

How to Run an Excel Formula Based on Cell Color (Yellow)

Authored by
stats writer

February 8, 2026

RECOMMENDED CITATION

stats writer (2026). *How to Run an Excel Formula Based on Cell Color (Yellow)*.

PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=129756>

Developing an Excel formula capable of executing an action strictly based on a cell's background color is a frequent requirement in advanced data analysis. While standard Excel formula functions, such as the powerful IF statement, excel at evaluating cell values, they are fundamentally unable to read formatting attributes directly. This limitation prevents a simple approach to check for a color like **yellow** and then trigger a specific calculation or response. Therefore, achieving color-based logic requires utilizing specialized, often overlooked, features within the Excel environment, specifically leveraging the capabilities of a Defined Name in conjunction with legacy macro functions.

The goal is to design a robust mechanism where, if a designated cell is confirmed to be **yellow**, a predetermined action--such as returning a specific text string or initiating a complex calculation--is executed. This capability is extremely valuable for dynamic reporting, quality control checks, or distinguishing critical data points without relying on separate indicator columns. We will demonstrate a practical, reliable method that bypasses the constraints of standard functions by employing a powerful, albeit dated, Excel technique.

Excel Formula Logic: Executing Actions Based on Cell Color

The Challenge: Why Standard Formulas Cannot Read Color

It is important to understand the fundamental limitation of standard Microsoft Excel functions. Functions like SUM, AVERAGE, and the IF statement are designed exclusively to process the underlying data value of a cell--the number, text, or date it contains. They cannot natively interact with or query formatting attributes, such as font style, border thickness, or background fill color. This distinction is crucial: Excel treats data and presentation (formatting) as separate layers. Attempting to write a function like `=IF(CellA1.Color="Yellow", "Yes", "No")` will simply result in a formula error because the required property attribute does not exist within the standard Excel formula language.

While features like Conditional formatting allow formatting to react to data changes, the reverse is not true using conventional methods. For example, if you use Conditional formatting to turn cells **yellow** when their value exceeds 100, the formula in another cell still cannot detect that resulting **yellow** color. This necessitates an advanced workaround that bridges the gap between the formatting layer and the data processing layer.

The Advanced Solution: Utilizing Defined Names and GET.CELL

To overcome Excel's intrinsic inability to read formatting, we must turn to a legacy macro function: GET.CELL. This function, dating back to Excel version 4.0, is a powerful but non-standard tool that can retrieve detailed information about a cell's attributes, including its color code. However,

`GET.CELL` cannot be directly entered into a worksheet cell; it must be encapsulated within a Defined Name.

By defining a custom name--which we will call **YellowCell**--and assigning the `GET.CELL` function to it, we create a pseudo-function that can be referenced like any standard formula within the worksheet. The `GET.CELL` function, when used with the appropriate argument, returns a numerical code corresponding to the interior color index of the target cell. Once we capture this color index, we can use a conventional IF statement to evaluate whether that returned code matches the index for **yellow**, thus achieving the desired conditional logic.

Practical Example: Identifying All-Star Basketball Players

To illustrate this technique, consider a dataset containing a list of basketball players. In this scenario, we have manually highlighted the names of All-Star players using a **yellow** fill color. Our objective is to automatically generate a designation ("All-Star" or "Not All-Star") in an adjacent column based purely on the presence or absence of the **yellow** background color in the player's name cell.

Suppose we have the following list of basketball players in Excel in which the yellow cells indicate that the player is an All-Star:

| | 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 | | | | | |

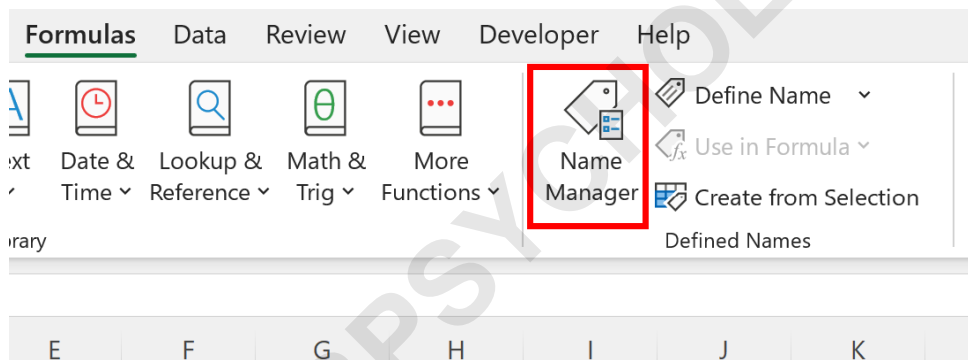
We aim to populate Column B with the appropriate status by checking the fill color in Column A. This process requires meticulously following the steps for setting up the necessary custom function using the **Name Manager** tool.

Step 1: Accessing the Name Manager Interface

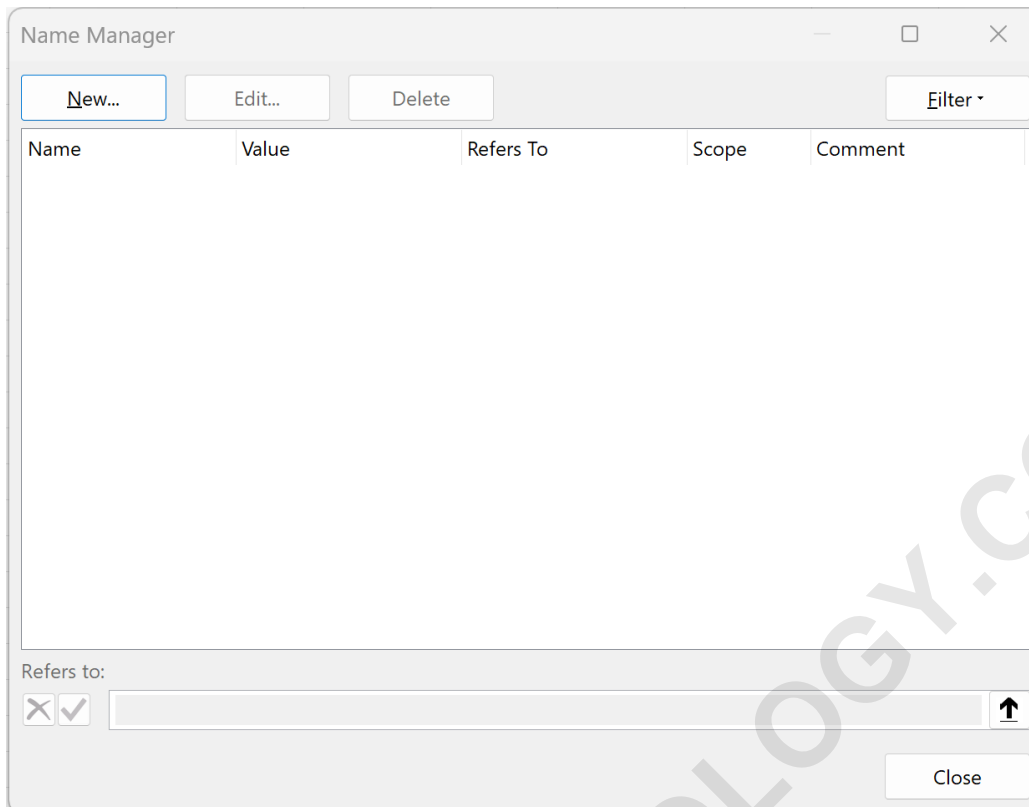
The foundation of this solution lies in defining the custom formula that reads the cell color. This is done through the **Name Manager**, which is the interface Excel uses to manage all Defined Names, ranges, and constants within a workbook. To begin, navigate to the main ribbon interface and locate the **Formulas** tab. Within the **Defined Names** group, you will find and click the **Name Manager** icon.

This action opens a dialog box that lists all current names defined in your workbook. Since we are creating a new custom function, the next logical step within this manager is to initiate the creation process. Click the **New** button located in the top left corner of the **Name Manager** window.

To do so, we can click the **Formulas** tab along the top ribbon, then click the **Name Manager** icon:



In the new window that appears, click the **New** button in the top left corner:



Step 2: Defining the YellowCell Function Using GET.CELL

The crucial step involves entering the specific macro formula required to extract the background color index. In the **New Name** dialog box, we must accurately populate two fields: **Name** and **Refers to**. The Name field establishes the easy-to-use reference for our custom function, while the Refers to field contains the technical instruction that leverages the GET.CELL command.

First, type **YellowCell** into the **Name** box. This is the name we will use in our final IF statement. Next, in the **Refers to** box, input the following formula: `=GET.CELL(38,Sheet1!A2)`. The number **38** is the specific argument within GET.CELL that instructs Excel to return the fill color index of the referenced cell. Note that `Sheet1!A2` should reference the cell immediately to the left of where you are creating the formula, and it must be entered as a relative reference. Ensure you specify the correct Sheet name if your data resides on a different tab. Click **OK** to save the Defined Name.

In the new window that appears, type **YellowCell** in the **Name** box, then type `=GET.CELL(38,Sheet1!A2)` in the **Refers to** box, then click **OK**:

| | A | B | C | D | E |
|----|----------------|----|---|---|---|
| 1 | Athlete | | | | |
| 2 | Andy | 0 | | | |
| 3 | Bob | 19 | | | |
| 4 | Chad | 19 | | | |
| 5 | Doug | 0 | | | |
| 6 | Eric | 19 | | | |
| 7 | Frank | 0 | | | |
| 8 | Greg | 19 | | | |
| 9 | Henry | 19 | | | |
| 10 | Isaac | 0 | | | |
| 11 | John | 0 | | | |
| 12 | Kendall | 0 | | | |
| 13 | Luke | 19 | | | |
| 14 | | | | | |
| 15 | | | | | |

Step 3: Determining the Specific Color Index Code

Before we can construct the final conditional formula, we must determine the exact numerical code that corresponds to the shade of **yellow** used in our spreadsheet. This code varies depending on the Excel theme, the specific color palette selected, and whether the color was applied via manual formatting or Conditional formatting (though this method primarily works reliably with manual formatting). To find this specific code, we apply our newly created **YellowCell** function to the column adjacent to our data.

Type **=YellowCell** into cell **B2** (the first cell where we want the result). Crucially, because GET.CELL was defined using a relative reference (A2), when this formula is dragged down, it dynamically checks the cell immediately to its left. Drag the formula down to apply it to all rows corresponding to the basketball players.

Next, type **=YellowCell** in cell **B2**, then click and drag this formula down to each cell in column B:

| | A | B | C | D | E |
|----|----------------|----|---|---|---|
| 1 | Athlete | | | | |
| 2 | Andy | 0 | | | |
| 3 | Bob | 19 | | | |
| 4 | Chad | 19 | | | |
| 5 | Doug | 0 | | | |
| 6 | Eric | 19 | | | |
| 7 | Frank | 0 | | | |
| 8 | Greg | 19 | | | |
| 9 | Henry | 19 | | | |
| 10 | Isaac | 0 | | | |
| 11 | John | 0 | | | |
| 12 | Kendall | 0 | | | |
| 13 | Luke | 19 | | | |
| 14 | | | | | |
| 15 | | | | | |

Upon reviewing the results in Column B, we observe that the formula returns a numerical value--in this specific example, **19**--for every row where the corresponding cell in Column A is **yellow**. All uncolored cells will return a different code, usually 0, 1, or another non-matching number depending on the spreadsheet's default settings. This number, **19**, is the critical index we must use in our final logic test.

Step 4: Implementing the Final IF Statement Logic

With the color index confirmed (in this case, **19**), we can now integrate this information into a standard IF statement. The logic is straightforward: if the output of our custom **YellowCell** function equals the color code 19, the player is an "All-Star"; otherwise, they are "Not All-Star."

Replace the temporary `=YellowCell` formula in cell **B2** with the definitive conditional logic below. This formula uses the **YellowCell** Defined Name as the primary condition check, ensuring efficiency and clarity in the final calculation.

Type the following formula into cell **B2**:

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

Once entered, click and drag this new IF statement down to the remaining cells in column B. This action propagates the conditional check across all player names, instantly classifying them based on the background color detected by the **YellowCell** function.

We can then click and drag this formula down to each remaining cell in column B:

| | A | B | C | D | E | F | G |
|----|----------------|------------------------|---|---|---|---|---|
| 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 in the resulting table, Column B now accurately returns either "All-Star" or "Not All-Star" depending on whether or not the corresponding cell in Column A has been filled with the specific shade of **yellow** that corresponds to index code 19. This technique successfully bypasses the limitations of standard functions and introduces formatting awareness into spreadsheet logic.

Important Considerations Regarding Color Codes and Updates

It is vital to recognize that the color code returned by the `GET.CELL(38, ...)` function is not universal for all shades of a color. Different shades of **yellow**, blue, or red, even if they appear similar, will have distinct color index numbers. For instance, the default light **yellow** might be 19, while a darker gold might be 44. This variability is why the intermediate step of applying `=YellowCell` directly to the column was essential: it allowed us to empirically determine the exact code (**19** in this example) required for the subsequent IF statement.

Furthermore, because `GET.CELL` is a legacy macro function, it operates differently from standard

formulas. It is not always guaranteed to update immediately when a cell's format changes. If you change a cell's color from red to **yellow**, the formula relying on **YellowCell** might not update until Excel performs a calculation cycle. In such cases, you may need to force a recalculation, typically by pressing the **F9** key on your keyboard, to ensure the logic reflects the most current formatting status.

Conclusion: Bridging Formatting and Calculation in Excel

While Microsoft Excel does not offer a straightforward formula to query cell formatting, the technique of combining a Defined Name with the legacy `GET.CELL` function provides a powerful and surprisingly simple workaround. By extracting the specific color index code associated with **yellow**, we successfully integrate visual attributes into the core data processing logic of the spreadsheet. This allows for automated decision-making and reporting based on visual cues, significantly enhancing the analytical capabilities of the workbook.

Further Excel Operations and Tutorials

The following tutorials explain how to perform other common operations in Excel: