How do I create an [IF FUNCTION] to return [YES] or [NO] in [GOOGLE SHEETS]?

Authored by

stats writer

November 18, 2025

Harnessing the power of conditional statements is fundamental to effective data analysis and automation within spreadsheet applications. Specifically, the IF function is the cornerstone of decision-making formulas in Google Sheets, allowing users to return specific values based on whether a condition is met or not. When tracking compliance, status updates, or success metrics, a simple, clear, binary outcome--such as "Yes" or "No"--provides instant clarity regarding the status of a variable or transaction. This detailed guide explores how to construct and deploy the versatile IF function specifically tailored to generate these straightforward, categorical responses, ensuring your data is not only analyzed but also interpreted with immediate precision.

Understanding the Core Structure of the IF Function

The IF function operates based on classic Conditional logic, requiring three distinct arguments separated by commas. These three components dictate the test being performed, the result if the test is true, and the result if the test is false. By correctly defining these arguments, we can instruct the spreadsheet to perform complex evaluations and return simple, readable text outputs like "Yes" or "No." Achieving this clarity in reporting is often crucial for dashboards and summary reports where stakeholders need quick answers without delving into numerical details.

The generalized syntax for this function is structured as follows, where each part plays a non-negotiable role in the evaluation process. Understanding the role of each argument is the first step toward mastering conditional formatting and reporting: `=IF(logical_expression, value_if_true, value_if_false)`. The entire operation hinges on the first argument, the **logical_expression**, which must resolve to either TRUE or FALSE. The subsequent arguments merely provide the corresponding output based on this resolution.

When aiming for a "Yes" or "No" output, the second and third arguments of the formula will be fixed strings. For instance, we set **value_if_true** to "Yes" and **value_if_false** to "No". It is critical to remember that text strings within any Google Sheets formula must be enclosed in double quotation marks to distinguish them from cell references or numerical constants. This practice ensures that the application interprets "Yes" and "No" as text data rather than attempting to process them as variables or named ranges.

Basic Syntax for a Binary Output

To implement the requirement of returning "Yes" or "No," we use a fundamental formula structure centered on comparing two values or checking a single condition. This setup is highly flexible but fundamentally simple, making it applicable across a vast array of scenarios, from inventory management to academic grading.

You can use the following basic syntax to create an IF function in Google Sheets that returns "Yes" or "No" as a result, assuming we are comparing the contents of cell **A2** against cell **B2** using the

"greater than or equal to" comparison operator:

=IF(A2>=B2, "Yes", "No")

For this particular formula, the **logical_expression** is `A2>=B2`. If the value stored in cell **A2** is indeed greater than or equal to the value in cell **B2**, the function evaluates the expression as TRUE, and subsequently returns the text string "Yes." Conversely, if the value in **A2** is strictly less than the value in **B2**, the expression evaluates as FALSE, and the function returns "No." This structure provides an elegant solution for quickly assessing performance against a set benchmark or target, offering immediate pass/fail visibility to the user.

This simple structure is incredibly powerful because it abstracts complex numerical comparisons into easily digestible text. The consistency of using "Yes" and "No" ensures that downstream analyses, such as counting successes or failures, are straightforward to implement using functions like `COUNTIF` or `SUMIF`, which rely on consistent text criteria. Furthermore, the use of string literals ensures that the output is easily interpretable by any user, regardless of their familiarity with spreadsheet functions.

Practical Application Example: Comparing Sales vs. Targets

To illustrate the practical utility of this conditional setup, consider a scenario common in business operations: tracking sales performance against predetermined targets. Suppose we have a datasheet containing two columns that detail the actual sales achieved and the ambitious sales targets set for ten different products over a specific reporting period. Our goal is to create a third column that explicitly indicates whether the target was met or exceeded.

The initial dataset might look something like the following, where Column A represents Actual Sales and Column B represents the Sales Target:

	A	B	C	D
1	**Sales**	**Sales Target**		
2	10	12		
3	17	15		
4	22	15		
5	24	20		
6	29	30		
7	30	30		
8	30	35		
9	24	40		
10	28	30		
11	35	30		
12				
13				
14				
15				
16				
17				
18				

We can type the following formula into cell **C2**--the first cell in our status column--to return "Yes" if the number of sales in cell **A2** is equal to or greater than the sales target specified in cell **B2**. This comparison precisely defines success as either meeting the goal exactly or surpassing it:

=IF(A2>=B2, "Yes", "No")

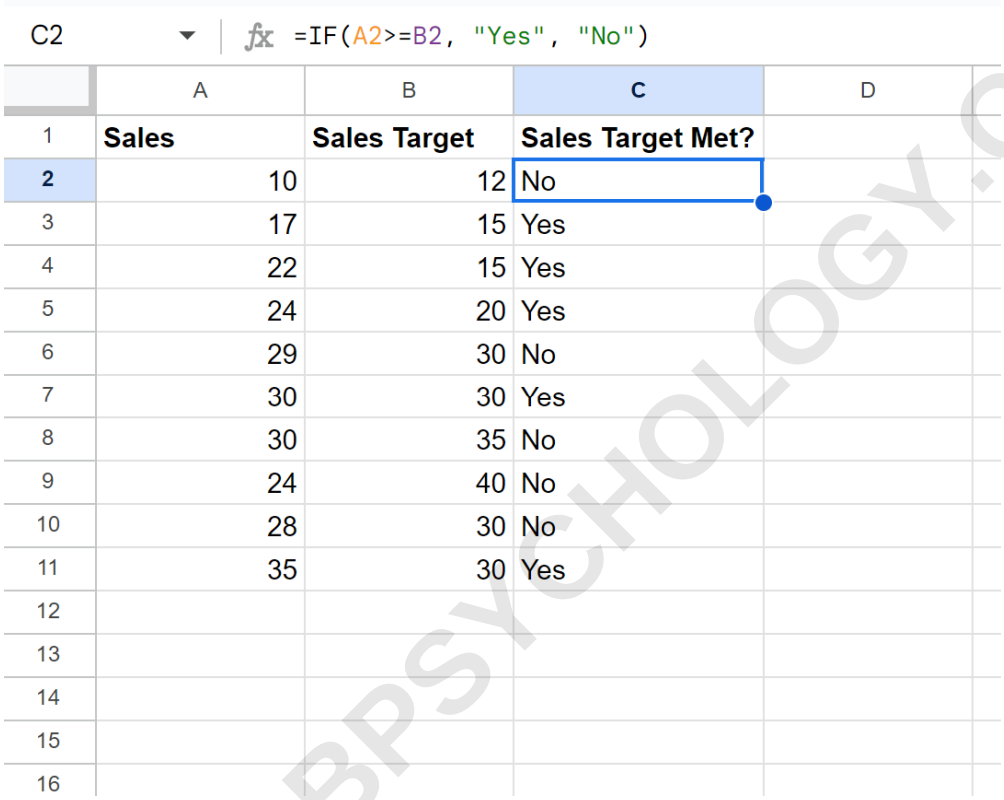
This formula immediately provides a status check for the first product listed. If the value in A2 is 1500 and the value in B2 is 1000, the condition `1500>=1000` is TRUE, and the output is "Yes." If, however, A2 was 900, the condition would be FALSE, yielding "No." This single formula encapsulates the entire logic required for evaluating success based on the criteria of achieving or surpassing the target.

Step-by-Step Implementation and Auto-Filling Formulas

Once the initial formula is accurately constructed in cell C2, the true efficiency of spreadsheet software comes into play through the process of auto-filling. Instead of manually writing and adjusting the formula for each of the subsequent nine rows, we can simply apply the logic down the entire column, thereby automating the status check for all products efficiently.

To propagate the formula, select cell **C2**, locate the small square box (the fill handle) at the bottom right corner of the cell, click and hold, and then drag this fill handle down to cell **C11** (assuming ten rows of data). As the formula is dragged, Google Sheets automatically adjusts the row references from `A2` and `B2` to `A3` and `B3`, and so forth, thanks to the use of relative referencing. This saves significant time and virtually eliminates the risk of manual data entry errors.

After the drag-and-fill operation is complete, the spreadsheet will immediately display the "Yes" or "No" status for every product, providing a clear visual audit of performance against targets:



	A	B	C	D
1	**Sales**	**Sales Target**	**Sales Target Met?**	
2	10	12	No	
3	17	15	Yes	
4	22	15	Yes	
5	24	20	Yes	
6	29	30	No	
7	30	30	Yes	
8	30	35	No	
9	24	40	No	
10	28	30	No	
11	35	30	Yes	
12				
13				
14				
15				
16				

The resulting column C dynamically returns either "Yes" or "No" depending on whether the sales value in the corresponding row of column A is greater than or equal to the sales target in column B. This approach provides an immediate, actionable metric, allowing analysts to quickly filter for products that did not meet their targets or to aggregate the total count of successful outcomes.

Alternative Logical Tests: Checking for Exact Equality

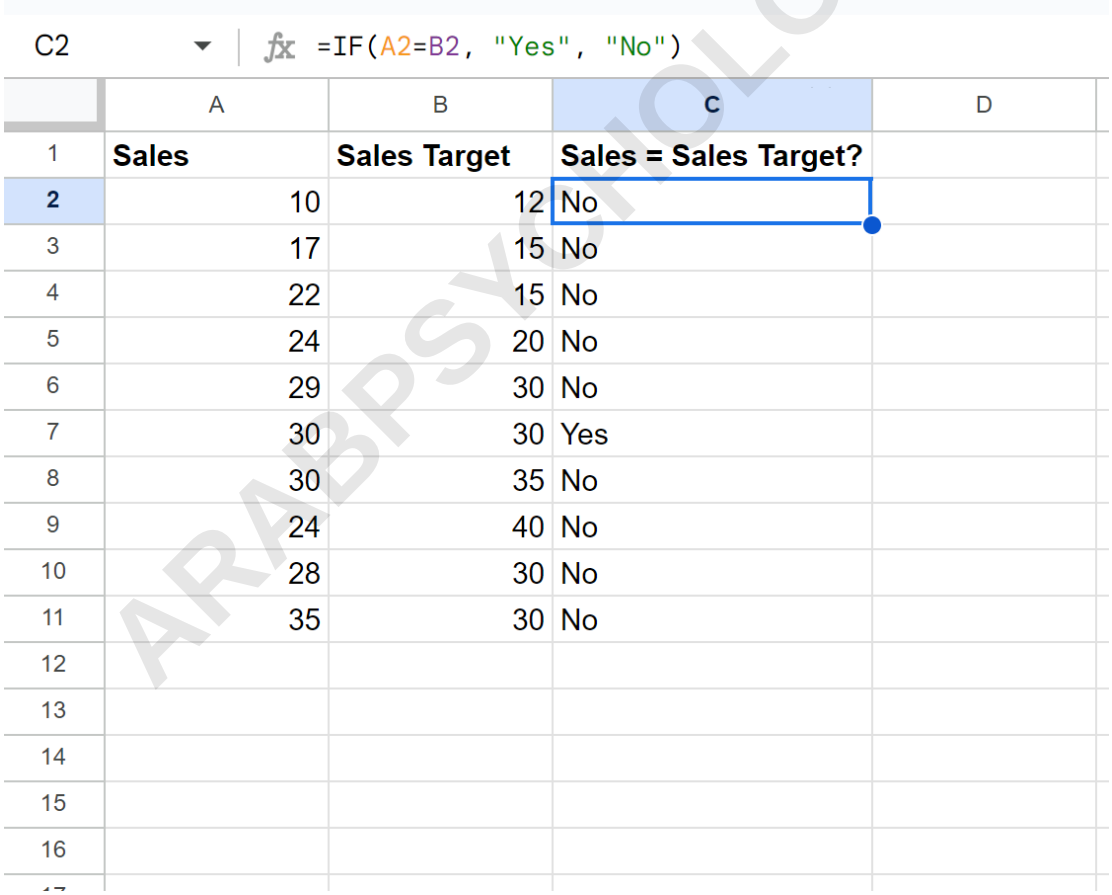
It is important to recognize that the strength of the IF function lies in its flexibility regarding the **logical_expression**. While the previous example used the "greater than or equal to" operator (`>=`), the conditional test can utilize any valid comparison operator. A common requirement is to check for strict equality, where success is defined only if the achieved value perfectly matches the target value, neither exceeding nor falling short.

For example, if the success criterion dictated that actual sales must be exactly equal to the sales target, we would modify the **logical_expression** using the equality comparison operator (`=`). This subtle change in the operator completely shifts the evaluation standard, making the criteria much stricter.

We could use the following formula to test if the values in cell **A2** and **B2** are exactly equal to each other, returning "Yes" if they are identical or "No" if they are not equal:

=IF(A2=B2, "Yes", "No")

Once this new formula is entered into C2, we can again use the drag-and-fill method to apply the new strict equality test across all remaining cells in column C. This recalculation will instantly update the status column based on the revised, stricter rule. Products that previously returned "Yes" because they exceeded the target might now return "No," highlighting the importance of selecting the correct logical test for the desired business outcome.



	A	B	C	D
1	**Sales**	**Sales Target**	**Sales = Sales Target?**	
2	10	12	No	
3	17	15	No	
4	22	15	No	
5	24	20	No	
6	29	30	No	
7	30	30	Yes	
8	30	35	No	
9	24	40	No	
10	28	30	No	
11	35	30	No	
12				
13				
14				
15				
16				

In this revised output, the formula returns "Yes" only if the sales and sales target are exactly equal. Otherwise, if the sales either exceed or fall short of the target, the formula returns "No." This demonstrates the precise control offered by selecting the appropriate comparison operator within

the IF function, ensuring the binary status reflects highly specific business rules.

Expanding Conditional Checks: Utilizing Diverse Comparison Operators

The ability of the IF function to handle various conditional criteria is determined entirely by the comparison operator used in the logical expression. Beyond the basic equality (`=`) and greater than or equal to (`>=`), Google Sheets supports a full suite of standard operators that allow for nuanced conditional testing, all while maintaining the simple "Yes"/"No" binary outcome.

A comprehensive understanding of these operators allows the user to design robust conditional checks. For instance, if you need to determine if a value is strictly below a certain threshold (e.g., inventory falling into a reorder zone), you would use the less than operator (`<`). If you need to ensure a value has been achieved but not surpassed (e.g., staying within a budget limit), the less than or equal to operator (`<=`) is appropriate. Furthermore, the "not equal to" operator (``) is extremely useful when confirming that two values are different from each other, serving as a powerful flag for discrepancies or mismatches.

Here is a list outlining the standard comparison operators available for use in the logical expression of your IF function, each of which can be used to drive a "Yes" or "No" result:

= (Equal to): Checks if two values are identical. Example: `=IF(A2=100, "Yes", "No")`
> (Greater than): Checks if the first value is strictly larger than the second. Example: `=IF(A2>B2, "Yes", "No")`
< (Less than): Checks if the first value is strictly smaller than the second. Example: `=IF(A2<B2, "Yes", "No")`
>= (Greater than or equal to): Checks if the first value is larger than or the same as the second. Example: `=IF(A2>=B2, "Yes", "No")`
<= (Less than or equal to): Checks if the first value is smaller than or the same as the second. Example: `=IF(A2<=10, "Yes", "No")`
(Not equal to): Checks if the two values are different. Example: `=IF(A2B2, "Yes", "No")`

Feel free to use whatever logical test you'd like in the first argument of the **IF** function depending on what specific condition you need to verify. The key remains constant: the logical expression must result in a clear TRUE or FALSE, which then triggers the corresponding "Yes" or "No" output.

Advanced Considerations and Handling Complexity

While the goal of generating a simple "Yes" or "No" output is often best met with a single, straightforward IF function, data complexity sometimes demands more intricate conditional logic. When multiple conditions must be met simultaneously, or when several mutually exclusive conditions must be checked sequentially, techniques like using logical operators (AND, OR) or

nesting IF function statements become necessary.

If, for instance, a transaction is only successful ("Yes") if Sales are greater than Target (A2 > B2) AND Inventory is above zero (C2 > 0), you would combine these requirements using the **AND** function within the logical expression: `=IF(AND(A2>=B2, C2>0), "Yes", "No")`. This structure maintains the simple "Yes"/"No" binary outcome while requiring two distinct conditions to be TRUE before returning "Yes." If either A2 is less than B2 OR C2 is zero or less, the entire AND statement resolves to FALSE, and the function returns "No."

For scenarios involving more than two potential outcomes (e.g., "High," "Medium," "Low"), relying solely on nested IF statements for many conditions can lead to formulas that are cumbersome and difficult to debug. In such cases, Google Sheets offers the `IFS` function, a more modern and streamlined alternative designed to handle multiple conditions sequentially without extensive nesting. However, even when using `IFS`, it is possible and often recommended to maintain the "Yes"/"No" output structure for certain specific outcomes or for general categorization, ensuring that the primary status remains a clear, unambiguous binary outcome.

Summary and Best Practices for Conditional Reporting

The ability to deploy the IF function to return a "Yes" or "No" status is a crucial skill for anyone working with data in Google Sheets. This approach turns raw quantitative data into clear, qualitative assessments, instantly communicating whether a condition has been met. The process is straightforward, relying on three critical elements: a logical expression using a comparison operator, the defined result for TRUE (e.g., "Yes"), and the defined result for FALSE (e.g., "No").

To ensure your conditional reporting is highly effective, always adhere to the following best practices. First, clearly define your success criteria--is success achieved by meeting the target (`=`), exceeding it (`>`), or both (`>=`)? Secondly, ensure that your text outputs ("Yes," "No") are consistently capitalized and spelled identically across all relevant formulas to facilitate accurate counting and filtering later. Finally, utilize the power of relative referencing and the drag-and-fill functionality to apply the formula across large datasets efficiently. By mastering these techniques, you transform simple data tables into powerful, self-auditing performance dashboards, driven by clean and effective Conditional logic.