How to Use Wildcards with the FILTER Function in Google Sheets: A Step-by-Step Guide

Authored by

stats writer

November 21, 2025

1. Introduction: Understanding Wildcards in Google Sheets Filtering

While powerful, the standard filtering capabilities within Google Sheets often require exact matches or rely on complex regular expressions for pattern recognition. For users needing to swiftly locate data based on partial text matches or specific character patterns, utilizing wildcard characters becomes essential. Traditional wildcard syntax, such as the asterisk (*) representing any sequence of characters and the question mark (?) representing any single character, is a common feature in many spreadsheet environments. However, the native FILTER function in Google Sheets does not inherently support these standard symbols directly in its criteria argument for text searches.

To effectively implement flexible, pattern-based searching--which is the fundamental purpose of wildcards--within the powerful array manipulation capabilities of the FILTER function, we must employ an auxiliary function. This combination allows you to define criteria that look for specific strings, phrases, or patterns within a column, regardless of the surrounding text. This method drastically improves data analysis efficiency, enabling users to quickly isolate and highlight specific records from large datasets based on inexact matches.

By integrating a specialized search function, we can mimic the behavior of wildcards, transforming the standard FILTER function from a rigid exact-match tool into a dynamic, pattern-matching engine. This article will detail the necessary syntax and provide comprehensive examples demonstrating how to harness this technique, ensuring you can leverage the power of flexible text searching within your spreadsheets, thus moving beyond the limitations of simple equals comparisons.

2. The Challenge of Standard Filtering

When working with textual data in Google Sheets, a common requirement is to filter a range based on whether a column **contains** a certain substring, rather than equaling it entirely. Standard conditional criteria used within the FILTER function typically involve boolean comparisons, such as checking if a cell is equal to, greater than, or less than a specified value. If you attempt to use the traditional wildcard symbols (* or ?) directly within the criteria argument of `FILTER`, the function will interpret them literally, often resulting in no matches or inaccurate results.

For instance, if we had a column of product names and wanted to retrieve all products containing the word "Pro," a simple criterion like `A2:A10="*Pro*"` would generally fail to produce the desired effect within the native FILTER function environment. Google Sheets requires a different approach for substring matching compared to database systems or other spreadsheet applications that natively recognize these symbols. This limitation necessitates a workaround that translates the concept of "contains" into a boolean (TRUE/FALSE) array that the `FILTER` function can process effectively.

The core challenge lies in generating this criteria array. The FILTER function requires a second argument that is an array or range of boolean values, where **TRUE** corresponds to rows that should be included in the final output. Therefore, to achieve wildcard-like behavior, we need a helper function capable of iterating through the target range and returning a number or value that can be coerced into a boolean **TRUE** for every cell that successfully contains the specified pattern, simulating the effect of the asterisk wildcard on both ends of the search term.

3. Integrating the `SEARCH` Function for Wildcard Functionality

To overcome the limitations of standard filtering criteria and successfully implement wildcard behavior, we utilize the SEARCH function. The primary role of SEARCH is to locate a specific substring within a text string and return the starting position of that substring. If the substring is found, SEARCH returns a positive number (its position); if it is not found, it returns a `#VALUE!` error. This characteristic is precisely what makes it an ideal candidate for creating the conditional array required by the FILTER function.

When applied to a range of cells, the SEARCH function outputs an array where each successful find generates a position number, and each failed search generates an error. In the context of the `FILTER` function, any positive number in the criteria array is automatically interpreted as **TRUE**, indicating a match, while the presence of the `#VALUE!` error is implicitly treated as **FALSE**, indicating a non-match. This automatic type coercion converts the numerical output of SEARCH directly into the necessary boolean criteria, effectively allowing us to search for any text string located anywhere within the target cell, thereby replicating the functionality of surrounding the search term with the **wildcard characters**.

It is important to note that the SEARCH function in Google Sheets is **case-insensitive**. If your filtering requirements demand a case-sensitive match, you would need to substitute the `SEARCH` function with the `FIND` function, which performs an identical task but distinguishes between uppercase and lowercase letters. For most general wildcard filtering purposes, however, the case-insensitivity of `SEARCH` provides greater flexibility and is the preferred method for generating the criteria array for `FILTER`.

4. Detailed Syntax Breakdown of the `FILTER` and `SEARCH` Combination

To implement this dynamic filtering technique, we combine the FILTER function and the SEARCH function into a single formula. Understanding the structure of this combined expression is key to mastering dynamic data retrieval in Google Sheets.

The general structure of the formula is as follows:

=FILTER(range_to_filter, search("search_term", criteria_range))

Let us break down the components of the example formula provided below, which filters the range **A2:C11** based on criteria in **A2:A11**:

=FILTER(A2:C11,search("avs",A2:A11))

The first argument of `FILTER`, `A2:C11`, is the `range_to_filter`. This defines the entire block of data--including all columns and rows--that you wish to return. The rows that meet the criteria will be extracted from this range and displayed in the result area. The second argument is the crucial conditional expression: `search("avs",A2:A11)`. Here, `"avs"` is the `search_term`, the specific substring we are looking for. The `criteria_range`, `A2:A11`, is the specific column within the larger range that the search will operate on. The `SEARCH` function performs the test row by row, generating an array of positions (if found) or errors (if not found). This numerical/error array is then processed by `FILTER`, yielding the filtered dataset. This particular formula will filter the cells in the range **A2:C11** to only contain the rows where the value in **A2:A11** contains "avs" somewhere within the cell's contents.

5. Practical Example: Filtering Player Data

To illustrate the efficacy of using `FILTER` with `SEARCH` for wildcard-like operations, let us consider a real-world scenario involving a dataset detailing various basketball players. This dataset includes information such as the Player's Name, their Team, and their average Points Per Game (PPG). Suppose we have this data organized in columns A, B, and C, starting from row 2.

The hypothetical dataset is structured as follows:

	A	B	C	D
1	**Team**	**Points**	**Assists**	
2	Mavs	22	5	
3	Mavs	29	4	
4	Spurs	30	4	
5	Mavs	23	7	
6	Spurs	29	8	
7	Cavs	14	7	
8	Hornets	10	8	
9	Spurs	12	12	
10	Cavs	18	10	
11	Nets	14	7	
12				
13				
14				
15				
16				
17				
18				
19				
20				

Our goal is to apply a filter to this range, **A2:C11**, but we only want to display rows where the Team name (found in column A) contains a specific, potentially partial string. For instance, we might want to find all teams whose names contain the abbreviation "avs" regardless of whether it appears at the beginning, middle, or end of the full team name. This is a classic scenario where simple equality comparison fails, demanding the dynamic text search capability provided by the combined `FILTER(SEARCH(...))` method.

This approach is particularly useful in large inventories, customer lists, or transaction logs where identifiers might contain codes or substrings that need to be quickly isolated. By using this method, the analyst avoids the tedious process of manually sorting or using simple filters multiple times for slightly different variations of a name or code. The use of `SEARCH` provides the robust text matching flexibility we are aiming for, ensuring that the results are comprehensive and accurate based on the partial match criteria.

6. Step-by-Step Implementation: Searching for Specific Substrings

Following our example, suppose we want to filter the **A2:C11** dataset to retrieve only the rows

where the value in the **Team** column (A2:A11) contains the string "avs" somewhere in the name. Since we are filtering based on the content of column A, this column will serve as our criteria range.

To apply this powerful dynamic filter, we will input the formula into an empty cell outside the data range, typically cell **A13**, which is where the filtered array will spill over. The exact formula needed to achieve this specific filtering requirement is:

=FILTER(A2:C11,search("avs",A2:A11))

Upon execution, the SEARCH function internally evaluates every cell from **A2** through **A11**. For any cell containing "avs," `SEARCH` returns the starting position number (a positive integer); for cells that do not contain "avs," it returns the `#VALUE!` error. The outer `FILTER` function then interprets the positive numbers as **TRUE** and the errors as **FALSE**, thus selecting only the corresponding rows from the **A2:C11** range and displaying them starting at **A13**. This action successfully simulates a wildcard search, retrieving all records matching the substring.

7. Analyzing the Filtered Results and Troubleshooting

Once the formula is entered, the filtered dataset appears immediately below the input cell. The following screenshot demonstrates the practical application of the formula and the resulting output:

A13 | =FILTER(A2:C11,search("avs",A2:A11))

	A	B	C	D
1	**Team**	**Points**	**Assists**	
2	Mavs	22	5	
3	Mavs	29	4	
4	Spurs	30	4	
5	Mavs	23	7	
6	Spurs	29	8	
7	Cavs	14	7	
8	Hornets	10	8	
9	Spurs	12	12	
10	Cavs	18	10	
11	Nets	14	7	
12				
13	Mavs	22	5	
14	Mavs	29	4	
15	Mavs	23	7	
16	Cavs	14	7	
17	Cavs	18	10	
18				
19				
20				
21				
22				
23				

Observe closely that the only rows displayed in the filtered output are those where the team name, located in the original column A, contains the string "avs." For example, both "Mavericks" and "Hawks" might be included if they contain the specified substring, demonstrating the successful implementation of the wildcard-like functionality. This outcome confirms that the combination of `FILTER` and SEARCH successfully targets and extracts records based on a partial match criterion.

A common troubleshooting point relates to the selection of the ranges. Always ensure that the `range_to_filter` (e.g., A2:C11) and the `criteria_range` (e.g., A2:A11) have the exact same number of rows. If the row counts differ, the `FILTER` function will return an error because it cannot reconcile the row indexes between the data array and the criteria array. Furthermore, if the formula returns a `#N/A` error, it typically means that **no rows matched the criteria**, indicating that the specified substring was not found anywhere in the target column.

8. Advanced Modification: Adapting the Search Criteria

One of the greatest advantages of using this formula structure is its immediate adaptability. To search for a different string or pattern, the user simply needs to replace the current search term (e.g., "avs") within the `SEARCH` function argument with the new desired substring. The rest of the formula structure remains identical, allowing for rapid querying of the dataset based on various partial matches.

For instance, if we now wish to isolate all players whose team names contain the string "ets," we only need to modify the `search_term` parameter. We can use the following formula, maintaining the same data and criteria ranges:

=FILTER(A2:C11,search("ets",A2:A11))

Applying this modified formula yields a new filtered result, which now only includes teams that match the substring "ets." This dynamic modification capability makes the `FILTER(SEARCH(...))` pattern an incredibly versatile tool for iterative data analysis and exploration within Google Sheets. The consistency in the formula structure ensures scalability and minimizes the potential for error when adapting the filter to new search requirements.

The subsequent screenshot illustrates the resulting output when searching for the new substring "ets":

A13 | fx | =FILTER(A2:C11,search("ets",A2:A11))

	A	B	C	D	E
1	**Team**	**Points**	**Assists**		
2	Mavs	22	5		
3	Mavs	29	4		
4	Spurs	30	4		
5	Mavs	23	7		
6	Spurs	29	8		
7	Cavs	14	7		
8	Hornets	10	8		
9	Spurs	12	12		
10	Cavs	18	10		
11	Nets	14	7		
12					
13	Hornets	10	8		
14	Nets	14	7		
15					
16					
17					
18					
19					
20					

Now the only rows displayed in the filtered dataset contain the string "ets" somewhere in the team name. This fluidity in criteria definition is the hallmark of effective spreadsheet automation.

9. Conclusion: Mastering Dynamic Filtering Techniques

Mastering the combination of the FILTER function with the SEARCH function provides a highly effective workaround for the lack of native wildcard characters support in standard Google Sheets criteria. By leveraging the numerical output of `SEARCH`--which is automatically interpreted as **TRUE** by `FILTER` when a match is found--users can perform complex, case-insensitive substring matches across large data arrays. This methodology fundamentally expands the analytical power available in Google Sheets, allowing for more precise and dynamic data retrieval based on partial text patterns.

This technique is not limited solely to text matching; the underlying principle of generating a boolean criteria array can be extended to implement other complex filtering logic, such as using the `ISNUMBER` function in conjunction with `MATCH` to filter based on values found in another list, or

combining multiple `SEARCH` statements using boolean logic (e.g., plus signs for OR logic, multiplication for AND logic). Understanding this mechanism is crucial for intermediate and advanced Google Sheets users who seek to automate complex data aggregation and reporting tasks.

In summary, while Google Sheets does not officially support traditional * and ? wildcard characters within its primary array functions like `FILTER`, the seamless integration of the SEARCH function provides a robust, flexible, and case-insensitive alternative. This powerful combination ensures that users can quickly find and highlight the exact data needed, significantly streamlining the process of cleaning, analyzing, and presenting data within the spreadsheet environment.