How to Check if Cell Contains Text from List in Google Sheets

Authored by

stats writer

November 21, 2025

Determining if a specific cell contains any text from a predefined list in Google Sheets requires a sophisticated approach beyond simple logical checks. While you might initially consider using a combination of the FILTER function and the COUNT function, a more efficient and powerful method for handling large datasets and complex partial matches involves using an array of functions, specifically combining ArrayFormula, TEXTJOIN, and REGEXMATCH.

This methodology is essential when dealing with situations where the target cell might contain other descriptive text alongside the key phrase you are searching for. For instance, searching for "Apple" within a cell containing "Red Delicious Apple" demands pattern matching rather than exact string equality. Leveraging these advanced formulas ensures accuracy and scalability across your worksheets.

The goal is to generate a dynamic formula that automatically iterates through every item in your defined search list, checks each against the target cell(s), and consolidates the results into a single, comprehensive output. If any segment of the target cell matches any item in the search list, the formula will return a positive result, typically represented as **TRUE**.

The Complete Formula for List Checking

The most robust formula structure utilized in Google Sheets for checking if a cell range contains text from a dynamic list relies on an integrated use of several powerful functions. This approach eliminates the need for manual dragging and copying by processing an entire column at once, making it ideal for maintaining clean, scalable spreadsheets.

The following formula snippet is designed to be entered into a single cell, typically the top cell of your results column, and will populate the entire column based on the criteria provided. It uses regular expressions to perform case-insensitive, partial matching against the list of predefined terms.

Here is the exact structure used to achieve this complex conditional check:

=ArrayFormula(IF(LEN(A2:A13), REGEXMATCH(A2:A13,".*(?i)("&TEXTJOIN("|", TRUE,$E$2:$E$4)&").*") ,""))

In this representative example, if the cells within the range **A2:A13** contain any of the text values defined in the list range **$E$2:$E$4**, the formula will return **TRUE** for that corresponding row; otherwise, it will return **FALSE**. The key elements here are the array processing capabilities and the construction of a dynamic regular expression pattern.

Understanding the internal logic of this formula is essential for customization. We must first ensure that the target cell range is not empty before attempting the demanding REGEXMATCH operation.

This validation step is handled efficiently by the IF and LEN functions, ensuring resource optimization.

Deconstructing the Formula Components: Power of ArrayFormula and TEXTJOIN

The effectiveness of this solution stems from the coordinated action of its constituent functions. The outermost function, ArrayFormula, is crucial because it allows the formula to work on an entire range (e.g., A2:A13) rather than just a single cell. Without ArrayFormula, you would have to manually apply the formula row by row, losing the efficiency of array processing.

Inside the structure, we utilize TEXTJOIN to efficiently format the list of search criteria into a usable regular expression. The range **$E$2:$E$4** (our list of terms) is concatenated into a single string, with the pipe symbol (`|`) acting as the delimiter. In regular expression syntax, the pipe symbol signifies an "OR" condition. Thus, if our list contains "Mavs," "Spurs," and "Rockets," TEXTJOIN converts this into the string "Mavs|Spurs|Rockets".

The second argument in the TEXTJOIN function, set to **TRUE**, tells the function to ignore any empty cells within the search range, which is a best practice for clean pattern generation. This seamless conversion is what allows the subsequent function, REGEXMATCH, to perform multiple checks simultaneously against the target cell content.

Implementing REGEXMATCH for Flexible Matching

The heart of the pattern matching process is the REGEXMATCH function. This function checks if a piece of text matches a regular expression (regex) pattern. The regex pattern constructed dynamically by TEXTJOIN and surrounding characters is what gives the formula its flexibility.

Let's analyze the complete regex pattern generated: `".*(?i)("&TEXTJOIN(...)&").*"`. The components serve specific purposes:

`".*"`: This prefix and suffix act as wildcards, allowing the pattern to match text anywhere within the cell content. This ensures we are checking for partial containment, not exact matches.
`(?i)`: This is a crucial inline modifier that ensures the entire regular expression performs a case-insensitive search. This means "mavs" will match "Mavs," preventing errors related to capitalization.
`"("&TEXTJOIN(...)&")"`: This segment injects our list of terms (joined by `|`) into the pattern, enclosed in parentheses to define the group of OR conditions (e.g., (Mavs|Spurs|Rockets)).

The combination of these elements provides powerful functionality, enabling us to test every cell in the target range against multiple search terms simultaneously, confirming whether any of the items

in the list exist within the cell's content, regardless of surrounding text or letter casing.

Practical Example Setup: Identifying Teams from a Specific Region

To demonstrate this formula in action, consider a scenario involving a dataset tracking basketball player performance. Suppose we have a list detailing the points scored by various players, and we need to quickly identify which rows correspond to teams originating specifically from the state of Texas.

Our initial dataset, hypothetically contained in columns A and B, shows the team name and associated scores. We are interested in creating a new column that marks each row as **TRUE** or **FALSE** based on whether the team name belongs to our predefined list of Texas teams. This is a common requirement in data analysis for filtering, conditional formatting, or subsequent calculations.

The initial dataset structure might look like this, showing scores for various teams:

	A	B	C	D
1	**Team**	**Points**		
2	Mavs	31		
3	Nets	24		
4	Mavs	23		
5	Lakers	23		
6	Warriors	20		
7	Thunder	19		
8	Spurs	15		
9	Mavs	19		
10	Spurs	23		
11	Rockets	40		
12	Hornets	37		
13	Spurs	20		
14				
15				
16				
17				
18				
19				
20				

In this specific context, we are focusing on three teams known to be from Texas: the Mavs (Dallas

Mavericks), the Spurs (San Antonio Spurs), and the Rockets (Houston Rockets). These three names form the basis of our search criteria list.

Defining the Search List and Range

Before applying the formula, the first step is to establish the list of terms we are searching for. It is best practice to place this list in a separate column, allowing for easy updates and clear referencing. We will designate column E for this purpose.

We create a dedicated list of the Texas teams in column E, starting at E2. This list (E2:E4) will serve as the input range for the TEXTJOIN function within our formula. Using absolute references (e.g., **$E$2:$E$4**) for this range is essential when using ArrayFormula to ensure the reference does not shift.

Visually, the spreadsheet now includes the target data in columns A and B, and our criteria list in column E:

	A	B	C	D	E
1	**Team**	**Points**			**List**
2	Mavs	31			Mavs
3	Nets	24			Spurs
4	Mavs	23			Rockets
5	Lakers	23			
6	Warriors	20			
7	Thunder	19			
8	Spurs	15			
9	Mavs	19			
10	Spurs	23			
11	Rockets	40			
12	Hornets	37			
13	Spurs	20			
14					
15					
16					
17					
18					
19					
20					

This clear separation of data (A:B) and criteria (E) enhances the maintainability of the spreadsheet.

If the list of Texas teams changes, only column E needs modification, and the results in the output column will automatically update.

Applying the Array Formula and Analyzing Results

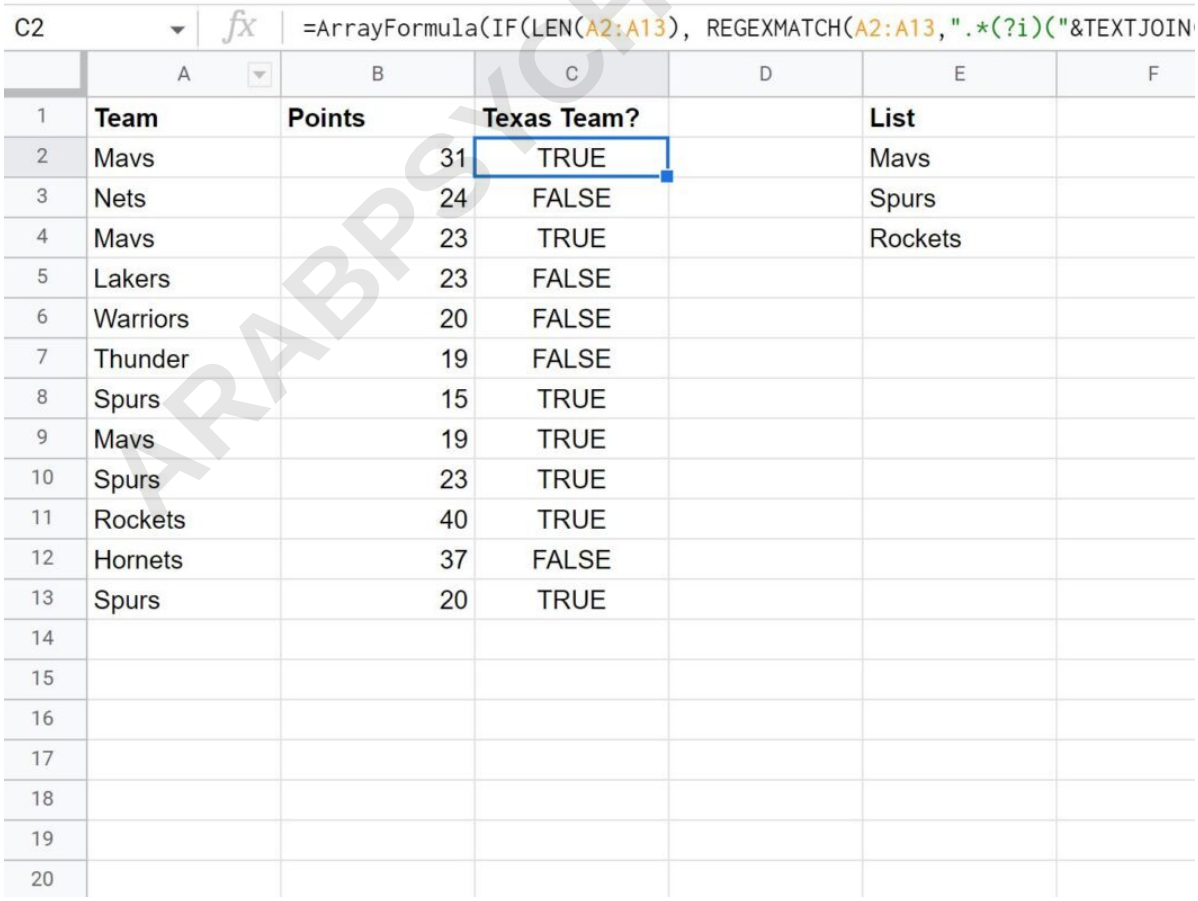
With the setup complete, we can now implement the full array formula into cell **C2**, which will be the header for our "Is Texas Team?" result column. The formula checks every entry in the team column (A2:A13) against the list of Texas teams (E2:E4).

The formula entered into cell C2 is:

=ArrayFormula(IF(LEN(A2:A13), REGEXMATCH(A2:A13,".*(?i)("&TEXTJOIN("|", TRUE,$E$2:$E$4)&").*") ,""))

Upon execution, the ArrayFormula automatically spills the results down column C, providing a boolean status for every row corresponding to the length of the data in column A. Rows that contain "Mavs," "Spurs," or "Rockets" will be flagged as **TRUE**.

The final output demonstrates the successful application of the pattern matching logic:



	A	B	C	D	E	F
1	**Team**	**Points**	**Texas Team?**		**List**	
2	Mavs	31	TRUE		Mavs	
3	Nets	24	FALSE		Spurs	
4	Mavs	23	TRUE		Rockets	
5	Lakers	23	FALSE			
6	Warriors	20	FALSE			
7	Thunder	19	FALSE			
8	Spurs	15	TRUE			
9	Mavs	19	TRUE			
10	Spurs	23	TRUE			
11	Rockets	40	TRUE			
12	Hornets	37	FALSE			
13	Spurs	20	TRUE			
14						
15						
16						
17						
18						
19						
20						

As illustrated, any row where the team name contains a match against our predefined list receives a value of **TRUE**, while all other rows, such as those for the Nets or Lakers, receive a value of **FALSE**. This instantaneous processing across the entire range highlights the efficiency of using array-based regular expression matching for complex data validation tasks.

Detailed Analysis of Match Outcomes

Examining the results in column C provides clear evidence of how the REGEXMATCH and ArrayFormula combination successfully evaluates each condition. The outcome is binary, making subsequent filtering or aggregation straightforward.

We can specifically analyze the results of several rows:

The first row received a value of **TRUE** because the team "Mavs" was successfully identified within the predefined list (E2:E4).
The second row received a value of **FALSE** because the team "Nets" is not present in the search list of Texas teams.
The third row also returned **TRUE**, confirming that "Mavs" is recognized irrespective of its position in the overall dataset.
The fourth row, featuring "Lakers," resulted in **FALSE**, correctly indicating the absence of a match with the Texas team criteria.

This method provides a robust and scalable solution for performing conditional checks across large datasets in Google Sheets, proving far superior to manually applying VLOOKUP or multiple nested IF statements for partial text containment checks.