Learning to Use Excel IF Statements Based on Cell Color

Authored by

Mohammed looti

November 11, 2025

The Challenge of Formatting-Based Logic in Excel

It is a common requirement in data analysis to execute a conditional action based on the visual attributes of a cell, such as its background fill color. Standard Microsoft Excel formulas, however, are inherently designed to process data values, not formatting properties. Functions like the ubiquitous IF statement can check if a number is greater than zero or if a text string matches a specific pattern, but they cannot natively detect whether a cell has a red background. This limitation poses a significant hurdle for users who rely on color-coding to convey critical information in their spreadsheets.

To overcome this architectural limitation, we must employ an advanced, though somewhat unconventional, technique that bypasses the constraints of modern Excel formulas. This method involves leveraging the capabilities of the legacy Excel 4.0 Macro functions, specifically the powerful **GET.CELL** function. While these functions are generally deprecated in favor of VBA (Visual Basic for Applications), they remain accessible and highly effective when embedded within a custom Defined Name.

This tutorial will guide you through the precise steps required to create a function that can reliably detect if a cell's background color is red, or any other color, and then use that information to drive complex conditional logic. This approach is essential when dealing with data sets where manual formatting, rather than automated Conditional Formatting rules, dictates the meaning of the cell.

Leveraging Excel 4.0 Macros via Defined Names

The core mechanism for checking cell color relies on the legacy function GET.CELL. This function is a component of the older Excel macro language and is specifically designed to retrieve information about a cell's formatting, location, or content. Crucially, it must be encapsulated within a **Defined Name** rather than being typed directly into a spreadsheet cell.

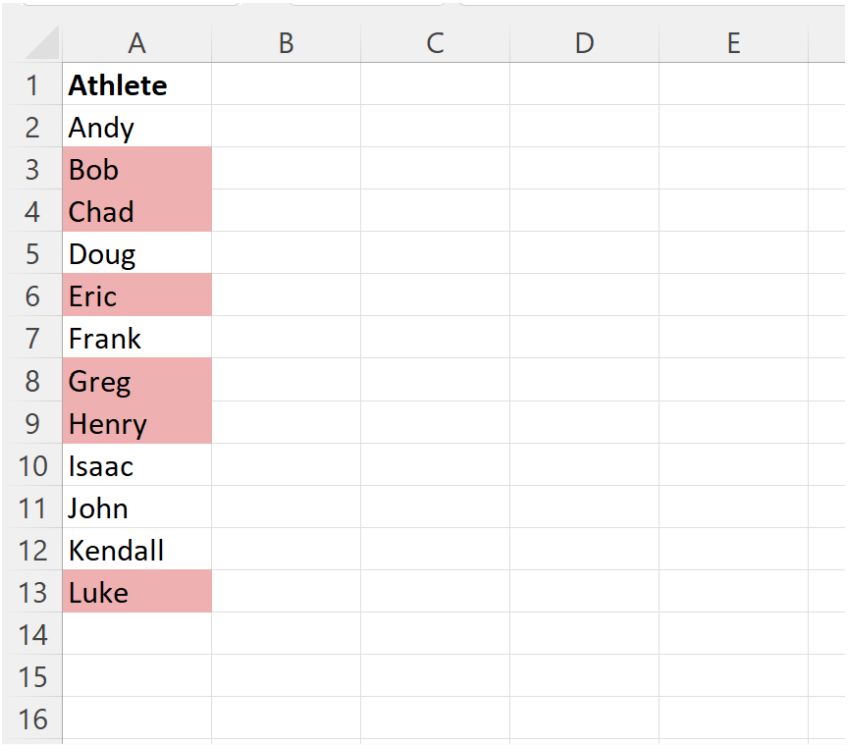
The syntax of **GET.CELL** requires two primary arguments. The first argument is an integer code that specifies the type of information to retrieve. For the purpose of identifying the background color, we use the code **38**. Code 38 instructs Excel to return the numerical color index corresponding to the cell's background fill. The second argument is a cell reference, which tells the function which cell to inspect.

By wrapping **GET.CELL(38, reference)** into a Defined Name, we create a custom function that can be called repeatedly across the spreadsheet. This allows us to dynamically check the color of any cell relative to the cell where the Defined Name is used, effectively giving our modern formulas the ability to "see" cell formatting.

Practical Example: Identifying All-Stars

Let us consider a practical scenario where we have a list of basketball players. In this data set, the cells in column A are manually formatted with a red background fill to indicate that the corresponding player is an All-Star. Our goal is to use an **IF** statement in column B that automatically returns the text "All-Star" if the cell in column A is red, and "Not All-Star" otherwise.

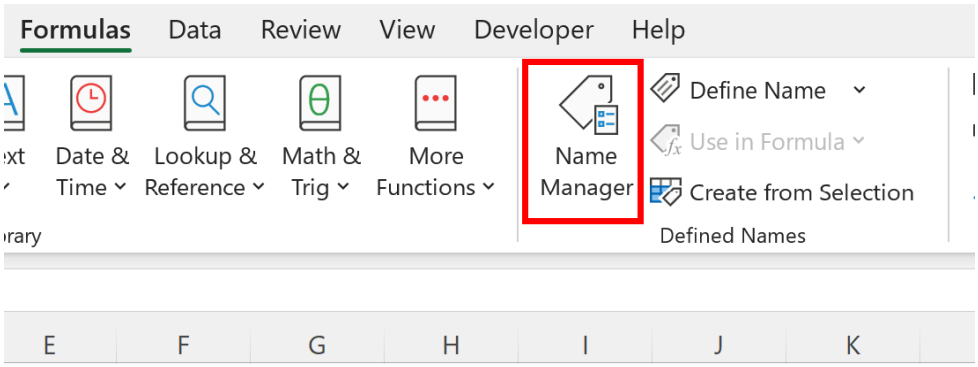
The initial data setup looks like this, where the red formatting holds the key information:



	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					

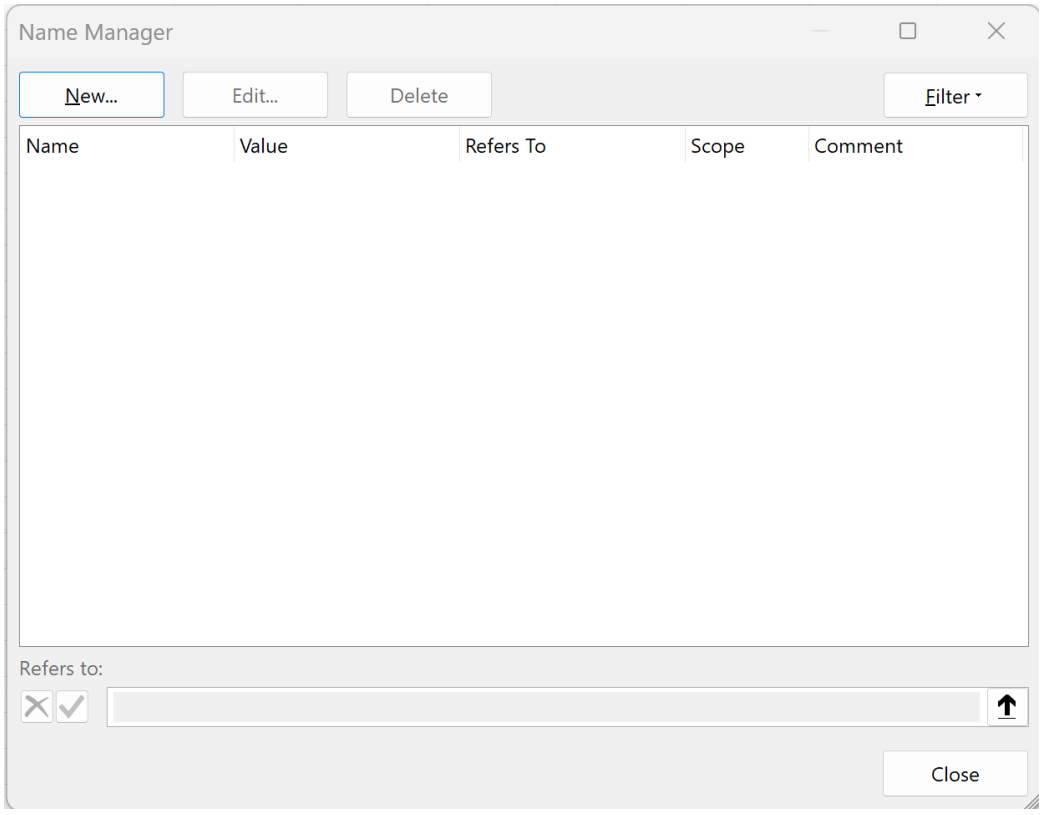
We will now proceed to create the custom function that can read the color information from column A. This involves navigating to the **Formulas** tab in the Excel ribbon and initiating the **Name Manager**, which is the control panel for creating and editing Defined Names.

To begin, click the **Formulas** tab, and then select the **Name Manager** icon. This action opens the dialog box necessary for defining our color-checking function.



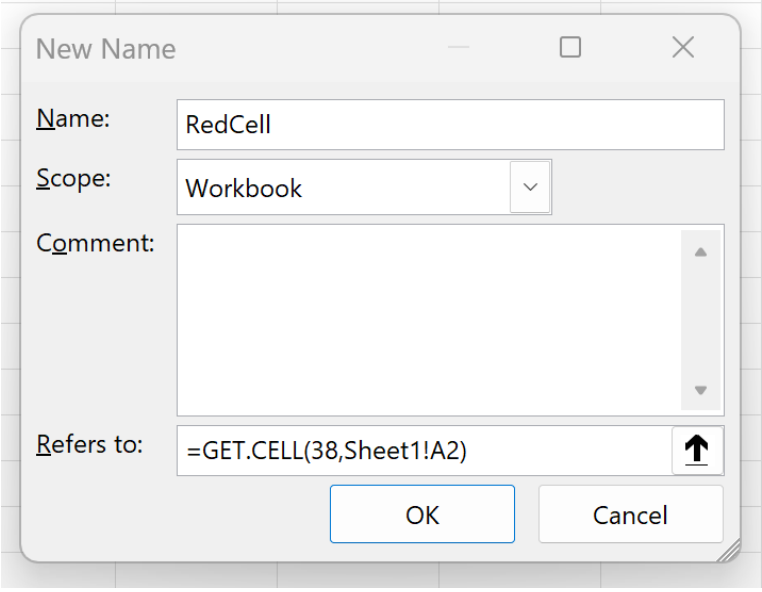
Defining the Custom Color Check Function

Once the **Name Manager** is open, the next critical step is to define the custom function that will retrieve the color index. In the Name Manager window, click the **New** button located in the top left corner.



In the subsequent "New Name" dialog box, we must accurately input the name and the reference formula. We will name our function **RedCell** for clarity. The formula itself must use relative referencing carefully so that it correctly checks the cell adjacent to where the formula is placed.

For the **Name** field, type **RedCell**. For the **Refers to** field, you must input the **GET.CELL** formula, ensuring that the reference is relative to the top-left cell of the range you intend to check. Assuming your data starts in A2, the reference should point to that cell, specified as an absolute reference to the sheet but a relative reference to the row/column, or simply a relative reference if the sheet name is used correctly in the context of the definition. The formula required is: **=GET.CELL(38,Sheet1!A2)**. This tells the function to retrieve the color index (38) of cell A2 on Sheet1.

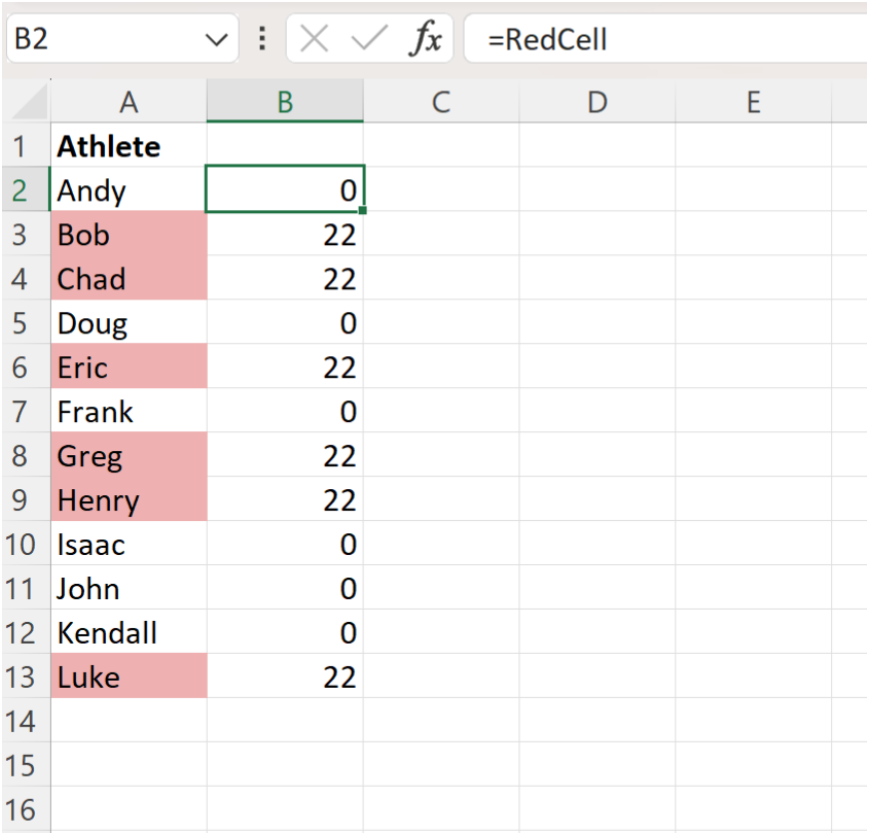


After entering the name and the formula, click **OK**. The custom function **RedCell** is now available for use throughout your workbook, allowing you to bypass standard formula limitations.

Executing the Color Check and Finding the Index

Before we can write the final **IF** statement, we must determine the exact numerical color code that Excel assigns to the specific shade of red used in our spreadsheet. It is crucial to understand that Excel uses a palette of indices, and different shades of red (or colors applied through different methods) will yield different numbers.

To find this color index, simply type **=RedCell** into cell **B2**, which is adjacent to the first cell we need to check (A2). Then, click and drag this formula down through the rest of column B.



Upon execution, we observe that the formula returns a color code of **22** for every cell in column A that is red. Cells that are not red will return a different code, typically 0 or 0, depending on their formatting status. This number, **22**, is the critical piece of information that we will use in our final conditional statement.

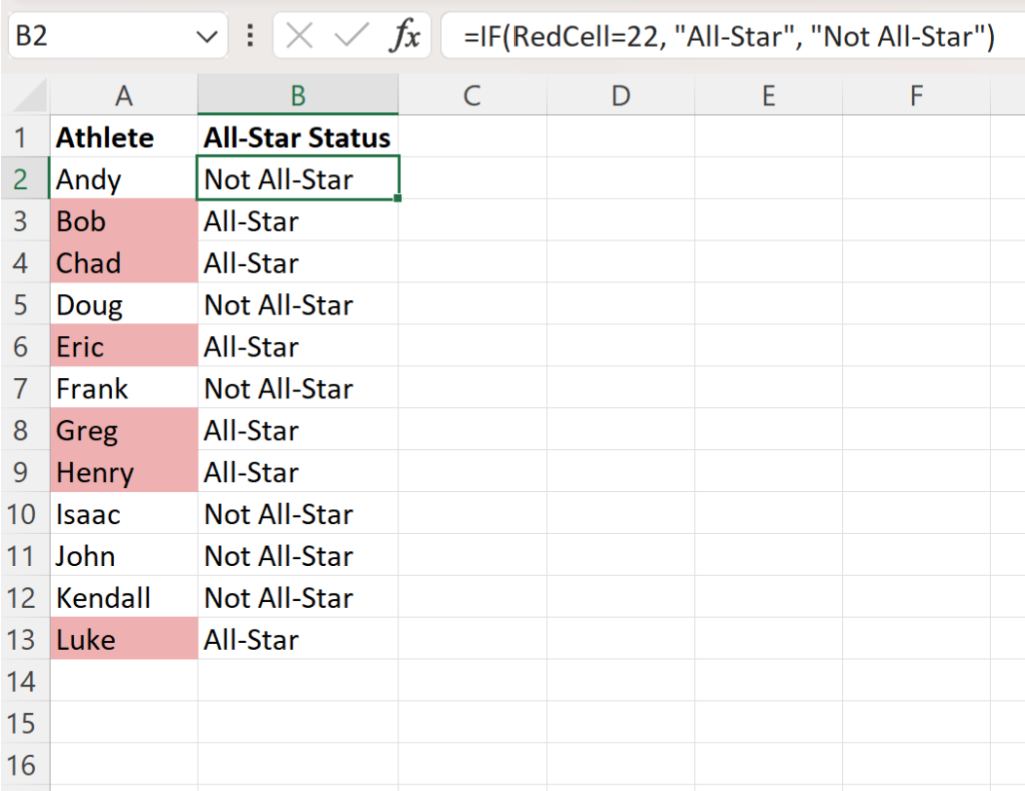
Implementing the Conditional Logic with IF

Now that we have successfully isolated the specific color code (**22**) representing the "All-Star" red, we can finalize the logic using a standard **IF** statement. This statement will check the output of our custom **RedCell** function against the identified code.

The logic is straightforward: if the result of **RedCell** equals 22, the player is an All-Star; otherwise, they are not. The following formula should be entered into cell **B2**:

=IF(RedCell=22, "All-Star", "Not All-Star")

After entering this formula, we apply it to the entire range by clicking and dragging the formula down to the remaining cells in column B.



	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						

The resulting column B successfully classifies each player based on the background color of their corresponding cell in column A. This demonstrates the seamless integration of the legacy macro function with modern conditional logic, providing a powerful, albeit non-standard, solution for formatting-dependent calculations.

Understanding Color Codes and Limitations

A crucial takeaway from this process is the variability of the color code. In this specific demonstration, the shade of red utilized was linked to the color code **22**. Had a different shade been applied--perhaps a lighter crimson or a darker maroon--the **GET.CELL(38, ...)** function would have returned a completely different numerical index.

This necessity of first determining the color code highlights why we had to create the **RedCell** custom function and test its output before constructing the final **IF** statement. If the formatting of the source data changes, the entire formula structure remains valid, but the target number (e.g., 22) within the **IF** statement must be updated to match the new color index.

Furthermore, users should be aware of the limitations inherent in using the Excel 4.0 Macro language. These functions are generally discouraged by Microsoft due to security concerns and their potential for instability across different versions or platforms. While they are indispensable for format-checking, alternative methods--such as ensuring all conditional logic is driven by a hidden

data column rather than visual formatting--are usually preferred for long-term, robust solutions. However, when the data source provides only visual cues, the Defined Name approach using **GET.CELL** is the only functional alternative outside of complex VBA scripting.

Additional Resources

If you are looking to expand your knowledge of advanced Excel techniques or explore other common operations that involve data manipulation and conditional logic, the following tutorials may prove useful:

How to create named ranges for simplified formula management.
Techniques for auditing and debugging complex formulas.
An introduction to VBA for custom functions that handle formatting rules.