Excel: List All Possible Combinations Between Lists

Authored by

stats writer

November 17, 2025

Mastering Combinatorics in Excel

Microsoft Excel is an immensely powerful tool, extending far beyond simple data entry and summation. One of the more complex tasks frequently encountered by data analysts and planners is the need to generate a complete list of all possible pairings--or the Cartesian Product--between two or more distinct lists. This requirement often arises in scenarios involving product configurations, scheduling, statistical sampling, or scenario modeling where every permutation must be accounted for. While specialized software can handle this, achieving this outcome directly within a spreadsheet environment requires a sophisticated, array-like formula utilizing core Excel functions.

Generating these extensive lists manually is prone to significant errors and becomes impractical as the source lists grow in size. Imagine trying to pair 50 items from one list with 50 items from another; the resulting 2,500 combinations necessitate an automated solution. The method we will detail relies on a specific combination of built-in functions designed to calculate the total number of required rows and then intelligently index items from the source lists based on the current row number. This technique ensures that every single possible combination is generated sequentially and without omission, transforming a potentially massive manual effort into a simple copy-and-paste operation.

The approach leverages the current position of the formula--specifically its row number--relative to the starting point of the desired output. By cleverly incorporating functions like ROW function, COUNTA function, INDEX function, and mathematical operations like integer division (via INT function) and the remainder (via MOD function), we construct a single formula capable of managing this Combinatorics challenge effectively. This method is essential for anyone seeking to automate complex data preparation within Excel.

The Power Formula: Generating Cartesian Products

The core functionality for listing all possible combinations between multiple lists in Excel resides in a comprehensive formula that elegantly manages the indexing of two separate ranges. This formula is often referred to as a "cross-join" or Cartesian Product generator within the spreadsheet context. Its primary role is twofold: first, it checks whether the formula has exceeded the total possible number of combinations, thus preventing error messages; and second, it accurately retrieves elements from the two input lists based on the current row position.

To list all pairings between two vertical data sets, specifically the values located in the range **A2:A4** and the range **B2:B4**, and output these combinations beginning in cell **D2**, you would deploy the following specific structure. This formula is robust and only requires adjustments to the referenced ranges and the starting output cell to adapt to different scenarios and list lengths.

=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))

The output of this formula is a combined text string, linking the item from the first list, a space delimiter, and the corresponding item from the second list. Understanding the logic embedded in this single line of code is key to utilizing it effectively. It represents an engineering solution that maps a one-dimensional sequence (the output rows) onto a two-dimensional structure (the combination matrix of the two lists). The absolute references (using the dollar sign, **$**) are critical, ensuring that when the formula is dragged down across subsequent rows, the reference ranges and the starting cell anchor remain fixed, guaranteeing accurate and sequential indexing.

Anatomy of the Combination Formula Components

To appreciate the formula's mechanism, we must break down its essential components, focusing on how it determines which element from List 1 (Range A) and List 2 (Range B) to retrieve for any given output row. The calculation hinges entirely on the difference between the current row number and the starting row of the output, effectively creating a zero-based counter for the combinations generated.

The first part, the ROW function condition, acts as the termination switch: **IF(ROW()-ROW($D$2)+1 > COUNTA($A$2:$A$4)*COUNTA($B$2:$B$4), "", ...)**. This checks if the current combination number (calculated by subtracting the starting row number from the current row number and adding one) exceeds the total possible number of combinations. The total count is determined by multiplying the counts of non-empty cells in the two ranges using the COUNTA function. If the limit is reached, it returns an empty string (""), stopping the output cleanly.

For retrieving the elements from the first list (List A), we utilize integer division: **INDEX($A$2:$A$4, INT((ROW()-ROW($D$2))/COUNTA($B$2:$B$4)+1))**. The core idea here is to cycle through the items in List A only after all items in List B have been paired with the current item from List A. The INT function truncates the result of the division, ensuring that the same index for List A is repeated for a number of rows equal to the length of List B. This controlled repetition is what drives the pairing logic for the first list.

Conversely, retrieving elements from the second list (List B) requires the remainder operator: **INDEX($B$2:$B$4, MOD(ROW()-ROW($D$2), COUNTA($B$2:$B$4))+1)**. The MOD function (Modulo) calculates the remainder of a division. When used with the combination counter and the length of List B, it produces a cyclical index that resets to zero after every full cycle through List B. By adding one to the result, we convert this zero-based remainder into a one-based index required

by the INDEX function, ensuring that List B cycles through all its values for every single item selected from List A.

Setting Up the Scenario: Teams and Positions

To illustrate the practical application of this powerful combination formula, let us consider a common planning scenario involving sports data. Suppose we need to generate a roster matrix listing every possible pairing between a defined set of basketball teams and a defined set of player positions. This matrix could be used for scheduling scrimmages, assigning roles, or statistical modeling.

We establish two source lists within our Excel sheet. List 1, representing the Team Names, is housed in column A, specifically cells A2 through A4. List 2, representing the Positions, is housed in column B, specifically cells B2 through B4.

List A (Teams): Lakers, Celtics, Bulls (3 items)
List B (Positions): Guard, Forward, Center (3 items)

The total number of possible combinations (the size of the Cartesian Product) is calculated simply by multiplying the count of items in List A by the count of items in List B (3 x 3 = 9). We anticipate an output matrix spanning nine rows, beginning in the designated output cell, **D2**. The visual representation of this setup confirms the source data structure:

	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					

This clear segregation of input data is essential for the formula's success. By maintaining these

ranges (A2:A4 and B2:B4) as the fixed input parameters, we ensure the formula can correctly count and reference the source data regardless of where the output is placed or how far down the formula is copied. The next step involves implementing the complex formula into the starting cell of our output area.

Implementing the Combination Generator

The implementation phase involves placing the master formula into the top-most cell of the designated output column--in our case, cell **D2**. Because the formula relies heavily on calculating its position relative to this starting cell (**ROW()-ROW($D$2)**), it is imperative that the starting cell reference is correct and absolute.

To initiate the combination listing, we carefully type or paste the full formula into cell **D2**:

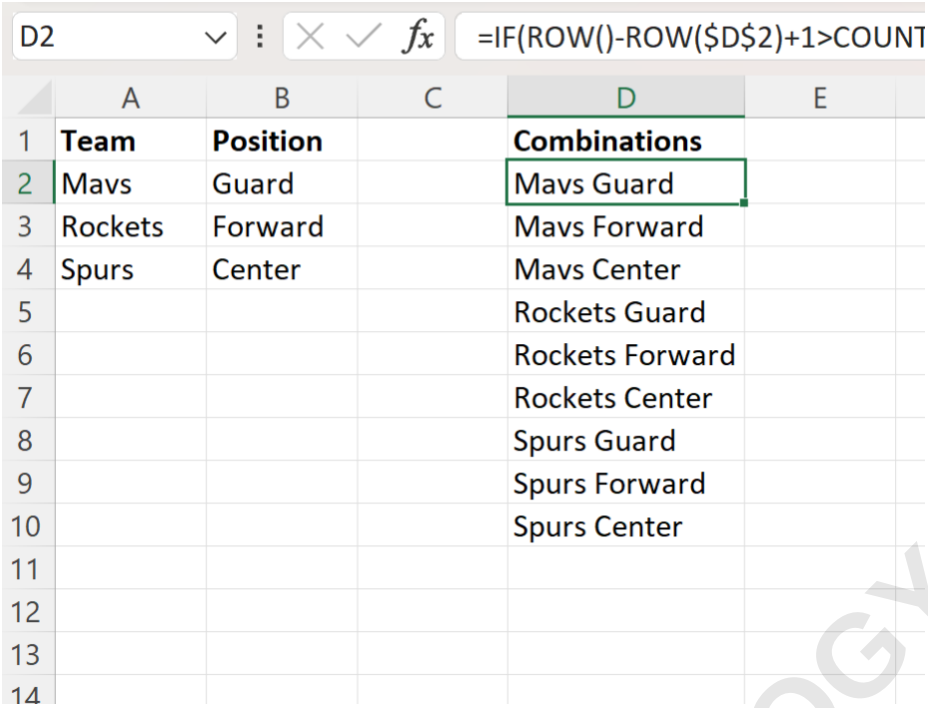
```
=IF(ROW()-ROW($D$2)+1>COUNTA($A$2:$A$4)*COUNTA($B$2:$B$4),"",INDEX($A$2:$A$4,INT((ROW()-ROW($D$2))/COUNTA($B$2:$B$4)+1))&" "&INDEX($B$2:$B$4,MOD(ROW()-ROW($D$2),COUNTA($B$2:$B$4))+1))
```

Once the formula is entered, the immediate result in **D2** should display the very first combination: "Lakers Guard". The true power of this method is unlocked when we propagate the formula down the column. We use the fill handle (the small square at the bottom-right corner of the selected cell) to drag the formula down column D.

The critical action here is dragging the formula further than the expected number of combinations (9, in this example). This is intentional. As the formula is dragged down, the internal counter based on the ROW function increases. When the counter exceeds the maximum total calculated by the COUNTA function multiplication, the **IF** condition switches on, and the formula returns an empty string (""), halting the output gracefully.

Analyzing the Results and Total Combinations

After successfully dragging the formula down column D until the output ceases, we can visually inspect the results. Column D now holds the comprehensive, sequential list of all possible pairings between the team names and the positions. This output represents the complete Cartesian Product of the two source lists, guaranteeing that no combination has been missed.



Observing the output, we confirm that there are exactly **9 possible combinations**, starting with 'Lakers Guard' and concluding with 'Bulls Center'. A quick inspection reveals the systematic manner in which the formula operates:

The first item (Lakers) is held constant while the second list cycles through all its elements (Guard, Forward, Center).
The index for the first list then increments (Celtics), and the second list cycles fully again.
This process repeats until all items in the first list have been paired with all items in the second list.

This organized generation of combinations demonstrates the efficacy of using mathematical functions (INT function and MOD function) combined with positional indexing (INDEX function) to solve complex Combinatorics problems entirely within a standard Excel sheet without needing VBA or external scripting.

Advanced Formatting: Utilizing TEXTSPLIT

While the primary combination formula efficiently generates the pairings as a single combined text string (e.g., "Lakers Guard"), practical applications often require these components to be separated into distinct columns for easier filtering, sorting, or further analysis. For modern versions of Excel (those supporting dynamic arrays), the TEXTSPLIT function provides an immediate and clean solution for this separation task.

The TEXTSPLIT function is designed specifically to take a text string and divide it into multiple

column or row outputs based on a specified delimiter. In our case, the two elements of the combination (Team Name and Position) are separated by a single space, which serves as our ideal delimiter. This eliminates the need for older, multi-step methods involving the Text-to-Columns feature or complex combinations of **LEFT**, **MID**, and **FIND** functions.

By deploying TEXTSPLIT, we can immediately create a clean table structure where the team name occupies one column and the position occupies the adjacent column, significantly enhancing the readability and utility of the generated data set. This transition from a concatenated string to a structured table is a crucial final step in preparing the combination data for reporting or database migration.

Step-by-Step TEXTSPLIT Implementation

To perform the separation, we initiate the TEXTSPLIT function in the column immediately adjacent to the combined output--in this example, starting in cell **E2**. The formula specifies the cell containing the combined text and defines the delimiter used during the combination process.

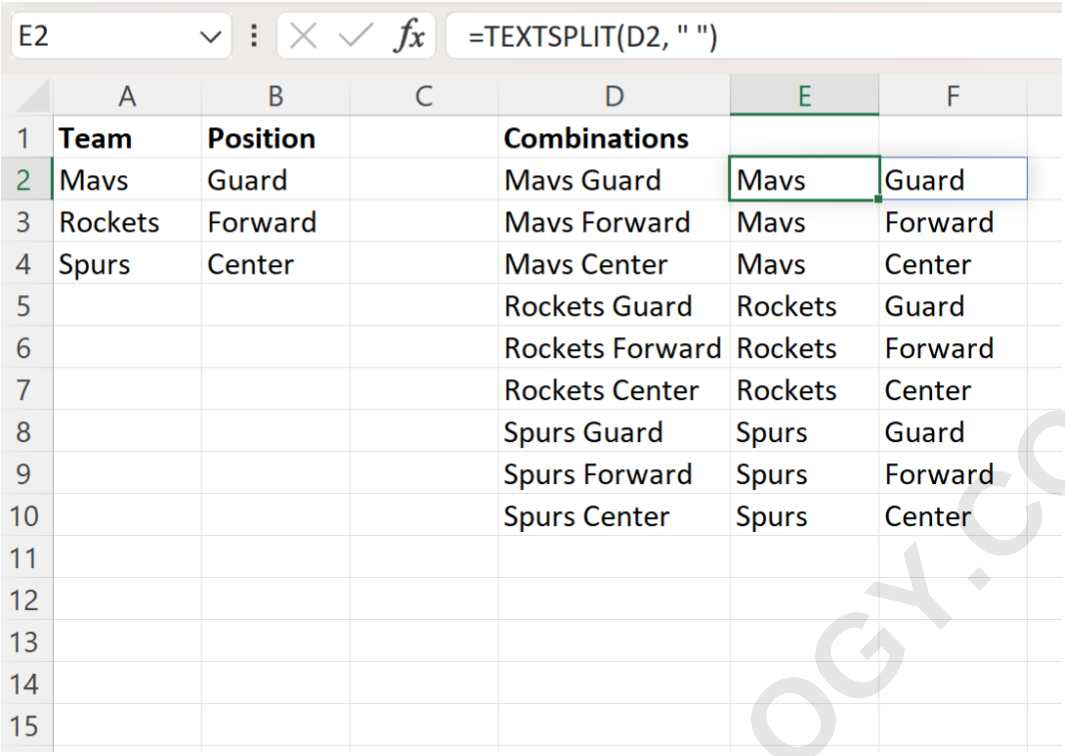
The formula is straightforward:

=TEXTSPLIT(D2, " ")

In this function:

D2 refers to the cell containing the combined text string (e.g., "Lakers Guard").
" " (a space enclosed in quotation marks) is the column delimiter, instructing Excel to split the text wherever a space is found.

Upon entering this formula in **E2**, the result will dynamically spill over into the adjacent cell, **F2**. Cell **E2** will show "Lakers," and cell **F2** will show "Guard." We then click and drag this formula down column E until it aligns with the last row of combinations generated in column D.



	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						

The final result is a structured, two-column table (Columns E and F) containing all possible combinations, perfectly parsed into their individual components. This dynamic array approach ensures efficiency and cleanliness, allowing for rapid data transformation suitable for further pivot table analysis, charting, or export.

Conclusion: Versatility and Next Steps

The combination of the INDEX function, Combinatorics logic (using INT function and MOD function), and the modern TEXTSPLIT function provides a robust, native Excel solution for generating and processing the complete Cartesian Product of two lists. This technique moves beyond simple lookups and demonstrates the capacity of Excel to handle complex data permutation tasks.

While this detailed example focused on two lists of equal length (3x3), the formula is designed to handle lists of unequal lengths seamlessly. The COUNTA function automatically determines the actual populated size of each range, adjusting the indexing logic accordingly, ensuring maximum flexibility. For instance, if List A contained 10 items and List B contained 5 items, the formula would correctly calculate 50 total combinations and execute the sequence accurately.

For those requiring combinations from three or more lists, the core logic must be extended by introducing additional nested **INDEX** components. While the complexity of the formula increases with each added list, the fundamental principle remains the same: using integer division (**INT**) to

control the slower-changing indices and the remainder function (**MOD**) to control the faster-changing index. Mastery of this two-list approach serves as the foundational skill for tackling increasingly sophisticated combinatoric requirements within the Excel environment.