

How to Easily Find Text in Google Sheets Using the SEARCH Function

Authored by
stats writer

November 21, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Find Text in Google Sheets Using the SEARCH Function*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=99100>

The Google Sheets environment is a remarkably powerful platform for data analysis, and mastering its functions is key to efficiently handling large quantities of information. Among the most useful functions for text manipulation and conditional filtering is the SEARCH function. Unlike basic string searching, **SEARCH** is specifically designed to locate a specific text string--or 'needle'--within a larger body of text--the 'haystack'--and return the starting position of that string. If the string is not found, it generates an error value. This powerful utility is instrumental when performing complex queries, such as searching for partial matches, validating data entry, or, as we will demonstrate, applying multiple criteria to filter extensive datasets.

A crucial characteristic that distinguishes **SEARCH** from its counterpart, the **FIND** function, is its inherent **case-insensitivity**. This means that whether you search for "apple" or "Apple," the function treats them identically, returning the same position. This flexibility makes **SEARCH** the preferred choice for user-facing applications where consistent capitalization cannot be guaranteed. Furthermore, when combined with logical functions or powerful array functions like FILTER, **SEARCH** transcends simple text location and becomes a cornerstone of advanced data extraction techniques, allowing users to efficiently sift through millions of cells to pinpoint exact matches or partial text segments based on sophisticated rules.

Understanding the fundamental application of the **SEARCH** function is the prerequisite for implementing complex multi-criteria filters. While it can be used alone to return a numeric position, its true power is unleashed when it is used to generate a Boolean or numeric array for other functions. When **SEARCH** successfully locates the specified text, it returns a position (a number greater than zero). In the context of array formulas, any non-zero number is interpreted as **TRUE**. Conversely, if **SEARCH** fails to find the text, it returns a `#VALUE!` error. This error, when passed into functions like FILTER, is interpreted as **FALSE**, enabling precise filtering based on the presence or absence of keywords across large columnar ranges. This mechanism is the foundation for creating highly dynamic and responsive spreadsheets capable of real-time data filtering.

Deconstructing the Syntax and Mechanics of **SEARCH**

The formal structure, or syntax, of the **SEARCH** function is elegantly simple yet incredibly versatile. It requires three specific arguments, though the third is optional. The structure is defined as `SEARCH(search_for, text_to_search,)`. The first argument, `search_for`, specifies the string or numerical value that the user is attempting to locate. This input is typically provided as a string enclosed in double quotes (e.g., "keyword") or a reference to a cell containing the desired search term. Precision in defining this string is paramount, as the function will look for an exact sequence of characters within the target cell.

The second argument, `text_to_search`, is the required range or cell containing the text where the

search will be performed. When utilizing **SEARCH** within an array context, such as with the [FILTER function](#), this argument often refers to an entire column or a specific vertical range (e.g., [A2:A100](#)). In this array context, the **SEARCH** function executes on every cell within that range simultaneously, generating an array of results--either numeric positions or `#VALUE!` errors--which the outer function then uses to evaluate truth conditions. This capability to process entire ranges non-iteratively is what makes these combined functions so efficient for handling large datasets.

The final argument, `starting_at`, is optional and specifies the character position within the `text_to_search` where the search should begin. By default, if this argument is omitted, the search starts at the very first character (position 1). While often unnecessary for simple presence/absence checks, utilizing `starting_at` becomes crucial in more niche scenarios, such as skipping a known prefix or header structure within a text string to locate a subsequent piece of information. Regardless of whether this third argument is used, the core principle remains: **SEARCH** is fundamentally a locator tool that returns a positive integer upon success and a predictable error upon failure, forming a powerful logic gate for conditional data manipulation.

Combining **SEARCH** and **FILTER** for Advanced Data Retrieval

While the **SEARCH** function is excellent for determining if a single cell contains a specific substring, it is not inherently designed to extract or filter rows from a sheet based on that criterion. This is where the integration with the [FILTER function](#) becomes essential. The [FILTER function](#) operates by taking a data range as its first argument and one or more conditional expressions as subsequent arguments. It then returns only those rows from the initial range for which all conditional expressions evaluate to **TRUE**. By embedding the **SEARCH** function as the conditional expression, we effectively instruct Google Sheets to "filter this data range, only keeping rows where the **SEARCH** function successfully found the keyword."

When **SEARCH** is used as a condition within **FILTER**, the error-handling behavior of **SEARCH** is implicitly managed. Since **FILTER** only accepts **TRUE** or **FALSE** logic, and a successful **SEARCH** returns a positive number (**TRUE**) while a failed search returns `#VALUE!` (**FALSE**), the combination works seamlessly. However, when we need to impose multiple criteria simultaneously--requiring a cell to contain Keyword A **AND** Keyword B--we simply chain multiple **SEARCH** functions together as separate filter conditions within the same **FILTER** formula. Because the [FILTER function](#) defaults to an **AND** logic (meaning all conditions must be met), this structure allows for highly precise and conjunctive filtering of large [datasets](#).

It is important to note the difference between **AND** logic and **OR** logic in this context. Using multiple **SEARCH** functions as separate arguments in the **FILTER** function creates an **AND** relationship: a row is returned only if *all* [SEARCH functions](#) successfully find their respective terms. If we wished to implement **OR** logic--where a row is returned if Keyword A *or* Keyword B is present--we would

need to adjust the syntax by summing or adding the results of the **SEARCH** functions and using the IFERROR function to handle the necessary error conversions, but the basic structure for the common **AND** condition remains the elegant chaining of multiple **SEARCH** instances.

Implementing Complex Multi-Criteria SEARCH Operations

For users who need to perform highly specific data extraction, requiring a single text field to contain several distinct terms simultaneously, the multi-criteria **SEARCH** operation is indispensable. This setup is particularly common in data cleaning, classification tasks, or inventory management where items might be tagged with multiple descriptors in a single cell. The core concept relies on the FILTER function enforcing the logical requirement that every individual **SEARCH** condition must resolve to a positive result for the entire row to be included in the output. This guarantees that only records meeting the stringent, comprehensive criteria are returned.

To visualize this requirement, consider the specific syntax necessary to achieve this conjunction: you list the range you want to return first, followed by the first **SEARCH** criterion, and then subsequent **SEARCH** criteria are added sequentially, separated by commas. Each comma acts as an implicit **AND** operator within the standard FILTER function structure. This methodical approach ensures that the output is precise and narrowly defined. Below is the basic structure demonstrating how to use the SEARCH function with multiple values in Google Sheets, requiring two specific strings, "Backup" and "Guard", to be present within the defined range.

You can use the following basic syntax to use the **SEARCH** function with multiple values in Google Sheets:

```
=FILTER(A2:A10, SEARCH("Backup", A2:A10), SEARCH("Guard", A2:A10))
```

This particular example will return all cells in the range **A2:A10** that contain both the string "Backup" and "Guard" somewhere in the cell.

The following example shows how to use this syntax in practice.

Practical Walkthrough: Filtering a Dataset Using Multiple Keywords

To illustrate the practical application of this combined **SEARCH** and FILTER methodology, let us consider a typical scenario involving a sports dataset. Suppose we maintain a spreadsheet detailing basketball players, where their position often includes multiple descriptors (e.g., "Starting Forward," "Backup Point Guard"). Our objective is to isolate players who specifically fill a backup role as a guard--meaning their position description must contain both the term "Backup" and the term "Guard." We are interested only in the contents of the 'Player' column (Range A2:A10) but the search criteria must be applied to the 'Position' column (Range A2:A10).

The initial dataset is structured with player names in column A and their positional descriptions in column B, along with ancillary data like points scored in column C. It is crucial to identify the exact range where the filtering criteria will be applied (the 'Position' column in this case) and the range whose values we wish to retrieve (the 'Player' column). This clear definition prevents common errors where users inadvertently apply the search criteria to the column they are trying to return, which may lead to incorrect or empty results. We will focus our criteria searches on the specific range of cells that holds the positional strings, which is **A2:A10** in the example below.

Observe the initial sample dataset, which provides the foundation for our filtering exercise. Note that some players have only one positional descriptor, while others have two. Our goal is to retrieve only the names of the players whose position column satisfies both conditions simultaneously, demonstrating the power of conjunctive logic enforced by the chained **SEARCH** functions within the FILTER function. This approach ensures maximum specificity in data retrieval from complex tables.

Suppose we have the following dataset in Google Sheets that contains information about various basketball players:

	A	B	C	D
1	Position	Points		
2	Starting Shooting Guard	22		
3	Starting Center	12		
4	Starting Point Guard	15		
5	Backup Shooting Guard	5		
6	Backup Center	4		
7	Backup Point Guard	9		
8	Backup Power Forward	6		
9	Starting Power Forward	28		
10	Backup Shooting Guard	7		
11				
12				
13				
14				
15				
16				
17				
18				
19				

Step-by-Step Formula Application and Execution

Following the visual confirmation of our dataset, we can proceed to construct and apply the filtering formula. Our requirement is strictly to find all cells in the Position column (A2:A10) that contain the exact string "Backup" **AND** the string "Guard" somewhere within the cell's text. We want the resulting output to list only the player names, which are also located in the A2:A10 range. We will input the resulting formula into an empty cell, typically outside the original dataset, such as cell **D1**, to prevent overwriting existing data and to clearly display the filtered result set.

The formula begins with the FILTER function, defining the output range first (A2:A10, containing the player names). This is immediately followed by the first criterion: `SEARCH("Backup", A2:A10)`. This instructs the function to check the position column for the presence of the word "Backup." Critically, we then add the second criterion, separated by a comma: `SEARCH("Guard", A2:A10)`. Since these are sequential arguments within FILTER, this creates the necessary **AND** relationship. A row will only pass the filter if both SEARCH functions return a positive position value (i.e., TRUE) for the corresponding row in the A2:A10 range.

After inputting the formula into cell **D1** and pressing Enter, Google Sheets executes the array operations instantly. The results shown in the output range (starting at D1) will be a dynamic list of the player names whose positional descriptions satisfied both criteria. The outcome clearly demonstrates that the formula correctly identified and returned the specific players whose roles included both "Backup" and "Guard." This successful execution validates the use of chained **SEARCH** functions as an efficient method for implementing complex, multi-layered text filtering rules across large spreadsheet arrays.

Now suppose we would like to find all cells in the Position column that contain both the word "Backup" and "Guard" somewhere in the cell.

We can type the following formula into cell **D1** to find each of these cells:

```
=FILTER(A2:A10, SEARCH("Backup", A2:A10), SEARCH("Guard", A2:A10))
```

The following screenshot shows how to use this formula in practice:

D1 fx =FILTER(A2:A10, SEARCH("Backup", A2:A10), SEARCH("Guard", A2:A10))

	A	B	C	D	E
1	Position	Points		Backup Shooting Guard	
2	Starting Shooting Guard	22		Backup Point Guard	
3	Starting Center	12		Backup Shooting Guard	
4	Starting Point Guard	15			
5	Backup Shooting Guard	5			
6	Backup Center	4			
7	Backup Point Guard	9			
8	Backup Power Forward	6			
9	Starting Power Forward	28			
10	Backup Shooting Guard	7			
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

The formula correctly returns the three cells that each contain the word "Backup" and "Guard" somewhere in the cell.

Expanding Retrieval: Returning Multiple Columns

In many analytical scenarios, merely retrieving the filtered key identifier (like the player name) is insufficient. Analysts often require the retrieval of the entire row or several adjacent columns associated with the filtered results. Fortunately, adjusting the syntax to return multiple columns is straightforward, requiring only a minor modification to the primary argument of the FILTER function, while leaving the conditional SEARCH functions unchanged.

To retrieve both the Player Name (Column A) and the Points Scored (Column B), we simply need to widen the initial range specified in the **FILTER** function. Instead of specifying `A2:A10` as the range to return, we adjust it to `A2:B10`. This tells Google Sheets that for every row that satisfies the subsequent **SEARCH** conditions, it should output the contents of both Column A and Column B. Crucially, the criteria ranges remain fixed to the column being searched (`A2:A10`, the Position column), regardless of how wide the output range is defined.

By making this minor adjustment, the function transforms from a simple list extractor into a row retriever, providing comprehensive information about the filtered entities. This flexibility underscores the robust nature of the array formulas in Google Sheets, allowing users to tailor their output structure without needing to rewrite complex logical criteria. The example below shows how this expansion is implemented to return the player's name and their corresponding points:

If you'd like to return the values in the points column for each of these players as well, simply change the filter range from **A2:A10** to **A2:B10** as follows:

=FILTER(A2:B10, SEARCH("Backup", A2:A10), SEARCH("Guard", A2:A10))

The following screenshot shows how to use this formula in practice:

D1 fx =FILTER(A2:B10, SEARCH("Backup", A2:A10), SEARCH("Guard", A2:A10))

	A	B	C	D	E
1	Position	Points		Backup Shooting Guard	5
2	Starting Shooting Guard	22		Backup Point Guard	9
3	Starting Center	12		Backup Shooting Guard	7
4	Starting Point Guard	15			
5	Backup Shooting Guard	5			
6	Backup Center	4			
7	Backup Point Guard	9			
8	Backup Power Forward	6			
9	Starting Power Forward	28			
10	Backup Shooting Guard	7			
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Advanced Considerations: Using SEARCH for OR Logic and Error Handling

While the combination of multiple **SEARCH** arguments in a FILTER function defaults to **AND** logic, achieving **OR** logic--where any one of the criteria must be met--requires slightly more complex

manipulation of the resulting arrays. To implement **OR** logic with SEARCH, one must ensure that the output of either successful search is counted as a positive result. This is often achieved by summing the results of the **SEARCH** functions. Since a successful SEARCH returns a positive number and a failed one returns an error, this requires wrapping each **SEARCH** function within an ISNUMBER or IFERROR function to convert the results into clean numeric (1 or 0) or Boolean (TRUE or FALSE) values before summing them.

A common approach for **OR** logic involves using `ISNUMBER(SEARCH("Term1", Range)) + ISNUMBER(SEARCH("Term2", Range))`. If the sum is greater than zero, it means at least one term was found, satisfying the **OR** condition. This summed result is then used as the single criterion in the FILTER function. Furthermore, when dealing with arrays, error handling is critical. The `#VALUE!` error generated by a failed **SEARCH** can sometimes interrupt the calculations of array functions. To preempt this, integrating `IFERROR(SEARCH("Term", Range), 0)` is a highly robust practice. This converts any error resulting from a failed search into a zero, ensuring that the filter calculation proceeds smoothly without interruption, particularly when performing arithmetic operations for **OR** logic.

For even more robust text matching, particularly when dealing with complex patterns, Google Sheets offers the REGEXMATCH function. While **SEARCH** handles simple, case-insensitive string matching, REGEXMATCH allows for the use of Regular Expressions--powerful patterns that can match arbitrary combinations of characters, structure, and position. Although slightly more complex to learn, REGEXMATCH can often consolidate multiple **SEARCH** conditions into a single expression, providing a highly scalable and efficient alternative for advanced text data manipulation within large datasets. Choosing between **SEARCH** and REGEXMATCH depends on the complexity and volume of the required text matching tasks.