Excel: The Difference Between SEARCH vs. FIND Functions

Authored by

Mohammed looti

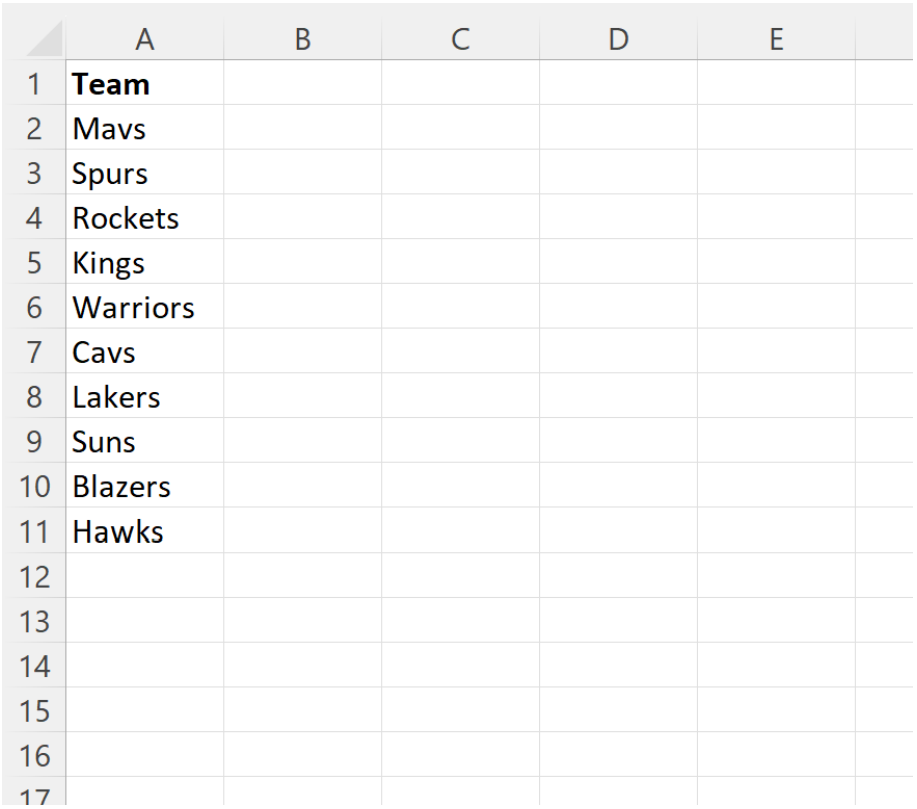
November 10, 2025

In the expansive toolkit of Microsoft **Excel**, manipulating and analyzing data often requires pinpointing specific characters or patterns within larger blocks of text. Both the **SEARCH** function and the **FIND** function are designed to locate the starting position of one **text string** within another. While their core purpose is identical--returning a numerical position--they are fundamentally different in two critical areas that dictate which one you should use in various data scenarios.

Understanding these distinctions is crucial for robust formula construction and accurate data processing. The two primary differences lie in their handling of capitalization and their ability to interpret special characters. Specifically:

The **SEARCH** function ignores capitalization, meaning it is not **case-sensitive**. The **FIND** function, however, requires an exact match, making it strictly **case-sensitive**.
The **SEARCH** function is capable of recognizing **wildcard characters** (such as the asterisk `*` and the question mark `?`) to perform approximate or pattern-based matching. The **FIND** function treats all input characters literally and does not support **wildcard characters**.

The following detailed examples, utilizing a small data set of basketball team names, will clearly illustrate how these differences manifest when applied in practice, helping you choose the correct function for your specific data cleansing and analysis needs.



	A	B	C	D	E
1	**Team**				
2	Mavs				
3	Spurs				
4	Rockets				
5	Kings				
6	Warriors				
7	Cavs				
8	Lakers				
9	Suns				
10	Blazers				
11	Hawks				
12					
13					
14					
15					
16					
17					

The Core Distinction: Handling of Case Sensitivity

The most immediate and frequently encountered distinction between these two functions centers on how they handle upper and lower case letters. When working with large datasets, especially those compiled from varying sources or manual entry, capitalization inconsistencies are common. This is where choosing between **SEARCH** and **FIND** becomes paramount to ensuring the reliability of your results. If your objective is simply to determine if a character exists and where it starts, regardless of its case, **SEARCH** offers unparalleled flexibility and ease of use.

The **FIND** function, by contrast, operates under a strict principle of textual identity. If you instruct **FIND** to locate a lowercase "a," it will only recognize and return the position of a lowercase "a." It will completely ignore an uppercase "A" that might appear earlier in the string, treating it as a different character entirely. This meticulous approach is invaluable when performing advanced parsing or validation where textual fidelity must be maintained, such as locating specific codes or syntax elements where case matters deeply.

Therefore, the choice often boils down to data quality tolerance. If you are dealing with user input where mistakes in capitalization are expected, or if you simply need a general location of a substring without worrying about precise case matching, **SEARCH** is the clear winner. If, however, you must ensure that the located substring matches your criteria exactly--capitalization included--then the reliability and strictness of **FIND** are essential for preventing logical errors in subsequent calculations or operations that depend on that positional data.

Practical Demonstration: Case Sensitivity in Action

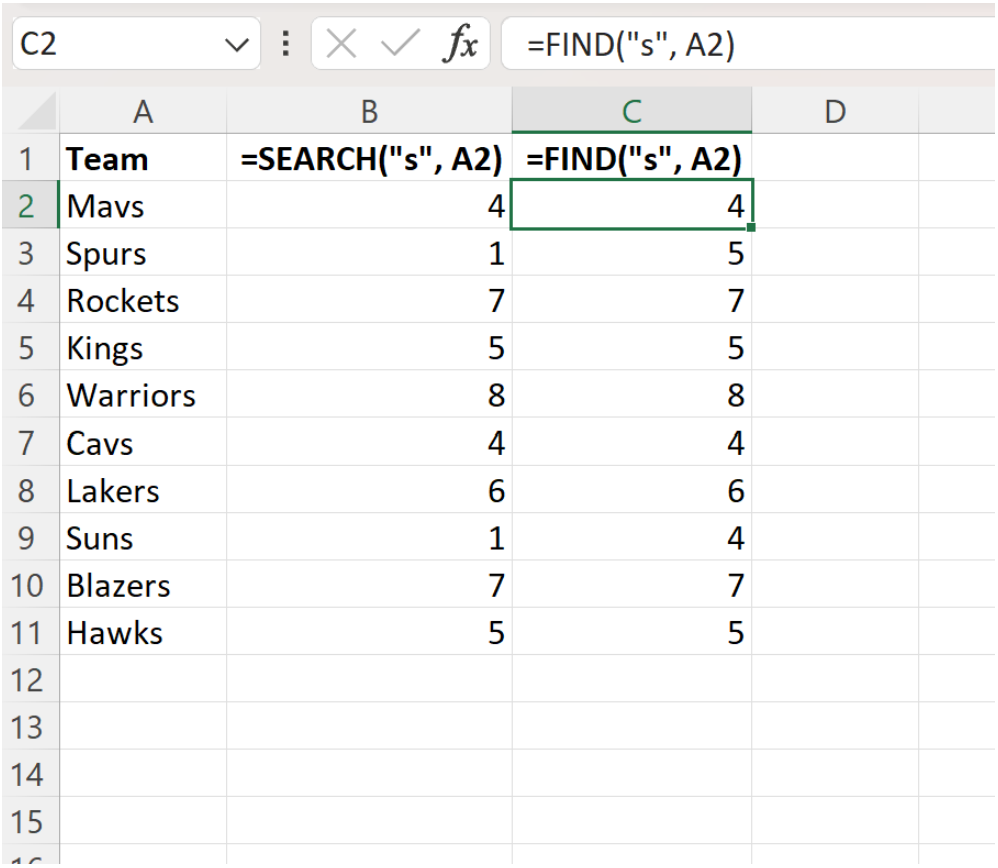
To fully grasp the implications of case sensitivity, let us examine how both functions handle the task of finding the first occurrence of the letter "s" within our list of team names. We are specifically looking for the lowercase character "s" in this scenario. We anticipate that this distinction will cause the functions to return different starting positions for names that begin with a capital "S," such as "Spurs."

We will implement the following formulas in columns B and C, applying them to the data in column A. In cell B2, we utilize **SEARCH**, which ignores case, and in cell C2, we use **FIND**, which enforces it. The goal is to identify the position of the first "s" in each team name.

We will input the subsequent formulas into cells **B2** and **C2**, and then apply them to the rest of the data set:

B2: **=SEARCH("s", A2)**
C2: **=FIND("s", A2)**

After entering these formulas, we drag them down through the respective columns to analyze the results across the entire list. The visual output clearly demonstrates the functional difference:



	A	B	C	D
1	**Team**	**=SEARCH("s", A2)**	**=FIND("s", A2)**	
2	Mavs	4	4	
3	Spurs	1	5	
4	Rockets	7	7	
5	Kings	5	5	
6	Warriors	8	8	
7	Cavs	4	4	
8	Lakers	6	6	
9	Suns	1	4	
10	Blazers	7	7	
11	Hawks	5	5	
12				
13				
14				
15				

Observe the result for the team name **Spurs**. Because the **SEARCH** function is not case-sensitive, it treats the initial capital "S" and the search term "s" as matches. Consequently, it returns **1**, indicating the letter "S" is found at the very first position. Conversely, the **FIND** function, being strictly case-sensitive, ignores the initial capital "S" and continues scanning the string until it encounters the first true lowercase "s," which is found at position **5**. This example powerfully illustrates that when the case of a character is variable or unknown, **SEARCH** provides a general location, while **FIND** provides a definitive, exact location based on the specified case.

Unlocking Flexibility: The Power of Wildcard Characters

The second fundamental divergence between these functions is the support for **wildcard characters**, a feature that significantly enhances the pattern matching capabilities of the **SEARCH** function. Wildcards allow users to search for text patterns rather than specific, rigid strings. Excel primarily supports two wildcards: the asterisk (`*`), which represents any sequence of characters (zero or more), and the question mark (`?`), which represents any single character. This capability transforms **SEARCH** into a robust tool for fuzzy matching and data validation, where the exact

content surrounding a key substring is irrelevant or unknown.

For instance, if you wanted to find the position of any word in a string that starts with "A" and ends with "E," regardless of the three characters in between, you could use a pattern like `A???E` with the **SEARCH** function. This flexibility allows analysts to quickly identify and extract data based on structural properties rather than memorized sequences. This is particularly useful in databases where identifiers or codes follow a known format but may have minor, variable elements.

The **FIND** function, however, treats the asterisk (`*`) and the question mark (`?`) as literal characters. If you attempted to search for `?rs` using **FIND**, it would search for the actual three-character string: question mark, followed by 'r', followed by 's'. Since this literal sequence is unlikely to exist in most text, **FIND** would typically fail to return a position, demonstrating its strict adherence to literal matching and its lack of support for generalized pattern recognition.

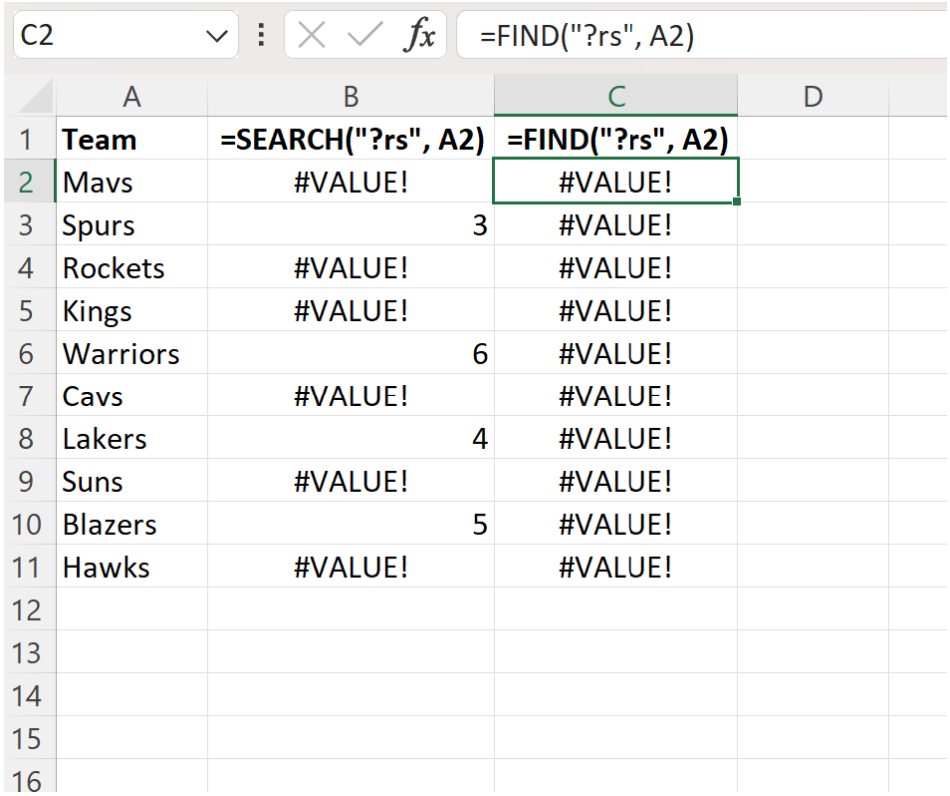
Practical Demonstration: Wildcards and Error Handling

Consider a scenario where we want to find the position of the substring "rs" in each team name, but we know that it is preceded by exactly one character--any character, regardless of what it is. This is a perfect application for the single-character wildcard, the question mark (`?`). We define our search pattern as `?rs`, meaning we are looking for the sequence of any character followed immediately by "rs."

We will input the following formulas into cells **B2** and **C2** to test the wildcard capability, and then copy them down the columns:

B2: **=SEARCH("?rs", A2)**
C2: **=FIND("?rs", A2)**

The resulting visual output demonstrates a dramatic difference in functionality:



	A	B	C	D
1	**Team**	**=SEARCH("?rs", A2)**	**=FIND("?rs", A2)**	
2	Mavs	#VALUE!	#VALUE!	
3	Spurs	3	#VALUE!	
4	Rockets	#VALUE!	#VALUE!	
5	Kings	#VALUE!	#VALUE!	
6	Warriors	6	#VALUE!	
7	Cavs	#VALUE!	#VALUE!	
8	Lakers	4	#VALUE!	
9	Suns	#VALUE!	#VALUE!	
10	Blazers	5	#VALUE!	
11	Hawks	#VALUE!	#VALUE!	
12				
13				
14				
15				
16				

The **SEARCH** function successfully interprets the **?** as a wildcard placeholder. For the string "Spurs," the pattern `?rs` matches the substring "urs" (where 'u' is the single placeholder character). Since this pattern starts at the third character of "Spurs," **SEARCH** correctly returns **3**. This confirms that the **SEARCH** function is capable of complex, pattern-based positional identification.

In stark contrast, the **FIND** function treats the search term `?rs` literally. Since none of the team names contain an actual question mark character followed by "rs," the function fails to locate the string in any row. This failure is communicated by the return of the **#VALUE!** error. The appearance of **#VALUE!** in this context signifies that the input value (the search string containing the wildcard) cannot be processed by the function in the way intended by the user, highlighting the severe limitation of **FIND** when dealing with non-literal pattern matching.

Summary of Syntax and Recommended Use Cases

Choosing between **SEARCH** and **FIND** depends entirely on the requirements of your data analysis task. Both share the same syntax structure: `FUNCTION(find_text, within_text, )`, where `find_text` is the string you are looking for, `within_text` is the cell or string being searched, and is an optional argument specifying where the search should begin (defaulting to 1).

However, their internal processing logic necessitates distinct applications. If your data analysis requires speed, general identification, and tolerance for case variations, **SEARCH** is highly

recommended. It is typically the superior choice for cleaning up messy datasets, identifying the presence of keywords regardless of capitalization, or performing lookups where you only know a pattern of characters rather than the exact string. Its ability to utilize **wildcard characters** makes it the most flexible text-positioning function in Excel.

Conversely, use **FIND** exclusively when precision is non-negotiable. If you are extracting substrings based on specific markers, validating data formats, or linking formulas that depend on the exact case of a character, the strict, **case-sensitive** nature of **FIND** provides the necessary assurance of accuracy. In summary, **SEARCH** is for general matching, and **FIND** is for exact matching.

Further Resources for Excel Mastery

Mastering these text manipulation functions opens the door to powerful data extraction and reporting techniques within Excel. Understanding the subtle yet critical differences between **SEARCH** and **FIND** ensures that your formulas are robust and perform reliably, regardless of the complexity or consistency of the source data. Continue to build your expertise by exploring other related functions that complement these positional tools.

The following tutorials explain how to perform other common tasks in Excel, often in conjunction with string-finding methods like those discussed above, such as extracting the text once the position has been identified: