Learn How to Use Excel Formulas to Check Cell Color and Perform Actions

Authored by

Mohammed looti

November 11, 2025

The Challenge of Conditional Formatting Checks in Excel

In advanced data analysis using Excel, users frequently encounter the need to perform a conditional operation based on a cell's visual properties, such as its background fill color. Standard IF statement functions are designed to evaluate the cell's underlying value, content, or calculated result, but they possess a crucial limitation: they cannot directly read formatting attributes. This constraint means that a simple formula like `=IF(A1 is yellow, "Yes", "No")` is impossible to construct using standard built-in functions.

Overcoming this technical hurdle requires leveraging legacy functionality within Excel--specifically, utilizing the powerful, yet often overlooked, capability of creating a Defined Name that incorporates an old-school macro function. This method allows us to bridge the gap between cell formatting and formula evaluation, enabling us to execute an action only if a cell's color matches a specified criterion, such as yellow.

This tutorial provides an expert-level walkthrough detailing how to configure Excel's environment to successfully evaluate cell colors, extract the necessary color code, and integrate this information into a robust conditional logic structure. We will demonstrate how to create a custom function using the **Name Manager** to detect the yellow fill and subsequently use an **IF** statement to return a specific outcome.

The Solution: Leveraging the Legacy GET.CELL Macro Function

To bypass the limitations of standard formulas, we must employ the GET.CELL function. This function originates from the Excel 4.0 Macro language, a precursor to modern VBA (Visual Basic for Applications). While generally deprecated for new development, these functions remain accessible when defined within the **Name Manager**, making them invaluable for tasks that standard functions cannot handle, such as retrieving formatting properties.

The GET.CELL function requires two primary arguments: the type of information to retrieve (the information number) and the cell reference. Specifically, Information Number **38** is the key parameter we need, as it instructs Excel to return the fill color index of the referenced cell. This index is a numerical code that Excel assigns to every possible color in its palette.

By defining a named range using this function, we effectively create a custom, lightweight formula that can be deployed across the entire worksheet. This custom formula, once created, will serve as the condition check within our final **IF** statement, allowing us to build the complex logic required to analyze the visual formatting. This procedure is significantly simpler than writing a full VBA script to achieve the same result.

Example Scenario: Identifying All-Star Players Based on Cell Color

Consider a practical scenario where we manage a list of basketball players. In this data set, players who have achieved "All-Star" status are visually highlighted by applying a yellow background fill to their names in Column A. Our objective is to automate the assignment of the label "All-Star" or "Not All-Star" in Column B by programmatically checking the background color of the corresponding cell in Column A.

Suppose our initial data table looks like this, where the yellow cells visually denote the status we wish to capture:

	A	B	C	D	E
1	**Athlete**				
2	Andy				
3	Bob				
4	Chad				
5	Doug				
6	Eric				
7	Frank				
8	Greg				
9	Henry				
10	Isaac				
11	John				
12	Kendall				
13	Luke				
14					
15					
16					

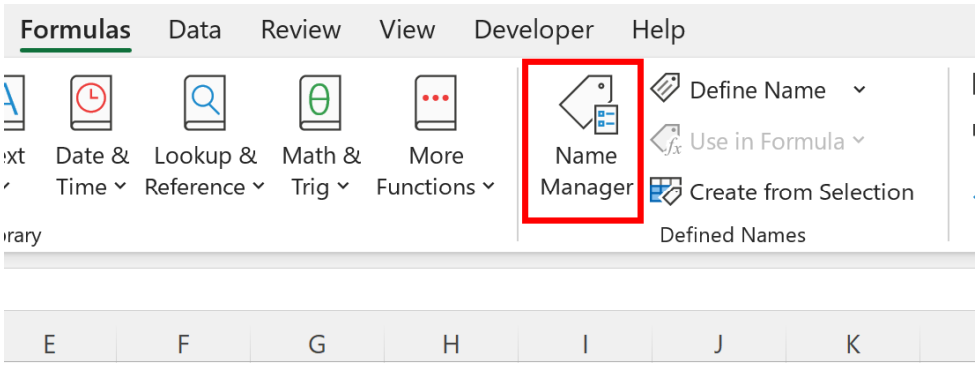
We aim to use a robust **IF** statement in column B to evaluate the color of the cell in column A. If the cell color is yellow, the formula should return "All-Star"; otherwise, it should return "Not All-Star." The subsequent steps will guide us through the critical setup required to make this conditional evaluation possible within standard Excel formulas.

Step-by-Step Implementation: Defining the Custom Name

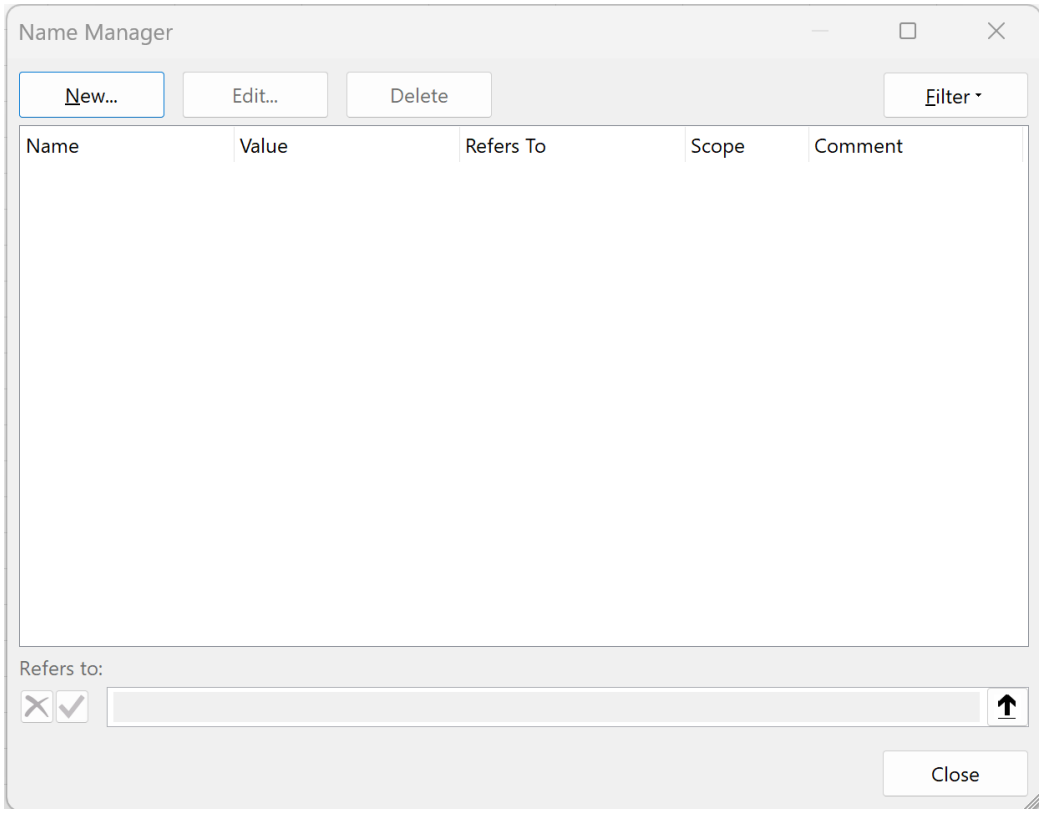
The first crucial step is setting up the custom function using the **Name Manager**. This definition ensures that the GET.CELL macro can be accessed cleanly within our worksheet formulas. To begin this process, navigate to the **Formulas** tab located along the top ribbon of the Excel

interface.

Within the **Formulas** tab, locate and click the **Name Manager** icon. This action will open a new dialogue box dedicated to managing defined names, which are essentially user-defined variables or custom formulas.

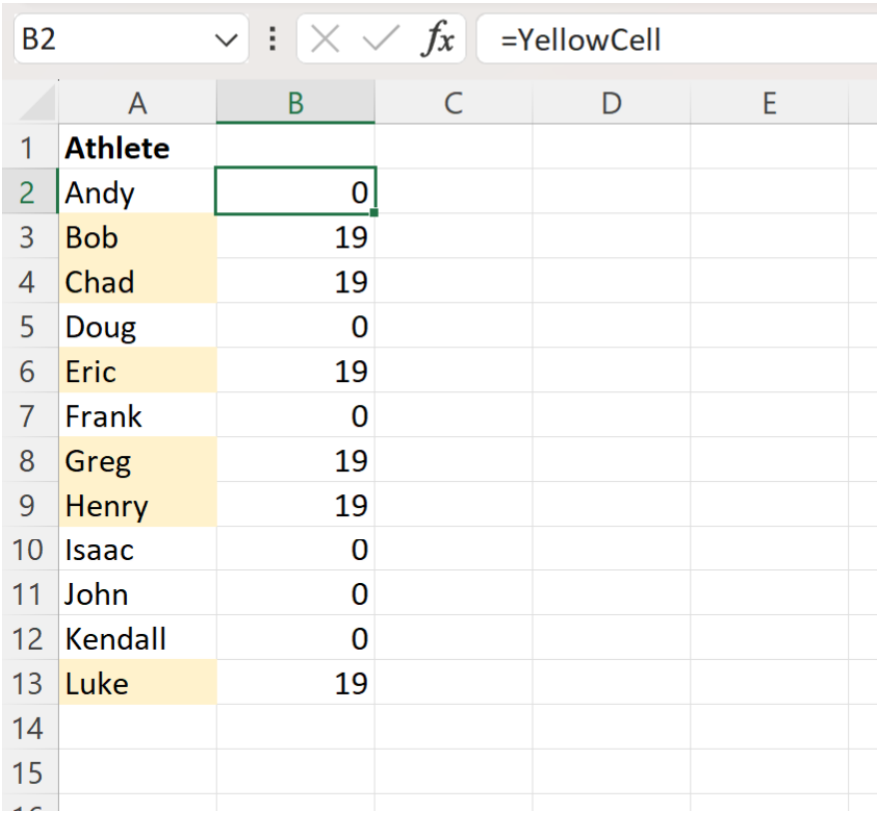


Once the **Name Manager** window appears, click the **New** button situated in the top-left corner. This initiates the creation of our custom macro-based function. In the subsequent "New Name" window, we must input the necessary parameters that define our color-checking utility.



In the **Name** box of the "New Name" window, type a descriptive name for your custom function,

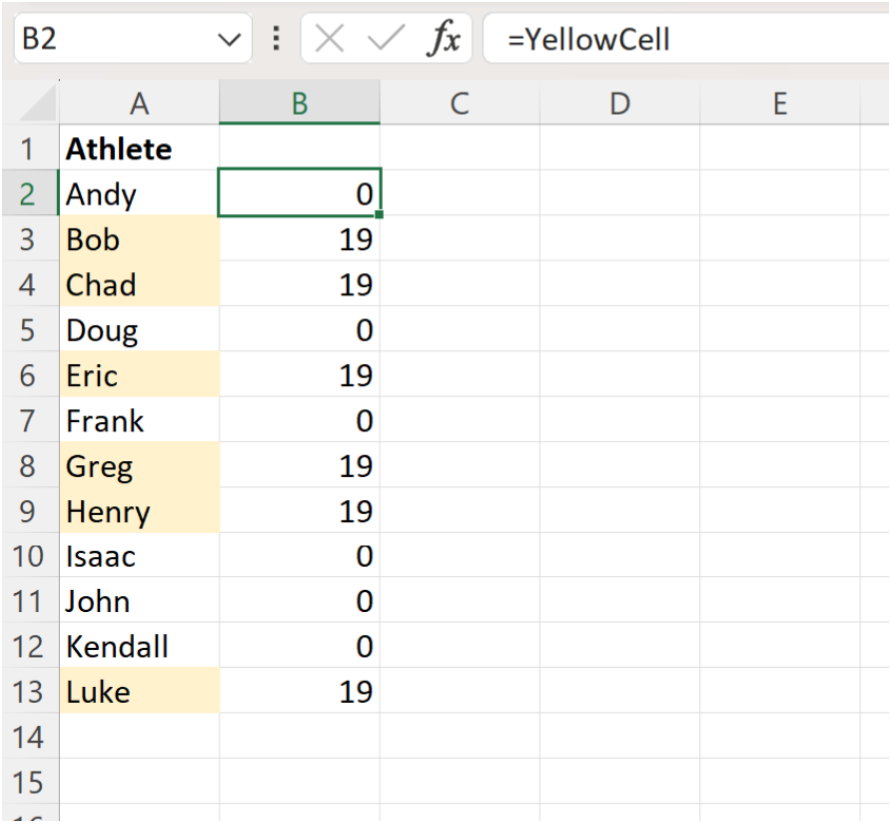
such as **YellowCell**. This name is what you will reference in your worksheet formulas later. Critically, in the **Refers to** box, input the following formula, ensuring the cell reference accurately reflects the location where the formula will be applied (here, we are referencing the first data point in column A, assuming the sheet name is Sheet1): **=GET.CELL(38,Sheet1!A2)**. After entering this, click **OK** to save the definition. The key component here is the information number **38**, which specifically requests the cell's background color index.



Applying the Color Code Logic

Now that the **YellowCell** defined name is established, we can use it directly in the worksheet to determine the numerical color code associated with the specific shade of yellow used in our data. It is essential to first identify this code because Excel assigns different index numbers to different hues, even within the same basic color family (e.g., pale yellow versus bright yellow).

In cell **B2**, enter the custom defined name as a formula: **=YellowCell**. Since the defined name refers to the color index of the corresponding cell (A2 in this case), this formula will return the color index number. Next, click and drag this formula down through column B to apply it to all corresponding cells in column A.



Upon executing the formula, the results in Column B reveal the specific color index numbers. In our particular example, the formula returns the color code **19** for every cell in column A that is shaded yellow. Cells that are not yellow will typically return a value of 0, 1, or another index depending on their default or applied color. This step is critical because it establishes the precise numerical value--in this case, **19**--that we must use as the condition within our final **IF** statement.

Constructing the Final Conditional Formula

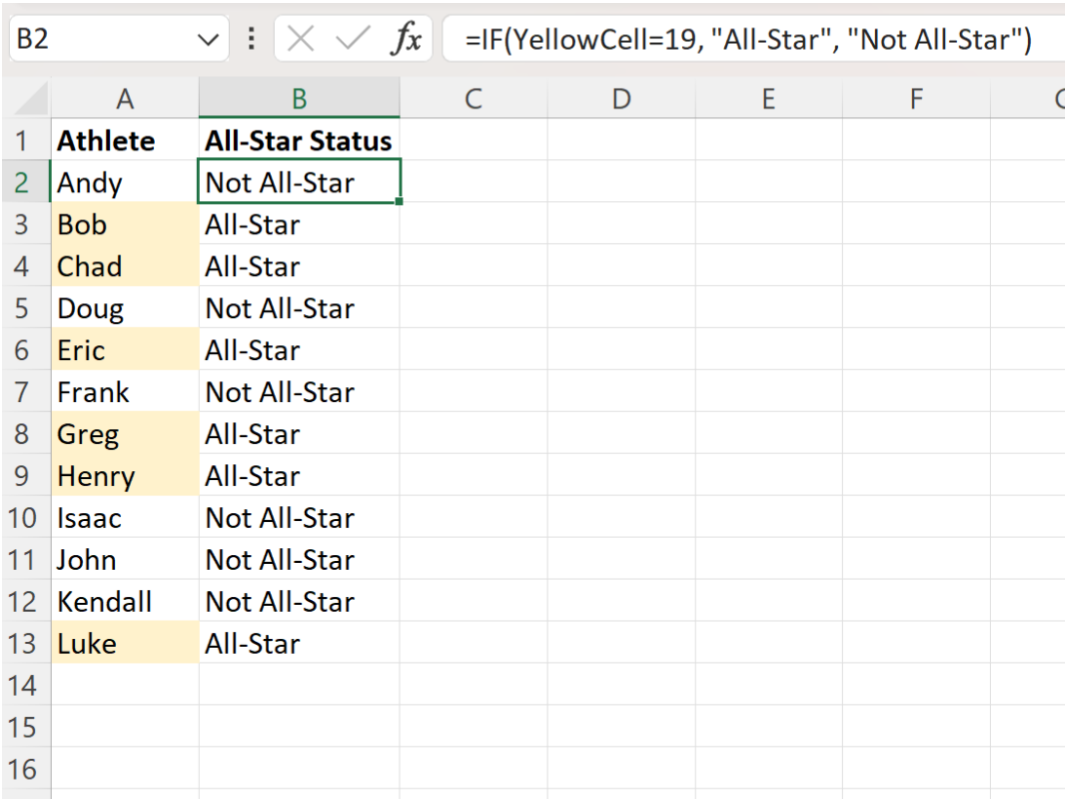
With the color index confirmed, we can now integrate this numerical condition into a standard **IF** statement to assign the desired text labels. This final formula structure is clean, powerful, and highly readable, relying on the custom function we created to perform the heavy lifting.

In cell **B2**, we input the following conditional formula. This formula checks if the value returned by our custom **YellowCell** function equals **19** (the color code for the specific shade of yellow used). If the condition is true, it returns "All-Star"; if false, it returns "Not All-Star."

=IF(YellowCell=19, "All-Star", "Not All-Star")

After inputting the formula into cell **B2**, the next step is to click and drag the formula handle

down to apply this logic to all remaining cells in column B. This action instantly populates the status column based entirely on the visual formatting of the player names in column A.



	A	B	C	D	E	F
1	**Athlete**	**All-Star Status**				
2	Andy	Not All-Star				
3	Bob	All-Star				
4	Chad	All-Star				
5	Doug	Not All-Star				
6	Eric	All-Star				
7	Frank	Not All-Star				
8	Greg	All-Star				
9	Henry	All-Star				
10	Isaac	Not All-Star				
11	John	Not All-Star				
12	Kendall	Not All-Star				
13	Luke	All-Star				
14						
15						
16						

As demonstrated by the final output, column B successfully returns "All-Star" only when the corresponding cell in column A is highlighted yellow, and "Not All-Star" otherwise. This successfully achieves the goal of making a formula execute an action based on cell color, a task impossible with standard functions alone.

Understanding Color Codes and Limitations

It is essential for users employing this technique to recognize that the color code returned by GET.CELL is entirely dependent on the specific shade used. As noted in this example, the particular shade of yellow utilized had a color code of **19**. Had a slightly darker, lighter, or custom yellow been used, the color code would be different.

This variability underscores why the initial step of using **=YellowCell** to extract the color index is mandatory. We cannot simply assume a color index (like **19**) without verification; the index must be derived from the actual formatting applied in the worksheet. Furthermore, if you plan to use this method for other colors (e.g., green or blue), you would repeat the initial procedure to determine the correct color index for those specific shades.

While the GET.CELL function is highly effective for this specific purpose, users should be aware that it relies on legacy Excel 4.0 Macro technology. While generally stable, this function may face limitations in certain highly restricted network environments or when dealing with complex compatibility issues across extremely old or new versions of Excel. For the majority of desktop use cases, however, this technique remains the most straightforward way to integrate formatting checks into formula logic without resorting to complex VBA solutions.

Additional Resources

For users interested in extending their capabilities with advanced formula techniques and conditional operations, the following tutorials explain how to perform other common and complex operations in Excel:

Tutorial on using nested **IF** statements for multiple conditions.

Guide to implementing conditional formatting based on formula outputs.

Exploring advanced uses of the **Name Manager** for dynamic ranging.

Detailed breakdown of color indexing in Excel environments.