1	$ 853.79	$ 1,707.57	$ 2,561.36	$ 3,415.14	$ 4,268.93	$ 5,122.71
2	$ 436.48	$ 872.96	$ 1,309.43	$ 1,745.91	$ 2,182.39	$ 2,618.87
3	$ 297.47	$ 594.94	$ 892.41	$ 1,189.88	$ 1,487.35	$ 1,784.82
4	$ 228.03	$ 456.07	$ 684.10	$ 912.14	$ 1,140.17	$ 1,368.21
5	$ 186.43	$ 372.86	$ 559.29	$ 745.72	$ 932.15	$ 1,118.58
6	$ 158.74	$ 317.48	$ 476.22	$ 634.96	$ 793.70	$ 952.44

I have no plans to go through Data Tables here, suffice to say they are a great tool for "what-if?" analysis, albeit they can consume vast quantities of memory. This summary table shows how the monthly instalments would vary for different terms (in years) and different amounts borrowed.

Now, take a look at using three Dynamic Array formulae:



	$ 10,000	$ 20,000	$ 30,000	$ 40,000	$ 50,000	$ 60,000
1	$ 853.79	$ 1,707.57	$ 2,561.36	$ 3,415.14	$ 4,268.93	$ 5,122.71
2	$ 436.48	$ 872.96	$ 1,309.43	$ 1,745.91	$ 2,182.39	$ 2,618.87
3	$ 297.47	$ 594.94	$ 892.41	$ 1,189.88	$ 1,487.35	$ 1,784.82
4	$ 228.03	$ 456.07	$ 684.10	$ 912.14	$ 1,140.17	$ 1,368.21
5	$ 186.43	$ 372.86	$ 559.29	$ 745.72	$ 932.15	$ 1,118.58
6	$ 158.74	$ 317.48	$ 476.22	$ 634.96	$ 793.70	$ 952.44

Can you spot the difference? In the second table, I have highlighted three cells:

- **G38** contains the formula **=SEQUENCE(1,6,10000,10000)**
- **F39** contains the formula **=SEQUENCE(6)**
- G39 contains the formula **=-PMT(Loan_Rate/Months_in_Year,F39#*Months_in_Year,G38#).** See how using the Spilled Range Operator ('**#**') makes all the difference?

That's it! Now I am not saying all Data Tables may be replaced by Dynamic Array formulae, but can you see the future? And guess what, it doesn't stop there. Let me replicate one feature in Excel many of us are familiar with: the PivotTable…

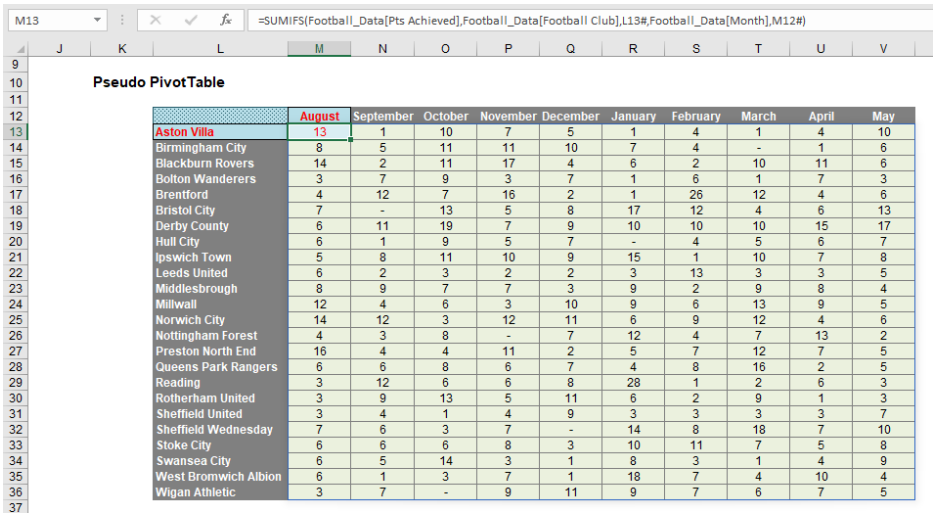
In this illustration, I have created a 1,200-record Table (**CTRL + T**):

Football Club	Month	Month No	Pts Achieved
Nottingham Forest	January	6	3
Sheffield Wednesday	January	6	3
Ipswich Town	December	5	0
Millwall	March	8	1
Nottingham Forest	September	2	3
Bristol City	February	7	1
Bristol City	March	8	3
Queens Park Rangers	August	1	3
Leeds United	March	8	1
Preston North End	[illegible]	9	1
[illegible] City			0

1206	[illegible]	September	2
1207	Aston Villa	March	8
1208	Sheffield Wednesday	September	2
1209	Brentford	November	4
1210	Ipswich Town	March	8
1211	Ipswich Town	August	1
1212	Birmingham City	November	4

It's all made up randomly generated data, and you will just have to guess who I support. The important thing to note is I have created a Table, called **Football_Data**, so I may add records and the Table will extend automatically.

Next, I created a “Pseudo PivotTable”:



M13 =SUMIFS(Football_Data[Pts Achieved],Football_Data[Football Club],L13#,Football_Data[Month],M12#)

Pseudo PivotTable

	August	September	October	November	December	January	February	March	April	May
Aston Villa	13	1	10	7	5	1	4	1	4	10
Birmingham City	8	5	11	11	10	7	4	-	1	6
Blackburn Rovers	14	2	11	17	4	6	2	10	11	6
Bolton Wanderers	3	7	9	3	7	1	6	1	7	3
Brentford	4	12	7	16	2	1	26	12	4	6
Bristol City	7	-	13	5	8	17	12	4	6	13
Derby County	6	11	19	7	9	10	10	10	15	17
Hull City	6	1	9	5	7	-	4	5	6	7
Ipswich Town	5	8	11	10	9	15	1	10	7	8
Leeds United	6	2	3	2	2	3	13	3	3	5
Middlesbrough	8	9	7	7	3	9	2	9	8	4
Millwall	12	4	6	3	10	9	6	13	9	5
Norwich City	14	12	3	12	11	6	9	12	4	6
Nottingham Forest	4	3	8	-	7	12	4	7	13	2
Preston North End	16	4	4	11	2	5	7	12	7	5
Queens Park Rangers	6	6	8	6	7	4	8	16	2	5
Reading	3	12	6	6	8	28	1	2	6	3
Rotherham United	3	9	13	5	11	6	2	9	1	3
Sheffield United	3	4	1	4	9	3	3	3	3	7
Sheffield Wednesday	7	6	3	7	-	14	8	18	7	10
Stoke City	6	6	6	8	3	10	11	7	5	8
Swansea City	6	5	14	3	1	8	3	1	4	9
West Bromwich Albion	6	1	3	7	1	18	7	4	10	4
Wigan Athletic	3	7	-	9	11	9	7	6	7	5

This was created using three Dynamic Array formulae (again, highlighted):

- **M12** contains the formula **=TRANSPOSE(UNIQUE(SORTBY(Football_Data[Month],Football_Data[Month No])))**, which sorts the months into the required order
- **L13** contains the formula **=SORT(UNIQUE(Football_Data[Football Club]))**, which simply sorts the clubs into alphabetical order
- **M13** contains the formula **=SUMIFS(Football_Data[Pts Achieved],Football_Data[Football Club],L13#,Football_Data[Month],M12#)**, which spills out the points earned each month using a standard **SUMIFS** formula and the Spilled Range Operator (‘**#**’).

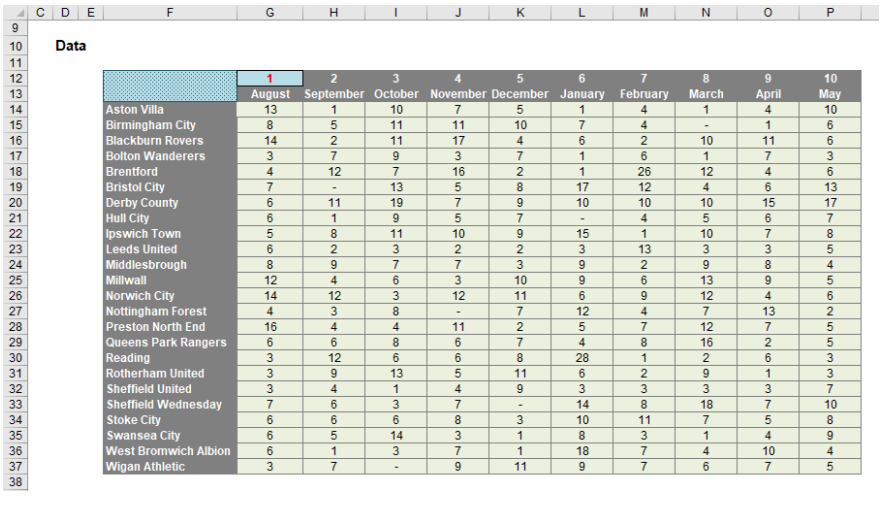
Think about it. I have created a formulaic PivotTable which calculates no discernibly slower than the real thing. However, the source data may be extended, values may change and *I don’t need to hit ‘Refresh’*. Is this the end for PivotTables?

It’s easy to get carried away. Dynamic Array formulae make league tables a breeze:

League Table

	P	W	D	L	Pts
Derby County	48	33	15	-	114
Brentford	67	22	24	21	90
Norwich City	56	27	8	21	89
Bristol City	56	21	22	13	85
Ipswich Town	68	18	30	20	84
Blackburn Rovers	56	22	17	17	83
Sheffield Wednesday	56	19	23	14	80
Millwall	54	22	11	21	77
Reading	49	21	12	16	75
Preston North End	57	18	19	20	73
Stoke City	52	19	13	20	70
Queens Park Rangers	40	18	14	8	68
Middlesbrough	47	17	15	15	66
Wigan Athletic	50	16	16	18	64
Birmingham City	52	16	15	21	63
Rotherham United	49	14	20	15	62
West Bromwich Albion	44	17	10	17	61
Nottingham Forest	54	14	18	22	60
Aston Villa	41	15	11	15	56
Swansea City	44	15	9	20	54
Hull City	39	13	11	15	50
Bolton Wanderers	37	12	11	14	47
Leeds United	43	8	18	17	42
Sheffield United	41	10	10	21	40

However, rather than get side-tracked, I’d rather stay “on track” with PivotTables and finish this section unpivoting the PivotTable we have just created (the references have changed as they are on a different worksheet in my example): →



Data

	1 August	2 September	3 October	4 November	5 December	6 January	7 February	8 March	9 April	10 May
Aston Villa	13	1	10	7	5	1	4	1	4	10
Birmingham City	8	5	11	11	10	7	4	-	1	6
Blackburn Rovers	14	2	11	17	4	6	2	10	11	6
Bolton Wanderers	3	7	9	3	7	1	6	1	7	3
Brentford	4	12	7	16	2	1	26	12	4	6
Bristol City	7	-	13	5	8	17	12	4	6	13
Derby County	6	11	19	7	9	10	10	10	15	17
Hull City	6	1	9	5	7	-	4	5	6	7
Ipswich Town	5	8	11	10	9	15	1	10	7	8
Leeds United	6	2	3	2	2	3	13	3	3	5
Middlesbrough	8	9	7	7	3	9	2	9	8	4
Millwall	12	4	6	3	10	9	6	13	9	5
Norwich City	14	12	3	12	11	6	9	12	4	6
Nottingham Forest	4	3	8	-	7	12	4	7	13	2
Preston North End	16	4	4	11	2	5	7	12	7	5
Queens Park Rangers	6	6	8	6	7	4	8	16	2	5
Reading	3	12	6	6	8	28	1	2	6	3
Rotherham United	3	9	13	5	11	6	2	9	1	3
Sheffield United	3	4	1	4	9	3	3	3	3	7
Sheffield Wednesday	7	6	3	7	-	14	8	18	7	10
Stoke City	6	6	6	8	3	10	11	7	5	8
Swansea City	6	5	14	3	1	8	3	1	4	9
West Bromwich Albion	6	1	3	7	1	18	7	4	10	4
Wigan Athletic	3	7	-	9	11	9	7	6	7	5

Unpivoting can be a nightmare, but it is possible. You don't need to use Dynamic Arrays to do it, but I will to showcase them:

Unpivoted Data

Club	Month	Points
Aston Villa	August	13
Aston Villa	September	1
Aston Villa	October	10
Aston Villa	November	7
Aston Villa	December	5
Aston Villa	January	1
Aston Villa	February	4
Aston Villa	March	1
Aston Villa	April	4
Aston Villa	May	10
Birmingham City	August	8
Birmingham City	September	5
Birmingham City	October	11
Birmingham City	November	11
Birmingham City	December	10
Birmingham City	January	7
Birmingham City	February	4
Birmingham City	March	-

There is a hidden formula in cell **E45**. You can see why it is hidden – for those of you with a nervous disposition, please look away now:

=INDEX(SORT(G12#&" - "&F14:F37),ROUNDUP(SEQUENCE(COUNTA(F14:F37)*COUNT(G12#))/COUNT(G12#),0),MOD(SEQUENCE(COUNTA(F14:F37)*COUNT(G12#))-1,COUNT(G12#))+1).

Oh dear. That's a horror. Rather than write 1,000 words trying to explain this, let me detail the concept instead. **SORT(G12#&" - "&F14:F37)** provides every combination of **Month Number** concatenated with a **Football Club**, separated by a " – " delimiter, *e.g.*

1 – Aston Villa, 2 – Aston Villa, …, 10 – Aston Villa, 1 – Birmingham City, 2 – Birmingham City, …

The problem is **SORT(G12#&" - "&F14:F37)** spills this into a 10-column by 24-row array. I want it as a list, so the entire rest of the formula simply forces the array down a column of 240 rows instead. **INDEX** is used to locate the next record in the array, with contrived formulae to determine the row and column numbers of the virtual grid.

SUMIFS is used to create the points total for each row, and to be honest, simpler formulae could have been used elsewhere too. But that's my point. As I have written this article, it's hard not to get carried away with all this and try and do everything in Dynamic Arrays. I have worked for years with Excel and been a keen advocate for keeping everything simple. Dynamic Arrays scare me that we may not help ourselves and write monsters like the formula above.

Maybe Excel's simpler functions and features will live on after all.

Calculation Order Concern

If it feels like you have aged a year since you started reading this, you probably have. There's a lot to get excited about and I have highlighted some of the issues too – many of which I am sure will be ironed out by the time everything becomes Generally Available. However, I am not sure the following concern will be going away any time soon.

When I calculate something in Excel, if I use the same formula, I must get the same answer, right? Well – not necessarily. Consider the following:

1. Calculation 1

Example

Using SEQUENCE and TRANSPOSE

Value for Formula 1: 3
Value for Formula 2: 4

Formula 1: #SPILL! — =SEQUENCE(H12)

Formula 2: 1 2 3 4 — =TRANSPOSE(SEQUENCE(H13))

2. Calculation 2

Example

Using SEQUENCE and TRANSPOSE

Value for Formula 1: 3
Value for Formula 2: 4

Formula 1: 1 — =SEQUENCE(H27)
2
Formula 2: #SPILL! 3 — =TRANSPOSE(SEQUENCE(H28))

In the example above, Calculations 1 and 2 are identical but deliver different results (*i.e.* different *#SPILL!* errors). Why?

- In Calculations 1 and 2, both values for Formula 1 and Formula 2 were originally set to 1. This causes no *#SPILL! errors*
- In Calculation 1, the value for Formula 2 (cell **H13**) was then changed to 4 with no error
- Then, in Calculation 1, the value for Formula 1 (cell **H12**) was changed to 3. This caused the resultant *#SPILL!* error in cell **K16**
- Next, in Calculation 2, the value for Formula 1 (cell **H27**) was changed to 3 with no error
- Then, in Calculation 2, the value for Formula 2 (cell **H28**) was changed to 4. This caused the resultant *#SPILL!* error in cell **I33**.

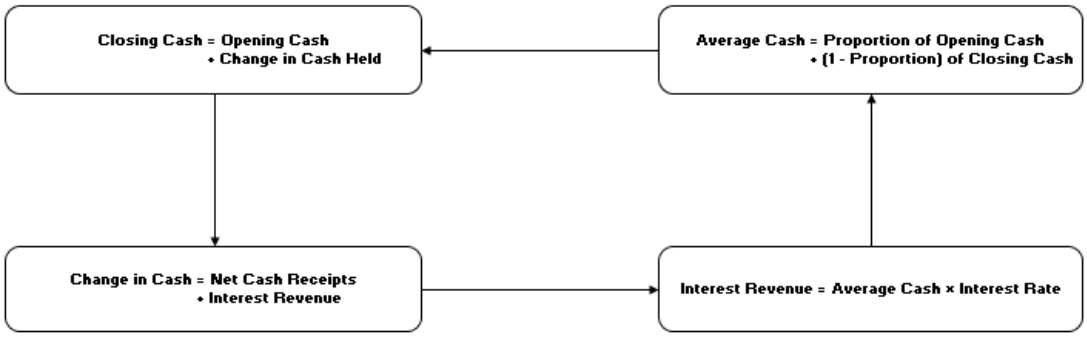
I am not sure what the solution is for this problem. Technically, *#SPILL!* is working correctly, but it doesn't seem right that two results may be generated in this instance depending upon what input I change first. The jury is out on this one.

As at the time of writing, all the features, functions and error messages are rolling out into the wonderful world of Office 365 (recently rebadged as Microsoft 365). Many users should now have them. Start getting excited now and consider taking the leap to Microsoft's subscription model. These are just a taste of some new functions to come to make a truly "dynamic" spreadsheet.

Top 3 Articles: #1 Calculating Interest without Circularity

For our 100th newsletter, we've decided to reproduce the three articles that have produced the most feedback in the past 99 issues. We are counting them down in reverse order. Here is the top dog, often first answer on Google too…

In a financial model, it is commonplace to have to calculate interest. For this illustration, let's assume we are calculating interest received on the business's average cash balance for certain periods of time (it could just as simply be interest paid on a debt balance, etc.). This gives rise to a perceived circular logic:



This problem can be solved algebraically in, er, a relatively straightforward manner without creating circularities – and is therefore our recommended approach.

In a newsletter, we wouldn't normally publish the following, but the derivation of the formula has proved to be one of our most popular web pages (see www.sumproduct.com/thought/interest-received). Therefore, we apologise for the following mathematical assault (for those not interested in the derivation, simply skip to the end) – unfortunately, Excel modelling sometimes boils down to solving simultaneous equations!

Let:

OB = opening cash balance for the period

CB = closing cash balance for the period

M = non-interest cash movement for the period

I = interest cash movement for the period

r = interest rate

t = tax rate (it is assumed this cannot equal 100%)

x = proportion into the period that the non-interest cash movements are assumed to occur, e.g.

- If x = 0%, this means the movement occurred at the start of the period
- If x = 100%, this means that the movement occurred at the end of the period
- If x = 50%, this means that the movement occurred midway through the period

So,

$$\mathbf{CB} = \mathbf{OB} + \mathbf{M}(1-\mathbf{t}) + \mathbf{I}(1-\mathbf{t}) \quad \text{and}$$

$$\mathbf{I}(1-\mathbf{t}) = (\mathbf{x}.\mathbf{OB} + (1-\mathbf{x}).\mathbf{CB}).\mathbf{r}.(1-\mathbf{t}) \quad \text{so (as } \mathbf{t} \neq 100\%)$$

$$\mathbf{I} = (\mathbf{x}.\mathbf{OB} + (1-\mathbf{x}).\mathbf{CB}).\mathbf{r}$$

$$= (\mathbf{x}.\mathbf{OB} + (1-\mathbf{x}).(\mathbf{OB} + \mathbf{M}(1-\mathbf{t}) + \mathbf{I}(1-\mathbf{t}))).\mathbf{r}$$

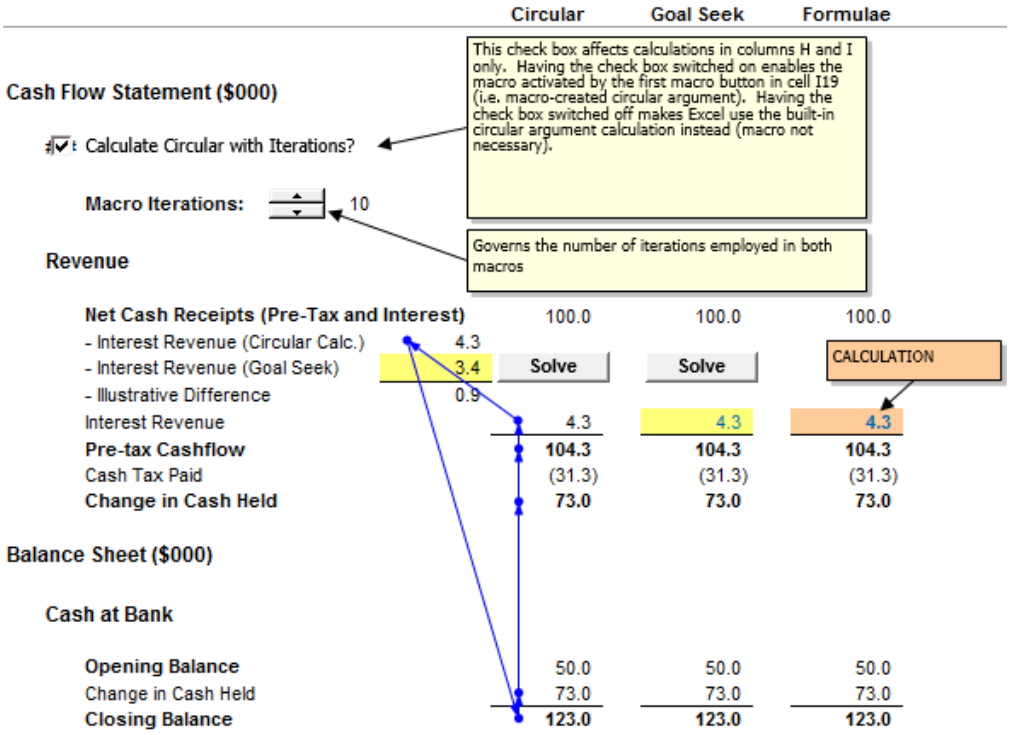
$$= \mathbf{OB}.\mathbf{r} + (1-\mathbf{x}).\mathbf{M}.(1-\mathbf{t}).\mathbf{r} + (1-\mathbf{x}).\mathbf{I}.(1-\mathbf{t}).\mathbf{r}$$

Therefore,

$$\mathbf{I}.(1-(1-\mathbf{x}).(1-\mathbf{t}).\mathbf{r}) = \mathbf{OB}.\mathbf{r} + (1-\mathbf{x}).\mathbf{M}.(1-\mathbf{t}).\mathbf{r}$$

$$\Leftrightarrow \quad \mathbf{I} = \frac{\mathbf{OB}.\mathbf{r} + (1-\mathbf{x}).\mathbf{M}.(1-\mathbf{t}).\mathbf{r}}{(1-(1-\mathbf{x}).(1-\mathbf{t}).\mathbf{r})}$$

Hence, we can calculate interest from this final equation and have no circular references or goal seek. Please see www.sumproduct.com/thought/interest-received for an example Excel file that illustrates this technique:



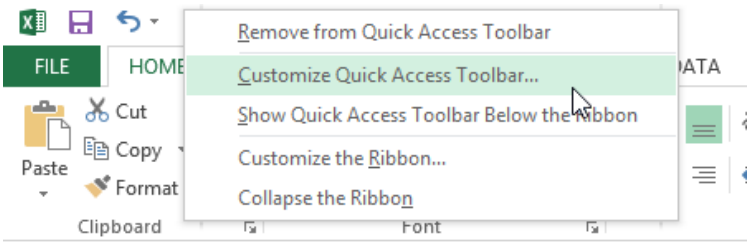
Best Excel Tip Ever – The Top Five

In newsletters gone by, we asked you – our readers – to vote for your favourite Excel tips of all time. Many moons ago we published the Top 5 – which we reproduce here, complete with old school screenshots. You have been warned!!

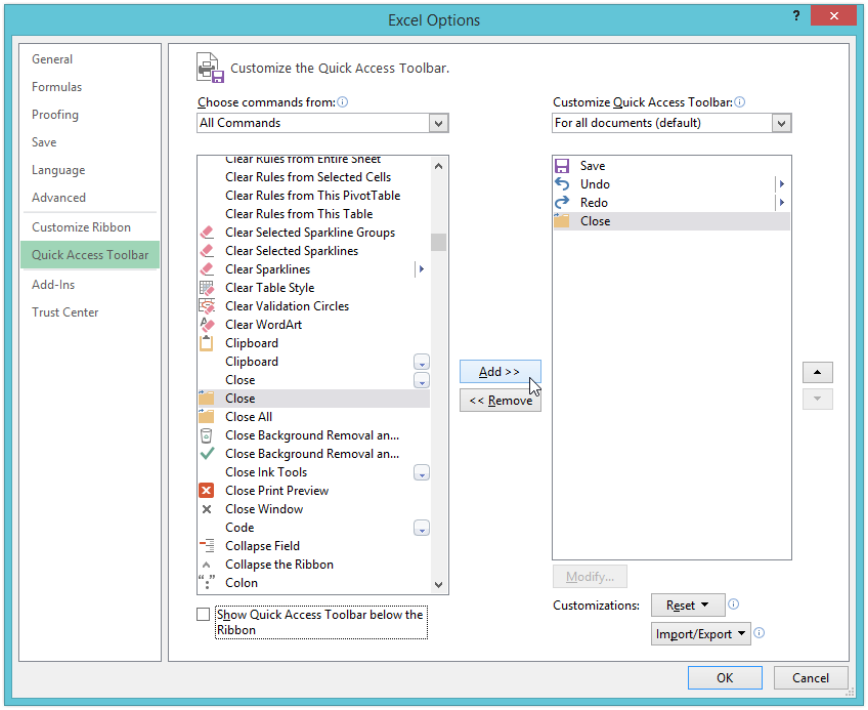
Number 5: Close Files, Not Excel

Ever closed that final file in Excel 2013 or later only for the application to close down as well? This is the Excel way of the world but there is a workaround and we thank fellow MVP **Wyn Hopkins** once more of Access Analytic for bringing this to our attention.

In Excel 2013 and later, simply right-click on the Quick Access Toolbar and select 'Customize Quick Access Toolbar…' *viz*.



In the subsequent dialog box, select 'All Commands' in the 'Choose commands from' drop down box and then select 'Close' (with the folder icon, please see the illustration below). Next, click on the 'Add>>' button to add it to the Quick Access Toolbar and finally click on 'OK' to exit the dialog box.



From now on, simply click on this 'Close' icon in the Quick Access Toolbar and you will never have to say goodbye to Excel again. Breaking up can just be so very hard to do!

Number 4: Finding Inconsistent Formulae Easily

Kim Ho and **Minh Lee** were two that suggested this one. Consider the following block of data:

	A	B	C	D	E	F	G	H	I	J	K	L
1	16	80	79	80	17	12	46	20	63	67	28	
2	64	58	72	39	63	90	73	15	29	36	45	
3	94	79	30	21	1	49	63	66	85	49	97	
4	87	73	36	88	44	27	59	0	1	21	19	
5	88	74	87	16	78	39	56	98	40	84	21	
6	96	98	15	63	59	89	70	36	99	25	50	
7	6	65	10	27	12	4	55	32	71	74	72	
8	64	0	51	1	14	34	18	81	46	62	94	
9	18	67	18	89	23	8	52	51	17	48	0	
10	7	47	57	31	24	38	30	5	90	75	37	
11	60	1	67	62	89	41	10	99	10	36	72	
12	12	53	3	0	53	58	29	95	28	7	65	
13	99	70	28	81	10	25	14	31	47	7	25	
14	42	82	51	46	18	79	33	63	9	53	20	
15	49	63	47	95	95	25	82	49	76	1	58	
16	55	88	77	95	73	60	25	37	20	87	48	
17	59	97	20	68	12	82	3	23	90	69	78	
18												

Let's assume this data is supposed to refer to a similar block of data elsewhere. How can we tell if the formula has been copied across and down correctly? Inspection by eye achieves nothing here.

One option is to use the keyboard shortcut **CTRL +** ` (the character is the key to the left of the 1 on a standard QWERTY keyboard):

	A	B	C	D	E	F	G	H	I	J	K	L
1	=A23	=B23	=C23	=D23	=E23	=F23	=G23	=H23	=I23	=J23	=K23	
2	=A24	=B24	=C24	=D24	=E24	=F24	=G24	=H24	=I24	=J24	=K24	
3	=A25	=B25	=C25	=D25	=E25	=F25	=G25	=H25	=I25	=J25	=K25	
4	=A26	=B26	=C26	=D26	=E26	=F26	=G26	=H26	=I26	=J26	=K26	
5	=A27	=B27	=C27	=D27	=E27	=F27	=G27	=H27	=I27	=J27	=K27	
6	=A28	=B28	=C28	=D28	=E28	=F28	=G28	=H28	=I28	=J28	=K28	
7	=A29	=B29	=C29	=D29	=E29	=F29	=G29	=H29	=I29	=J29	=K29	
8	=A30	=B30	=C30	=D30	=E30	=F30	=G30	=H30	=I30	=J30	=K30	
9	=A31	=B31	=C31	=D31	=E31	=F31	=H42	=I42	=J42	=K42	=L42	
10	=A32	=B32	=C32	=D32	=E32	=F32	=G32	=H32	=I32	=J32	=K32	
11	=A33	=B33	=C33	=D33	=E33	=F33	=F42	=H33	=I33	=J33	=K33	
12	=A34	=B34	=C34	=D34	=E34	=F34	=F43	=H34	=I34	=J34	=K34	
13	=A35	=B35	=C35	=D35	=E35	=F35	=F44	=H35	=I35	=J35	=K35	
14	=A36	=B36	=C36	=D36	=E36	=F36	=F45	=H36	=I36	=J36	=K36	
15	=A37	=B37	=C37	=D37	=E37	=F37	=F46	=H37	=I37	=J37	=K37	
16	=A38	=B38	=C38	=D38	=E38	=F38	=F47	=H38	=I38	=J38	=K38	
17	=A39	=B39	=C39	=D39	=E39	=F39	=F48	=H39	=I39	=J39	=K39	
18												

This shortcut toggles cell values with their content (*i.e.* formulae). This will show formulae which have not been copied across properly, but this is still fraught with user error (can you spot the relevant cells?) and would be cumbersome with vast arrays of data.

Instead, there is a simpler, automatic approach. Select all of the data (click anywhere in the range and press **CTRL + *** – see below for more on this shortcut). Then use the keyboard shortcut **CTRL + ** *viz*.

	A	B	C	D	E	F	G	H	I	J	K	L
1	16	80	79	80	17	12	46	20	63	67	28	
2	64	58	72	39	63	90	73	15	29	36	45	
3	94	79	30	21	1	49	63	66	85	49	97	
4	87	73	36	88	44	27	59	0	1	21	19	
5	88	74	87	16	78	39	56	98	40	84	21	
6	96	98	15	63	59	89	70	36	99	25	50	
7	6	65	10	27	12	4	55	32	71	74	72	
8	64	0	51	1	14	34	18	81	46	62	94	
9	18	67	18	89	23	8	52	51	17	48	0	
10	7	47	57	31	24	38	30	5	90	75	37	
11	60	1	67	62	89	41	10	99	10	36	72	
12	12	53	3	0	53	58	29	95	28	7	65	
13	99	70	28	81	10	25	14	31	47	7	25	
14	42	82	51	46	18	79	33	63	9	53	20	
15	49	63	47	95	95	25	82	49	76	1	58	
16	55	88	77	95	73	60	25	37	20	87	48	
17	59	97	20	68	12	82	3	23	90	69	78	
18												

This automatically selects all of the cells whose contents are different from the comparison cell in each row (for each row, the comparison cell is in the same column as the active cell).

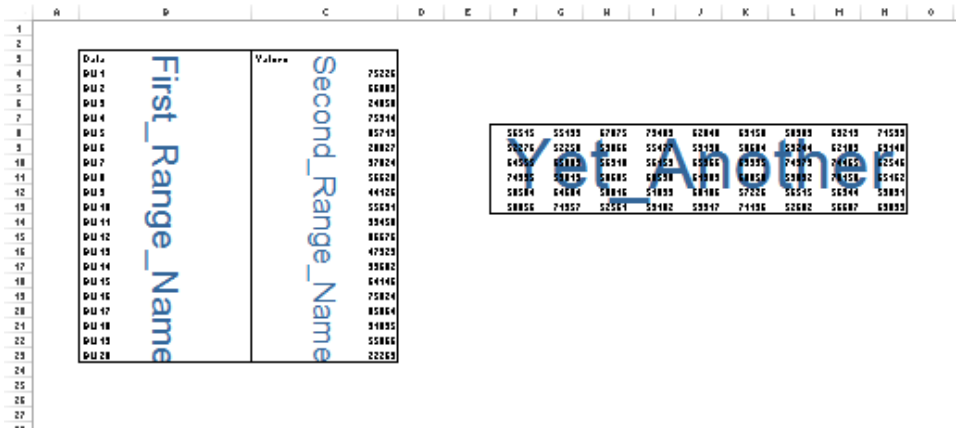
**CTRL + SHIFT + ** selects all cells whose contents are different from the comparison cell in each column (for each column, the comparison cell is in the same row as the active cell). In this example, where a formula is supposed to be copied across and down, there will be no difference.

These cells can now be highlighted and reviewed at leisure.

Number 3: The 39 Steps of Range Names

Excel MVP **Bob Umlas** was a great proponent of the following tip for identifying range names quickly.

An interesting quirk relating to range names is what happens if you actually reduce the scale of Zoom View (ALT + W + Q) to 39% or below:



It can be a simple way of tracking down some of those pesky critters.

Number 2: Selecting an Active Range

Interestingly, this one was most popular with the Excel MVPs, including **Ken Puls** and **Frederic le Guen**. Consider you have been working with an Excel range.

	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36
7						

Clicking anywhere in this range and then pressing **CTRL + *** will then select the whole range,

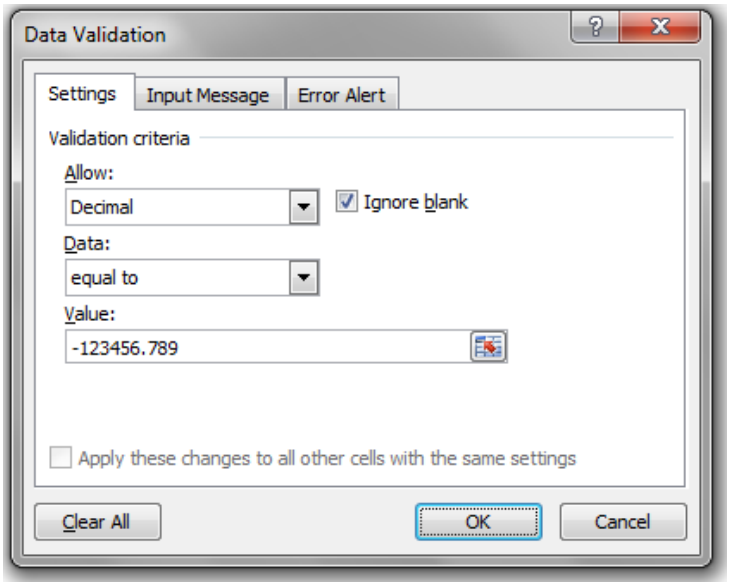
	A	B	C	D	E	F
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30
6	31	32	33	34	35	36
7						

Number 1: Demonic Data Validation

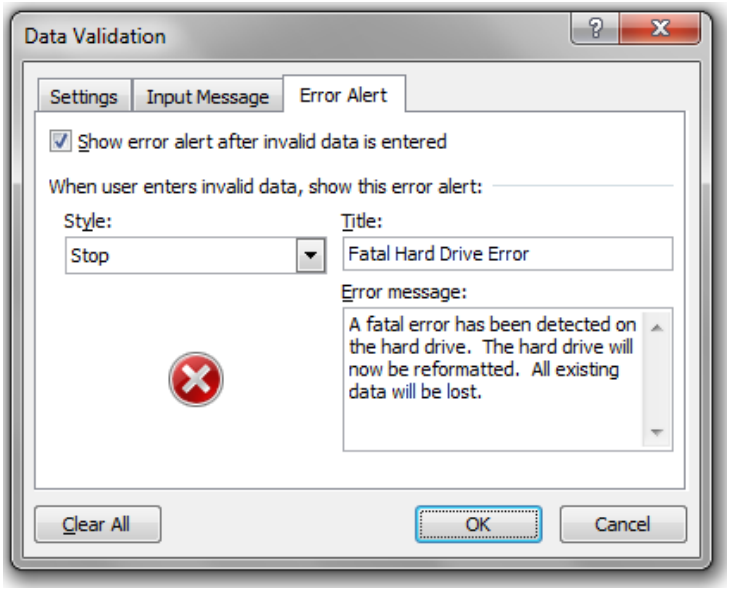
We have some twisted individuals reading these newsletters! By far and away the most popular 'trick' (in all sense of the word!) was this monster first divulged in our very first newsletter – so it seems appropriate to bring it up once more in our 100th! We have elected not to name all the people who suggested this – partly to save printing costs and partly to protect the guilty. You know who you are!!

Data Validation is a useful way to control what end users can type into a worksheet cell (see www.sumproduct.com/thought/data-validation). You can use this functionality to play a trick. Please use this at your own risk: if you get fired, you will get no sympathy here.

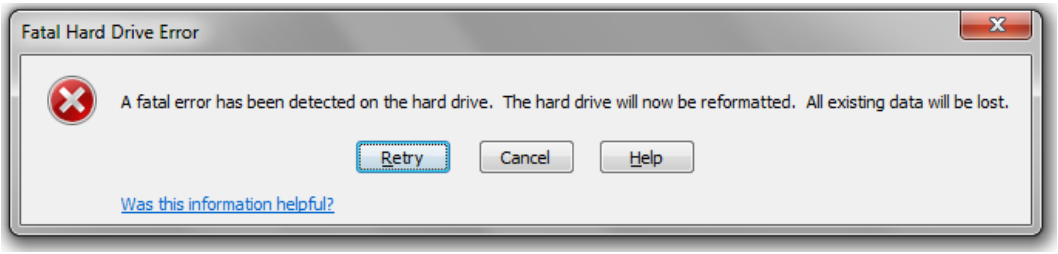
If someone is unfortunate to leave a spreadsheet unprotected, simply highlight the whole worksheet and then activate Data Validation (**ALT + D + L**). In the 'Settings' tab, select settings similar to the following (the aim is to pick a number the user won't use):



Then, select the 'Error Alert' tab:



Now, de-select the range and wait for your victim to use the worksheet. As soon as they type an invalid entry, they will be greeted with the following error alert:



Who says spreadsheets can't be fun..?

Beat the Boredom Challenge

With many of us currently "working from home" / quarantined, there are only so Zoom / Teams calls and virtual parties you can make before you reach your (data) limit. Perhaps they should measure data allowance in blood pressure millimetres of mercury (mmHg). To try and keep our readers engaged, we will continue to reproduce some of our popular ***Final Friday Fix*** *challenges from yesteryear in this and upcoming newsletters. One suggested solution may be found later in this newsletter. Here's this month's…*

This time, we are drawing the problem from our consulting work. The problem here relates to data validation. Normally, you can use data validation to restrict inputs to only values that come from a list. However, what if you want values from your data validation to subsequently populate your list?

The request that we had was innocuous enough. We were required to create a data validation input cell that will take values from a list and populate a dropdown. If a value is entered that does not exist in the list, we want a prompt to check whether this value is correct, or whether it was entered in error. This should work similar to the "Warning" option of data validation which will allow you to enter a value and keep it with a warning message, despite not meeting the data validation criteria.

Here's the tricky thing though. As values are entered, any new values should be included into the data validation list for future reference.

Unit	Name
Accounting	Amy
Accounting	Billy
Accounting	Faye
Finance	
Finance	
Finance	

List of items
Amy
Billy
Charlie
Debbie
Eric

This leaves us in a sticky situation, because a formula that adds a new value to the data validation list will no longer trigger the warning criteria in the data validation. That is, whilst the value might not exist in the data validation list when you are typing it in, a formula-driven list will pick up the value before the data validation check is applied, thus ignoring the data validation effectively and allowing the new value to be entered in without warning.

So, this is the challenge this month: can you find a solution that will allow you to enter a value using a drop down list, check if manually entered items are intended to be added to the list, and allow users to cancel their actions if entered in error?

Sound easy? Try it. One solution just might be found later in this newsletter – but no reading ahead!

Visual Basics

We thought we'd run an elementary series going through the rudiments of Visual Basic for Applications (VBA) as a springboard for newer users. This month, we thought we would look at "do-ing" some events...

There are a number of different ways that we can refer to Excel cells in a worksheet. In this series, we've been using a fairly consistent approach to referencing cell ranges – that is, using the **Range** object available in VBA.

However, there are a few other options that are available at your disposal that you may run across. It's important to understand how these different codes work, so we'd like to list out a few for you:

VBA code	
Range("C4")	Refers to the cell "C4"
Range("C" & x)	Refers to the cell in column C and row number 'x' (this can be a variable used in your code for looping)
Range("RangeName")	Refers to the named range "RangeName"
Cells(4,3)	Refers to the 4th row and the 3rd column (i.e. C4)
Cells	Refers to all cells in a worksheet
[C4]	Refers to the cell "C4"
[RangeName]	Refers to the named range "RangeName"
Rows(x)	Refers to the entirety of row 'x'
Columns(x)	Refers to the entirety of the x'th column
Columns("C")	Refers to the entirety of column "C"
Application.Goto Reference:="RangeName"	Selects the named range "RangeName"

So you can use any of these different approaches to refer to different cells, named ranges, rows and columns. For example, you could use:

- Range("C4").Value = 10
- Cells.ClearContents
- [MyTargetCell].PasteSpecial xlPasteAll
- Columns("C").Delete

Hopefully this will help to make sense of the macros that you see in your day-to-day environment!

More next time.

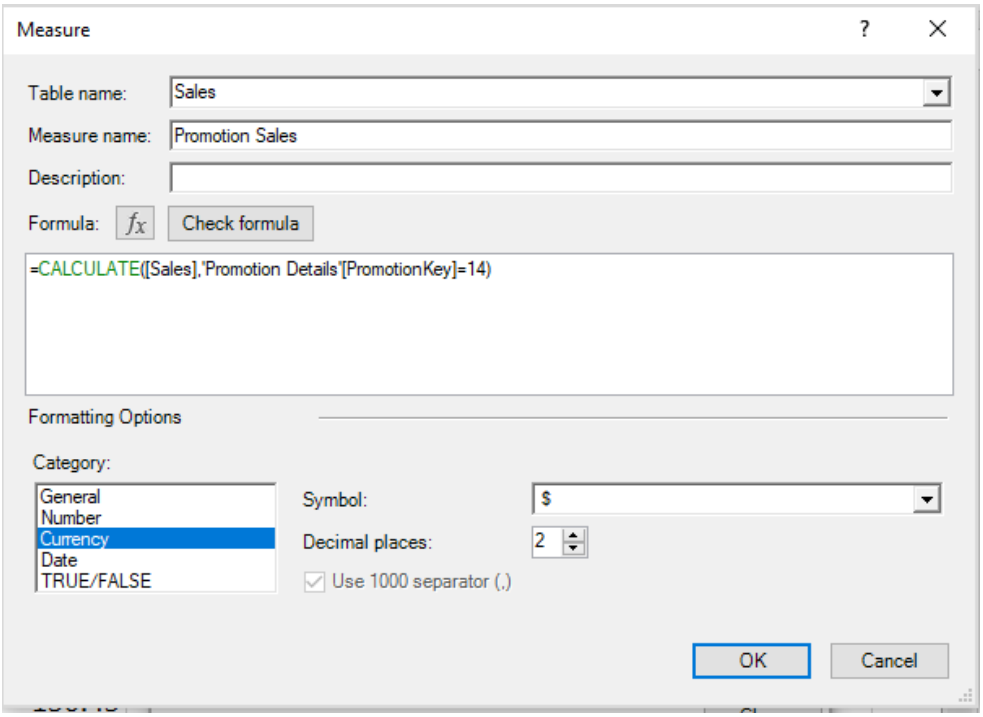
Power Pivot Principles

We continue our series on the Excel COM add-in, Power Pivot. This month, we consider hoe CALCULATE can benefit from connected tables.

In last month's article, we created relationships between all of the imported tables in Power Pivot; with these links we can now create measures that utilise these links.

Let's create the measure 'Promotion Sales' which we envisage to calculate the total amount of sales that were made under the promotion key of '14':

=CALCULATE([Sales],'Promotion Details'[PromotionKey]=14)



The **CALCULATE** function is not limited to using filters from the same table as it's expression, it can accept filters from other tables as well. To show this, let's insert our new measure into our PivotTable:

Year	2016	
Year	**Sales**	**Promotion Sales**
2016		
Jan	$6,779.00	
Feb	$7,749.55	
Mar	$7,925.10	
Apr	$9,260.70	
May	$10,556.85	
Jun	$10,230.45	
Jul	$42,138.05	$145.00
Aug	$115,169.70	$73.95
Sep	$117,152.60	$130.45
Oct	$124,468.30	
Nov	$125,286.20	
Dec	$157,338.05	
Grand Total	**$734,054.55**	**$349.40**

Thanks to linked tables we can now see how much sales were made with promotions throughout the year. Aren't connected tables great?

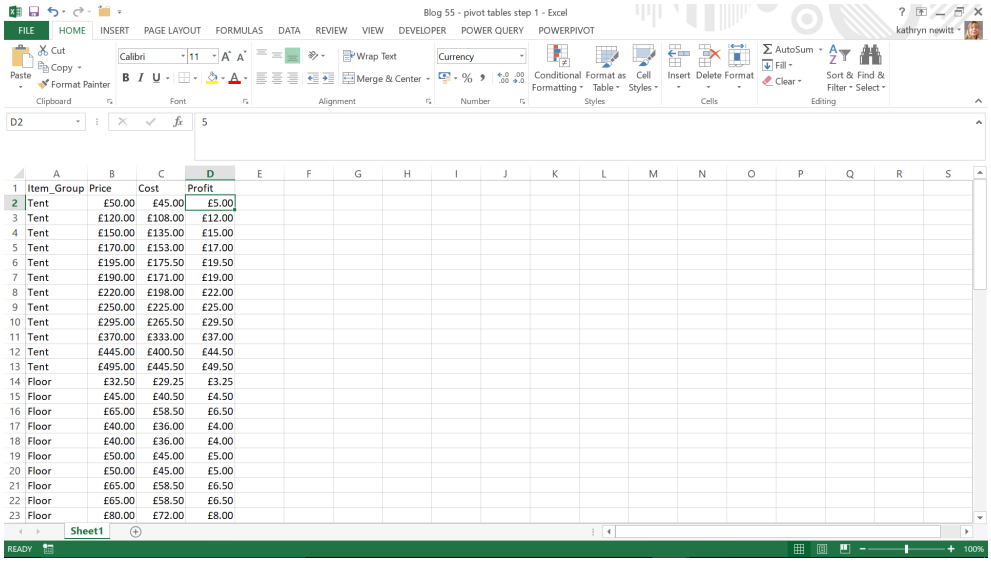
More *Power Pivot Principles* next month.

Power Query Pointers

Each month we'll reproduce one of our articles on Power Query (Excel 2010 and 2013) / Get & Transform (Office 365, Excel 2016 and 2019) from www.sumproduct.com/blog. If you wish to read more in the meantime, simply check out our Blog section each Wednesday. This month, we look at how Power Query may reduce the size of a PivotTable workbook.

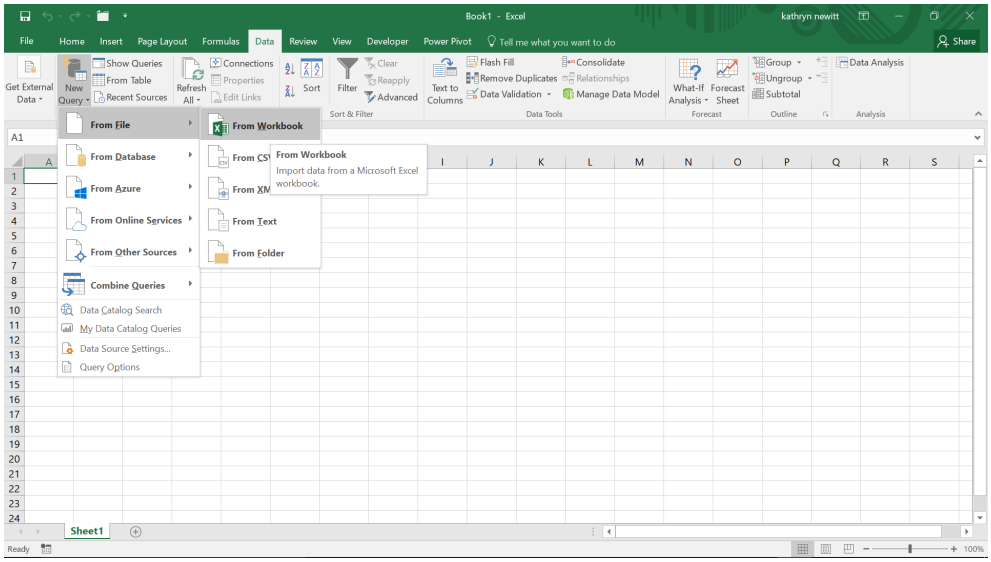
In other articles, we have focused on how well Power Query and Power Pivot work together. Power Query is not exclusive to Power Pivot though, as it may be used with ordinary PivotTables. There is a good reason for doing this too, which we'll reveal at the end. Talk about cliff-hangers, eh?

I start with data that we have copied (not uploaded using Power Query) from an **Items** table, just to provide some typical data that we might like to put in a PivotTable.

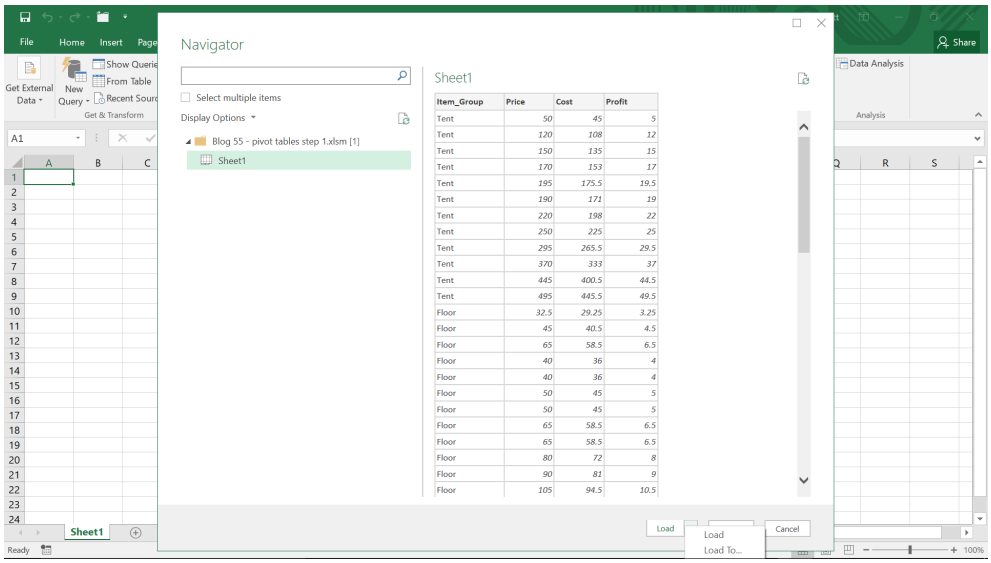


	Item_Group	Price	Cost	Profit
2	Tent	£50.00	£45.00	£5.00
3	Tent	£120.00	£108.00	£12.00
4	Tent	£150.00	£135.00	£15.00
5	Tent	£170.00	£153.00	£17.00
6	Tent	£195.00	£175.50	£19.50
7	Tent	£190.00	£171.00	£19.00
8	Tent	£220.00	£198.00	£22.00
9	Tent	£250.00	£225.00	£25.00
10	Tent	£295.00	£265.50	£29.50
11	Tent	£370.00	£333.00	£37.00
12	Tent	£445.00	£400.50	£44.50
13	Tent	£495.00	£445.50	£49.50
14	Floor	£32.50	£29.25	£3.25
15	Floor	£45.00	£40.50	£4.50
16	Floor	£65.00	£58.50	£6.50
17	Floor	£40.00	£36.00	£4.00
18	Floor	£40.00	£36.00	£4.00
19	Floor	£50.00	£45.00	£5.00
20	Floor	£50.00	£45.00	£5.00
21	Floor	£65.00	£58.50	£6.50
22	Floor	£65.00	£58.50	£6.50
23	Floor	£80.00	£72.00	£8.00

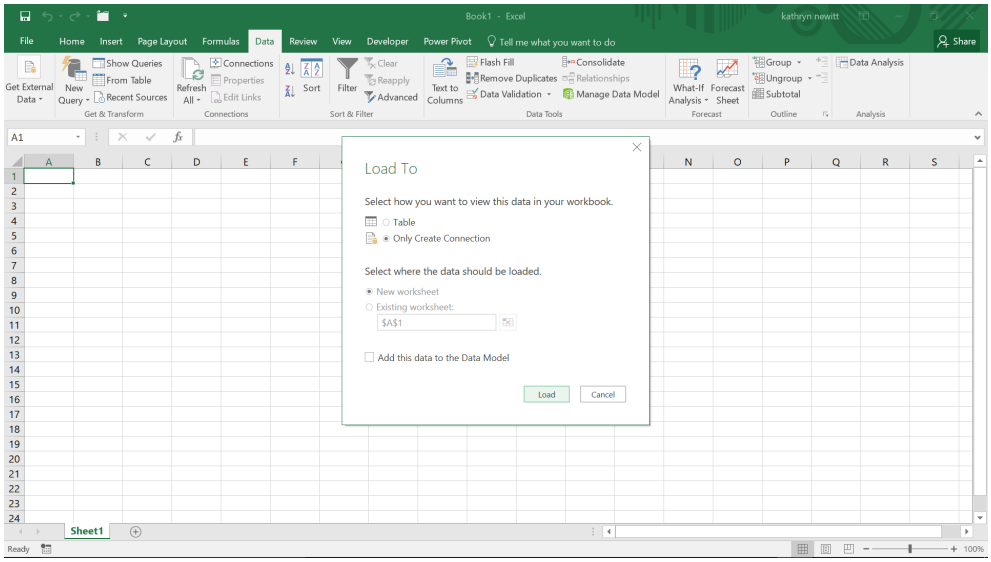
All the data has been copied and pasted so that there are no formulae. From another blank workbook, we use Power Query to connect to this workbook. In this new workbook, on the Data tab, choose 'New Query' and select 'From File' and then 'From Workbook' on the dropdown.



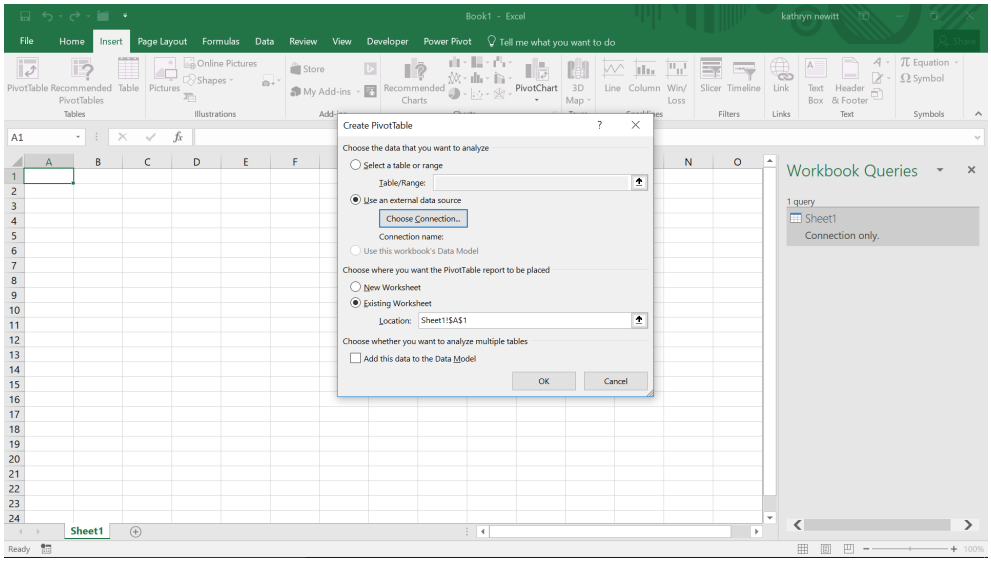
We may select the original workbook.



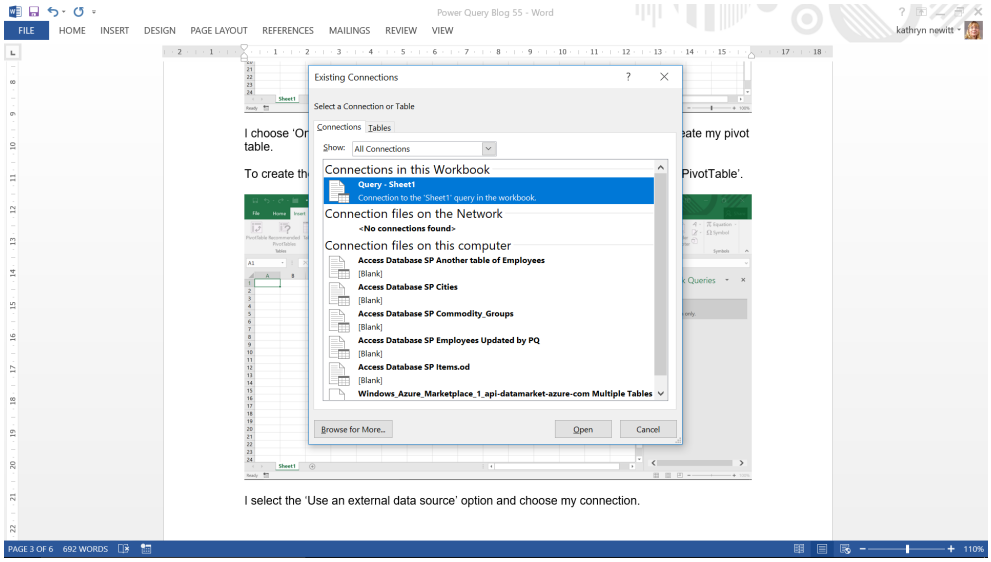
We don't want to load the data, we just want to make a connection to it, so we'll choose the 'Load To…' option.



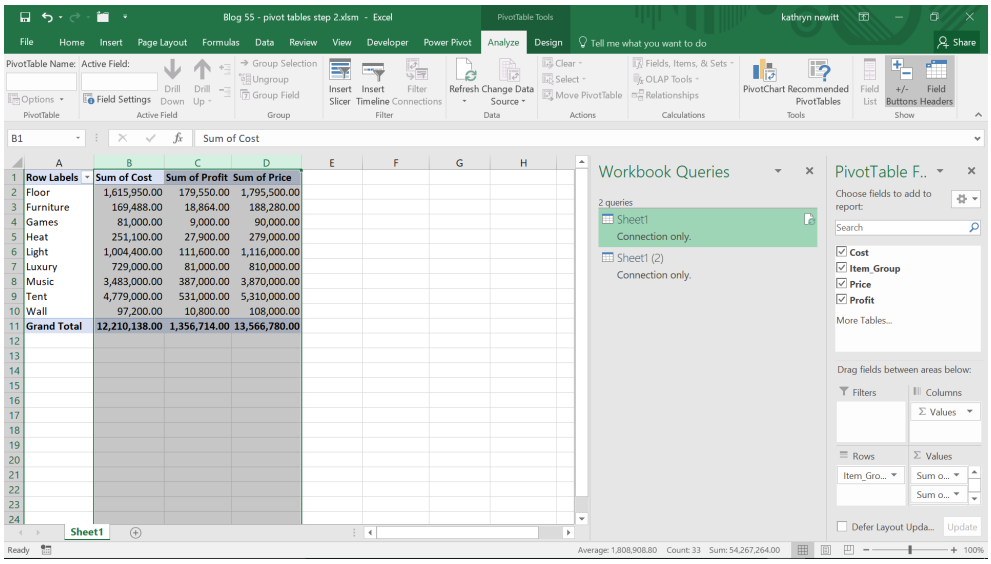
Choose 'Only Create Connection' – we want the connection to be available when we create a PivotTable, but we don't need to load the data to the workbook. To create the PivotTable, go to the Insert tab and in the Tables section choose PivotTable:



Select the 'Use an external data source' option and choose the connection.



Our connection is at the top, 'Query – Sheet1', so we'll open this and create a simple PivotTable as shown below.



Row Labels	Sum of Cost	Sum of Profit	Sum of Price
Floor	1,615,950.00	179,550.00	1,795,500.00
Furniture	169,488.00	18,864.00	188,280.00
Games	81,000.00	9,000.00	90,000.00
Heat	251,100.00	27,900.00	279,000.00
Light	1,004,400.00	111,600.00	1,116,000.00
Luxury	729,000.00	81,000.00	810,000.00
Music	3,483,000.00	387,000.00	3,870,000.00
Tent	4,779,000.00	531,000.00	5,310,000.00
Wall	97,200.00	10,800.00	108,000.00
Grand Total	12,210,138.00	1,356,714.00	13,566,780.00

So, what is the advantage of using Power Query? Well, for today's example, our data file was huge – over a million rows. This is how each step has been stored:

- step 1 is the data in the workbook
- step 2 is my connection only query and the data shown as a PivotTable.

Name	Date modified	Type	Size
Blog 55 - pivot tables step 2	27/11/2017 13:50	Microsoft Excel Macr...	126 KB
Blog 55 - pivot tables step 1	27/11/2017 13:42	Microsoft Excel Macr...	2,734 KB

Therefore, using Power Query is one way to reduce the size of the PivotTable workbook. There are other ways, but Power Query will also allow us to clean and transform the data prior to creating a PivotTable.

More next month!

Power BI Updates

Unfortunately, the updates have come out after the printing deadline for this newsletter, but don't worry, we'll report them all in next month's newsletter.

More next month we're sure!

The A to Z of Excel Functions: FIXED

Given we have such an action-packed 100th issue this month, there is only time for one (sorry!).

"Can we fix it?"

This function rounds a number to the specified number of decimals, formats the number in decimal format using a period (full stop, ".") and commas, and returns the result as text.

The **FIXED** function employs the following syntax to operate:

FIXED(number, [decimals], [no_commas]).

The **FIXED** function has the following arguments:

- **number:** this is required and represents the **number** you wish to round and convert to text
- **decimals:** this is optional and represents the number of digits to the right of the decimal point
- **no_commas:** this is also optional. This is a logical value that, if TRUE, prevents **FIXED** from including commas in the returned text.

It should be noted that:

- numbers in Microsoft Excel can never have more than 15 significant digits, but **decimals** may be as large as 127
- if **decimals** is negative, **number** is rounded to the left of the decimal point (*e.g.* 10's, 100's, …)
- if you omit **decimals**, it is assumed to be 2 (not zero)
- if **no_commas** is FALSE or omitted, then the returned text includes commas as usual
- the major difference between formatting a cell containing a number by using a command (*e.g.* on the 'Home' tab, in the 'Number' group, click the arrow next to 'Number', and then click 'Number') and formatting a number directly with the **FIXED** function is that **FIXED** converts its result to text. A number formatted with the 'Cells' command is still a number.

Please see our example below:

	A	B	C
1	Data		
2	1234.567		
3	-1234.567		
4	44.332		
5			
6			
7	Formula	Description	Result
8	=FIXED(A2,1)	Rounds the number in A2 one digit to the right of the decimal point.	1,234.6
9	=FIXED(A2,-1)	Rounds the number in A2 one digit to the left of the decimal point.	1,230
10	=FIXED(A3,-1,TRUE)	Rounds the number in A3 one digit to the left of the decimal point, without commas (the TRUE argument).	-1230
11	=FIXED(A4)	Rounds the number in A4 two digits to the left of the decimal point, even though not specified.	44.33
12	=FIXED(A2)+FIXED(A3)	Even though these are two text values, the "+" still adds them and produces a text result (0.00). The three decimal places is presumably attributable to the number of decimal places prevalent in the two values.	0.000
13			

More Excel Functions next month (don't worry, there will be more than one!)…

Beat the Boredom Suggested Solution

Earlier in this month's newsletter, we posed an interesting problem. Data validation warnings can help you check a value you enter into a cell, ensuring that it either comes from a pre-defined list or to confirm that you want to add your value to the cell. However, the problem arises in this specific scenario, where a new value will add to the list in a dynamic way, because a formula-driven list will pick up the value before the data validation check is applied. Because of this, when the data validation is checked, it sees the value in the list, and proceeds to treat it as a valid value.

So, how do we get around this?

First of all, we need to create a dynamic list using range names. If you don't know what Dynamic Range Names are, please check out the following link. In particular, we're going to use the **OFFSET** approach.

To begin, let's create a counter to keep track of which names already exist in the list. This will form the basis of the named range. We can use the formula

=IF(COUNTIF($C$2:C3,[@Name])=1,MAX($D$2:D2)+1,0)

The idea is that we're going to look through the **Name** column and check to see if this is the first instance of the name appearing. If so, we're going to increment a counter. Thus, if this is a value contains the fifth unique name in the list, it will have a counter result of '5'.

Elsewhere in the spreadsheet, we may create an index that looks at the numbers we have assigned, and form them into a list. This is simply an **INDEX(MATCH)** function to pull things into line, such as

=IFERROR(INDEX(Table2[Name],MATCH(G3,Table2[Counter],0)),"")

The **MATCH** function finds the first instance of the name, and the **INDEX** function pulls it into a table so that we can put it into our named range accordingly.

Unit	Name	Counte
All Teams	Amy	1
All Teams	Billy	2
All Teams	Charlie	3
All Teams	Debbie	4
All Teams	Eric	5
Accountin	Amy	0
Accountin	Billy	0
Accountin	Faye	6
Finance		0
Finance		0
Finance		0

Index	Names
1	Amy
2	Billy
3	Charlie
4	Debbie
5	Eric
6	Faye
7	
8	
9	
10	
11	
12	

So far, so good. Now, here's where it gets tricky. Adding a new name updates the list of names faster than the data validation can check for, so it won't warn us when we're putting a new name in. To get around this and provide the warning that we're looking for, we're going to need a macro to help us out.

The macro is going to run every time we make a change to the worksheet, to test if we actually want to keep or reject the value that we've just entered. We'll check whether the change exists in the area that we're targeting, and if not, then we'll ignore the rest of the macro. We can do this with the following code (assuming that your data validated cells are in the range called '**List_Names**':

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim AppliedRange As Range

Application.EnableEvents = False

Set AppliedRange = Application.Intersect(Target, Range("List_Names"))

If AppliedRange Is Nothing Then

    Exit Sub

Else
```

Once we have identified that we're working in the data validated cells, we need to check off a few things that will invalidate our results. If we're selecting multiple cells, that will cause us issues, so we need to set up an error trap if that's the case. Also, if we're deleting a value, rather than adding a new name in place, we don't want to run the code either – no need to data validate our deletion. Therefore, we need the following items as well:

```
Dim TargetValue As String
On Error GoTo ExitSub 'Does not work with multiple rows selected
TargetValue = Target.Value
On Error GoTo 0

If TargetValue = "" Then GoTo ExitSub 'Skip if clearing contents from cell
```

Now we are at the stage where we need to test if our names have appeared before. We can call a **MATCH** function to check where the name appears in our list, and an **INDEX(MATCH)** to determine how many times it's appeared so far.

```
Dim InList As Integer
Dim NumberInlist As Integer
InList = 0
NumberInlist = 0
On Error Resume Next
InList = Application.WorksheetFunction.IfError(Application.WorksheetFunction.Match(TargetValue, Range("List_From"), 0), 0)
NumberInlist = Application.WorksheetFunction.IfError(Application.WorksheetFunction.Index(Range("List_From").Offset(0, 1), _
    Application.WorksheetFunction.Match(TargetValue, Range("List_From"), 0)), 0)
On Error GoTo 0
```

Here's the sneaky bit now. If the name we've just entered appears exactly once, then we know that it didn't exist previously in the list. Therefore, we can test to see if the number of times it's appears is greater than one [1]. If so, then we don't need to do run any warning, because the data validation is working the way we want it to.

However, if there is exactly one item in the list, then we want to pop up a message box and check to see if we really do want to enter it and add it to the list. If not, we should delete the value that was just entered. That's what this next block of code does:

```
If InList > 0 And NumberInlist > 1 Then
    'Do nothing - already exists in client group
Else

    Dim MsgBoxResult As Integer

    MsgBoxResult = MsgBox("This will create a new client group.  Do you wish to continue?", _
        vbYesNo, "New client group")

    If MsgBoxResult = 6 Then
        'Yes result - do nothing
    Else
        Target.ClearContents
    End If

End If
```

A **MsgBox** function will return a value based on what button is clicked. In our case, clicking the 'Yes' button when it asks you if you want to continue gives us a value of '6', which we can check for. If the value returned doesn't equal six (*e.g.* if the user clicks on 'No', 'Cancel' or anything else), then the cell we're looking at will have the contents removed, and it will effectively undo the act of typing a new name in place.

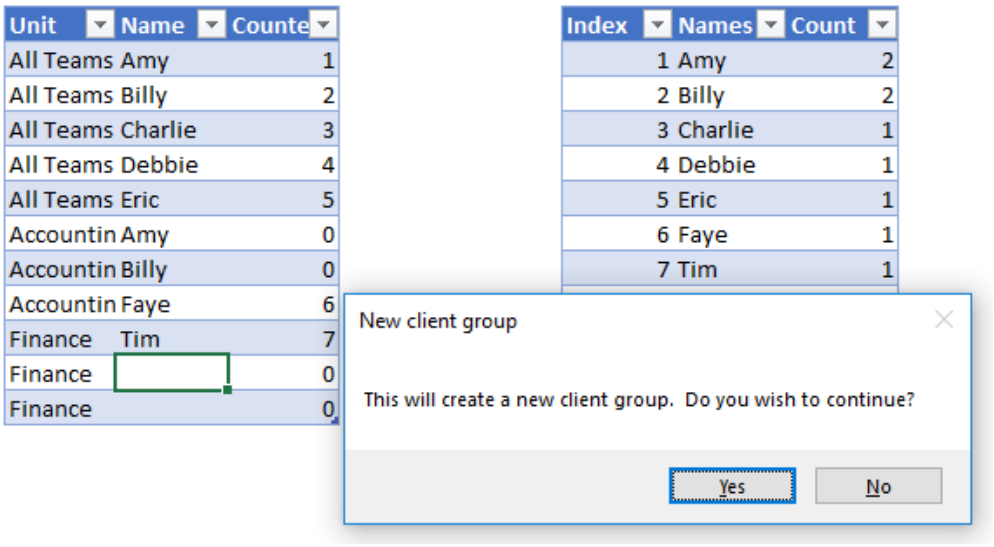
```
End If

ExitSub:

Application.EnableEvents = True

End Sub
```

Finally, we just need to clean up after ourselves to re-enable events (we disabled it initially because deleting the value would also trigger this check) and to provide a break place (here called **ExitSub**) to allow errors to skip through the main code content.



How did you go? Did you find a formula-based solution that didn't require VBA? Let us know, we'd be keen to hear if you think you have a better way to do this!

Until next time.

Upcoming SumProduct Training Courses - COVID-19 update

Due to the COVID-19 pandemic that is currently spreading around the globe, we are suspending our in-person courses until further notice. However, to accommodate the new working-from-home dynamic, we are switching our public and in-house courses to an online delivery stream, presented via Microsoft Teams, with a live presenter running through the same course material, downloadable workbooks to complete the hands-on exercises during the training session, and a recording of the sessions for your use within 1 month for you to refer back to in the event of technical difficulties. To assist with the pacing and flow of the course, we will also have a moderator who will help answer questions during the course.

If you're still not sure how this will work, please contact us at training@sumproduct.com and we'll be happy to walk you through the process.

Location	Course	Date	Date	Duration	Duration
Online (Australia)	Power Pivot, Power Query and Power BI	7 - 9 Apr 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	14 Apr 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	15 - 16 Apr 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	10 - 12 May 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	17 May 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	18 - 19 May 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	15 - 17 Jun 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	22 Jun 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	23 - 24 Jun 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	19 - 21 Jul 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	26 Jul 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	27 - 28 Jul 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	23 - 25 Aug 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	30 Aug 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day

Location	Course	Date	Date	Duration	Duration
Online (Australia)	Financial Modelling	31 Aug - 1 Sep 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	29 Sep - 1 Oct 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	6 Oct 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	7 - 8 Oct 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	3 - 5 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	10 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	11 - 12 Nov 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days
Online (Australia)	Power Pivot, Power Query and Power BI	8 - 10 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	3 Days
Online (Australia)	Excel Tips and Tricks	15 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	1 Day
Online (Australia)	Financial Modelling	16 - 17 Dec 2021	09:00-17:00 AEDT	(-1 day) 22:00-07:00 GMT	2 Days

Key Strokes

Each newsletter, we'd like to introduce you to useful keystrokes you may or may not be aware of. This year, we thought we'd revisit each function key in depth (there are 12 – one for each month of the year!). Given it's March, let's look at the **F3** tips:

Keystroke	What it does
F3	Paste names
CTRL + F3	Open Name Manager
SHIFT + F3	Function wizard
CTRL + ALT + F3	New name
CTRL + SHIFT + F3	Create names

There are *c.*550 keyboard shortcuts in Excel. For a comprehensive list, please download our Excel file at www.sumproduct.com/thought/keyboard-shortcuts. Also, check out our new daily **Excel Tip of the Day** feature on the www.sumproduct.com homepage.

Our Services

We have undertaken a vast array of assignments over the years, including:

- **Business planning**
- **Building three-way integrated financial statement projections**
- **Independent expert reviews**
- **Key driver analysis**
- **Model reviews / audits for internal and external purposes**
- **M&A work**
- **Model scoping**
- **Power BI, Power Query & Power Pivot**
- **Project finance**
- **Real options analysis**
- **Refinancing / restructuring**
- **Strategic modelling**
- **Valuations**
- **Working capital management**

If you require modelling assistance of any kind, please do not hesitate to contact us at contact@sumproduct.com.

Link to Others

These newsletters are not intended to be closely guarded secrets. Please feel free to forward this newsletter to anyone you think might be interested in converting to "the SumProduct way".

If you have received a forwarded newsletter and would like to receive future editions automatically, please subscribe by completing our newsletter registration process found at the foot of any www.sumproduct.com web page.

Any Questions?

If you have any tips, comments or queries for future newsletters, we'd be delighted to hear from you. Please drop us a line at newsletter@sumproduct.com.

Training

SumProduct offers a wide range of training courses, aimed at finance professionals and budding Excel experts. Courses include Excel Tricks & Tips, Financial Modelling 101, Introduction to Forecasting and M&A Modelling.

Check out our more popular courses in our training brochure:

Drop us a line at training@sumproduct.com for a copy of the brochure or download it directly from www.sumproduct.com/training.

Sydney Address: SumProduct Pty Ltd, Suite 803, Level 8, 276 Pitt Street, Sydney NSW 2000
New York Address: SumProduct Pty Ltd, 48 Wall Street, New York, NY, USA 10005
London Address: SumProduct Pty Ltd, Office 7, 3537 Ludgate Hill, London, EC4M 7JN, UK
Melbourne Address: SumProduct Pty Ltd, Ground Floor, 470 St Kilda Road, Melbourne, VIC 3004
Registered Address: SumProduct Pty Ltd, Level 14, 440 Collins Street, Melbourne, VIC 3000

contact@sumproduct.com
www.sumproduct.com
+61 3 9020 2071