how to Check if String Contains Another String in VBA?

Authored by

stats writer

November 21, 2025

VBA (Visual Basic for Applications) serves as the powerful scripting language embedded within Microsoft Office applications, such as Excel, Word, and Access. Its primary utility lies in enabling users to automate complex, repetitive tasks and customize application functionalities beyond standard capabilities. One of the most fundamental operations in any programming environment is the ability to manipulate and analyze text data, often referred to as strings. Efficiently determining whether a larger block of text contains a specific sequence of characters--a substring--is a crucial skill for data processing and validation.

Introducing the InStr() Function for Substring Checks

The core mechanism used within VBA to determine if one string resides within another is the built-in **Instr() function**. The name "Instr" stands for "In String," and its sole purpose is to return the starting position of the first occurrence of one string inside another. This simple positional return value provides the definitive answer regarding containment, moving beyond simple boolean checks.

Unlike functions that merely return **True** or **False**, **Instr()** returns a numeric value. If the substring is successfully found, the function returns an integer greater than zero, indicating the character position where the search string begins. Crucially, if the substring is not present anywhere in the larger text, the function returns **0** (zero). This numerical output allows for precise control flow in your VBA code, enabling you to execute specific actions based on where the substring is located, or if it exists at all.

Leveraging **Instr()** is far superior to attempting manual character-by-character comparison loops, especially when dealing with large volumes of data. It is highly optimized and standard across all VBA environments, ensuring that your code is both efficient and easily maintainable for tasks like data validation and text automation.

Detailed Syntax and Parameters of InStr()

To effectively utilize the **Instr()** function, it is essential to understand its required and optional parameters. The basic structure defines how the search should be conducted, specifying where it starts, what text to search, and what text to look for.

The standard syntax for the function is as follows:

Instr(, string1, string2,)

Here is a breakdown of what each parameter represents:

start: This is an optional argument specifying the numeric position within **string1** where the search should begin. If omitted, the search starts from the first character (position 1).

string1: This is the required argument representing the primary string (the larger body of text) within which you are searching.
string2: This is the required argument representing the substring you are actively attempting to find within **string1**.
compare: This is an optional argument that defines the type of string comparison to be performed. It can be set to **vbBinaryCompare** (case-sensitive, default) or **vbTextCompare** (case-insensitive).

The return value of **Instr()** dictates the outcome of the containment check. If the function returns any value greater than 0, it confirms that **string2** is present in **string1**. Conversely, a return value of 0 immediately confirms that the target substring was not found within the text being examined. This numerical result must be tested (e.g., using `<> 0`) within conditional statements to drive the logic of your VBA macro.

Example: Use VBA to Check if String Contains Another String

To illustrate the practical use of **Instr()**, let us consider a common scenario in data analysis involving a dataset housed in a Microsoft Excel worksheet. We have a list of team names, and we want to programmatically extract only those rows where the team name contains a specific sequence of letters.

Suppose we have the following dataset starting in column A of our Excel sheet, detailing information about various basketball teams:

	A	B	C	D	E	F
1	**Team**	**Points**				
2	Mavs	100				
3	Lakers	114				
4	Cavs	94				
5	Spurs	99				
6	Rockets	104				
7	Hornets	115				
8	Warriors	100				
9	Magic	103				
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

Our objective is to identify which teams contain the specific substring "avs" anywhere in their name. This search must be automated, and any matching rows (Team Name and Score) must be copied over to a dedicated output area, specifically starting in columns D and E. This process is ideally handled by a VBA macro.

We will implement a loop structure that iterates through each team name in the "Team" column (A2 through A9). Inside the loop, we employ the **Instr()** function to perform the substring check. If the function confirms the presence of "avs", the corresponding row data will be copied. We also utilize an independent counter variable (`i_num`) to manage the destination row placement, ensuring that the filtered results are presented sequentially in columns D and E without any gaps.

Analyzing the VBA Code for String Containment

We can use the following syntax in VBA to check if each string in the Team column contains "avs" and, if so, return the entire row in the columns D and E:

Sub StringContains()
Dim i As Integer, i_num As Integer

For i = 2 To 9

```
If InStr(1, LCase(Range("A" & i)), "avs") <> 0 Then
i_num = i_num + 1
Range("D" & i_num & ":E" & i_num) = Range("A" & i & ":B" & i).Value
End If
Next i
End Sub
```

The code initializes two integer variables: **i** serves as the row counter for iterating through the source data (rows 2 through 9), and **i_num** acts as the destination row counter, initialized implicitly to zero. The **For** loop systematically checks every relevant row. The conditional statement, `If InStr(...) <> 0 Then`, is the core logic. It executes the search and determines if a match occurred. The comparison `<> 0` explicitly checks for any positive position return, confirming containment.

If the **Instr()** function returns a value other than zero, the conditional block is executed. First, **i_num** is incremented, ensuring we move to the next available output row. Second, the entire range from columns A and B (the source data for the current row **i**) is copied directly into columns D and E at the position specified by **i_num**. This efficient data transfer mechanism highlights the power of VBA for data manipulation.

Handling Case Sensitivity with LCase()

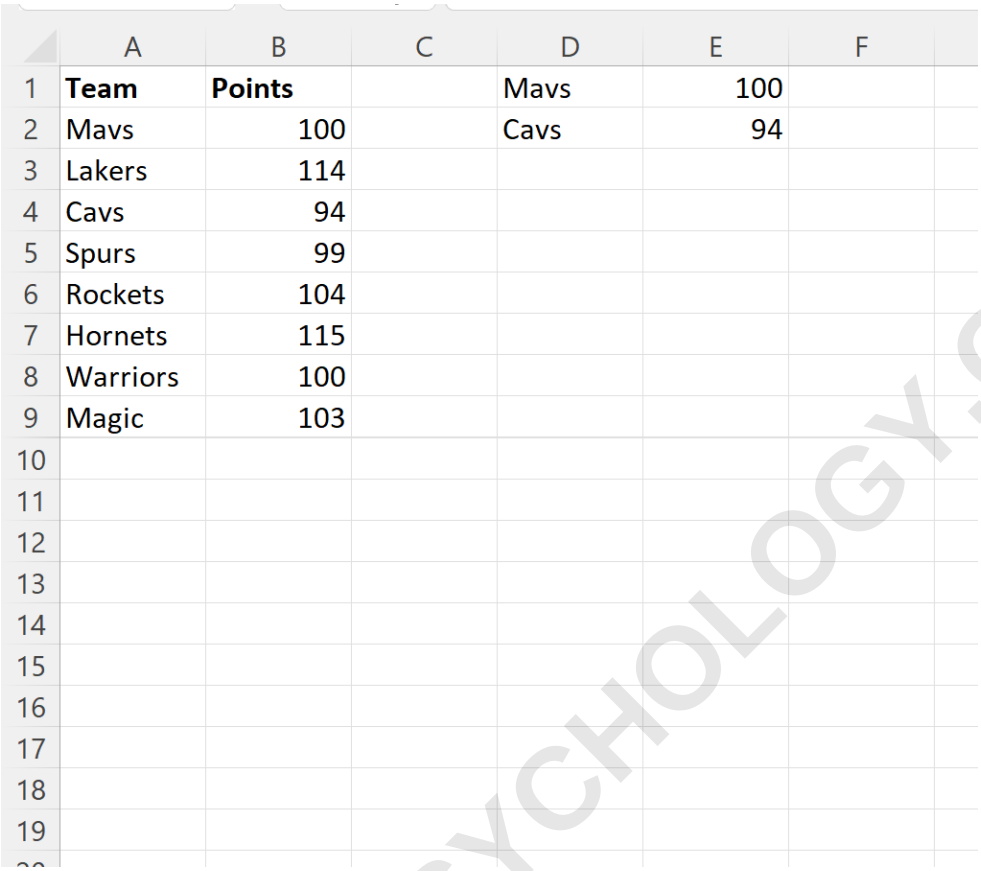
A critical consideration when performing string comparisons in VBA is case sensitivity. By default, the **Instr() function** uses **vbBinaryCompare**, meaning it is case-sensitive. If our goal is to find "avs" regardless of how it is capitalized in the source data (e.g., "Mavs" vs. "MAVS"), we must explicitly manage this aspect.

In the provided code example, we achieve case-insensitivity through the use of the **LCase()** function: `LCase(Range("A" & i))`. The **LCase()** function converts the entire content of the cell (**string1**) to lowercase before it is passed to **Instr()**. Since the search substring (**string2**), which is "avs", is also specified in lowercase, the comparison becomes inherently case-insensitive, ensuring that both "Mavs" and "Cavs" are identified correctly.

While using the optional fourth parameter **vbTextCompare** is another valid method for case-insensitive searching, applying **LCase()** directly to the input string provides visual confirmation of the normalization step, often leading to code that is easier to debug and maintain when complex string cleaning operations are involved.

Interpreting the Output and Expanding Search Criteria

When we execute this macro, we receive the following output:



	A	B	C	D	E	F
1	**Team**	**Points**		Mavs	100	
2	Mavs	100		Cavs	94	
3	Lakers	114				
4	Cavs	94				
5	Spurs	99				
6	Rockets	104				
7	Hornets	115				
8	Warriors	100				
9	Magic	103				
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

The output successfully displays only the rows that contained the substring "avs" somewhere in the Team column. In this example, only two teams (Mavs and Cavs) satisfied the containment criteria, confirming the successful execution of the search and data transfer logic.

To search for a different substring, the process is straightforward: simply replace the string literal "avs" within the code block with the new target string of your choice. This modularity allows the same subroutine to be adapted quickly for various filtering and data extraction tasks across different datasets.

The power of **Instr()** lies in its flexibility. By integrating it into conditional logic, you can easily build sophisticated data processing systems that automatically categorize, filter, or manipulate text based on the presence of predefined keywords or partial matches.

Further Resources and Documentation

The **Instr() function** is a cornerstone of string handling in VBA. For advanced usage, including

detailed specifications on the `compare` argument and handling edge cases (such as searching within null strings), consult the official Microsoft documentation.

The following tutorials explain how to perform other common tasks using VBA: