EXCEL DYNAMIC ARRAYS

STRAIGHT TO THE point

BILL JELEN

Straight to the Point

The Straight to the Point e-books are designed to thoroughly cover one targeted aspect of Excel.

Version 1: September 25, 2018

Version 2: September 27, 2018. New in this version:

- General copyediting and typo corrections
- "Using the New Array Reference Notation: E3#" on page 8
- "Using FILTER with Multiple Conditions" on page 18
- "Understanding Unique Versus Distinct" on page 19
- "Creating a Crosstab Report with Three Formulas" on page 42
- "Using an Array Reference as Part of a Reference" on page 48
- "Using Dynamic Arrays for Dependent Validation" on page 58
- "What Is the VBA Story?" on page 58

Version 2a: September 28, 2018. Typo corrections in Index & Table of Contents

Version 2b: September 29, 2018. More copyediting corrections.

EXCEL DYNAMIC ARRAYS
Straight to the Point

Bill Jelen

Holy Macro! Books
PO Box 541731, Merritt Island FL 32953

Excel Dynamic Arrays Straight to the Point

First Published: September 25, 2018
Version 2b: September 29, 2018

Author: Bill Jelen

Copyeditor: Kitty Wilson

Cover Design & Compositor: Suat M. Ozgur

Cover Illustration: Shannon Mattiza, 6'4 Productions

Indexer: Nellie Jay

Published by: Holy Macro! Books, PO Box 541731, Merritt Island, FL 32953

Distributed by Independent Publishers Group, Chicago, IL

ISBN 978-1-61547-265-9 PDF, 978-1-61547-377-9 ePub, 978-1-61547-165-2 Mobi

Table of Contents

About the Author

Bill Jelen is the host of MrExcel.com and the author of 54 books about Microsoft Excel, including *Excel Gurus Gone Wild*, *Pivot Table Data Crunching*, and *Excel 2019 In Depth*. He made more than 80 guest appearances on TV's *Call for Help* with Leo Laporte and was voted guest of the year on the *Computer America* radio show.

Jelen writes the monthly Excel column for *Strategic Finance* magazine. He has produced more than 2100 episodes of his daily video podcast *Learn Excel from MrExcel*. Before founding MrExcel.com in 1998, Jelen spent 12 years "in the trenches" as a financial analyst for the accounting, finance, marketing, and operations departments of a publicly held company. Today his company automates Excel reports for hundreds of clients around the world. The website answers more than 30,000 questions a year – for free – for readers all over the world. Jelen hails from Merritt Island, Florida.

Acknowledgments

Thanks to Joe McDaid and the entire Calc team for undertaking a major change to the calculation engine in Excel.

Copyeditor Kitty Wilson and Suat Ozgur were invaluable during the creation of this book. As I was creating screenshots and content, Suat was working almost in real-time, doing layout of the book.

When I released the book on Tuesday September 25, 2018, several of my MVP friends spotted typos or factual errors. Thank you to Roger Govier, Charles Williams, Ingeborg Hawighorst, Liam Bastick, and Jon Peltier for spotting these issues.

INTRODUCTION

What Will the Headlines Say About Dynamic Array Formulas?

The Ignite 2018 conference is likely to spark headlines like these:

- 7 New Functions Introduced at Ignite!
- Sort, Filter, or Remove Duplicates with New Easy Array Formulas
- One Excel Formula Can Spill to Many Cells

But I don't think any of these headlines really capture what was released to the Office Insiders channel on September 24, 2018.

The new Dynamic Array functions are just one side effect of an effort to completely rewrite the calculation engine in Excel. Joe McDaid and the rest of the Calc team have laid the groundwork for all future functions in Excel. Yes, the first crop—**SORT**, **SORTBY**, **FILTER**, **UNIQUE**, **SEQUENCE**, and **RANDARRAY**—are awesome and powerful, but they are just the first of many new functions that will come to Office 365 over the coming years.

Figure 1 *One of many tweets about Dynamic Arrays.*

In the past, many people tried to learn Array Formulas and failed. My friend Mike "ExcelIsFun" Girvin wrote a 350-page book to explain how to do everything with array formulas. Entire chapters of that book will be reduced to single sentences, thanks to **UNIQUE** and other new functions.

You will notice that I did not include **SINGLE** in the list of functions above. Yes, this is a new function. But it won't be a function for the masses. It will be a function for 1 out of every 10,000 Excellers. Very few people ever intentionally used Implicit Intersection in Excel. As you will learn if you brave the complexities of Chapter 9, it was Implicit Intersection that made array formulas so complex in legacy Excel.

This Book Is Not the *Comprehensive* Guide to Dynamic Arrays

In this book I will show you a lot of ways that you can use the new Dynamic Arrays. But the real story will develop as 100 million people start putting these new features to work. That is when people will invent and discover new ways to do awesome things with these new features. I will show you some ways that I can imagine in Chapter 10. But the people using Excel will come up with many more ways to use them.

Dynamic Array Formulas and Their Offspring Are Office 365 Exclusive

Dynamic Arrays came to public preview on September 24, 2018. But they are only in preview, and Office 2019, also released in September does not include Dynamic Array formulas. For those of you still stuck in the paradigm of buying a perpetual license to Office every three or six years, you will have to wait until Office 2022 or Office 2025 to get all of the awesomeness of these new features.

If you haven't realized it yet, subscribing to Office 365 is the only way to get cool new Excel features. The perpetual licenses (for example, Excel 2016, Excel 2019) are being released for the last holdouts, the people who are at the South Pole or on Mars or top-secret government employees who are not allowed to download their bits from the Internet.

How This Book Is Organized

In Chapter 1, you will learn about the concept of a formula spilling to adjacent cells. You will see how a non-empty cell can block the spill and how to correct that. You will hear the first mention of Implicit Intersection and how the **SINGLE** function can solve that.

Chapters 2 through 7 cover each of the six new functions: **SORT**, **SORTBY**, **FILTER**, **UNIQUE**, **SEQUENCE**, and **RANDARRAY**.

Chapter 8 covers combining the new arrays.

Chapter 9 is based on an excellent Joe McDaid video. You will learn the vocabulary the Excel Calc team uses to refer to arrays in legacy Excel. You will see the concepts of Lifting, Broadcasting, Pairwise Lifting, Array Truncation, and Implicit Intersection. These five terms are most likely completely new to you. For this reason, I capitalize them and set them in blue throughout this book. If you are unsure about anything that appears in blue type, flip to Chapter 9 for more information on the concept. To watch Joe's video, start at the 24 minute mark of https://www.microsoft.com/en-us/businessapplicationssummit/video/BAS2018-2140

Chapter 10 provides 24 examples of likely hundreds of new ways to combine Dynamic Arrays with practically every function in Excel. September 24, 2018, will not go down as the day that Joe McDaid gave us 7 new functions. It will be the day that Joe gave us the ability to do hundreds or thousands of new, powerful things.

1 - FORMULAS CAN NOW SPILL

This chapter covers the new **=A2:A20** formula, the **#SPILL!** error, and the new **SINGLE** function required in place of Implicit Intersection.

Let's start with the basic array formula. Go to cell **E3**. Type **=A2:C10**, as shown here. In the past, you would have had to wrap that formula in an aggregation function and maybe use **Ctrl+Shift+Enter**.

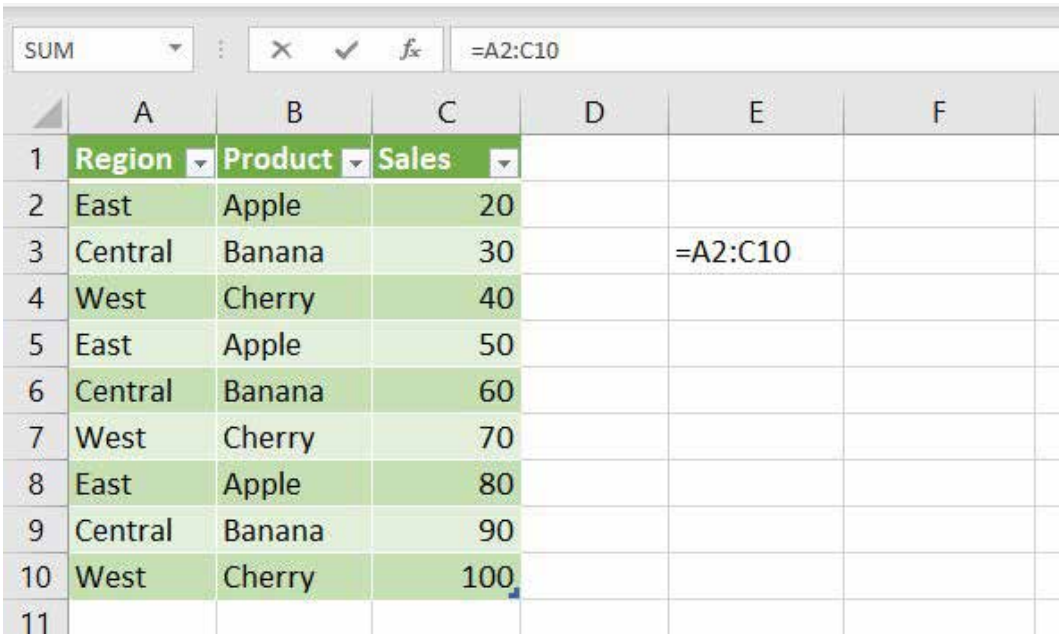


Figure 2 *One cell, pointing to many cells.*

But now you can simply press **Enter**. Excel returns 27 answers—and the answers spill into the adjacent cells! Look at the formula in the formula bar...there aren't any curly braces, which means no one pressed **Ctrl+Shift+Enter**.

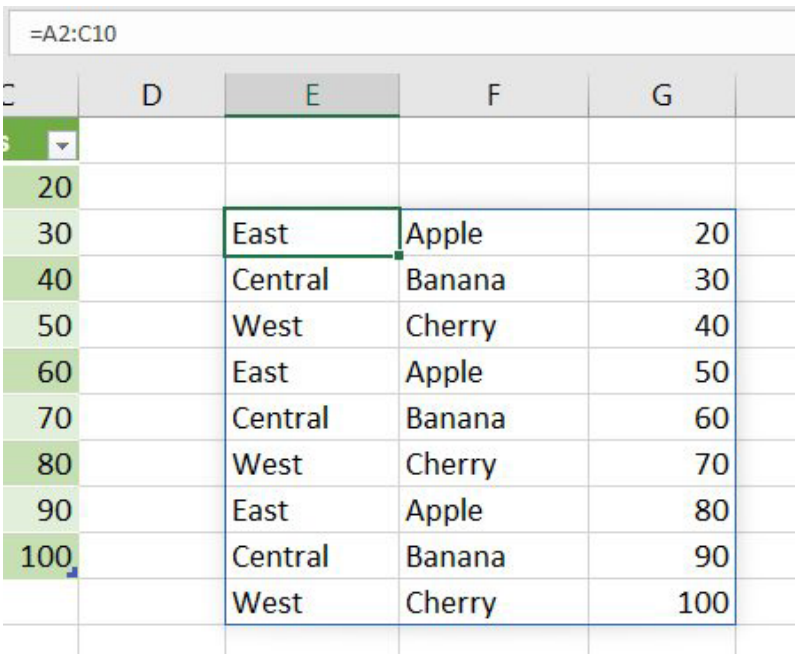


Figure 3 *One formula brought all of these answers.*

Cell **E4** contains the value Central, and although the formula bar contains a formula for this cell, it is grayed out.

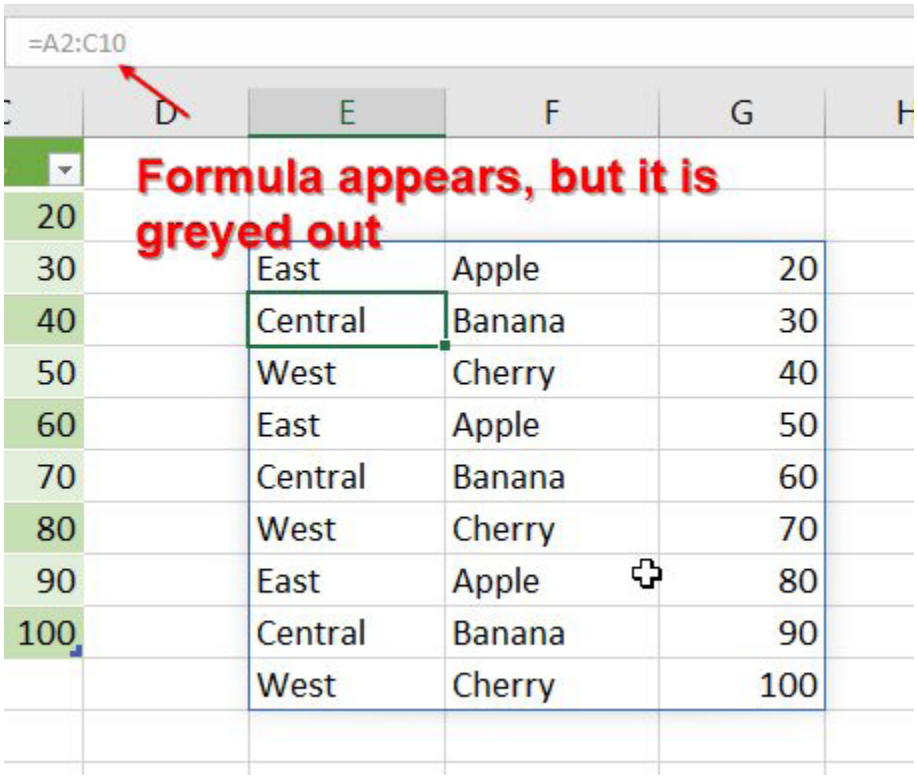


Figure 4 *We are in a whole new world here, Captain.*

Just to prove that **E4** does not contain a formula, here are some queries typed into the Immediate pane in VBA. VBA reports that there is a value in **E4** but no formula.

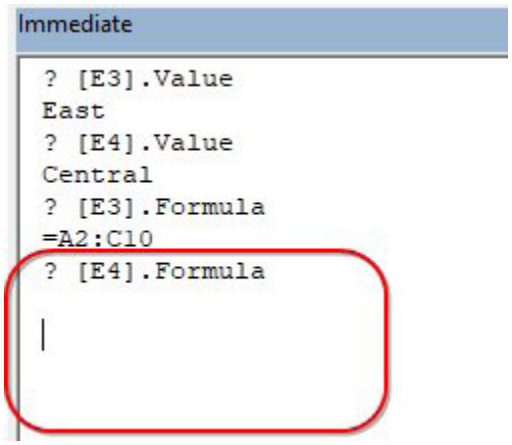


Figure 5 *VBA is reporting no formula in* **E4**.

But if you select **D1:H20** and then select **Go To Special**, **Formulas**, you see **E3:G12** shown as formula cells. This is not supposed to happen. Joe McDaid says it is a bug. In the near future, an update will be pushed out where only **E3** will be selected.

One of the first questions on YouTube in response to my first Dynamic Array videos: Can you copy and paste values? Yes! You would select **E3:G11**, select **Copy**, select a new cell, and select **Paste Values**.

WHAT HAPPENS IF A FORMULA CAN'T SPILL?

What if we put some stuff in the way and enter one of the Dynamic Array formulas, what will happen? (I like the way you are thinking: Let's try to break it.)

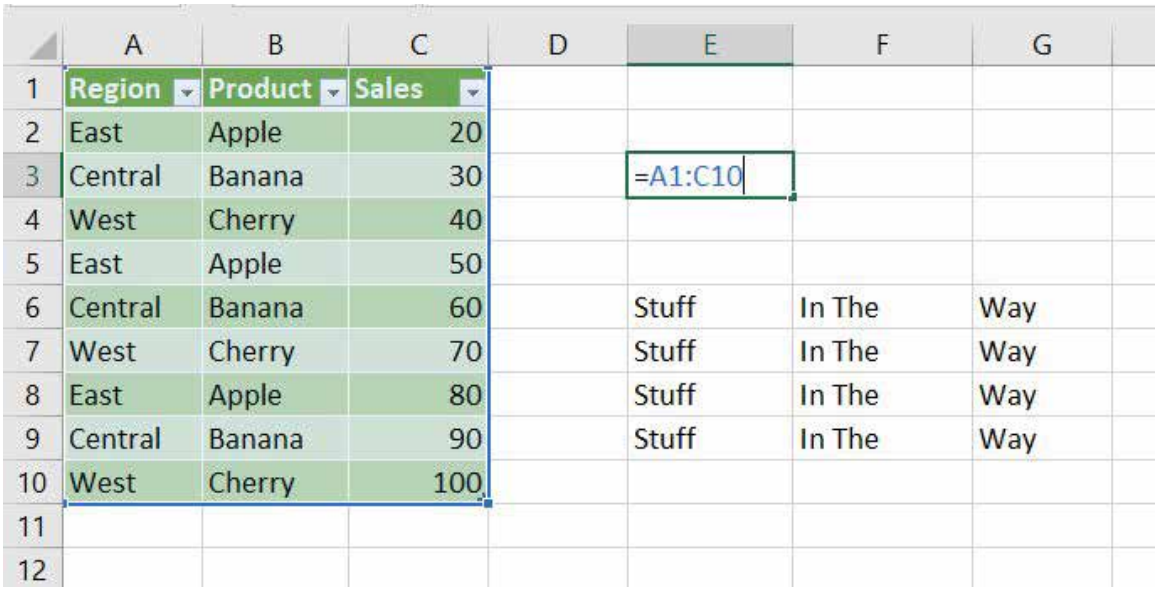


Figure 6 *How will Excel deal with this?*

This is quite possibly the first published screenshot of the new **#SPILL!** error. This is Excel's way of saying that it can't return *all* the results, so it won't return *any* of the results. The little drop-down to the left of the cell gives you the choice Select Obstructing Cells, so perhaps you can cut them and paste them elsewhere.

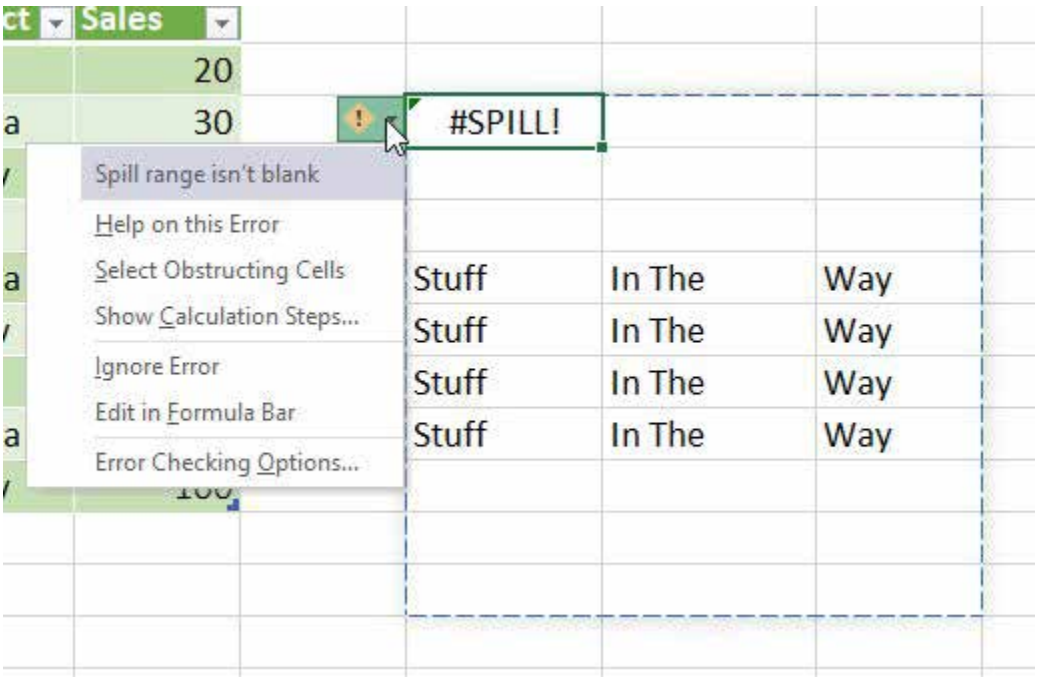


Figure 7 *We've got a* **#SPILL!** *error in Aisle E...*

As soon as you clear out the obstructing cells, the results appear.

	A	B	C	D	E	F	G
1	Region	Product	Sales				
2	East	Apple	20				
3	Central	Banana	30		Region	Product	Sales
4	West	Cherry	40		East	Apple	20
5	East	Apple	50		Central	Banana	30
6	Central	Banana	60		West	Cherry	40
7	West	Cherry	70		East	Apple	50
8	East	Apple	80		Central	Banana	60
9	Central	Banana	90		West	Cherry	70
10	West	Cherry	100		East	Apple	80
11	South	Guava	110		Central	Banana	90
12					West	Cherry	100
13					South	Guava	110
14							

Figure 8 *Clear the obstructing cell, and the results appear.*

There are several types of **#SPILL!** errors. The one above is **Spill Range Is Not Blank**. Other errors include:

- **Indeterminate Size:** You can't use a volatile argument such as **RANDBETWEEN** inside of **SEQUENCE**.
- **Extends Beyond Worksheet's Edge:** You can't **=SORT(C:C)** from **E2**.
- **Table Formula:** You can't use a Dynamic Array inside a table.
- **Out of Memory:** You need to reference a smaller range.
- **Spill into Merged Cells:** A Dynamic Array cannot spill into a merged cell.
- **Unrecognized/Fallback:** Excel can't reconcile the error.

If Your Formula Points to a Table, the Array Will Expand

Did you notice what else happened in the previous figure? I typed a new row in the table in **columns** **A:C**. The formula expanded to include the extra row. This happened because **A1:C11** is defined as a **Ctrl+T** table. It would also happen if you inserted new rows in the middle of a regular range.

While Dynamic Array formulas can point to a table, you cannot include Dynamic Array results in a table.

What Is Really Happening Behind the Scenes?

In this example, the formula and the results of that formula are stored in cell **E3**. As shown here, you may be able to catch Excel in a weird state where the value in **E3** reveals the double-byte characters stored after **Apple**. In this case, Excel is storing the whole array in **E3** and displaying it in the spill range.

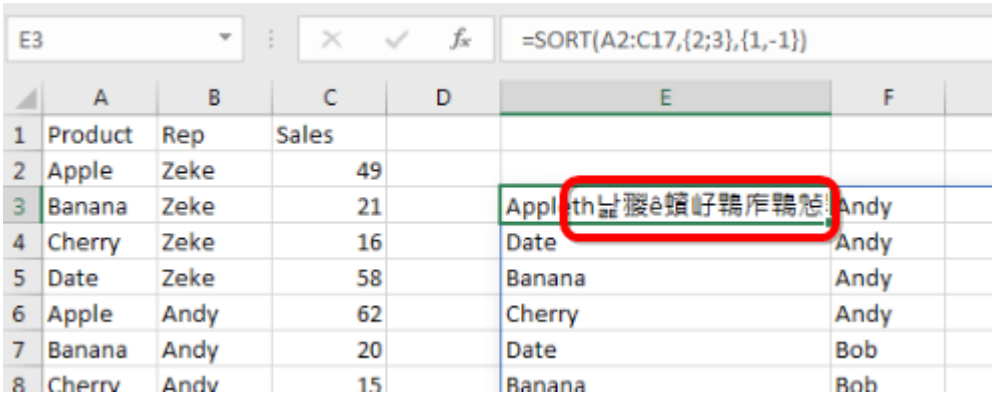


Figure 9 *Double-byte characters hold the entire array in* **E3** *but are not supposed to be seen.*

USING THE NEW ARRAY REFERENCE NOTATION: E3#

How will you ever refer to **E3:G13** if you don't know how tall the range is going to be? For this, you add the spilled range operator (#) after the cell containing the array formula.

For example, **=E3** refers to Apple. **=E3#** refers to the entire array that starts in **E3**. This is unofficially called Array Reference notation. You will see many examples of this in Chapter 10.

WHAT ABOUT IMPLICIT INTERSECTION?

I know a lot of Excel gurus. Who is the person who knows every single picayune fact about Excel? Is it Charley Kyd? Bob Umlas? Jordan Goldmeier? Well, whoever it is, by now that person is surely bellowing from the back of the room, "What about Implicit Intersection!!!"

Oh, wow. Implicit Intersection is about as common as flying pigs, but someone had to bring it up. Have you *ever* seen anyone actually use Implicit Intersection in the wild? Do we really need to talk about this?

So I don't bore you to tears, I'll just provide an example to show how it works: If you enter **=C$2:C$10** anywhere in rows **2** through **10**, the formula will return from the row where it was entered. I've seen one person use this in real life, and she had no idea what she was doing. I've seen Mike Girvin cover it in a YouTube video. Bob Umlas has written about it. If you were a fan of it, you might want to know how Implicit Intersection is going to work in this brave new world.

If you really desperately *need* Implicit Intersection, wrap your formula in the **SINGLE** function, as shown here.

	A	B	C	D	E	F	G
1	Region	Product	Sales				
2	East	Apple	20		Implicit intersection		
3	Central	Banana	30		requires SINGLE		
4	West	Cherry	40				
5	East	Apple	50				
6	Central	Banana	60		60	=SINGLE($C$2:$C$11)	
7	West	Cherry	70				
8	East	Apple	80				
9	Central	Banana	90		90	=SINGLE($C$2:$C$11)	
10	West	Cherry	100				
11	South	Guava	110				
12							

Figure 10 *Use the* **SINGLE** *function when you need Implicit Intersection.*

This is the crazy-amazing story of the backward compatibility in action here: If you use **SINGLE** in an Office 365 file and then open that file in Excel 2013, Joe McDaid will rewrite the formula as Implicit Intersection in Office 365. And if you open an Excel 2013 function that has Implicit Intersection, Joe McDaid will rewrite that formula to use **SINGLE**.

2 - THE SORT FUNCTION

A SIMPLE SORT WITH ONE ARGUMENT

Sorting with a formula in Excel used to require an insane combination of formulas. Take a look at the following data, which is used throughout this chapter.

	A	B	C
1	Original Data		
2			
3	Name	Team	Score
4	Amanda	Red	559
5	Bernardo	Blue	999
6	Carol	Red	808
7	Daniel	Blue	200
8	Daniela	Red	903
9	Eduardo	Blue	656
10	Gabriel	Red	404
11	Helena	Blue	526
12	Isabella	Red	559
13	Joao Pedro	Blue	485
14	José	Red	354
15	Stephanie	Blue	700
16	Thiago	Red	693
17			

Figure 11 *Data in* **A3:C11**.

Before September 24, 2018, in order to sort this data with a formula, you would have had to use a handful of functions: **RANK**, **COUNTIF**, **MATCH**, **INDEX**, and **INDEX**. Once you'd finished this set of formulas, you would have been ready for a nap.

E	F	G	H	I
Old Way To Sort With A Formula				
Rank of Original Data	Numbers	Position	Name	Score
=RANK(C4,$C$4:$C$16)+COUNTIF(C$3:C3,C4)	1	=MATCH(F4,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G4)	=INDEX($C$4:$C$16,G4)
=RANK(C5,$C$4:$C$16)+COUNTIF(C$3:C4,C5)	2	=MATCH(F5,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G5)	=INDEX($C$4:$C$16,G5)
=RANK(C6,$C$4:$C$16)+COUNTIF(C$3:C5,C6)	3	=MATCH(F6,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G6)	=INDEX($C$4:$C$16,G6)
=RANK(C7,$C$4:$C$16)+COUNTIF(C$3:C6,C7)	4	=MATCH(F7,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G7)	=INDEX($C$4:$C$16,G7)
=RANK(C8,$C$4:$C$16)+COUNTIF(C$3:C7,C8)	5	=MATCH(F8,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G8)	=INDEX($C$4:$C$16,G8)
=RANK(C9,$C$4:$C$16)+COUNTIF(C$3:C8,C9)	6	=MATCH(F9,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G9)	=INDEX($C$4:$C$16,G9)
=RANK(C10,$C$4:$C$16)+COUNTIF(C$3:C9,C10)	7	=MATCH(F10,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G10)	=INDEX($C$4:$C$16,G10)
=RANK(C11,$C$4:$C$16)+COUNTIF(C$3:C10,C11)	8	=MATCH(F11,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G11)	=INDEX($C$4:$C$16,G11)
=RANK(C12,$C$4:$C$16)+COUNTIF(C$3:C11,C12)	9	=MATCH(F12,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G12)	=INDEX($C$4:$C$16,G12)
=RANK(C13,$C$4:$C$16)+COUNTIF(C$3:C12,C13)	10	=MATCH(F13,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G13)	=INDEX($C$4:$C$16,G13)
=RANK(C14,$C$4:$C$16)+COUNTIF(C$3:C13,C14)	11	=MATCH(F14,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G14)	=INDEX($C$4:$C$16,G14)
=RANK(C15,$C$4:$C$16)+COUNTIF(C$3:C14,C15)	12	=MATCH(F15,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G15)	=INDEX($C$4:$C$16,G15)
=RANK(C16,$C$4:$C$16)+COUNTIF(C$3:C15,C16)	13	=MATCH(F16,$E$4:$E$16,0)	=INDEX($A$4:$A$16,G16)	=INDEX($C$4:$C$16,G16)

Figure 12 *The old way to sort with a formula.*

To greatly simplify our sorting in such a situation, Joe McDaid and his team have brought us **SORT** and **SORTBY**.

Let's start with **SORT**. Here is the syntax: **=SORT(array, [sort_index], [sort_order], [by_col])**.

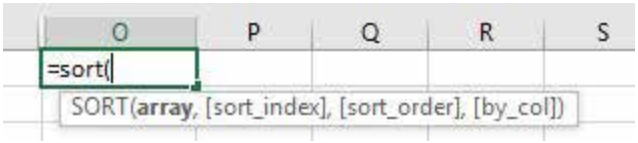


Figure 13 *The* **SORT** *Function.*

Let's say you want to sort **A3:C16** by the Score field. Score is the third column in the array, so your **sort_index** value will be **3**.

The choices for the **sort_order** value are **1** for ascending or **-1** for descending. I am not complaining, but there will never be support for sorting by color, formula, or a custom list using this function.

Figure 14 *Specify* **3** *as the* **sort_index** *value and* **-1** *as the* **sort_order** *value, for descending.*

The fourth argument will rarely be used. It is possible in the Sort dialog to sort by columns instead of rows, but 99.9% of people sort by rows. If you need to sort by columns, specify **True** for the final argument, **by_col**. This argument is optional and defaults to **False**.

Figure 15 *If you need to sort by columns, use* **True** *for the final argument,* **by_col**.

Here are the results of the single formula entered, in this case, in cell **O2**. Thanks to the new calc engine, the formula spills into adjacent cells.

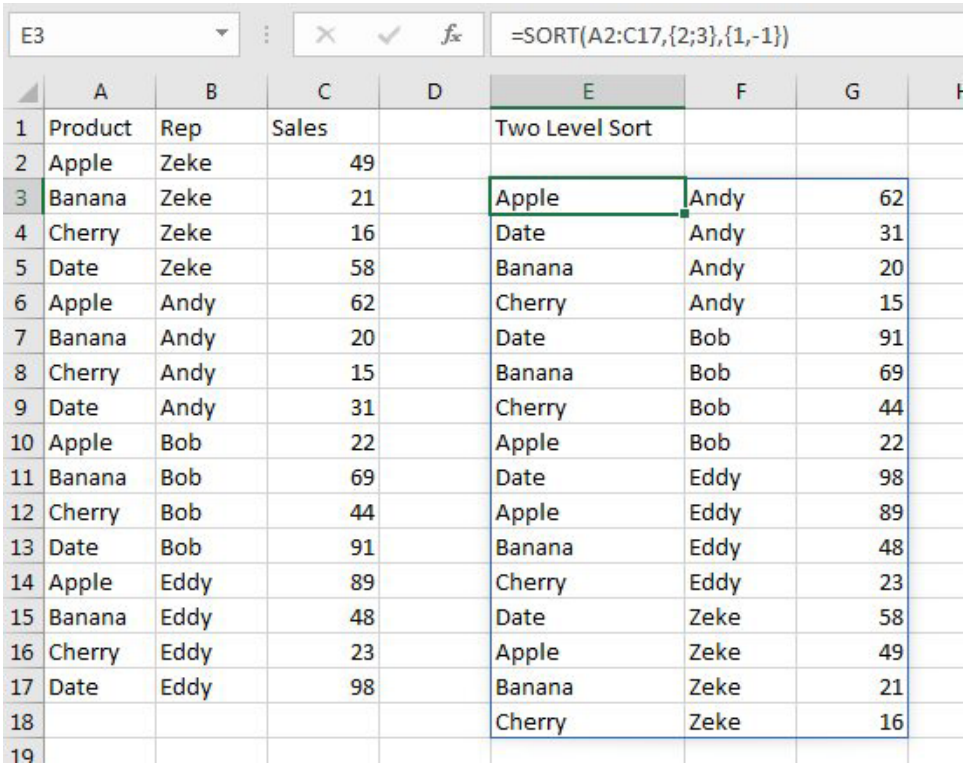
=SORT(A3:C16,3,-1,FALSE)

	O	P	Q	R
	Name	Team	Score	
	Bernardo	Blue	999	
	Daniela	Red	903	
	Carol	Red	808	
	Stephanie	Blue	700	
	Thiago	Red	693	
	Eduardo	Blue	656	
	Amanda	Red	559	
	Isabella	Red	559	
	Helena	Blue	526	
	Joao Pedro	Blue	485	
	Gabriel	Red	404	
	José	Red	354	
	Daniel	Blue	200	

Figure 16 *The original data is now sorted.*

A SORT BASED ON TWO OR MORE COLUMNS OF RESULTS

What if you need a two-level sort, such as sorting by column two ascending and column three descending? In this case, you supply array constants for the second and third arguments: **=SORT(A2:C17,{2;3},{1;-1})**.



	A	B	C	D	E	F	G
1	Product	Rep	Sales		Two Level Sort		
2	Apple	Zeke	49				
3	Banana	Zeke	21		Apple	Andy	62
4	Cherry	Zeke	16		Date	Andy	31
5	Date	Zeke	58		Banana	Andy	20
6	Apple	Andy	62		Cherry	Andy	15
7	Banana	Andy	20		Date	Bob	91
8	Cherry	Andy	15		Banana	Bob	69
9	Date	Andy	31		Cherry	Bob	44
10	Apple	Bob	22		Apple	Bob	22
11	Banana	Bob	69		Date	Eddy	98
12	Cherry	Bob	44		Apple	Eddy	89
13	Date	Bob	91		Banana	Eddy	48
14	Apple	Eddy	89		Cherry	Eddy	23
15	Banana	Eddy	48		Date	Zeke	58
16	Cherry	Eddy	23		Apple	Zeke	49
17	Date	Eddy	98		Banana	Zeke	21
18					Cherry	Zeke	16
19							

Figure 17 *Two-level sort.*

A RANDOM SORT USING SORT AND RANDARRAY

Difficult scenarios like random drug testing and random with no repeats become mind-numbingly simple when you combine **SORT** with **RANDARRAY**.

Say that you want to sort 13 names randomly and without repeats. To do this in post-September 24, Excel, you use **=SORTBY(A4:A16,RANDARRAY(13))** , as shown below. (You'll read more about **RANDARRAY** in Chapter 8.)

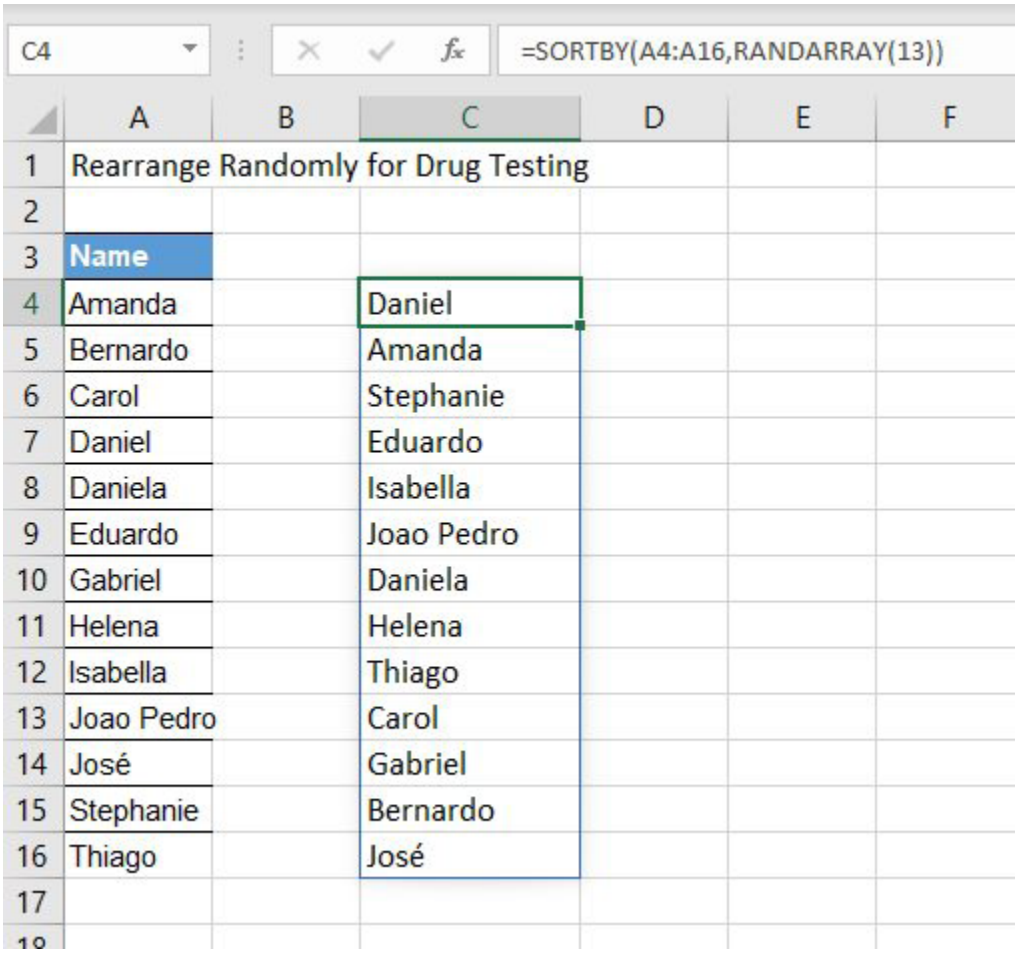


Figure 18 *Sorting randomly without repeats.*

WHAT'S LEFT FOR CTRL+SHIFT+ENTER?

Now that we have these new functions, is **Ctrl+Shift+Enter** completely dead? No. There are still two uses for it.

First, let's say you want to make sure no one can delete or insert any rows in rows **1:20**. In this case, you can select **Z1:Z20**, type **=1**, and press **Ctrl+Shift+Enter**. If anyone then tries to delete or insert a row or rows, Excel will not allow the changes because users aren't allowed to change part of an array.

Another use: Let's say you want only the top three results from the **SORT** function. You could select three cells, type **SORT(...)**, and press **Ctrl+Shift+Enter**. This will prevent the results from spilling beyond the bounds of the original formula.

In this help topic, Microsoft says that it is supporting **Ctrl+Shift+Enter** formulas for backward compatibility only, and we should all stop using them now.

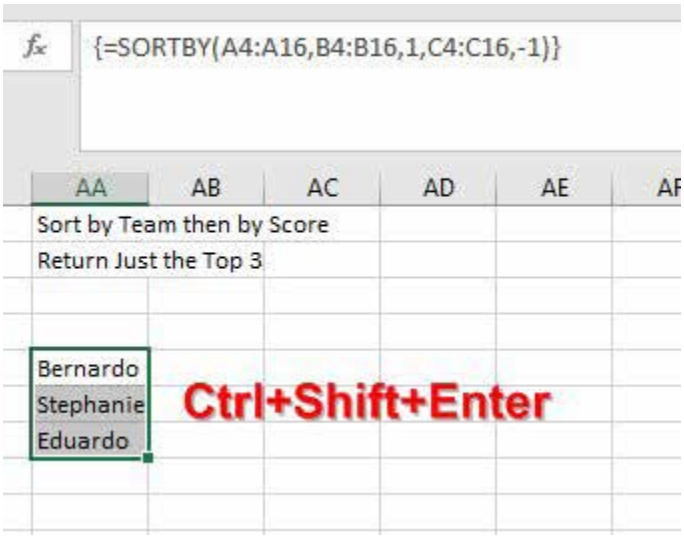


Figure 19 **Ctrl+Shift+Enter** *will still limit the size of a returned array.*

3 - The SORTBY Function

A SORT BY SOMETHING THAT IS NOT IN THE RESULTS

The **SORTBY** function's syntax is **=SORTBY(array, by_array1, sort_order1,)**...

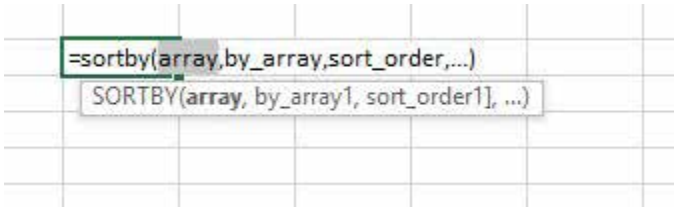


Figure 20 *It is possible to sort by something else.*

Say that you want to sort by team and then score, and you want to show only the names. In this case, you can use **SORTBY** as shown here.

Figure 21 *Sorting column* **A** *by column* **B** *and column* **C**.

4 - THE FILTER FUNCTION

USING THE FILTER FUNCTION WITH ONE CONDITION

The new **FILTER** function accepts an array, keeps the rows you specify, and returns the results to a spill range.

This chapter uses the following data set,

	A	B	C
1	FILTER function in Excel		
2			
3	Name	Team	Score
4	Amanda	Red	559
5	Bernardo	Blue	291
6	Carol	Red	808
7	Daniel	Blue	200
8	Daniela	Red	903
9	Eduardo	Blue	656
10	Gabriel	Red	404
11	Helena	Blue	526
12	Isabella	Red	559
13	Joao Pedro	Blue	485
14	José	Red	354
15	Stephanie	Blue	270
16	Thiago	Red	693
17			

Figure 22 *Name, team, and score.*

The syntax of the **FILTER** function is **FILTER(array, include, [if_empty])**.

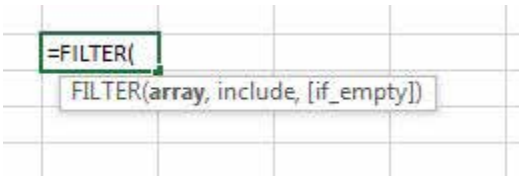


Figure 23 **FILTER** *syntax.*

Say that you want to retrieve all rows from the data set where the team is Blue. Type **Blue** in cell **F1**. The formula entered in **E4** is **=FILTER(A4:C16,B4:B16=F1,"None Found")**.

Notice that you don't have to use the **F4** key or dollar signs in the formula. This single formula returns multiple results. There is no need to copy it anywhere, so there is no need for absolute references. Also, you do not have to press **Ctrl+Shift+Enter**.

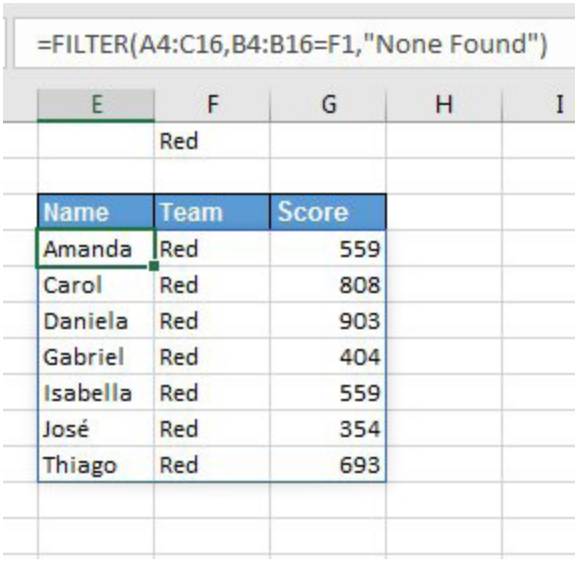


Figure 24 *One* **FILTER** *formula returns all Red team members.*

If you now type **Blue** in **F1**, you get all of the Blue team members.

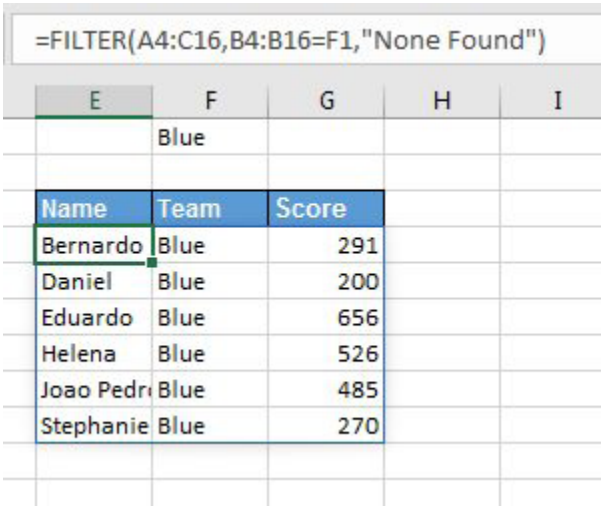


Figure 25 *Change the value in* **F1** *to change the results.*

The optional third argument of **FILTER** is illustrated here. **[if_empty]** specifies the text to return in case there are no results. If you change the value in **F1** to **Lime**, you get the text specified in the formula.

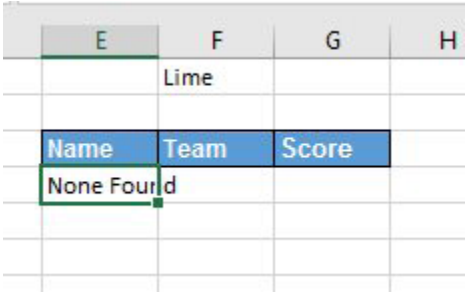


Figure 26 *Specify what to return if nothing is found.*

The result above is not very complete. What if you have a formula that really needs a numeric answer in the third column? In this case, you can specify an array constant as the [if_empty] argument: **=FILTER(A4:C16,B4:B16=F1,{"None Found","No Team",0})**.

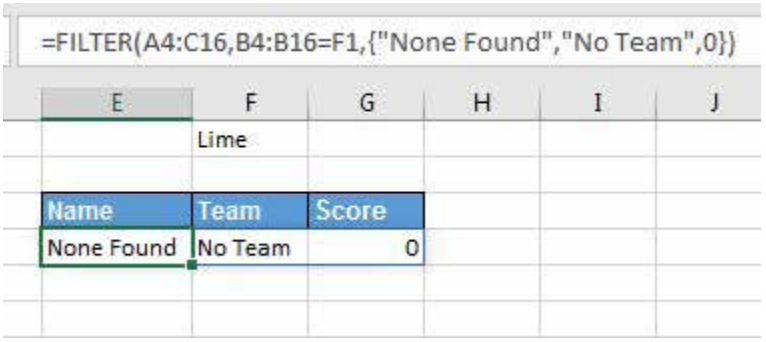
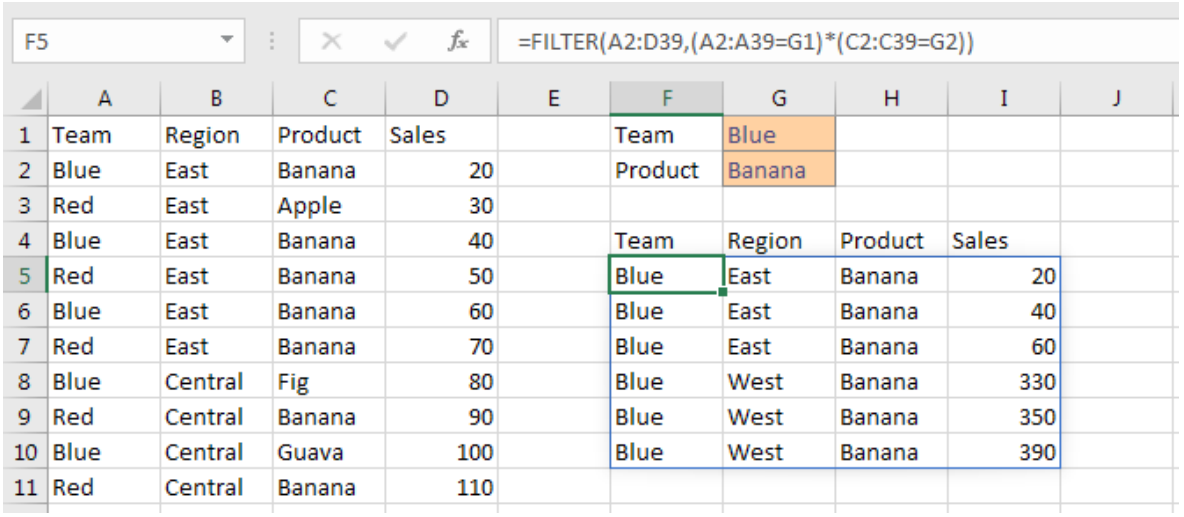


Figure 27 *Will the third argument accept an array? Yes.*

USING FILTER WITH MULTIPLE CONDITIONS

Say you have to combine two criteria, and both criteria have to be true. Wrap each item in parentheses and multiply them together. In the figure below, the formula is **=FILTER(A2:D39,(A2:A39=G1)*(C2:C39=G2))**. Thanks to Smitty Smith for this technique.



	A	B	C	D	E	F	G	H	I	J
1	Team	Region	Product	Sales		Team	Blue			
2	Blue	East	Banana	20		Product	Banana			
3	Red	East	Apple	30						
4	Blue	East	Banana	40		Team	Region	Product	Sales	
5	Red	East	Banana	50		Blue	East	Banana	20	
6	Blue	East	Banana	60		Blue	East	Banana	40	
7	Red	East	Banana	70		Blue	East	Banana	60	
8	Blue	Central	Fig	80		Blue	West	Banana	330	
9	Red	Central	Banana	90		Blue	West	Banana	350	
10	Blue	Central	Guava	100		Blue	West	Banana	390	
11	Red	Central	Banana	110						

Figure 28 *Multiply two sets of conditions to join with* **AND**.

If you want to see all records that are either Fig or Guava, join the conditions with a plus sign:
=FILTER(A2:D39,(C2:C39="Fig")+(C2:C39="Guava"))

For all team Blue records with Fig or Guava:
=FILTER(A2:D39,((C2:C39="Fig")+(C2:C39="Guava"))*(A2:A39=G1))

The more you use these new Dynamic Array formulas, the more amazing possibilities you see.

5 - The UNIQUE Function

SYNTAX OF THE UNIQUE FUNCTION

The new **UNIQUE** function is part of the Dynamic Array formula collection. It returns the distinct values from an array.

The syntax of this function is **=UNIQUE(array, [by_col], [occurs_once])**. In this syntax, **array** is any array. For **by_col**, the choices are **True** to compare by column and **False** to compare by row. It seems that comparing by row is the logical choice, and it is the default if you omit the second argument. The optional **occurs_once** argument is interesting and brings me to a small rant.

UNDERSTANDING UNIQUE VERSUS DISTINCT

Consider this list: Apple, Apple, Banana, Cherry, Cherry. What would you say are the unique items in the list? If you are not a database pro, you would say the correct answer is Apple, Banana, Cherry. But several features in Excel would say that Banana is the only item in the list that has no duplicates. For example, if you select **Home**, **Conditional Formatting**, **Highlight Cells Rules**, **Duplicate Values**, **Unique**, Excel will highlight only Banana. Database pros say that Apple, Banana Cherry is a list of *distinct* values and that Banana is the only *unique* item.

Who would ever care about a product that was sold once? Unless you are listing one-hit wonders, this definition of unique seems useless. We'll send this one out as a long-distance dedication to Casey Kasem in the afterlife.

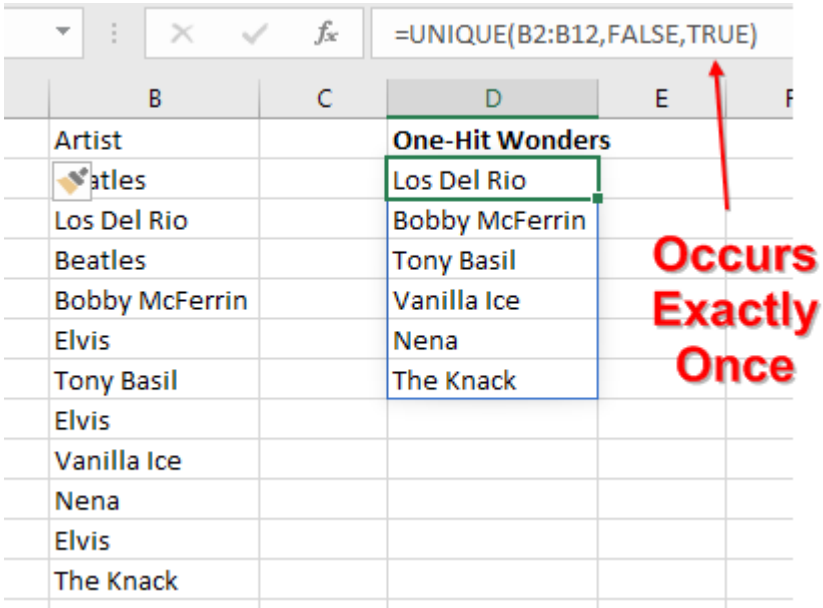


Figure 29 *Return a list of items that occur exactly once.*

The great news is that the **UNIQUE** function can be used a couple ways. In this case, you could have it return Apple, Banana, Cherry (the default), or you could change the third argument and get just Banana.

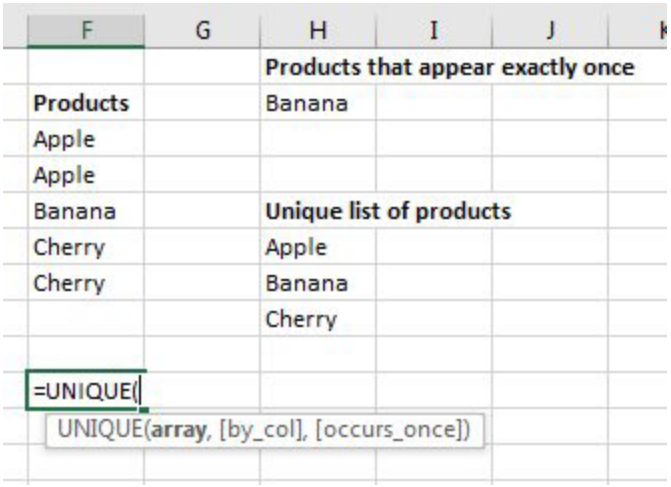


Figure 30 *Which would you say is the true unique list?*

The rest of the examples in this chapter use the following data set.

	A	B	C	D
3	Team	Name	Product	Score
4	Red	Amanda	Apple	559
5	Blue	Bernardo	Orange	291
6	Red	Carol	Lemon	808
7	Blue	Daniel	Lime	200
8	Red	Daniela	Apple	903
9	Blue	Eduardo	Orange	656
10	Red	Gabriel	Lemon	404
11	Blue	Helena	Lime	526
12	Red	Isabella	Apple	559
13	Blue	Joao Pedro	Orange	485
14	Red	José	Lemon	354
15	Blue	Stephanie	Lime	270
16	Red	Thiago	Apple	693
17	Red	Amanda	Orange	170
18	Blue	Bernardo	Lemon	180
19	Red	Carol	Lime	190
20	Blue	Daniel	Apple	200
21	Red	Daniela	Orange	210
22	Blue	Eduardo	Lemon	220
23	Red	Gabriel	Lime	230
24	Blue	Helena	Apple	240
25	Red	Isabella	Orange	250
26	Blue	Joao Pedro	Lemon	260
27	Red	José	Lime	270
28	Blue	Stephanie	Quince	280
29	Red	Thiago	Kiwi	290

Figure 31 *Team, name, product, and score data.*

Using **UNIQUE** on this data is simple. Because the second and third arguments are optional, you can use **=UNIQUE(C4:C29)** to return a list of unique products.

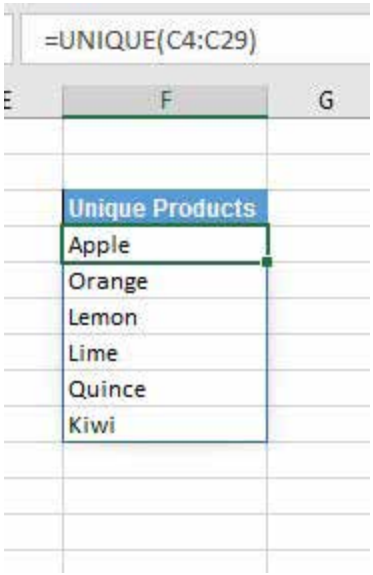


Figure 32 *Returning a list of unique products.*

The chapter of Mike Girvin's **Ctrl+Shift+Enter** book on getting unique values just became one sentence. I love his book (and I am the publisher of that book), but it is about to become a pamphlet with all of these simplications!

What if you need every unique combination of name and product? You can use a two-column array: **=UNIQUE(B4:C29)**.

Figure 33 *Every unique combination of two values.*

6 - COMBINING FUNCTIONS

NESTING ARRAY FUNCTIONS: SORT AND UNIQUE

The examples in this brief chapter are amazing. Need to sort a list of unique products? No problem: With the data shown in Figure 31, use **=SORT(UNIQUE(C4:C29,FALSE,FALSE))**.

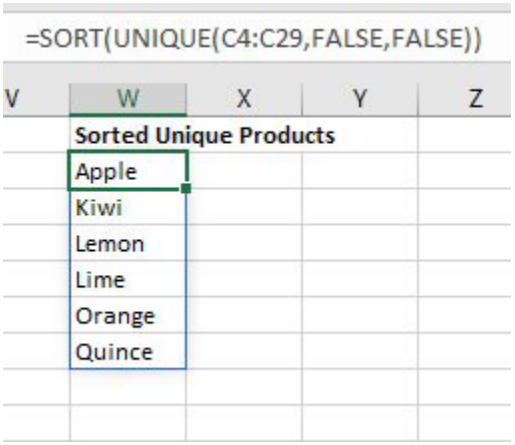


Figure 34 *Get the unique values and sort, all in one formula.*

NESTING ARRAY FUNCTIONS: SORT, UNIQUE, AND FILTER

Say that you want to filter the names in column **B** in Figure 31 so you get just the rows where the team in column **A** is Blue. Say also that you want to then get just the unique values and sort them. To do all this, you just use **=SORT(UNIQUE(FILTER(B4:B29,A4:A29="Blue"),FALSE))**.

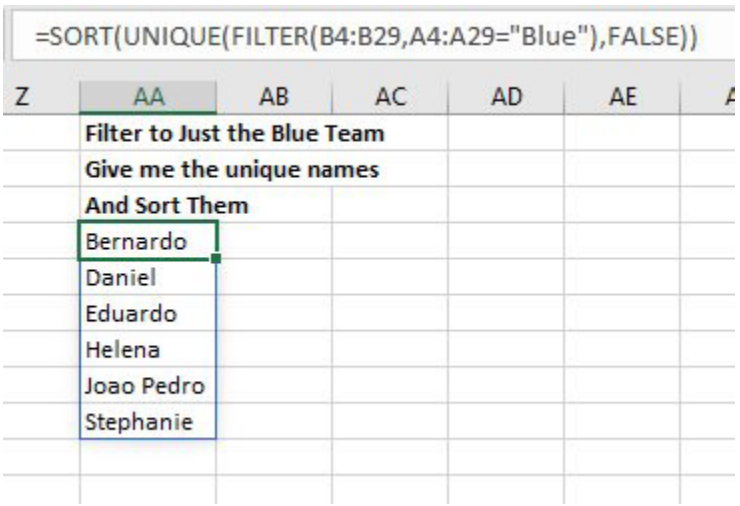


Figure 35 *Nesting array functions is no problem.*

7 - The SEQUENCE Function

GENERATING A RANGE OF SEQUENTIAL NUMBERS

I have a cool trick in my Power Excel seminars where I show how you can Ctrl+drag the fill handle from a cell with the number 1, and Excel will extend to 2, 3, 4. It is an obscure trick, and not many people know it. When I first saw the **SEQUENCE** function, I thought it might be for people who had never discovered this trick. But as you will see in this chapter and in Chapter 10, the **SEQUENCE** function is the key to making many other Excel functions into array functions.

SEQUENCE generates an array of numbers. The syntax is **=SEQUENCE(rows, [columns], [start], [step])**. While **=SEQUENCE(10)** generates 1 through 10, you can customize the numbers with **=SEQUENCE(10,3,5,10)** to generate a 10-row by 3-column array starting at 5 and stepping by 10.

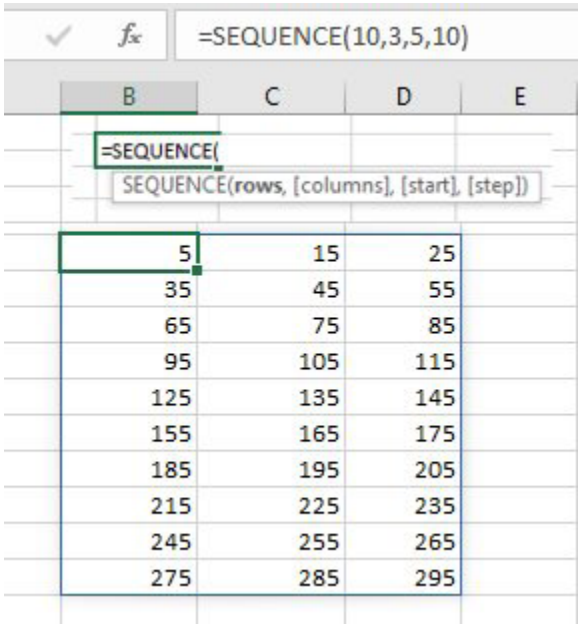


Figure 36 *Generating a sequence of numbers.*

The **RANDARRAY** function is great for driving Monte Carlo analysis. To use it, you just specify how many rows and columns of random numbers you want.

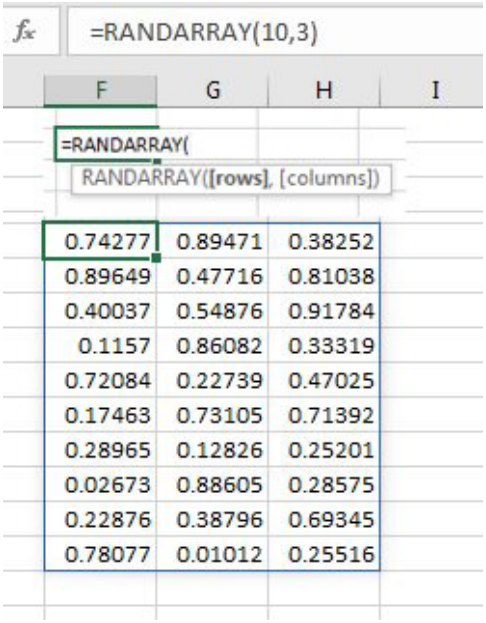


Figure 37 *Finding 30 random numbers.*

Using SEQUENCE Inside Another Function

The figure below calculates the interest payment for each of the first five months of a loan. You have to key in the numbers 1 through 5 in **A7:A11** or use **=ROW(1:1)** and copy down. Having to use the **=ROW(1:1)** hack to generate a sequence of number will go away soon.

	A	B
1	Price	495000
2	Term	360
3	Rate	4.25%
4	Payment	($2,435.10)
5		
6	Month	Interest
7	1	($1,753.13)
8	2	($1,750.71)
9	3	($1,748.29)
10	4	($1,745.85)
11	5	($1,743.41)
12		($8,741.39)
13		

Figure 38 *Five formulas to calculate interest payments for five months.*

For the period argument of the **IPMT** function, I tried putting in **SEQUENCE(5)**, but, darn it, the formula spilled and gave me five results.

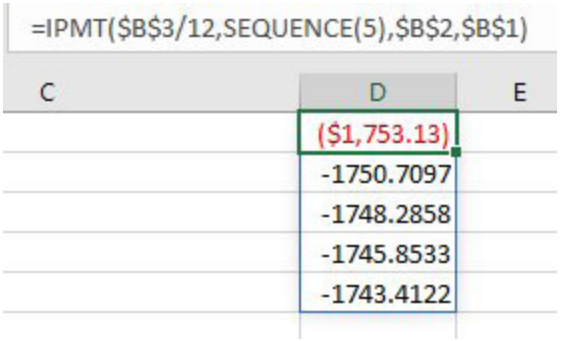


Figure 39 *I wanted one answer instead of five.*

> *The formatting in the image above is one buggy thing about these new Dynamic Arrays. If your formula is going to spill to five cells, you should format them first. In this case, Excel formatted the first cell, but the formatting does not get copied. When I asked Excel Project Manager Joe McDaid about this, he replied: "It is a known issue. We wanted to get you all of the functionality now and fix the formatting later." That is a fair point. I want the functionality now and can worry about formatting later.*

To return a single answer when Excel wants to spill to five cells, you can use a wrapper function, such as **SUM**: **=SUM((IPMT($B$3/12,SEQUENCE(5),$B$2,$B$1))**.

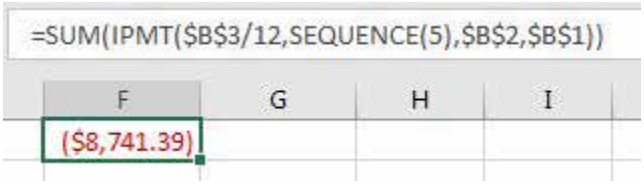


Figure 40 *Five answers summarized to one cell.*

Say that you want to figure out how much interest you will pay in year 3 of a loan. You don't need to create a loan amortization table to do this. The formula shown below does it all at once. You still use the **IPMT** function. But for the third argument, you can specify **SEQUENCE(12,1,37,1)** to generate periods 37 to 48.

In the image below, the starting month is in column **F**. The formula figures out the interest for 12 months, starting from the month number in column F.

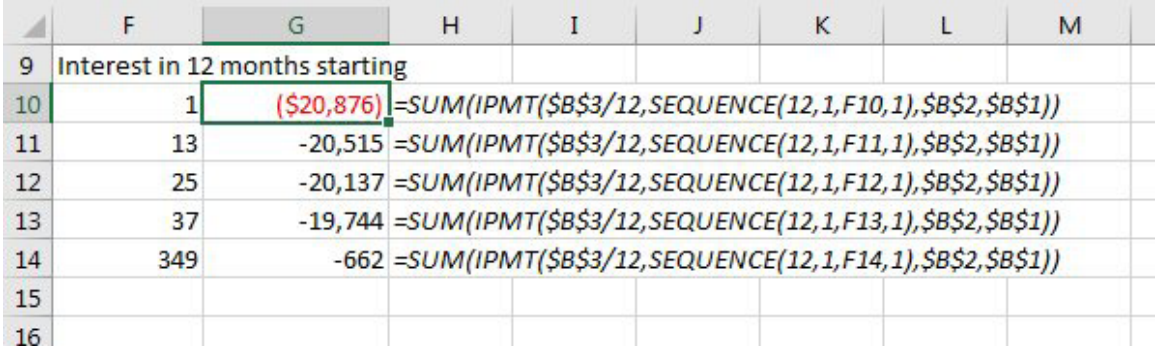
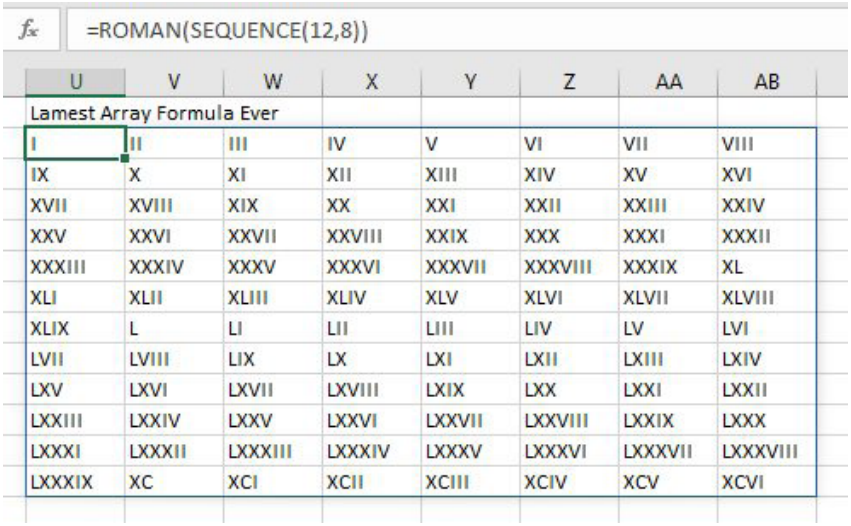


Figure 41 *Easier than the full amortization table.*

In this book, you've read about **SINGLE**, **SORT**, **SORTBY**, **FILTER**, **UNIQUE**, **SEQUENCE**, and **RANDARRAY**. But the new array functions are not limited to these seven functions. In a demonstration of the lamest array function ever, I present an array of **ROMAN** functions, generated using the **SEQUENCE(12,8)** function inside **ROMAN**. Virtually every Excel formula is now an array formula, with no need for **Ctrl+Shift+Enter**.



U	V	W	X	Y	Z	AA	AB
Lamest Array Formula Ever							
I	II	III	IV	V	VI	VII	VIII
IX	X	XI	XII	XIII	XIV	XV	XVI
XVII	XVIII	XIX	XX	XXI	XXII	XXIII	XXIV
XXV	XXVI	XXVII	XXVIII	XXIX	XXX	XXXI	XXXII
XXXIII	XXXIV	XXXV	XXXVI	XXXVII	XXXVIII	XXXIX	XL
XLI	XLII	XLIII	XLIV	XLV	XLVI	XLVII	XLVIII
XLIX	L	LI	LII	LIII	LIV	LV	LVI
LVII	LVIII	LIX	LX	LXI	LXII	LXIII	LXIV
LXV	LXVI	LXVII	LXVIII	LXIX	LXX	LXXI	LXXII
LXXIII	LXXIV	LXXV	LXXVI	LXXVII	LXXVIII	LXXIX	LXXX
LXXXI	LXXXII	LXXXIII	LXXXIV	LXXXV	LXXXVI	LXXXVII	LXXXVIII
LXXXIX	XC	XCI	XCII	XCIII	XCIV	XCV	XCVI

Figure 42 *Leave it to MrExcel to find the world's lamest use of the new Dynamic Array formulas.*

8 - The RANDARRAY Function

GENERATING AN ARRAY OF RANDOM NUMBERS WITH RANDARRAY

The syntax for **RANDARRAY** is **=RANDARRAY([rows],[columns])**. The **RANDARRAY** function generates a series of random numbers between 0 and 1.

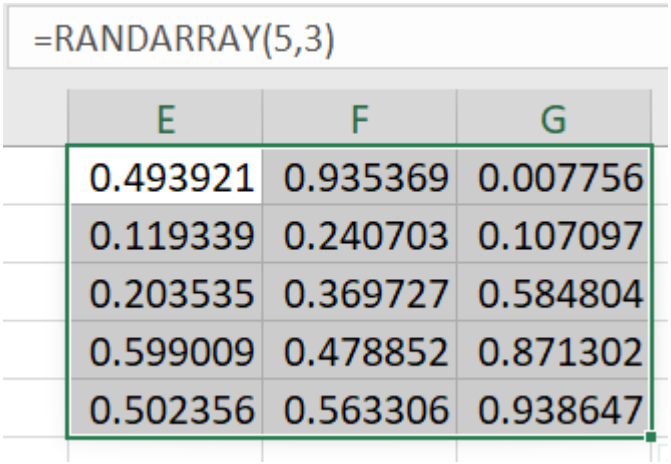


Figure 43 *Generating an array of 5 rows by 3 columns of random numbers.*

SIMULATING RANDBETWEEN

What if you want random numbers between 11 and 19? Then you are back to doing the same sort of calculations you had to do before **RANDBETWEEN** came along: **=ROUNDDOWN(RANDARRAY(1000)*(19-11+1),0)+11**.

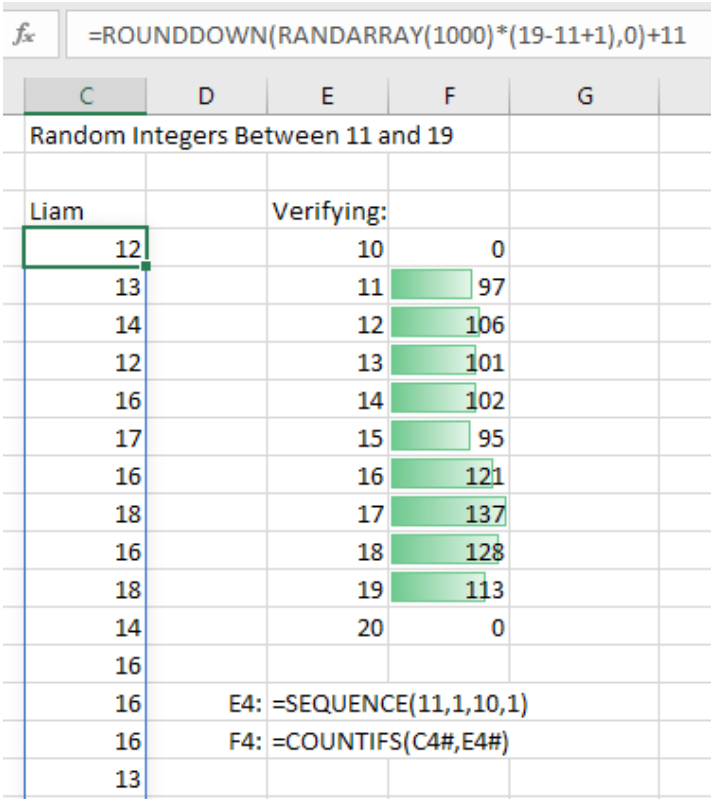


Figure 44 *An array of random numbers between 11 and 19.*

Excel MVP Liam Bastick of SumProduct.com is not just the wittiest Excel MVP but quickly pointed out that my original formula `=ROUND(RANDARRAY(10,3)*9,0)+10` would incorrectly include some 10s and not enough 19s. He points out: "Since `RAND` and `RANDARRAY` generate a random number greater than or equal to 0 but strictly less than 1, your original formula is biased against the two limits as they are only half as likely to occur as the other numbers."

USING RANDARRAY FOR MODELING AND SIMULATION

Because it produces a series of random numbers between 0 and 1, the `RANDARRAY` function can be great for performing modeling or simulations.

In August 2018, Excel MVP Oz du Soleil challenged six YouTube creators to make a video using four Excel ingredients. My video was about the asteroid named Bennu. It analyzed 100,000 trials of whether Bennu would strike Earth in the 2182–2196 time frame, creating a new tourist attraction named the Bennu Crater.

The Excel file used 100,000 cells with `RAND` inside `NORM.INV`. Each formula was along the lines of `=NORM.INV(RAND(),$H$4,$H$5)`. To the right of those 100,000 formulas were another 100,000 formulas doing `=VLOOKUP(V10,$N$23:$O$179,2,TRUE)`. Now, admittedly, I could have made these 200,000 formulas into 100,000 formulas with `=VLOOKUP(NORM.INV(RAND(),$H$4,$H$5),$N$23:$O$179,2,TRUE)`.

The result of the `VLOOKUP` was either a `0` or a `1`, with `0` meaning no Earth impact and `1` meaning Earth impact. In 100,000 trials, Bennu hits Earth about 44 times, or a 1-in-2273 chance of creating a new tourist attraction like Meteor Crater in the Arizona desert.

The original file weighs in at 3195 KB and took 13 seconds to recalculate.

When the new Dynamic Array formulas came along, I was able to replace the 200,000 calculation cells with a single array formula:

```
=SUM(VLOOKUP(NORM.INV(RANDARRAY(100000),$H$4,$H$5),$N$23:$O$179,2,TRUE))
```

The size of the workbook shrunk to 37 KB, and recalculation time dropped to 6.5 seconds.

9 - Why CSE Arrays Were So Hard: Implicit Intersection

If you've ever pressed **Ctrl+Shift+Enter** in the past, this chapter will shed a lot of light on why array formulas were difficult in legacy Excel.

In the past, I could copy an array formula from Mike Girvin's **Ctrl+Shift+Enter** book and use it. I could even explain how the formula was working if you let me watch a few steps with the Evaluate Formula dialog box. Once, I even talked about array formulas for an entire hour at Tanja Kuhn's Trainer Days event in Lucerne, Switzerland. Eckhard Pfeiffer was in attendance and even gave me a nod that I was not completely wrong in my understanding. Now, I was speaking English to an audience of German speakers, so maybe I was wrong, and they were all too polite to say it.

But I never really understood array formulas until I watched Joe McDaid's 30-minute presentation on legacy array formulas in the fall of 2018. I occasionally learn one new tip from an Excel presentation, but every second of this 30-minute presentation was new to me. Afterward, I told Joe it was the best 30-minute investment I've ever made in my Excel life.

The new array formulas are so easy, it seems odd to put this 400-level material in this book. But it is near the back of the book, so a lot of people won't read it. But I hope you do.

A Quick Glossary

You need to understand the following terms for this chapter and Chapter 10:

- **Legacy Excel:** The Excel that you used through September 23, 2018.
- **Scalar:** A single value or a single cell.
- **Whilst:** How people in the UK and Australia say "while."

Legacy Excel Used Arrays Far More Often Than We Realized

In legacy Excel, you were using array calculations in circumstances such as these:

- When a name wasn't just a simple range
- When using a formula in conditional formatting
- When pressing **F9** whilst characters in the formula bar were highlighted

Even regular formulas in the grid exhibited array calculation behaviors.

The following sections describe five array calc behaviors:

1. Lifting
2. Pairwise Lifting
3. Broadcasting
4. Implicit Intersection
5. Array Truncation

Implicit Intersection occurs when a range is supplied to something that expects a scalar, and Excel selects the value in the same row or column as the formula.

In the next figure, if you ask for **LEN(A2:A7)** from row **2**, the answer will be the length of **A2**.

	A	B
1	Word	Length
2	heart	=LEN(A2:A7)
3	blip	
4	backbone	
5	bacon	
6	werewolf	
7	chunky	
8		

Figure 45 *Implicit Intersection causes the answer to be based on* **A2**.

If you copy that formula to row **4**, the answer will be the answer from row **4**.

	A	B
1	Word	Length
2	heart	5
3	blip	
4	backbone	8
5	bacon	
6	werewolf	
7	chunky	
8		

Figure 46 *Copy the formula to row* **4**, *and the answer is based on* **A4**.

Implicit Intersection also works sideways. If you ask for the **=LEN(A1:F1)** from column **D**, the answer will be the length of **D1**.

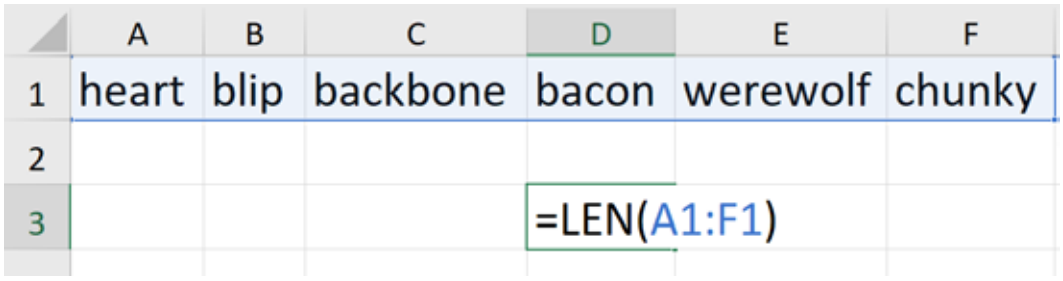


Figure 47 `LEN` *returns the answer based on* `D1`.

Bacon has a length of 5. Chunky has a length of 6. Copy the formula from column `D` to column `F`, and the pasted formula will return the answer from `F1`.

Implicit Intersection was working all the time in legacy Excel, and we simply don't realize it. I once gave out one of my collectible Excel Guru prizes for someone who showed me the following Excel formula. It was, honestly, the first time I had ever seen Implicit Intersection in the wild, and the person had no clue she was using it. The only reason that `=VLOOKUP(A$2:A$10,ProdTable,3,False)` works in `D2` is because Implicit Intersection is pointing to only `A2` out of `A2:A10`.

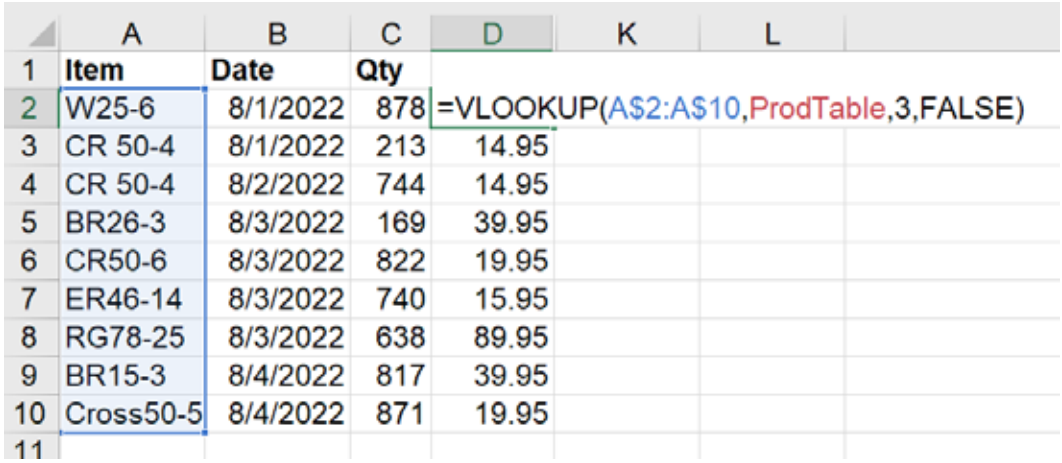


Figure 48 *You should not use* `VLOOKUP` *for every value in* `A`*, but Implicit Intersection let this crazy formula work.*

BREAKING IMPLICIT INTERSECTION

Implicit Intersection fails when you enter it in a cell that does not intersect the data. In the figure below, a formula pointing to `A1:F1` entered in column `G` will return a `#VALUE!` error.

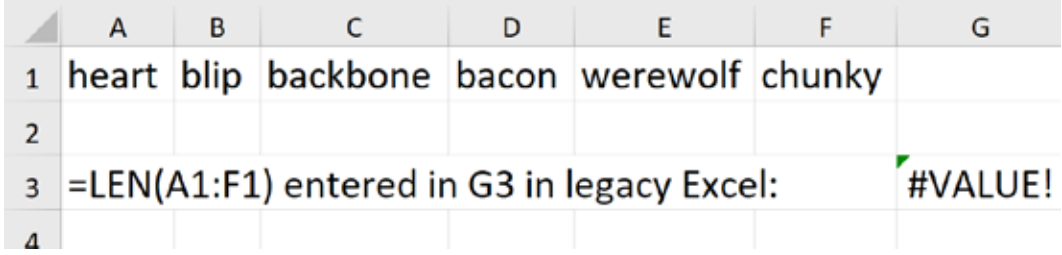


Figure 49 *With nothing intersecting the formula, a* `#VALUE!` *error results.*

Here's another thing that will break Implicit Intersection: In the image below, the formula `=LEN(A1:F3)` is entered in column `B`.

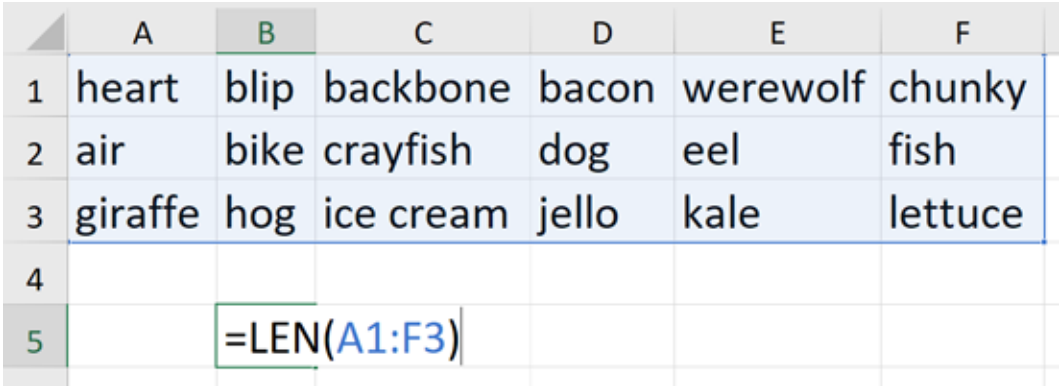


Figure 50 *More than one cell intersects with the formula.*

This formula clearly has an intersection with the range. But Excel does not know if you want the length of blip, bike, or hog, so you get a **#VALUE!** error.

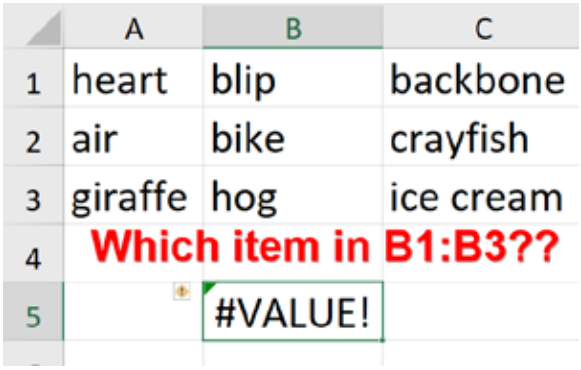


Figure 51 *Excel does not know which to use, so you get a* **#VALUE!** *error.*

In legacy Excel, why does **=SUM(B2:B6*C2:C6)** fail with a **#VALUE!** error? It is because the formula is entered in column **C**, and Implicit Intersection causes the **#VALUE!** error.

If you replace **SUM** with **SUMPRODUCT**, the answer works because **SUMPRODUCT** is designed to ignore Implicit Intersection.

Implicit Intersection is the bad actor that caused many formulas in legacy Excel to fail. You did not realize that Implicit Intersection was the culprit, but it frequently was the problem.

LIFTING WHEN A SCALAR IS EXPECTED BUT AN ARRAY IS PROVIDED

What happens when legacy Excel is expecting a single value, but you provide a range of values? Excel will pass each of the values in the array to the function and will return an array of the same dimensions as the array. This is called Lifting.

In the figure below, the **LEN** function is expecting only one value.

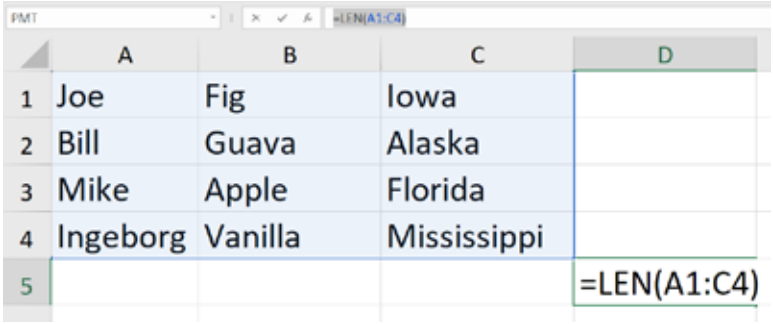


Figure 52 LEN *expects a scalar.*

When you pass a range of values to it, Excel imposes Lifting on that function and calculates 12 versions of the **LEN** function in memory. You can see the temporary results in the formula bar in the figure below.

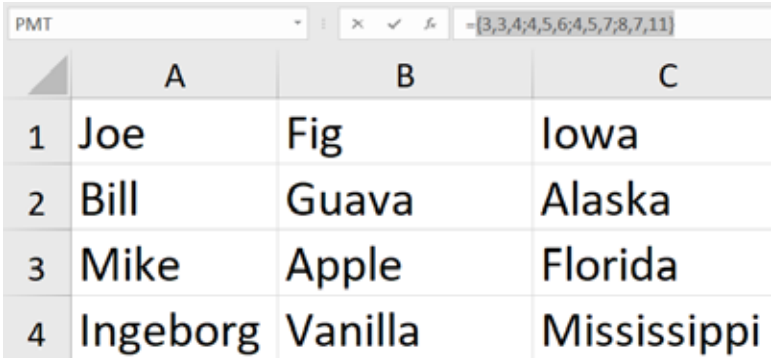


Figure 53 *The formula bar shows the 12 results.*

If you tried to use **=LEN(A1:C4)** in legacy Excel without pressing **Ctrl+Shift+Enter**, Excel would apply Lifting to the function and calculate 12 results, but the answer that appeared in the cell would be different, depending on where you entered the formula.

If you entered that formula anywhere in columns **A:C** or rows **1:4**, the Implicit Intersection rules would kick in, Excel would not be able to figure out which cell you wanted, and you would get a **#VALUE!** error.

If you entered that formula in **D5** or anywhere below or to the right, you would get the top-left answer, or **3**, because **LEN("JOE")** is **3**. This illustrates the concept of Array Truncation.

UNDERSTANDING ARRAY TRUNCATION

When there are not sufficient cells to output an array result, Excel pulls another trick out of its hat: Array Truncation. Even if the **LEN** function returns **{3,3,4;4,5,6;4,5,7;8,7,11}**, if you are entering the formula in one cell, you will see only the **3** because Excel has employed Array Truncation.

USING A WRAPPER FUNCTION IN LEGACY EXCEL

A common approach in legacy Excel was to pass the **LEN(A1:C4)** function to a wrapper function such as **SUM**. Again, without pressing **Ctrl+Shift+Enter** in legacy Excel, the formula **=SUM(LEN(A1:C4))** anywhere in columns **A:C** or rows **1:4** gets you a **#VALUE!** error. Entering the same formula in **D4** would have surprisingly, returned the correct result, **67**, without the use of **Ctrl+Shift+Enter**.

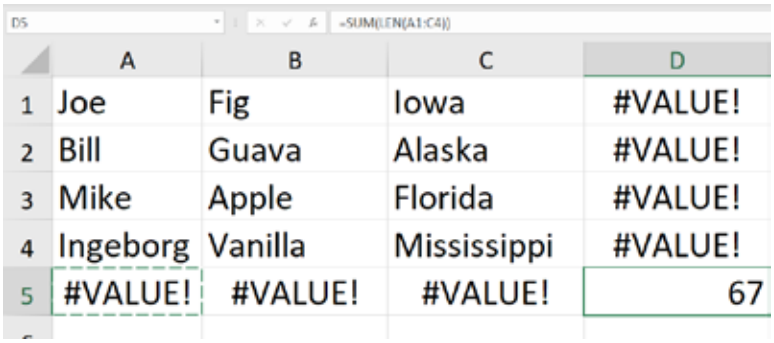


Figure 54 *The formula works only when it does not encounter implicit intersection.*

PREVENTING IMPLICIT INTERSECTION WITH CTRL+SHIFT+ENTER

In legacy Excel, pressing **Ctrl+Shift+Enter** for a single-cell formula prevents Implicit Intersection. This means that when someone pressed **Ctrl+Shift+Enter**, they were often making sure that the formula would not be subject to Implicit Intersection. I never knew this is why I was pressing **Ctrl+Shift+Enter**.

Ctrl+Shift+Enter also controls the size of the output. If you wanted to return a 4x5 array with the **LINEST** function or a 12x1 array with the **FREQUENCY** function, then selecting the appropriate size range, typing the formula, and pressing **Ctrl+Shift+Enter** would enter all of the array from the functions in those cells.

Five legacy functions in Excel were able to prevent Implicit Intersection: **SUMPRODUCT**, **MMULT**, **MINVERSE**, **ROWS**, and **COLUMNS**. When you flip through Mike Girvin's **Ctrl+Shift+Enter** book, any time he did not use **Ctrl+Shift+Enter**, he was likely using **SUMPRODUCT**, **MMULT**, or **ROWS**.

In legacy Excel, using **=SUM(LEN(A1:C4))** followed by **Ctrl+Shift+Enter** would yield the correct answer. You could enter this in any cell, including cells in columns **A:C** or rows **1:4**. I have used this formula in legacy Excel. I has mistakenly assumed you needed **Ctrl+Shift+Enter** probably because my first attempt to use the formula intersected with the cells, and it would not work without **Ctrl+Shift+Enter**.

Legacy Excel was very capable of Lifting a function and providing many results. The issue was that Implicit Intersection would often block the answer. If Implicit Intersection did not get in the way, then Array Truncation would reduce the answer to the first answer from the array.

Thanks to Dynamic Arrays, you no longer need to worry about Implicit Intersection screwing up the results. The formula **=SUM(LEN(A1:C4))** entered in any cell, without **Ctrl+Shift+Enter,** will return the correct result.

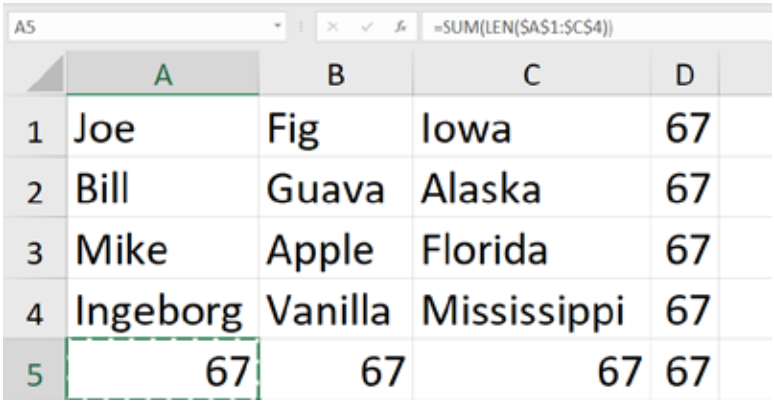


Figure 55 *With Dynamic Arrays, the formula works everywhere.*

FROM LIFTING TO PAIRWISE LIFTING

If Excel is expecting two scalars and you provide two arrays, Excel applies Pairwise Lifting to the arrays. For example, the result of the following is 14 in legacy Excel.

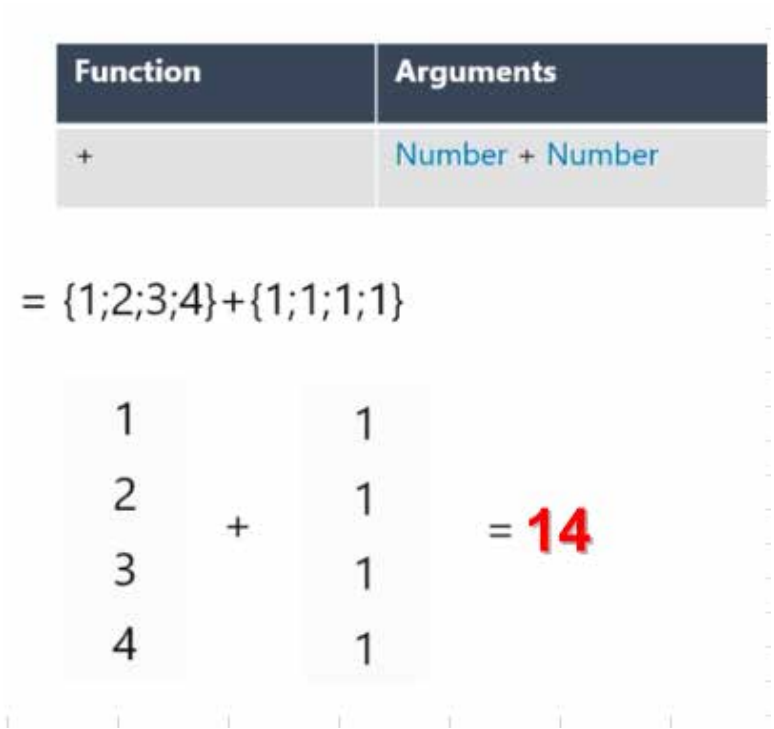


Figure 56 *Both arrays have the same number of rows, so you get an answer thanks to Pairwise Lifting.*

BROADCASTING MAKES ALL ARRAYS THE SAME SIZE

When Excel has to deal with two arrays that are of different sizes, Excel expands the arrays until they have the same size. With a "1 by array" or a "scalar," Excel copies the value. Otherwise, Excel fills the extra space in the arrays with `#N/A`. I had seen this behavior before, but I never new that it's called Broadcasting.

How to Read Excel Array Constants

An array constant consists of curly braces with values inside, separated by commas and semicolons. One of these symbols means next column, and the other one means next row. Which is which? Here is an easy way to remember it.

The semicolon appears very near the Enter or Return key on U.S. keyboards. Thus, you can remember that a semicolon means to press Return and go to the next row.

If a semicolon already means next row, then the only thing left for the comma is to mean next column. But if you need a memory trick, just remember that comma *and* column *both start with C.*

A Simple Broadcasting Example

Say that you have a 3-row-by-1-column array. Multiply it by a scalar or a single-cell array, and Excel will make copies of the **3** in the second array and fill the rest of the newly resized array with **3**s. When Excel does the multiplication for **=SUM({1;2;3}*3)**, you get **18**.

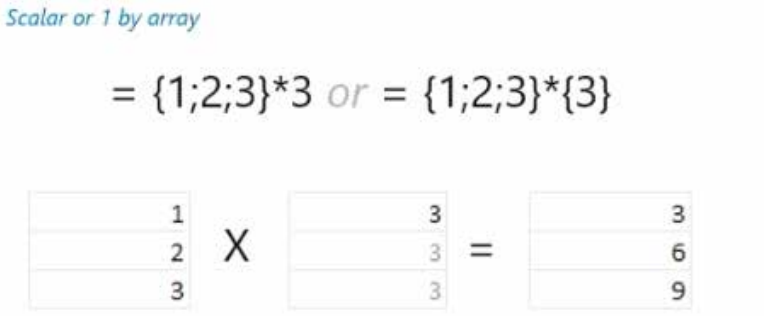


Figure 57 *Excel copies the* **3** *to the other cells in the larger array.*

But here is a counterintuitive example. If you multiply the **{1;2;3}** array by **{3;3}**, Excel does not know what to do for the third cell in the second array. (In this case, maybe you could argue that it should have been **3**, but couldn't it have been **{17;2}**?) The answer to **=SUM({1;2;3}*{3;3})** will have an intermediate result of **SUM(3,6,#N/A)**, and that **#N/A** error will cause the answer to be **#N/A**.

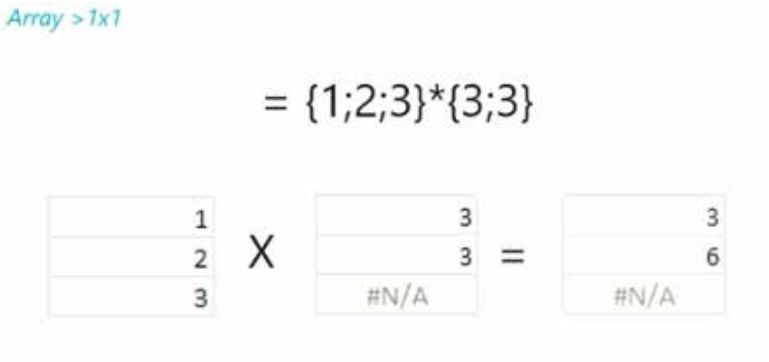


Figure 58 *Excel does not know how to fill the third cell in the second array.*

Say that you enter this formula in a single cell in legacy Excel: **=1+{1;1;1}+{1,1,1,1}+{1,1}+{1;1;1;1}**

Surprisingly, the answer is 5. Why is it 5?

Follow this logic: The widest part of the formula is the third term, **{1,1,1,1}**, so the resultant array has to be 4 cells wide. The tallest part of the formula is the second term, **{1;1;1}**, so the resultant array has to be 3 rows tall.

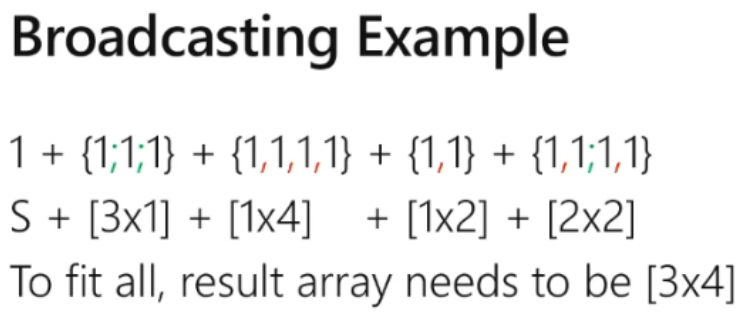


Figure 59 *Excel broadcasts all arrays to be 3x4.*

When Excel tries to resize everything to be 3x4 cells, you get this:

1	1	1	1		1	1	1	1		1	1	1	1		1	1	#	#		1	1	#	#
1	1	1	1	+	1	1	1	1	+	1	1	1	1	+	1	1	#	#	+	1	1	#	#
1	1	1	1		1	1	1	1		1	1	1	1		1	1	#	#		#	#	#	#

Figure 60 *The* **#N/A** *errors show up in the fourth and fifth arrays.*

Notice that each **#** sign in those results is really **#N/A**, but the column is not wide enough to show the whole error.

If you add everything together, you will end up with a 2x2 array of 5 and the rest of the array all **#N/A**.

5	5	#	#
5	5	#	#
#	#	#	#

Figure 61 *Only 4 of the 12 Broadcasting cells have answers.*

In legacy Excel, if you enter that formula without **Ctrl+Shift+Enter**, Array Truncation happens, and you get 5. If you enter that formula with **Ctrl+Shift+Enter**, Excel tries to do **5+5+5+5** plus a whole bunch of **#N/A**, and you get **#N/A**

HOW DO LIFTING, BROADCASTING, ARRAY TRUNCATION, AND IMPLICIT INTERSECTION AFFECT DYNAMIC ARRAYS?

As mentioned more than a few times already, Excel has seven new functions. **SORT**, **SORTBY**, **FILTER**, **UNIQUE**, **SEQUENCE**, and **RANDARRAY** all provide new functionality. That is great. The seventh function does not fit with the rest. **SINGLE** does something that was already possible in Excel, but it forces you to be explicit about the fact that you really mean to invoke Implicit Intersection. The inclusion of **SINGLE**, in conjunction with the discussion in this chapter, reveals a lot about the new Dynamic Arrays. In the past, array formulas worked only when you did some magic—such as using **Ctrl+Shift+Enter** or using **SUMPRODUCT** or **MMULT**—to turn off Implicit Intersection. With the release of Dynamic Arrays, Implicit Intersection is turned off all the time. If you specifically need Implicit Intersection, you can still use it, but you do it by using the **SINGLE** function.

There you have it: Implicit Intersection is why so many of your attempts to do array calculations failed in legacy Excel.

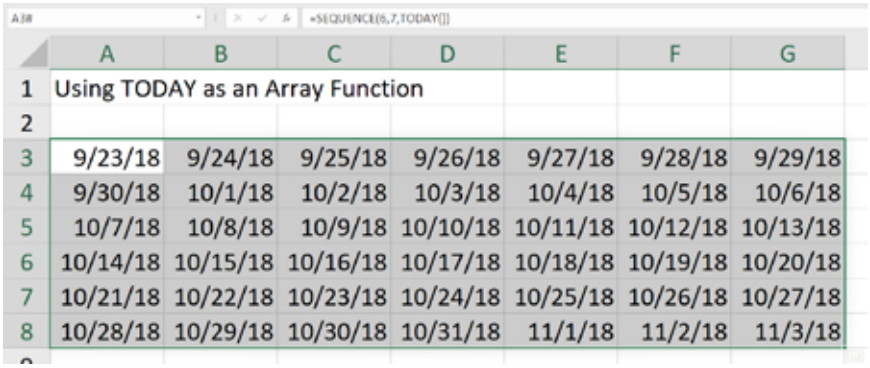
10 - Other Functions That Are Now Dynamic Arrays

The story announced at Ignite does not end at **SORT**, **SORTBY**, **FILTER**, **UNIQUE**, **SINGLE**, **SEQUENCE**, and **RANDARRAY**. At one point, I thought that the new Dynamic Arrays would make every Excel function into an array function. I thought that perhaps this chapter would be a hilarious chapter with 373 examples of making every single Excel function into an array function. But I am not sure that is possible. For example, what if you have a function that is not expecting a scalar? How are you going to make **NOW** or **TODAY** into array functions if the function is not expecting a scalar that you can turn into an array via Lifting?

The 24 examples in this chapter are designed to give you a starting point, to show you what is possible.

USING TODAY AND SEQUENCE FOR A CALENDAR

Say that you use **TODAY()** as the third argument with the **SEQUENCE** function. **=SEQUENCE(6,7,TODAY())** will generate 7 columns and 6 rows of a calendar.

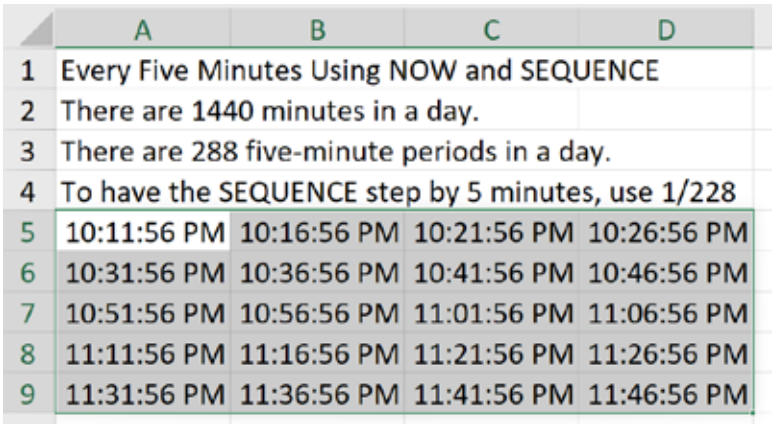


	A	B	C	D	E	F	G
1	Using TODAY as an Array Function						
2							
3	9/23/18	9/24/18	9/25/18	9/26/18	9/27/18	9/28/18	9/29/18
4	9/30/18	10/1/18	10/2/18	10/3/18	10/4/18	10/5/18	10/6/18
5	10/7/18	10/8/18	10/9/18	10/10/18	10/11/18	10/12/18	10/13/18
6	10/14/18	10/15/18	10/16/18	10/17/18	10/18/18	10/19/18	10/20/18
7	10/21/18	10/22/18	10/23/18	10/24/18	10/25/18	10/26/18	10/27/18
8	10/28/18	10/29/18	10/30/18	10/31/18	11/1/18	11/2/18	11/3/18

Figure 62 *Using* **SEQUENCE** *with* **TODAY**.

NOW AND SEQUENCE

You can combine **NOW** with **SEQUENCE**. Each minute is 1/1440 of a day. Therefore, to have the array formula return every 5 minutes, use 1/288 as the **step** argument in **SEQUENCE**.



	A	B	C	D
1	Every Five Minutes Using NOW and SEQUENCE			
2	There are 1440 minutes in a day.			
3	There are 288 five-minute periods in a day.			
4	To have the SEQUENCE step by 5 minutes, use 1/228			
5	10:11:56 PM	10:16:56 PM	10:21:56 PM	10:26:56 PM
6	10:31:56 PM	10:36:56 PM	10:41:56 PM	10:46:56 PM
7	10:51:56 PM	10:56:56 PM	11:01:56 PM	11:06:56 PM
8	11:11:56 PM	11:16:56 PM	11:21:56 PM	11:26:56 PM
9	11:31:56 PM	11:36:56 PM	11:41:56 PM	11:46:56 PM

Figure 63 *Each cell is 5 minutes later.*

GENERATING SEQUENTIAL LETTERS WITH CHAR, SEQUENCE, AND TEXTJOIN

The **CHAR** function returns a letter based on the ASCII code. **=CHAR(65)** is A, and **=CHAR(90)** is Z. You can use **SEQUENCE** inside **CHAR** to generate a sequence of letters, and then you can pass that array to **TEXTJOIN** to show the sequence of letters with a hyphen (or any other character) in between.

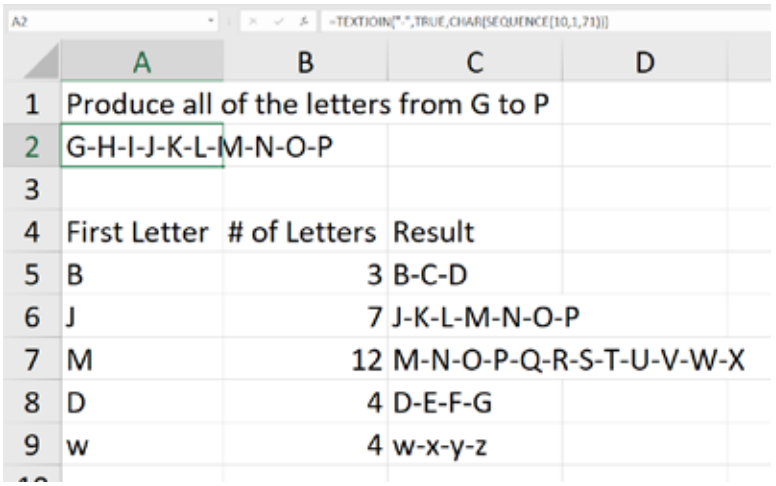


Figure 64 *Generating sequential letters.*

RETURNING THE *N* LARGEST ITEMS USING LARGE

Say that you want to return the *N* largest values from a range. In the following example, type any number in **G1**. The formula in **G2** will return the *N* largest numbers.

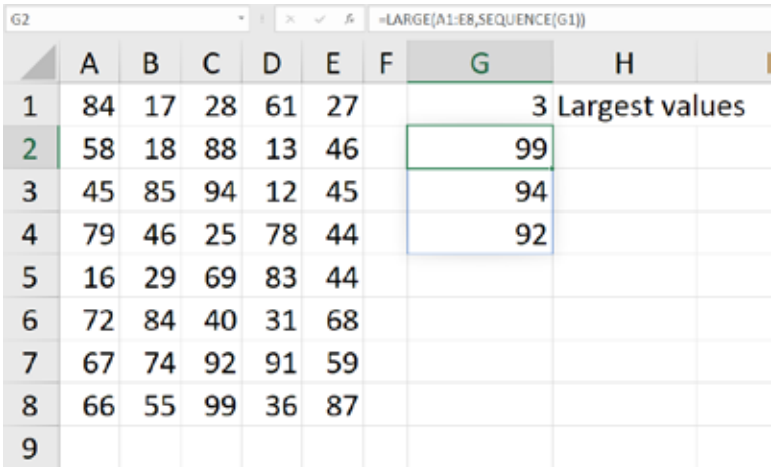


Figure 65 *Returning the three largest values from a range.*

RETURNING THE *N* SMALLEST ITEMS HORIZONTALLY

Imagine that you want to make a horizontal list of the *N* smallest items from a range. In the following example, the **TRANSPOSE** function turns the results to horizontal with **=TRANSPOSE(SMALL(A1:E8,SEQUENCE(G1)))**.

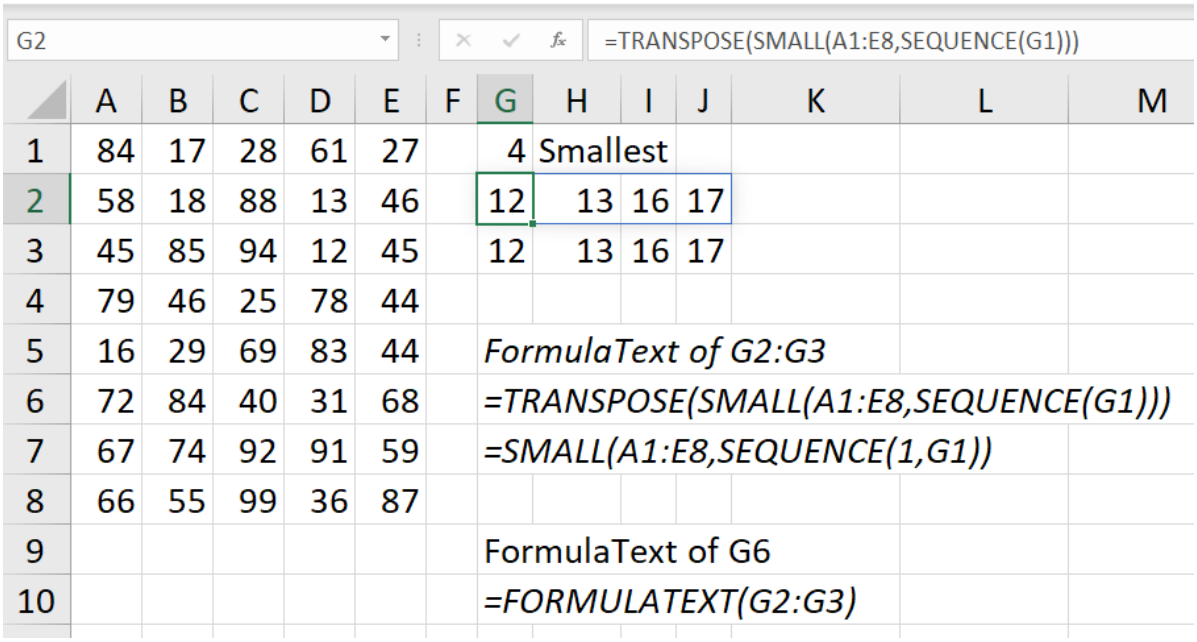


Figure 66 *It is possible to transpose while finding the smallest cells.*

TRANSPOSING WITH A SHORTER FORMULA

At ModelOff competitions, the prize money is sometimes awarded to the team with the shortest formula. Look again at Figure 66, above. The formula in G2 is essentially using **SEQUENCE(4)** to return a vertical sequence that is then made horizontal with **TRANSPOSE**.

You can do the same thing more expediently by using **SEQUENCE(1,G1)** to return a horizontal array—and there is no need to use **TRANSPOSE**. This formula, shown in **G3**, is much shorter than the formula in **G2**.

SHOWING FORMULAS FOR A RANGE WITH FORMULATEXT

I use **FORMULATEXT** frequently to document the formulas shown in a book. Figure 66, above, previous page shows how to use **=FORMULATEXT(G2:G3)** as a Dynamic Array formula.

CREATING A CROSSTAB REPORT WITH THREE FORMULAS

As the co-author of *Pivot Table Data Crunching*, I love a good pivot table. But Excel Project Manager Joe McDaid and Excel MVP Roger Govier both pointed out that the three formulas shown here simulate a pivot table and do not have to be refreshed.

To build the report, **=SORT(UNIQUE(C2:C392))** provides a vertical list of customers starting in **F6**. Then, **=TRANSPOSE(SORT(UNIQUE(A2:A392)))** provides a horizontal list of products starting in **G5**.

When you specify **F6#** and **G5#** in arguments of **SUMIFS**, Excel applies Broadcasting to the arrays to return a two-dimensional result: **=SUMIFS(D2:D392,C2:C392,F6#,A2:A392,G5#)**.

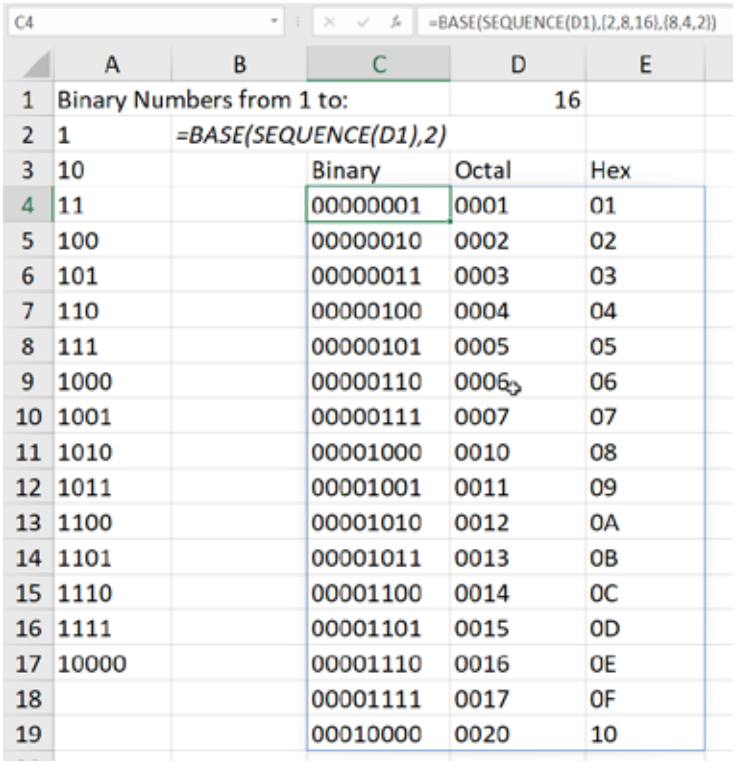
G6 =SUMIFS(D2:D392,C2:C392,F6#,A2:A392,G5#)

	A	B	C	D	E	F	G	H	I
1	**Product**	**Date**	**Customer**	**Revenue**	F6:	=SORT(UNIQUE(C2:C392))			
2	DEF	2/8/2017	ABC Stores	20610	G5:	=TRANSPOSE(SORT(UNIQUE(A2:A392)))			
3	DEF	2/23/2017	ABC Stores	8116	G6:	=SUMIFS(D2:D392,C2:C392,F6#,A2:A392,G5#)			
4	DEF	8/16/2017	ABC Stores	7032					
5	DEF	5/2/2018	ABC Stores	18290			ABC	DEF	XYZ
6	XYZ	1/6/2017	AT&T	2401		ABC Stores	0	54,048	0
7	XYZ	3/2/2017	AT&T	6765		AT&T	0	271,339	227,598
8	DEF	3/9/2017	AT&T	21357		BankUnited	406,326	0	0
9	DEF	3/13/2017	AT&T	5370		CUNA Insurance	0	51,240	0
10	XYZ	4/4/2017	AT&T	4124		Exxon	294,138	185,286	224,935
11	DEF	4/7/2017	AT&T	12318		Ford	0	0	57,516
12	DEF	4/19/2017	AT&T	10195		General Electric	247,092	321,759	0
13	XYZ	4/26/2017	AT&T	13412		General Motors	280,967	233,435	235,761
14	DEF	5/16/2017	AT&T	20020		Lowe's	0	31,369	0

Figure 67 *Three formulas replace a pivot table.*

DISPLAYING NUMBERS AS BINARY, OCTAL, OR HEX BY USING BASE

It is possible to convert a series of decimal numbers to binary, octal, or hexadecimal by using the **BASE** function and the **SEQUENCE** array. The first example below, in **A2**, delivers an array of the binary numbers from 1 to 16 with **=BASE(SEQUENCE(D1),2)**. In this case, **SEQUENCE** delivers the numbers 1 through 16, and the radix is 2 to indicate binary numbers. The more advanced function in **C4** uses the **{2,8,16}** array constant as the radix and **{8,4,2}** as the number of digits. This delivers binary in column **C**, octal in column **D**, and hexadecimal in column **E**. This is an example of Pairwise Lifting.



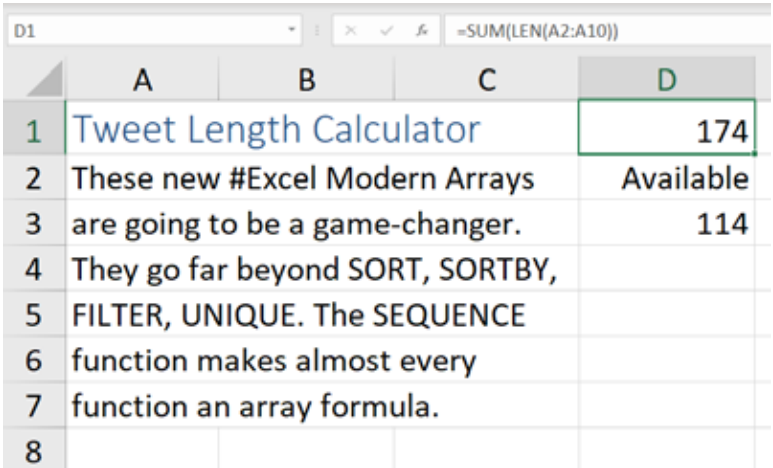
C4 =BASE(SEQUENCE(D1),{2,8,16},{8,4,2})

	A	B	C	D	E
1	Binary Numbers from 1 to:			16	
2	1	=BASE(SEQUENCE(D1),2)			
3	10		Binary	Octal	Hex
4	11		00000001	0001	01
5	100		00000010	0002	02
6	101		00000011	0003	03
7	110		00000100	0004	04
8	111		00000101	0005	05
9	1000		00000110	0006	06
10	1001		00000111	0007	07
11	1010		00001000	0010	08
12	1011		00001001	0011	09
13	1100		00001010	0012	0A
14	1101		00001011	0013	0B
15	1110		00001100	0014	0C
16	1111		00001101	0015	0D
17	10000		00001110	0016	0E
18			00001111	0017	0F
19			00010000	0020	10

Figure 68 *Generating binary, octal, and hex numbers.*

SUMMING THE LENGTHS OF MANY CELLS

You can find the sum of the lengths of several text cells by using **=SUM(LEN(A2:A10))**. Using such a formula no longer requires **Ctrl+Shift+Enter**.



D1 =SUM(LEN(A2:A10))

	A	B	C	D
1	Tweet Length Calculator			174
2	These new #Excel Modern Arrays			Available
3	are going to be a game-changer.			114
4	They go far beyond SORT, SORTBY,			
5	FILTER, UNIQUE. The SEQUENCE			
6	function makes almost every			
7	function an array formula.			
8				

Figure 69 *Checking the length of your tweets.*

USING A FORMULA TO CONVERT TEXT TO COLUMNS

The formula shown below in **B2** is based on a formula from Rick Rothstein at the MrExcel.com Message Board. The formula uses **TRIM**, **MID**, **SUBSTITUTE**, and **REPT**. You can add a **SEQUENCE** function to return all words from a single formula — and you can see that even this formula becomes simpler with the use of the **SEQUENCE** function.

	A	B	C	D	E	F
1	Split a sentence with SUBSTITUTE	Split	a	sentence	with	SUBSTITUTE
2	=TRIM(MID(SUBSTITUTE($A1," ",REPT(" ",99)),SEQUENCE(1,A4)*99-98,99))					
4	5					
5	=COUNT(UNIQUE(FIND(" ",A1,SEQUENCE(LEN(A1)-3)))+1					

Figure 70 *Using* **SEQUENCE(1,5)** *to parse with a single formula.*

SUMMING ALL VLOOKUPS

How do you calculate a number of **VLOOKUP**s and then sum them? In legacy Excel, you could only do this with the ancient **LOOKUP** function. In fact, it was one of only two reasons that Excel tricksters would reach for the **LOOKUP** function. With Dynamic Arrays, you can instead use **VLOOKUP**, and you will likely use the old **LOOKUP** function only when you have the lookup vector and results vector in opposite orientations. The single formula in **J9** does all the **VLOOKUP** calculations and sums them: **=SUM(VLOOKUP(H2:H26,J2:K6,2))**.

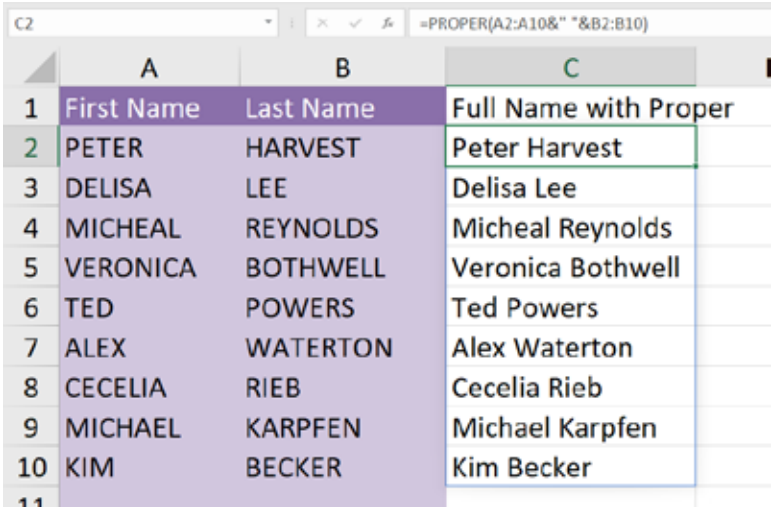
=SUM(VLOOKUP(H2:H26,J2:K6,2))

	F	G	H	I	J	K
	Invoice	Rep	Amount		Sale	Bonus
	1001	Carole	12835		0	0
	1002	Andy	19634		5000	5
	1003	Carole	898		10000	12
	1004	Hector	7747		20000	50
	1005	Andy	6239		25000	100
	1006	Gary	19867			
	1007	Andy	27537		Total Bonus	
	1008	Dale	679		652	
	009	Flo	24240			
	1010	Kevin	25314			
	1011	Larry	12375			
	1012	Dale	27502			
	1013	Flo	7822			
	1014	Flo	4356			
	1015	Larry	8980			
	1016	Dale	29937			
	1017	Kevin	22942			
	1018	Gary	5695			
	1019	Jill	21025			
	1020	Hector	9686			
	1021	Dale	11289			
	1022	Ed	4248			

Figure 71 *Calculating all* **VLOOKUP**s *and summing them.*

FINDING THE PROPER CASE OF ALL NAMES WITH ONE FORMULA

This example uses Pairwise Lifting, which is described in Chapter 9. The **PROPER** function receives a range of first names, a single scalar of a space, and a range of last names identical to the range of first names. If you used the formula **=PROPER(A2:A10&" "&B2:B10)**, the result is a 9x1 array containing the names.

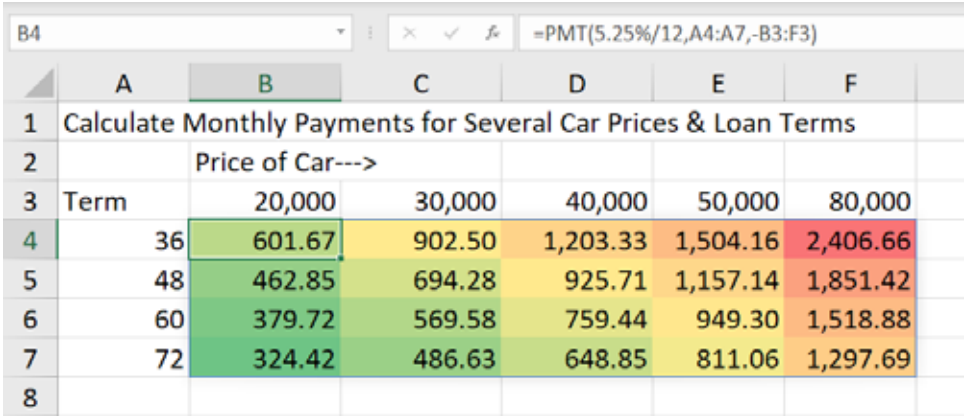


	A	B	C
1	First Name	Last Name	Full Name with Proper
2	PETER	HARVEST	Peter Harvest
3	DELISA	LEE	Delisa Lee
4	MICHEAL	REYNOLDS	Micheal Reynolds
5	VERONICA	BOTHWELL	Veronica Bothwell
6	TED	POWERS	Ted Powers
7	ALEX	WATERTON	Alex Waterton
8	CECELIA	RIEB	Cecelia Rieb
9	MICHAEL	KARPFEN	Michael Karpfen
10	KIM	BECKER	Kim Becker

Figure 72 *Concatenating and using* **PROPER**.

REPLACING A WHAT-IF DATA TABLE WITH ONE FORMULA

This example uses Broadcasting, as described in Chapter 9, to calculate a number of monthly car payments using the **PMT** function. The interest rate payment is a single value. The **term** argument is expecting a scalar, but the formula passes a 4x1 array. The present value is passing a scalar, but the formula passes a 1x5 array. Through the magic of Broadcasting, Excel knows that is has to generate a 4 row x 5 column array and pass those 20 calculations to **PMT**. I wondered if the conditional formatting color scales would work on the formula results, and as you can see here, they do!

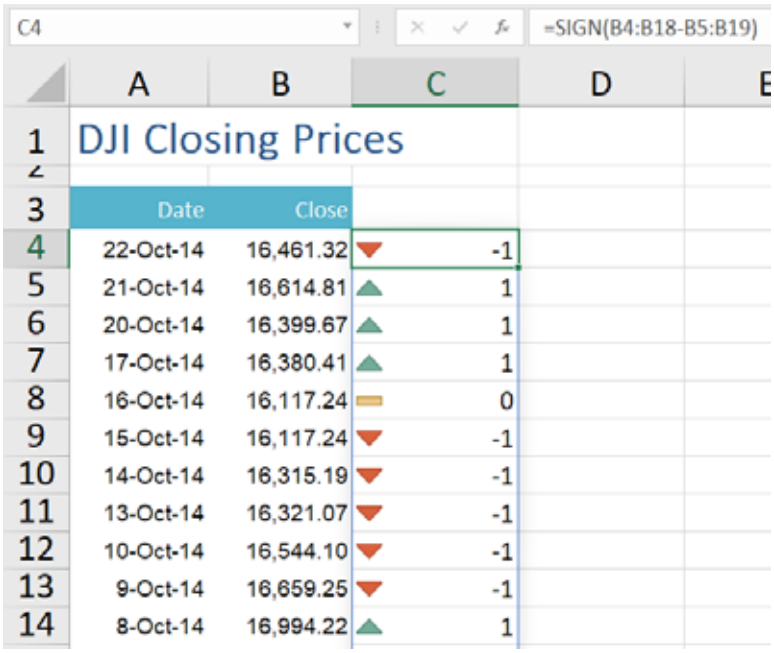


	A	B	C	D	E	F
1	Calculate Monthly Payments for Several Car Prices & Loan Terms					
2		Price of Car--->				
3	Term	20,000	30,000	40,000	50,000	80,000
4	36	601.67	902.50	1,203.33	1,504.16	2,406.66
5	48	462.85	694.28	925.71	1,157.14	1,851.42
6	60	379.72	569.58	759.44	949.30	1,518.88
7	72	324.42	486.63	648.85	811.06	1,297.69
8						

Figure 73 *A single Dynamic Array formula replaces a data table.*

APPLYING UP/FLAT/DOWN ICONS BY USING THE SIGN FUNCTION

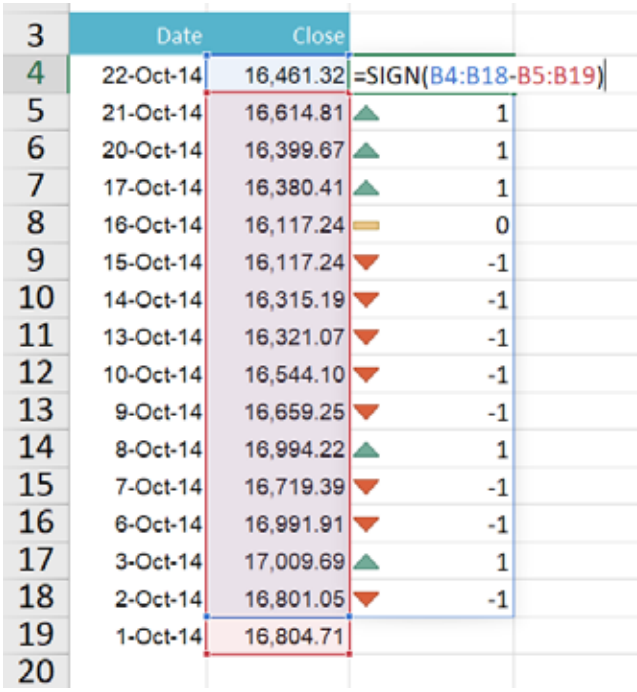
Here's another conditional formatting example: Whenever I have to use the "three triangles" icon set (which is clearly two triangles and a rectangle...but I digress), I add a range of **SIGN** functions that subtract today from yesterday. The example below uses Pairwise Lifting (as described in Chapter 9) on two arrays that are offset one from the other. The formula in **C4** is **=SIGN(B4:B18-B5:B19)**.



	A	B	C	D
1	DJI Closing Prices			
3	Date	Close		
4	22-Oct-14	16,461.32	-1	
5	21-Oct-14	16,614.81	1	
6	20-Oct-14	16,399.67	1	
7	17-Oct-14	16,380.41	1	
8	16-Oct-14	16,117.24	0	
9	15-Oct-14	16,117.24	-1	
10	14-Oct-14	16,315.19	-1	
11	13-Oct-14	16,321.07	-1	
12	10-Oct-14	16,544.10	-1	
13	9-Oct-14	16,659.25	-1	
14	8-Oct-14	16,994.22	1	

Figure 74 *Using* **SIGN** *to find the difference between two arrays.*

If you did not skip Chapter 9, you will remember that Pairwise Lifting involves two arrays that have the same number of rows. In this example, both of the arrays are in column **B**, but one starts in **B4** (for the value from today) and the other one starts in **B5** (for the value from yesterday). The figure below shows the **C4** formula in edit mode so you can see the two arrays.



3	Date	Close		
4	22-Oct-14	16,461.32	=SIGN(B4:B18-B5:B19)	
5	21-Oct-14	16,614.81	1	
6	20-Oct-14	16,399.67	1	
7	17-Oct-14	16,380.41	1	
8	16-Oct-14	16,117.24	0	
9	15-Oct-14	16,117.24	-1	
10	14-Oct-14	16,315.19	-1	
11	13-Oct-14	16,321.07	-1	
12	10-Oct-14	16,544.10	-1	
13	9-Oct-14	16,659.25	-1	
14	8-Oct-14	16,994.22	1	
15	7-Oct-14	16,719.39	-1	
16	6-Oct-14	16,991.91	-1	
17	3-Oct-14	17,009.69	1	
18	2-Oct-14	16,801.05	-1	
19	1-Oct-14	16,804.71		
20				

Figure 75 *Subtracting the previous day from the current day.*

USING THE SPILLED RANGE OPERATOR TO POINT TO AN ARRAY

This example is an anagram generator. An anagram, as you might remember from your school days, is a word or phrase rearranged to form another word or phrase. In the next image, for example, the phrase "EXCEL IS COOL " is rearranged to be " SOX COLLCIEE ." Using random sequences is not the most efficient way to create anagrams, but it can help you find something close. For example, given the result shown below, I can mentally rearrange it and suggest to PowerPivotPro.com's Rob Collie that perhaps he could name his next dog SOXCE COLLIE just to have an anagram of EXCEL IS COOL.

A1: EXCEL IS COOL

	A	B	C	D	E	F	G	H
1	EXCEL IS COOL	14						
2	E	13			Sox Collciee			
3	X	4		S				
4	C	7		O	A1: Input Cell			
5	E	14		X	A2: =MID(A1,SEQUENCE(B1,1)			
6	L	10			B1: =LEN(A1)			
7		1			B2: =SORTBY(SEQUENCE(B1),RANDARRAY(B1))			
8	I	12		C	D2: =SORTBY(A2#,B2#)			
9	S	2		O	E2: =PROPER(CONCAT(D2#))			
10		5		L				
11	C	11		L				
12	O	8		C				
13	O	3		I				
14	L	9		E				
15		6		E				

Figure 76 *An anagram generator using Dynamic Arrays.*

The worksheet in this case is flexible because you can type any length of the phrase in **A1** to generate the anagram.

Because I knew that I wanted to be able to potentially stop calculating and type new numbers in column **B**, I built this example in several steps even though I knew it could have been far more efficient. But I am glad I did it in several steps because it provides an example of the new spilled formula operator and the array reference notation.

In cell **D2**, I had to sort the array from **A2** by the array in **B2**. As the worksheet author, I don't know how many characters you will be typing in **A1**, so I don't know how many items will be in the **A2** or **B2** arrays.

I actually dreamt that I would have to be using **OFFSET(A2,0,0,LEN(A1),1)** or **A2:INDEX(A2:A999,LEN(A1)**. But neither one is necessary because of the new spilled range operator. When you refer to **A2#** in a formula, you are saying that you want to refer to the array starting in **A2**. This is an **<AngelsSing>awesome</AngelsSing>** new part of Dynamic Array formulas.

Let's walk through creating the five formulas in this example:

1. Type a phrase in **A1**.
2. Cell **B1** calculates **LEN(A1)**. This is a helper cell and is reused three times in the following formulas.

3. Cell **A2** applies Lifting with the **MID** function to return an array of each character in the text. The formula is **=MID(A1,SEQUENCE(B1),1)**.
4. Cell **B2** generates random integers that do not repeat by using **=SORTBY(SEQUENCE(B1),RANDARRAY(B1))**.
5. Cell **D2** sorts the letters in **A2#** by the numbers in **B2#** with **=SORTBY(A2#,B2#)**.
6. Cell **E2** uses **PROPER** and **CONCAT** with the **D2#** array to generate the anagram. This is the formula: **=PROPER(CONCAT(D2#))**.

USING AN ARRAY REFERENCE AS PART OF A REFERENCE

Excel MVP Wyn Hopkins of AccessAnalytic.com.au discovered that you can use a reference such as **A2#** as the first part of a cell reference. Say that you have a dynamic array starting in **A2**. Other array formulas in **B:C** will shrink and grow with **A2#**, but they are not part of **A2#**.

Say that you want to refer to **A2:Bn**, where **n** is however many rows are in the **A2#** array, you can use **A2#:C2**, and the reference will expand or contract.

F3 =VLOOKUP(E3,A3#:C3,3,FALSE)

	A	B	C	D	E	F	G	H
1			5					
2	Month	Rate	Note					
3	1	0.03	Normal		4	Super Bonus		
4	2	0.02	Normal					
5	3	0.04	Normal	A3:	=SEQUENCE(C1)			
6	4	0.05	Super Bonus	B3:	=VLOOKUP(A3#,RateTable,2,0)			
7	5	0.01	Normal	C3:	=IF(B3#=0.05,"Super Bonus","Normal")			
8				F3:	=VLOOKUP(E3,A3#:C3,3,FALSE)			
9								

Figure 77 **A3#:C3** *will refer to* **A3:C7** *for now.*

GENERATING A SERIES OF MONTHS

This example shows how to generate months, month names, or ends of months. In the figure below, **A5** generates a series of month starting dates for the year 2019 with **=DATE(2019,SEQUENCE(12),1)**. To generate month names, you can either refer to **A6# =TEXT(A6#,"MMMM")**, as in **B6**, or perform the calculation in a single column with **=TEXT(DATE(2019,SEQUENCE(12),1),"MMM")**, as in **D6**. To find the ends of months, use **=EOMONTH(DATE(2019,SEQUENCE(12),0)**, as in **C6**.

	A	B	C	D	E
1		A4: =DATE(2019,SEQUENCE(12),1)			
2		B4: =TEXT(A5#,"MMMM")			
3		C4: =EOMONTH(DATE(2019,SEQUENCE(12),1),1)			
4		D4: =TEXT(DATE(2019,SEQUENCE(12),1),"MMM")			
5	1/1/2019	January	2/28/2019	Jan	
6	2/1/2019	February	3/31/2019	Feb	
7	3/1/2019	March	4/30/2019	Mar	
8	4/1/2019	April	5/31/2019	Apr	
9	5/1/2019	May	6/30/2019	May	
10	6/1/2019	June	7/31/2019	Jun	
11	7/1/2019	July	8/31/2019	Jul	
12	8/1/2019	August	9/30/2019	Aug	
13	9/1/2019	September	10/31/2019	Sep	
14	10/1/2019	October	11/30/2019	Oct	
15	11/1/2019	November	12/31/2019	Nov	
16	12/1/2019	December	1/31/2020	Dec	

Figure 78 *Generating 12 months as dates or text.*

Oddly, the formula **EOMONTH(A6#,0)** does not work but returns a **#VALUE** error. This is a common problem with any of the functions that were originally in the Analysis ToolPak: Formulas containing those functions tend to have problems with arrays. The Excel team could fix this, but then some backward compatibility would fail.

FORECASTING WITH AN ARRAY

This example shows how to generate a forecast by using **FORECAST.ETS** or **FORECAST**. In **=FORECAST.ETS(A146#,$B$2:$B$145,$A$2#,1,1)**, the period is provided by a **SEQUENCE(12,1,146)** function in **A146**. The actual sales are the range **B2:B145**. The array of known period numbers points to **SEQUENCE(145)** in **A2#**.

	A	B	C	D	E	F
1	Period	Actuals	Forecast			
144	143	$502,721				
145	144	$717,589				
146	145		=FORECAST.ETS(A146#,$B$2:$B$145,$A$2#,1,1)			
147	146		$319,786			
148	147		$384,513			
149	148		$379,924			
150	149		$389,482			
151	150		$377,043			
152	151		$383,433			
153	152		$397,995			
154	153		$381,759			
155	154		$422,830			
156	155		$526,939			
157	156		$685,556			

Figure 79 *Generating 12 months of a forecast.*

FORECASTING 12 MONTHS BY 5 YEARS

The previous example calculates an array with 12 monthly cells of **FORECAST.ETS**. This example wraps that array into a **SUM** function to return the forecast for all of 2019. This step works even if the formula is a bit convoluted.

=SEQUENCE(5,1,2019) generates 2019 through 2023 in column **D**.

The formula in **E2** is:

```
=SUM(
  FORECAST.ETS(
  EOMONTH(DATE(D2,SEQUENCE(12),1),0),
  $B$2:$B$145,
  $A$2#,1,1))
```

It might be easier to envision this formula in the orange text box in the figure below.

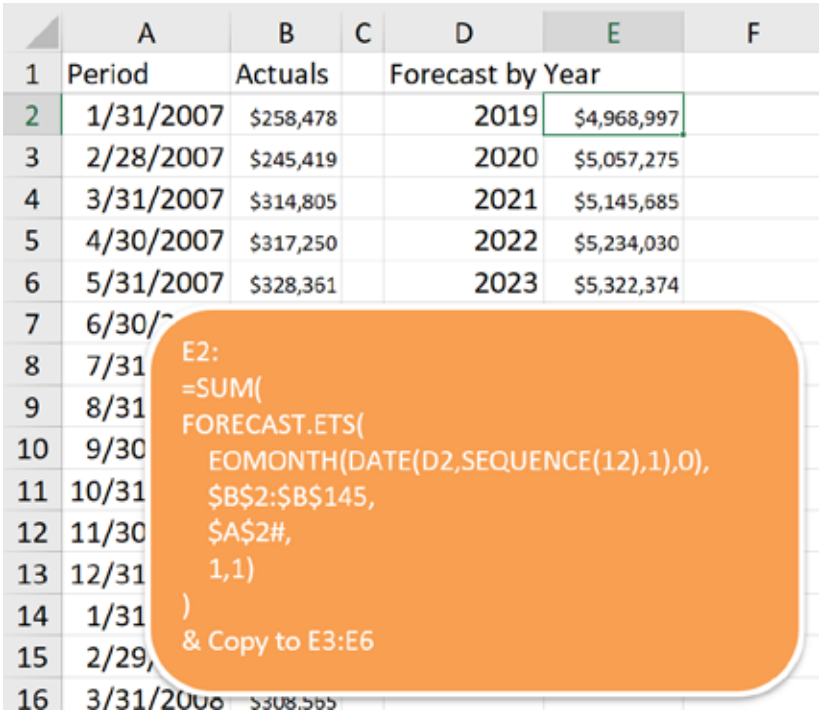


Figure 80 *Creating a formula by year.*

Inside **FORECAST.ETS**, you have to provide the period for which you want the forecast. The function is expecting a scalar, but instead you pass it a series of 12 month-ending dates by using **SEQUENCE** inside **EOMONTH**. For the known **Y**s, you use the actual sales in **B2:B145**. For the known **X**s, you use the dates in **A2#**.

When the formula is working, you copy **E2** and paste to **E3:E6**.

Why does this have to be five separate formulas? Couldn't you replace the **D2** reference to the year with a **SEQUENCE(1,5,2019)** array formula? The answer is no. With two vertical arrays in the same function, Excel would try to do Pairwise Lifting. But the arrays are not the same size, so you would get an error.

TRANSPOSING ONE ARRAY TO PREVENT PAIRWISE LIFTING

One strategy to prevent Pairwise Lifting is to make the years a horizontal array. The next figure shows the years as **=SEQUENCE(1,5,2019)** in **E1**. One array formula in **E3** returns 12 rows of months and 5 columns of years:

```
=FORECAST.ETS(
  EOMONTH(
    DATE(E1#,SEQUENCE(12),1),0),
    $B$2:$B$145,
    $A$2#,1,1)
```

Individual **SUM** functions in **E2:I2** sum the forecast for each year. The five-year forecast sum in **J2** is $25,728,361.

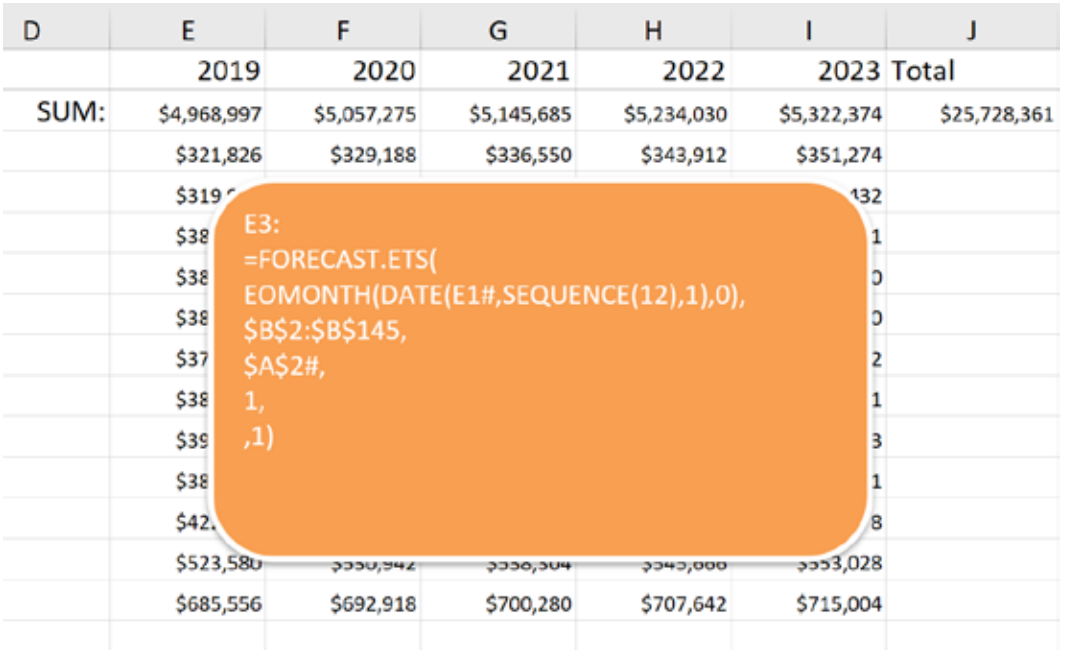


Figure 81 *If you run the months sideways, the formula works.*

FORECASTING ALL FIVE YEARS IN ONE FORMULA

If you pass the **FORECAST.ETS** formula into the **SUM** function, you can duplicate the $25,728,361 answer, as shown below.

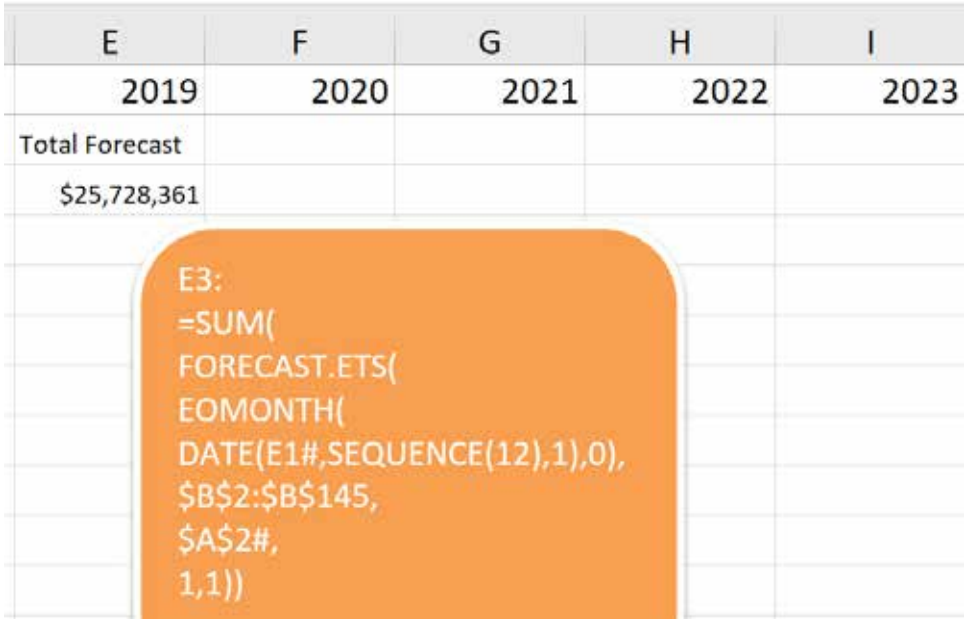


Figure 82 *One formula does all of the forecasts and sums them.*

COMBINING ARRAY FORMULAS TO SIMPLIFY CUBE FORMULAS

This example is a fairly long case study about cube formulas. Recently, while I was updating the 6th edition of *Pivot Table Data Crunching* for Microsoft Press, I wanted to expand on Mike Alexander's chapter on OLAP tools to show how to convert a pivot table to cube formulas.

It turns out that this is much harder than expected. The first steps are easy, though:

1. Select one cell in your data worksheet.
2. Select **Insert**, **Pivot Table**.
3. In the Create PivotTable dialog, choose **Add This Data to the Data Model**.

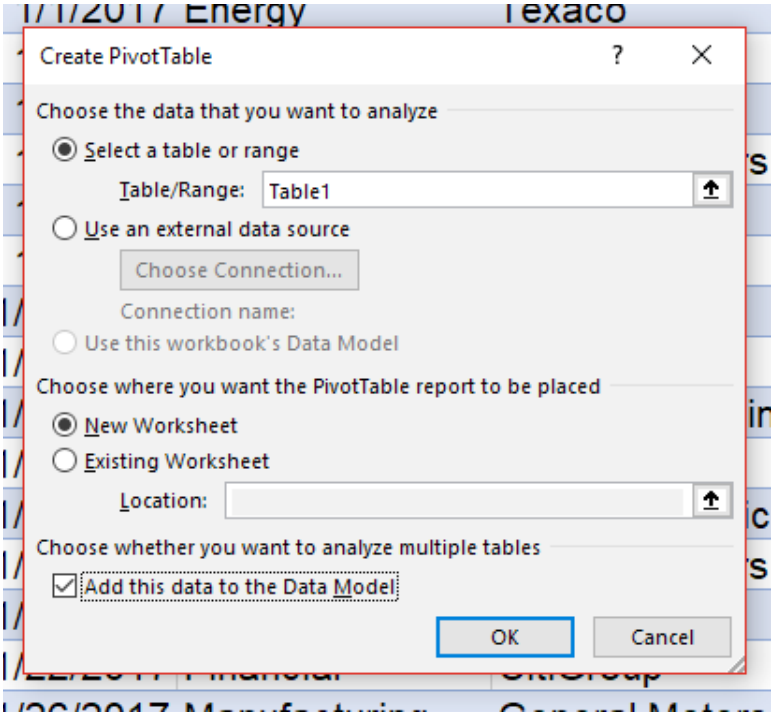
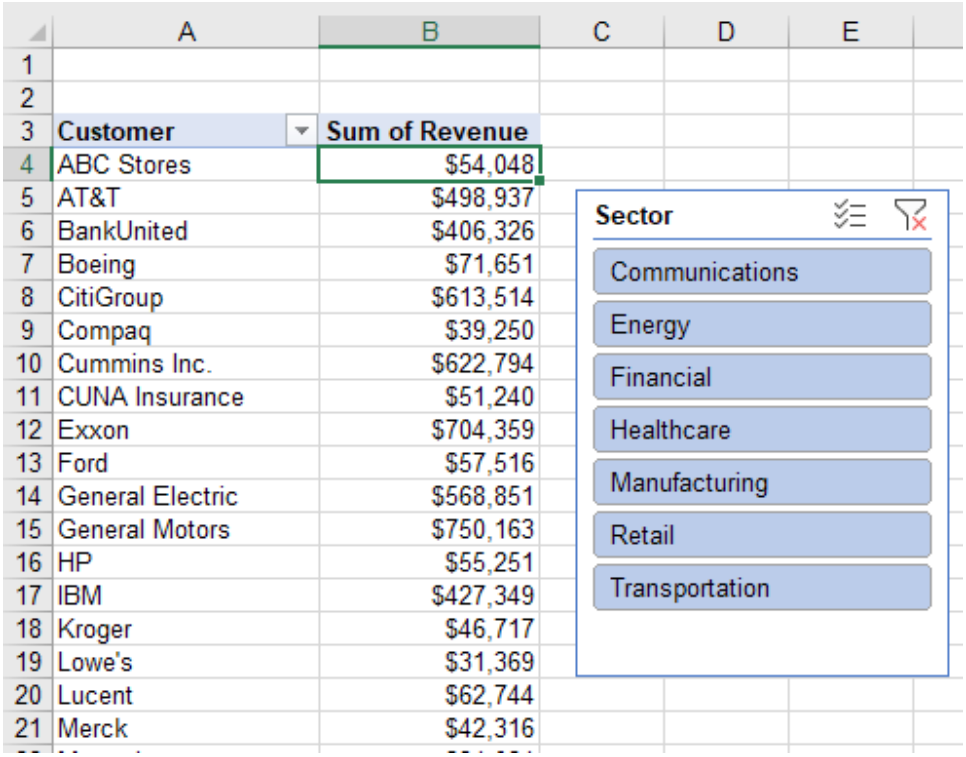


Figure 83 *Creating a pivot table using the data model.*

4. Build a pivot table with Customer in Rows, Revenue in Values, and Sector as a slicer.



	A	B	C	D	E
1					
2					
3	Customer	Sum of Revenue			
4	ABC Stores	$54,048			
5	AT&T	$498,937			
6	BankUnited	$406,326			
7	Boeing	$71,651			
8	CitiGroup	$613,514			
9	Compaq	$39,250			
10	Cummins Inc.	$622,794			
11	CUNA Insurance	$51,240			
12	Exxon	$704,359			
13	Ford	$57,516			
14	General Electric	$568,851			
15	General Motors	$750,163			
16	HP	$55,251			
17	IBM	$427,349			
18	Kroger	$46,717			
19	Lowe's	$31,369			
20	Lucent	$62,744			
21	Merck	$42,316			

Figure 84 *Building a simple pivot table.*

5. On the Analyze tab, choose **OLAP Tools**, **Convert to Formulas**.

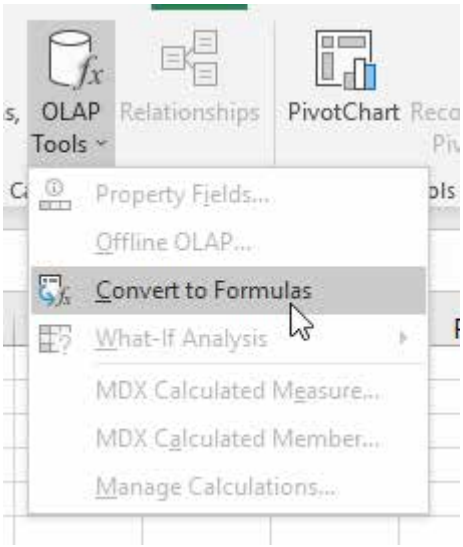


Figure 85 *Converting the pivot table to formulas.*

The formulas are shown in the next figure. The Sum of Revenue and Revenue numbers are great, but the crazy way that Excel creates the customer names is not flexible. If more customers are added later, they won't be included in the report.

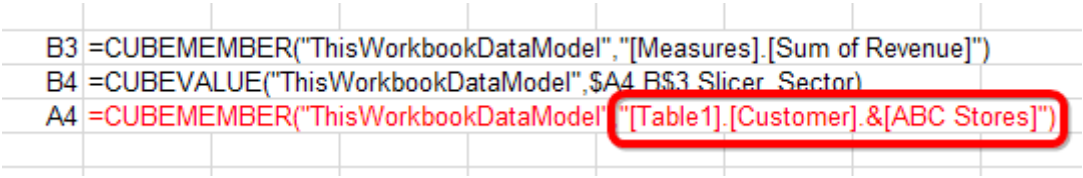


Figure 86 *Convert to Formulas is not flexible.*

The other problem is that the list of customers does not collapse as the slicer changes. The numbers are updated, but the customers do not disappear.

6	BankUnited	
7	Boeing	71651
8	CitiGroup	
9	Compaq	39250
10	Cummins Inc.	622794
11	CUNA Insurance	
12	Exxon	
13	Ford	57516
14	General Electric	568851
15	General Motors	750163
16	HP	55251
17	IBM	427349
18	Kroger	
19	Lowe's	
20	Lucent	
21	Merck	
22	Motorola	31021

Figure 87 *When you choose from the slicer, customers are not filtered out.*

Instead of using Convert to Formulas, you can use the **CUBESET** and **CUBERANKEDMEMBER** functions, as described by Rob Collie at PowerPivotPro.com.

Even with Rob's technique, the list of customers will not collapse in response to the slicer.

	A	B	C
1	Definition of Revenue		
2	Sum of Revenue	=CUBEMEMBER("ThisWorkbookDataModel","[Measures].[Sum of Revenue]")	
3	Complete Set of Customers		
4	All Customers	=CUBESET("ThisWorkbookDataModel", "[Range].[Customer].children", "All Customers",2,"[Measures].[Sum of Revenue]")	
5	27	=CUBESETCOUNT(A4)	
6	**Customer**	**Revenue**	
7	Wal-Mart		Array in A7#
8	General Motors	$750,163	=CUBERANKEDMEMBER("ThisWorkbookDataModel",$A$4,SEQUENCE(A5))
9	Exxon		
10	Cummins Inc.	$622,794	Formula in B7
11	CitiGroup		=IFNA(CUBEVALUE("ThisWorkbookDataModel",A7,A$2,Slicer_Sector),"")
12	General Electric	$568,851	
13	AT&T		
14	IBM	$427,349	
15	BankUnited		
16	Southwest Airlines		
17	SBC Communications		
18	Boeing	$71,651	
19	Lucent		
20	P&G	$60,299	
21	State Farm		
22	Ford	$57,516	
23	HP	$55,251	
24	ABC Stores		
25	CUNA Insurance		
26	Phillip Morris	$50,030	
27	Kroger		
28	Merck		
29	Compaq	$39,250	
30	Sears		
31	Texaco		
32	Lowe's		
33	Motorola	$31,021	

Figure 88 *Using* **CUBESET** *does not solve the problem.*

I encountered this problem at Excelapalooza. Several people there were quite confident they could write some MDX to solve the problem. But so far, the only result is a convoluted one from former Excel project manager Dany Hoter. I appreciate Dany's time, but the simple fact is that Dynamic Array formulas now provide a quick and easy way to do this.

One solution is to take those ugly formula results and move them out to the right. Then you can use the **FILTER** function to bring back any customers where the **LEN** of the answer is greater than 0. The formula in **A8** is **=FILTER(H7:I36,LEN(I7:I36)>0,"None Selected")**.

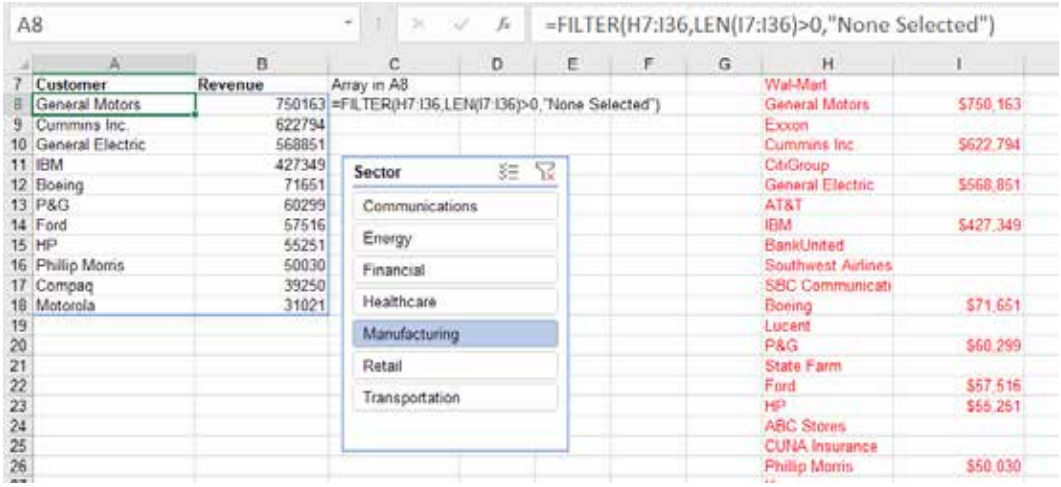


Figure 89 *Using* **FILTER** *to return only the customers with revenue.*

When you choose different sectors from the slicer, the report updates, as shown below.

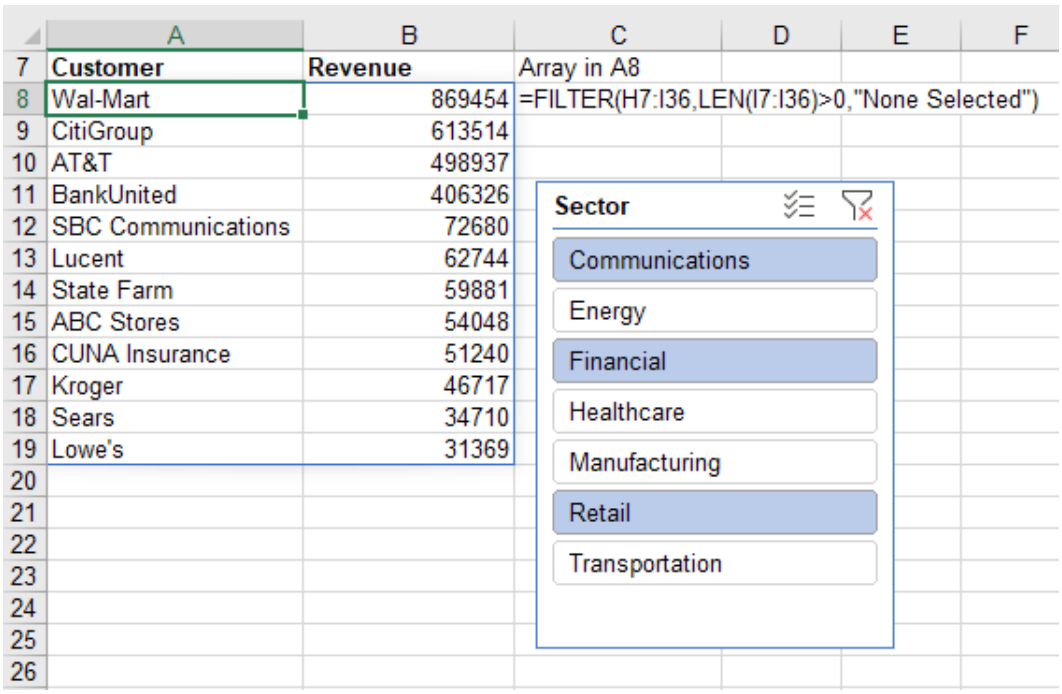


Figure 90 *Success!*

Note that you have to use **LEN(I7:I36)>0** instead of **I7:I36>0** in the formula.

USING DYNAMIC ARRAYS FOR DEPENDENT VALIDATION

The Data Validation feature lets you choose from a dropdown list in Excel. It works great until someone wants to have two lists. The items in the second list are dependent on what is chosen in the first list. This is called dependent validation.

In the figure below, the items for the first dropdown list appear in **D4#**, thanks to **=SORT(UNIQUE(B4:B23))**. The validation in **H3** points to **=D4#**. The list for the second validation appears in **E4#** because of the formula **=FILTER(A4:A23,B4:B23=H3,"Choose Class First")**. The validation in **H5** uses **=E4#**.

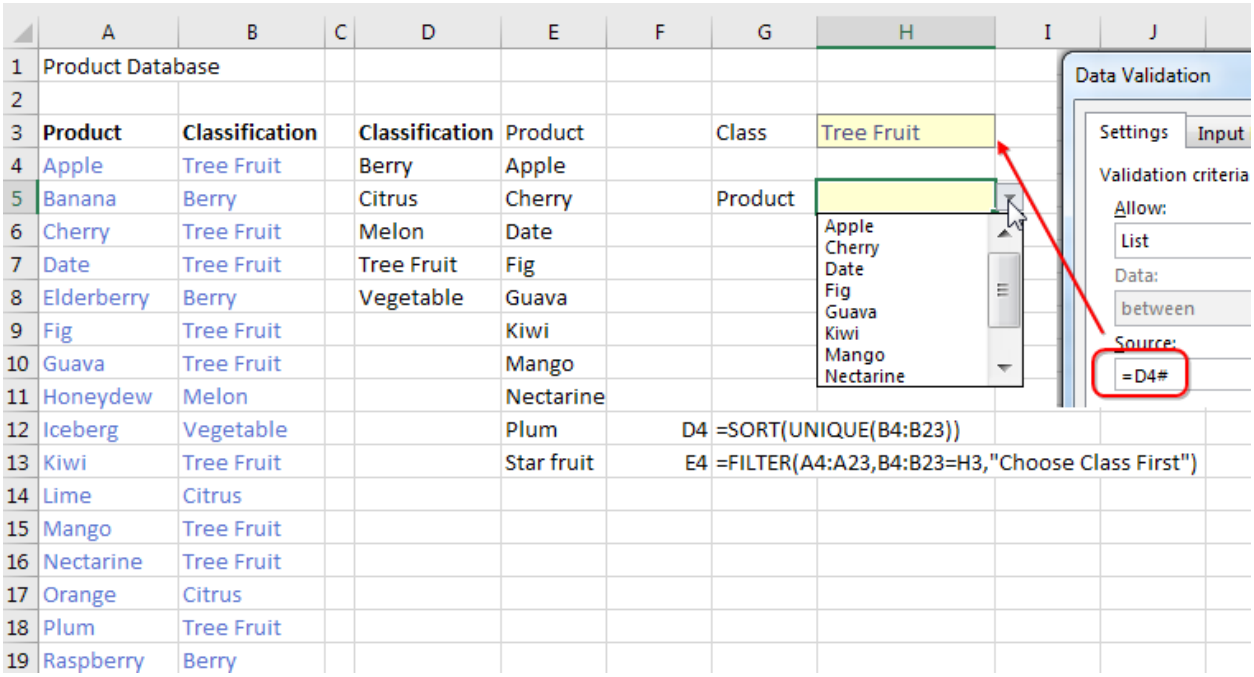


Figure 91 *Excel MVP Liam Bastick worked out the logic for dependent validation.*

THERE WILL BE HUNDREDS MORE EXAMPLES

While I have provided 24 examples in this chapter, this is just the beginning of a whole new world for Excel formulas. If you invent some great new use for Dynamic Arrays, tweet me @MrExcel with the formula.

WHAT IS THE VBA STORY?

The VBA story is coming. There will be new properties and methods. But they are not out as of September 27, 2018.

Index

O

P

R

S

T

U

V

W

Y