How to Use the “If Contains” Formula in Google Sheets

Authored by

stats writer

December 5, 2025

The ability to check if a specific substring exists within a larger text string is a fundamental requirement in data analysis and manipulation across various spreadsheet platforms. In Google Sheets, achieving this "If Contains" logic requires combining several powerful functions. The most straightforward and classic formula for determining if a cell contains a specific piece of text is: `=IF(ISNUMBER(SEARCH(text, cell)),"Yes","No")`. This elegant construction checks for the presence of the desired content (text) within a target cell, returning "Yes" if the substring is found and "No" if it is absent. This method leverages the strengths of the IF function, conditional logic, and the position-finding capabilities of the SEARCH function to deliver a clear, binary result.

While the combination of SEARCH and ISNUMBER function is highly effective, modern data operations often benefit from the precision offered by regular expressions. Google Sheets provides a dedicated function, REGEXMATCH, specifically designed to handle pattern matching. This approach simplifies the formula structure significantly, making the intent immediately clearer when dealing with complex patterns or simple substring checks. Using REGEXMATCH allows users to implement the "If Contains" logic with far greater flexibility, especially when needing to check for multiple possible strings simultaneously or using advanced wildcards.

To determine if a cell contains a certain string using the powerful regular expression syntax, you can employ the following structure. This method is preferred by many professionals due to its conciseness and native support for pattern matching:

=IF(REGEXMATCH(B1, "this"), 1, 0)

In this specific example, the IF function evaluates the result of REGEXMATCH. If cell B1 successfully matches the string "this" anywhere within its content, the REGEXMATCH returns TRUE, and the formula subsequently returns a **1**. Conversely, if the string is not found, REGEXMATCH returns FALSE, resulting in the output of a **0**. This simple structure provides a boolean (TRUE/FALSE) outcome represented numerically.

The following comprehensive examples will illustrate exactly how to deploy and modify the REGEXMATCH formula to handle both single-string containment checks and more complex scenarios involving multiple possible search terms.

Understanding the Core Logic of "If Contains"

To effectively implement the "If Contains" logic, it is essential to understand that we are performing a **conditional check**. This check relies on a function that searches for a substring and reports back whether or not the search was successful. Both the SEARCH function/ISNUMBER combination and the REGEXMATCH function are designed to achieve this, but they operate using

different mechanisms. The final step is always wrapping this successful/unsuccessful search operation within the powerful IF function, which dictates what value should be returned based on the search result.

When using the IF function, the syntax is universally `IF(logical_expression, value_if_true, value_if_false)`. In the context of "If Contains," the `logical_expression` is the search mechanism itself (e.g., `ISNUMBER(SEARCH(...))` or `REGEXMATCH(...)`). If the search finds the target text, the expression evaluates to **TRUE**, triggering the `value_if_true`. If the text is missing, it evaluates to **FALSE**, triggering the `value_if_false`. This separation of searching and reporting allows for maximum flexibility in output, whether you choose numeric values, text strings, or even other formulas.

Case Sensitivity: SEARCH vs. REGEXMATCH

A critical distinction between the two primary methods of implementing "If Contains" lies in how they handle case sensitivity. The SEARCH function in Google Sheets is inherently **case-insensitive**. This means that searching for "apple" will successfully match "Apple," "APPLE," or "aPpLe." This is often desirable for general data scrubbing where capitalization errors are common, but it can be restrictive when exact matching is required.

Conversely, the standard REGEXMATCH function is generally **case-sensitive**. Searching for "Lakers" using REGEXMATCH will not match "lakers." For precise technical matching, REGEXMATCH offers superior control. If you require a case-insensitive match using REGEXMATCH, you must use a specific regular expression modifier, often prefixed with `(?i)` at the start of your pattern, although for simple containment checks, many users prefer sticking to the SEARCH function to avoid unnecessary complexity.

Example 1: Using "If Contains" For One Specific String

This first practical scenario demonstrates how to set up a column that conditionally tags data based on the presence of a single, defined keyword. We are using the REGEXMATCH approach here, which is ideal for checking structured data like team names or product codes. The goal is to return a numerical flag (1 or 0) indicating whether a specific team name is present in the data set.

The following formula is applied across a range of cells (starting in B2, for instance) to check if the corresponding cell in column A contains the string "Lakers." If the condition is met, it returns **1**; if not, it returns **0**:

```
=IF(REGEXMATCH(A2, "Lakers"), 1, 0)
```

D2 =IF(REGEXMATCH(B2, "Lakers"), 1, 0)

	A	B	C	D	E
1	**Player**	**Team**	**Points**	**Lakers?**	
2	Andy	Lakers	13.4	1	
3	Bob	Mavericks	7.8	0	
4	Carl	Spurs	13.7	0	
5	Dave	Warriors	22.3	0	
6	Eric	Mavericks	27.8	0	
7	Fred	Mavericks	20.8	0	
8	George	Spurs	12.7	0	
9	Harold	Lakers	8.2	1	
10	Isaiah	Warriors	12.5	0	
11	Joe	Warriors	30.2	0	
12	Ken	Spurs	22.4	0	
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

As shown in the illustration, the formula accurately identifies all instances where the word "Lakers" appears in column A. This is a powerful technique for creating binary filters in large spreadsheets. The use of numeric outputs (1/0) is especially useful when the resulting column needs to be used in calculations, such as summing the count of records that meet the specified criterion.

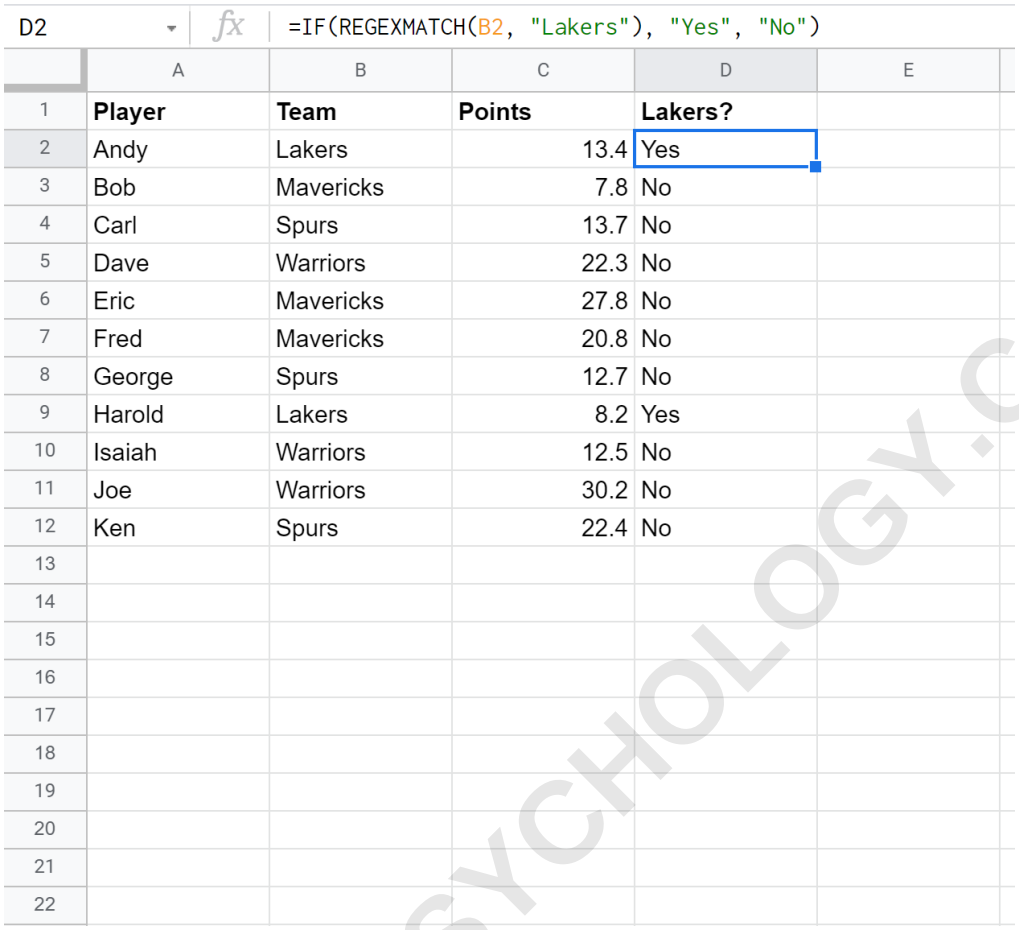
Flexible Output: Returning Text Strings Instead of Numbers

While numeric outputs (1 and 0) are excellent for data processing and aggregation, sometimes a more user-friendly textual label is required for direct readability. The beauty of the IF function is its versatility, allowing us to easily swap out the numeric results for descriptive text strings.

To return the more intuitive labels "Yes" and "No," simply replace the numeric values in the IF function's arguments with the desired strings, ensuring they are enclosed in double quotes. The underlying logic remains identical; only the output display changes:

```
=IF(REGEXMATCH(A2, "Lakers"), "Yes", "No")
```

The image below confirms how this small modification changes the resulting column, making it much clearer to users without requiring mathematical interpretation:



	A	B	C	D	E
1	**Player**	**Team**	**Points**	**Lakers?**	
2	Andy	Lakers	13.4	Yes	
3	Bob	Mavericks	7.8	No	
4	Carl	Spurs	13.7	No	
5	Dave	Warriors	22.3	No	
6	Eric	Mavericks	27.8	No	
7	Fred	Mavericks	20.8	No	
8	George	Spurs	12.7	No	
9	Harold	Lakers	8.2	Yes	
10	Isaiah	Warriors	12.5	No	
11	Joe	Warriors	30.2	No	
12	Ken	Spurs	22.4	No	
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

Example 2: Using "If Contains" For One of Several Strings (OR Logic)

A common requirement in data filtering is checking if a cell contains text matching any value from a list of possibilities. This is known as implementing "OR" logic. With standard functions, this would require nesting multiple IF and OR functions, leading to complex and unwieldy formulas. However, one of the greatest advantages of using the REGEXMATCH function is its ability to handle multiple search terms efficiently using the regular expression OR operator.

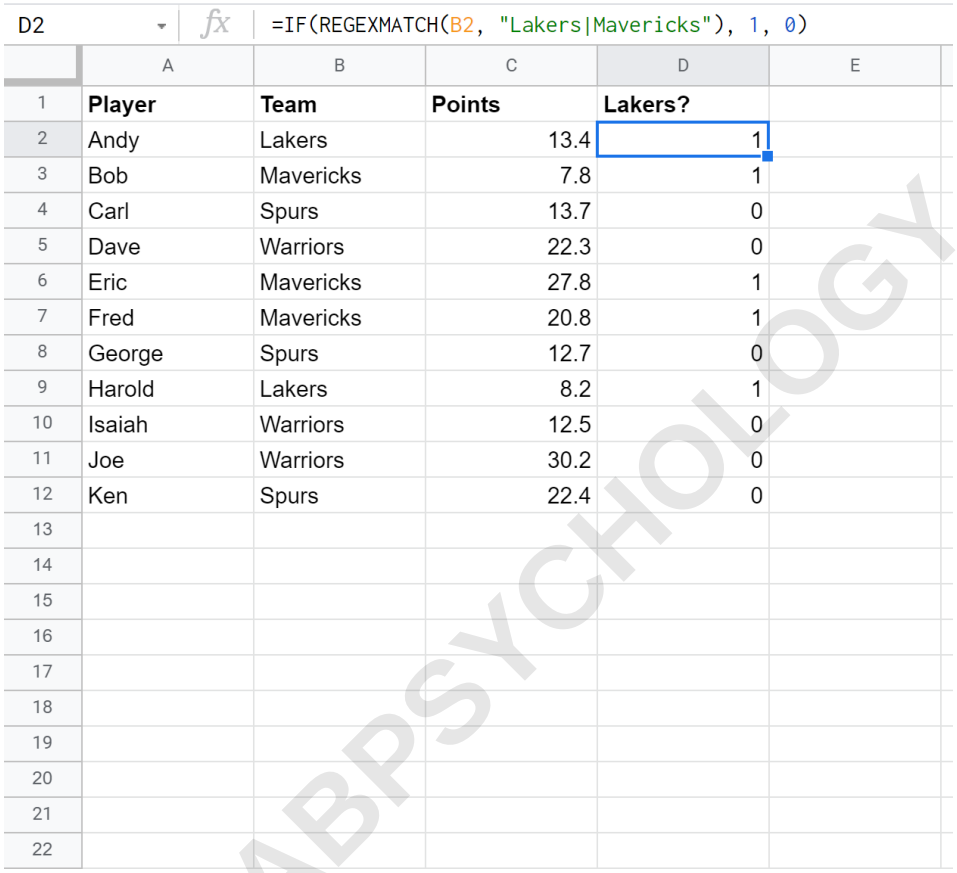
In regular expression syntax, the pipe symbol (|) acts as the OR operator. By placing this symbol between the strings you wish to match, you instruct REGEXMATCH to return TRUE if it finds the first string OR the second string OR any subsequent string defined in the pattern. This significantly simplifies the logic for multi-condition checks.

To demonstrate this, consider a scenario where we want to flag rows associated with either "Lakers" or "Mavericks." The formula is structured by including both terms within the pattern

argument, separated by the pipe:

```
=IF(REGEXMATCH(A2, "Lakers|Mavericks"), 1, 0)
```

This formula will return a **1** if the cell A2 contains the string "Lakers" or the string "Mavericks," and a **0** otherwise. Notice how this single pattern handles the complexity that would otherwise require nested spreadsheet functions.



	A	B	C	D	E
1	**Player**	**Team**	**Points**	**Lakers?**	
2	Andy	Lakers	13.4	1	
3	Bob	Mavericks	7.8	1	
4	Carl	Spurs	13.7	0	
5	Dave	Warriors	22.3	0	
6	Eric	Mavericks	27.8	1	
7	Fred	Mavericks	20.8	1	
8	George	Spurs	12.7	0	
9	Harold	Lakers	8.2	1	
10	Isaiah	Warriors	12.5	0	
11	Joe	Warriors	30.2	0	
12	Ken	Spurs	22.4	0	
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

Advanced Alternatives: Using FILTER, QUERY, and ARRAYFORMULA

While IF and REGEXMATCH are excellent for column-level checks, advanced users often look for solutions that can process entire ranges or filter data dynamically. Google Sheets offers powerful alternatives that integrate containment logic directly into filtering or querying operations.

The QUERY function, for example, allows you to use SQL-like syntax, including the `CONTAINS` or `MATCHES` clause, which simplifies filtering based on substring presence without explicitly using IF logic. Similarly, using the FILTER function combined with REGEXMATCH inside an ARRAYFORMULA allows for the automatic spilling of filtered results across a sheet based on the containment criterion.

These methods are highly recommended when dealing with large datasets or when the goal is not merely to return a flag (1 or 0) but to extract the matching records themselves. For instance, to extract all rows where column A contains "Lakers" or "Mavericks," you could use:

```
=FILTER(A:Z, REGEXMATCH(A:A, "Lakers|Mavericks"))
```

This demonstrates the flexibility and power that REGEXMATCH provides when combined with other array manipulation functions within Google Sheets.

Summary of Best Practices for Substring Checks

Choosing the right formula for "If Contains" depends entirely on the specific requirements of your analysis, primarily focusing on whether you require case sensitivity and how many strings you need to check simultaneously.

For simple, case-insensitive checks involving only one substring, the classic combination `=IF(ISNUMBER(SEARCH(...)), "Yes", "No")` remains reliable and highly efficient.

For checks involving multiple strings or requiring strict **case sensitivity**, or if you plan to use complex patterns (like checking for strings at the beginning or end of a cell), the IF(REGEXMATCH(...), ...) approach is superior due to its direct handling of regular expressions and the logical OR operator (`|`).

When dealing with large datasets where filtering or extraction is the primary goal, integrate REGEXMATCH within FILTER or QUERY functions for dynamic and scalable solutions.

Mastering these techniques ensures that you can efficiently categorize, filter, and analyze textual data within Google Sheets, transforming raw information into actionable insights using powerful conditional logic.