Generating Cartesian Products: A Comprehensive Guide to Combining Lists in Excel

Authored by

Mohammed looti

November 10, 2025

Generating a comprehensive list of all possible pairings, formally known as the Cartesian product, between two or more disparate data sets is a fundamental requirement in advanced Excel analysis. This process, often referred to simply as generating **combinations**, ensures that every item in the first list is systematically matched with every item in the second list, producing an exhaustive output table. While contemporary versions of Excel (specifically Microsoft 365) now include powerful dynamic array functions that streamline this operation significantly, many users still rely on traditional or older versions where these modern tools are unavailable. For these users, mastering a complex, array-like formula structure is essential for achieving the required combinatorial output. This intricate formula structure relies heavily on standard, long-established Excel functions, including the INDEX function, COUNTA function, INT function, and MOD function, which collectively manage the iteration and indexing necessary to produce every single combination sequentially down a column.

The Advanced Formula for Generating Combinations

The following robust, non-dynamic array formula is the cornerstone of generating all potential pairings between two distinct sets of spreadsheet data in older or non-subscription versions of Excel. It is imperative to recognize that this method is dependent on relative row positioning. The formula is designed to be entered once and then copied down across the required number of rows, leveraging the current row number--obtained through the ROW function--to systematically calculate which unique combination corresponds to that specific position in the output column. Understanding the structure below is key to customizing it for any dataset.

```
=IF(ROW()-ROW($D$2)+1>COUNTA($A$2:$A$4)*COUNTA($B$2:$B$4),"",INDEX($A$2:$A$4,INT((ROW()-ROW($D$2))/COUNTA($B$2:$B$4)+1))&" "&INDEX($B$2:$B$4,MOD(ROW()-ROW($D$2),COUNTA($B$2:$B$4))+1))
```

This specific configuration is tailored to calculate and output all pairings between the values situated in the range **A2:A4** (designated as List 1) and the values contained in the range **B2:B4** (designated as List 2). The resulting concatenated combinations are displayed sequentially, beginning from the designated starting output cell, **D2**. Critical to the success of this formula are the absolute references, denoted by dollar signs (e.g., `$A$2:$A$4`). These absolute references ensure that when the formula is copied down the column, the defined source data ranges remain fixed and unchanged. Conversely, the dynamic, relative calculation generated by the ROW function correctly handles the iteration, generating a unique index for each combination row.

Deconstructing the Formula Logic: Iteration and Termination

To truly master and adapt this powerful technique, it is essential to systematically dissect the formula into its core operational components. The entire logic hinges on two mathematical principles: first, accurately calculating the total number of combinations possible, and second, using integer mathematics--specifically the INT function and the MOD function--to generate the precise index number required for selecting items from both List 1 and List 2 in the current row. This method guarantees a flawless progression through the Cartesian product.

The Iteration Counter (`ROW()-ROW($D$2)+1`): This critical segment serves as the engine of the formula, generating a simple, monotonically increasing counter. If placed in cell D2, the result is 1; in D3, it is 2; and so forth. This counter provides the fundamental numeric input that drives the subsequent mathematical calculations needed to iterate through all potential pairings.

The Stop Condition (`IF(Counter > COUNTA(A:A)*COUNTA(B:B), "", ...)`): The controlling `IF` statement at the beginning of the formula provides the necessary termination mechanism. It checks if the current iteration counter value exceeds the total number of possible pairings. The total number of pairings is dynamically calculated by multiplying the count of items in List 1 by the count of items in List 2, determined accurately using the COUNTA function. Once the counter surpasses this maximum count, the formula returns an empty string (`""`), effectively halting the combination output sequence.

Generating Indices for List 1 (The Dividend Logic)

To correctly determine which item from the first list (A2:A4) should be selected for the current row's combination, we must employ integer division, which is accomplished in Excel using the INT function. The key requirement for List 1 is that its items must repeat in blocks where the block size is exactly equal to the number of items in List 2. For instance, if List 2 has three items, List 1's first item must appear three times consecutively before the formula shifts to List 1's second item.

The core fragment responsible for calculating the List 1 index is: `INDEX($A$2:$A$4, INT((ROW()-ROW($D$2))/COUNTA($B$2:$B$4)+1))`.

The iteration counter is first divided by the size (count of items) of List 2, calculated via `COUNTA($B$2:$B$4)`.

The INT function then truncates any fractional remainder, effectively performing integer division. This action ensures that the index number for List 1 remains constant until enough rows have passed to complete a full cycle of List 2.

Finally, adding `+1` adjusts the resulting index. This adjustment is necessary because the INDEX function requires its row or column number argument to be 1-based, not 0-based. This structure successfully manages the block repetition required for the first list.

Generating Indices for List 2 (The Remainder Logic)

In contrast to List 1, which repeats in blocks, List 2 must cycle through all its items repeatedly for every single item selected from List 1. This cyclical behavior is perfectly achieved using the mathematical remainder operator, which is implemented in Excel through the powerful MOD function (Modulo).

The section responsible for calculating the List 2 index is: `INDEX($B$2:$B$4, MOD(ROW()-ROW($D$2), COUNTA($B$2:$B$4))+1)`.

The MOD function calculates the remainder when the iteration counter is divided by the size of List 2.

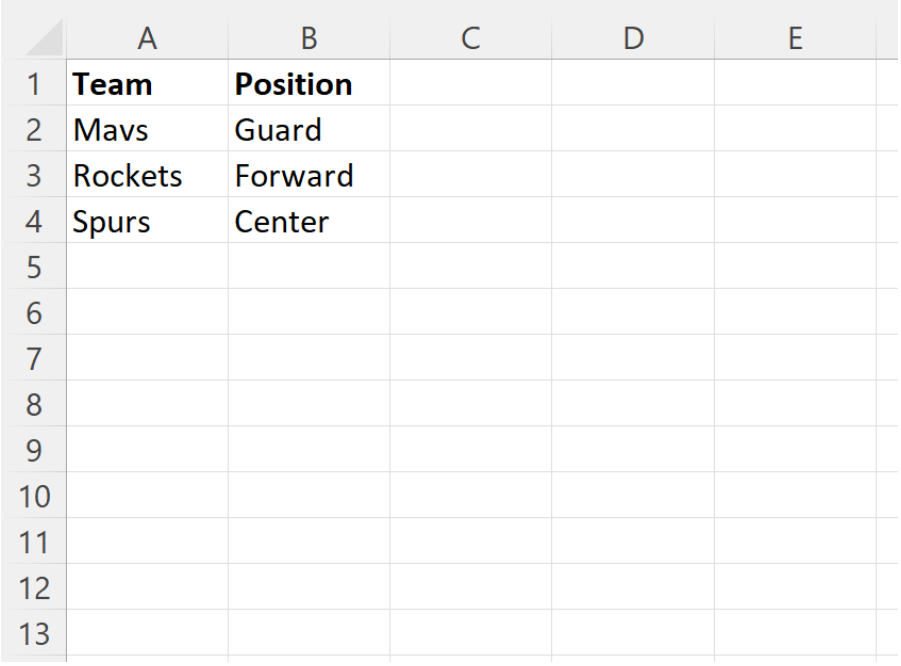
Mathematically, the remainder will cycle perpetually from 0 up to (N-1), where N represents the total size of List 2. This cycle naturally selects the items from List 2 one by one.

As with List 1, adding `+1` to the remainder is crucial. This step converts the standard 0-based result of the modulo operation into the 1-based index that the INDEX function requires, guaranteeing that List 2 iterates through its items correctly for every repetition block of List 1.

Practical Application Example: Generating a Sports Roster

To solidify the understanding of this complex formula, let us apply it to a practical scenario. Imagine the necessity of structuring a comprehensive sports database where we must meticulously list every possible pairing between a defined set of basketball team names and a standard set of corresponding player positions. This is a classic combinatorial problem.

The input data is clearly structured, with the team names residing in Column A (specifically A2:A4) and the player positions listed in Column B (B2:B4), as shown in the initial setup below.



	A	B	C	D	E
1	**Team**	**Position**			
2	Mavs	Guard			
3	Rockets	Forward			
4	Spurs	Center			
5					
6					
7					
8					
9					
10					
11					
12					
13					

Our objective is to generate an exhaustive list of all **Cartesian product** pairings involving the three distinct team names and the three defined positions. Based on the fundamental rules of combinatorics, we anticipate a total output of 3 teams multiplied by 3 positions, resulting in precisely 9 possible unique combinations.

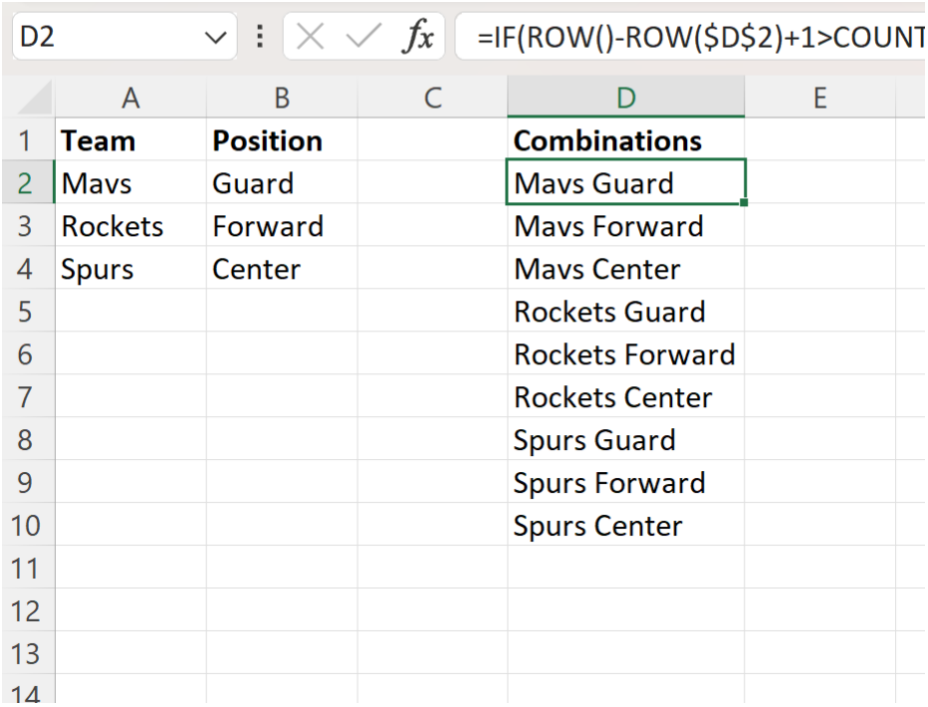
Applying the Combination Formula

To commence the combination listing process, the sophisticated master formula must be accurately entered into the designated starting cell, which is **D2** in this working example. Recall that the formula utilizes absolute references (e.g., `$D$2`, `$A$2:$A$4`) to ensure the source data ranges and the starting point for calculation normalization remain fixed, preventing unintended shifts when the formula is copied.

=IF(ROW()-ROW($D$2)+1>COUNTA($A$2:$A$4)*COUNTA($B$2:$B$4),"",INDEX($A$2:$A$4,INT((ROW()-ROW($D$2))/COUNTA($B$2:$B$4)+1))&" "&INDEX($B$2:$B$4,MOD(ROW()-ROW($D$2),COUNTA($B$2:$B$4))+1))

Once the formula is correctly placed in **D2**, the subsequent and equally vital step is to deploy the formula across the entire required range of rows. This is efficiently achieved by utilizing the fill handle--the small green square located at the bottom-right corner of the active cell--and dragging it

down column D. This process must continue until the formula output transitions from displaying calculated results to returning empty cells. This exact termination behavior is dictated and managed by the initial `IF` stop condition built into the formula's logic.



	A	B	C	D	E
1	**Team**	**Position**		**Combinations**	
2	Mavs	Guard		Mavs Guard	
3	Rockets	Forward		Mavs Forward	
4	Spurs	Center		Mavs Center	
5				Rockets Guard	
6				Rockets Forward	
7				Rockets Center	
8				Spurs Guard	
9				Spurs Forward	
10				Spurs Center	
11					
12					
13					
14					

As visually confirmed in the image provided above, Column D now contains the comprehensive, sequential list of all possible pairings. Each row successfully concatenates the team name and the corresponding position, separated by a space character. Aligning with our preliminary calculation, we can observe that the process has yielded precisely 9 possible unique combinations between the two original source lists.

Separating Combinations Using TEXTSPLIT (Modern Enhancement)

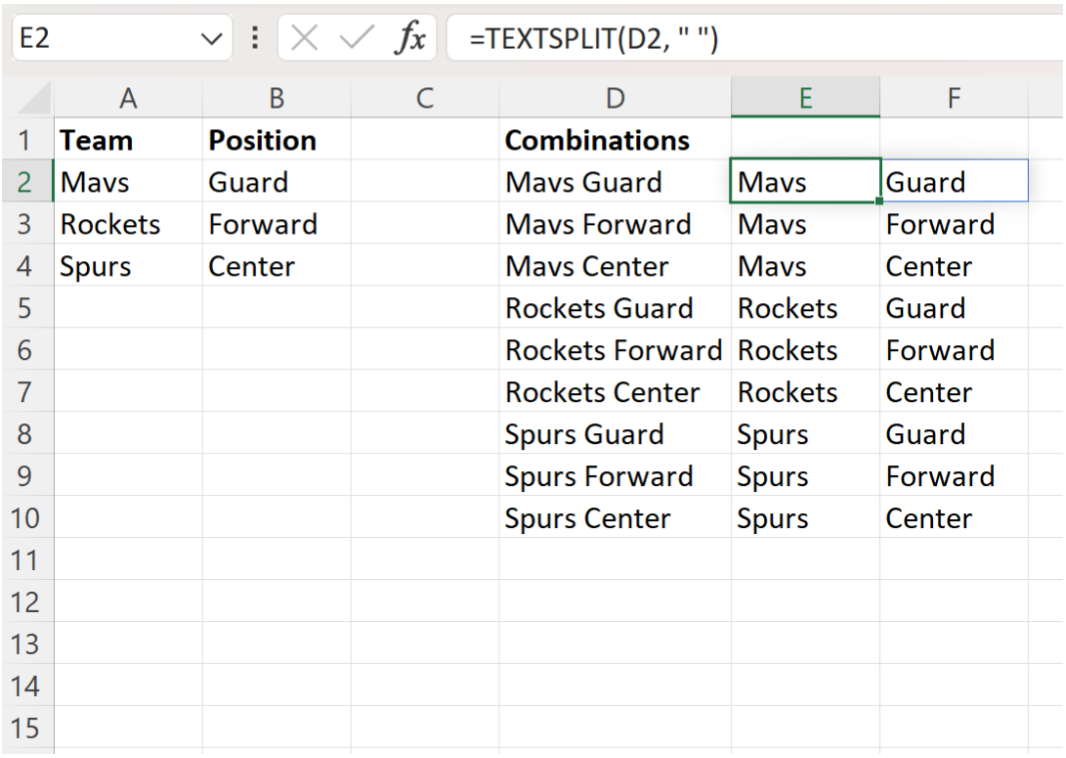
The formula structure detailed previously successfully generates the pairings, but it outputs them as a single, combined text string within one cell (e.g., "Knicks Center"). For subsequent data manipulation, reporting needs, or seamless database import, it is frequently necessary to separate the two constituent elements of the combination--Team and Position--into distinct, adjacent columns. For users operating on modern, subscription-based versions of Excel (Microsoft 365), the specialized **TEXTSPLIT** function provides an exceptionally elegant and highly efficient solution to this parsing requirement.

To cleanly split the concatenated output residing in Column D into its separate components, Team and Position, simply enter the following formula into cell **E2**:

=TEXTSPLIT(D2, " ")

The TEXTSPLIT function is specifically engineered to segment text within a source cell based on a designated delimiter. In this implementation, we instruct the function to analyze the content of cell **D2** and use the space character (" ") as the critical delimiter point for separating the string into columns. Because the output in D2 is part of a dynamic array calculation in Microsoft 365, this single formula entered into E2 will automatically "spill" its results, populating both columns E and F across the entire range of combination data without needing to be dragged down. Users relying on older Excel versions would still need to drag a compatible parsing formula down or utilize the legacy "Text to Columns" feature.

Following the application of the TEXTSPLIT function (and ensuring it is applied throughout the range), the combination data is now perfectly organized and structured:



	A	B	C	D	E	F
1	**Team**	**Position**		**Combinations**		
2	Mavs	Guard		Mavs Guard	Mavs	Guard
3	Rockets	Forward		Mavs Forward	Mavs	Forward
4	Spurs	Center		Mavs Center	Mavs	Center
5				Rockets Guard	Rockets	Guard
6				Rockets Forward	Rockets	Forward
7				Rockets Center	Rockets	Center
8				Spurs Guard	Spurs	Guard
9				Spurs Forward	Spurs	Forward
10				Spurs Center	Spurs	Center
11						
12						
13						
14						
15						

All possible pairings between the teams and positions are now displayed cleanly across two distinct columns. This sophisticated two-step methodology--first generating the combinations using

the complex INDEX function logic, and subsequently parsing the output using TEXTSPLIT function--offers a highly robust and essential framework for managing demanding data pairing requirements within spreadsheet environments. The capability to generate these comprehensive lists is fundamental across diverse professional domains, including statistical modeling, data preparation pipelines, and the generation of comprehensive test cases in software quality assurance.

Additional Resources for Advanced Excel Techniques

For professionals committed to further enhancing their proficiency in data manipulation, array handling, and complex combinatorial analysis within spreadsheet software, the following tutorials detail other common, advanced techniques in Excel. Mastering these functions allows users to perform sophisticated analyses without reliance on external programming tools or VBA scripting.

Tutorial on utilizing modern Dynamic Array functions (such as SEQUENCE, UNIQUE, and FILTER) for dramatically simpler combination generation in Microsoft 365.

A practical guide to leveraging the Power Query Editor for merging tables and generating the Cartesian product in a flexible, non-formulaic data preparation environment.

A detailed statistical explanation of the conceptual differences between permutations and combinations in quantitative analysis.