How to Get the Day of the Week Name in VBA Using WeekdayName

Authored by

stats writer

February 24, 2026

Introduction to Automated Date Formatting via Visual Basic for Applications

In the modern landscape of data management and business intelligence, the ability to automate repetitive tasks within **Microsoft Excel** is a fundamental skill for analysts and developers alike. **Visual Basic for Applications** (VBA) serves as the primary engine for this automation, providing a robust environment where users can create sophisticated **macros** to streamline complex workflows. One of the most frequent challenges encountered when processing a large **data set** involves the conversion of raw date values into human-readable formats. Specifically, translating a numerical date into its corresponding day of the week is essential for creating intuitive reports, scheduling logs, and temporal analysis models.

The **WeekdayName** **function** in VBA is a specialized tool designed to address this specific requirement by returning a string representing the name of the weekday. While Excel offers built-in cell formatting to display dates as day names, utilizing **VBA** provides a programmatic advantage, allowing the developer to store the actual string value in a cell or use it within a larger logical framework. This distinction is critical when the output needs to be exported to other systems or used in text-based comparisons where formatting alone would be insufficient. By mastering this function, users can significantly enhance the clarity and professionalism of their automated spreadsheets.

The efficiency of the **WeekdayName** function lies in its simplicity and its ability to interface seamlessly with other date-related utilities within the **API**. Whether you are building a financial dashboard that tracks weekly performance or a project management tool that categorizes tasks by day, understanding the nuances of this function is paramount. In the following sections, we will explore the technical specifications, the underlying syntax, and a practical implementation strategy to help you leverage the full potential of date manipulation in your **Excel** projects.

The Technical Mechanics and Syntax of WeekdayName

To effectively implement the **WeekdayName** function, one must first understand its formal syntax and the **parameters** it accepts. The function is structured to take a specific numerical input and transform it into a meaningful text string. The primary argument is the weekday number, which is a required **Integer** representing the day of the week. By default, in most regional settings, the number 1 corresponds to Sunday, 2 to Monday, and so forth, continuing through to 7 for Saturday. This mapping is the foundation upon which the function operates, ensuring that the correct string is retrieved from the system's localized calendar definitions.

Beyond the mandatory numerical input, the **WeekdayName** function offers an optional **Boolean** parameter known as "Abbreviate." This parameter dictates the length of the returned string. When set to **False** (which is the default behavior), the function returns the full name of the day, such as

"Wednesday." Conversely, when set to **True**, the function returns a shortened version, typically the first three letters, such as "Wed." This flexibility is particularly useful when designing compact user interfaces or headers where space is at a premium. An example of this in action would be calling `WeekdayName(3, False)`, which returns the full string "Tuesday," representing the third day of the standard week.

Furthermore, a third optional parameter, "FirstDayOfWeek," allows the developer to define which day should be treated as the start of the week. This is a critical feature for global applications, as the definition of the "first" day can vary by culture and industry. For instance, while many Western calendars begin on Sunday, some business environments or international standards might prioritize Monday. By adjusting this **data type**, the **WeekdayName** function ensures that the numerical input is interpreted correctly within the context of the user's specific requirements, making it a versatile tool for international **software development**.

Distinguishing Between Weekday and WeekdayName Functions

A common point of confusion for those new to **Visual Basic for Applications** is the distinction between the **Weekday** function and the **WeekdayName** function. While they sound similar and are often used in tandem, they serve entirely different purposes in the **object model**. The **Weekday** function is designed to take a date object or a date string as an input and return an **integer** value. For example, applying the **Weekday** function to "1/1/2023" would yield the number 1. It identifies the "index" of the day rather than its name.

In contrast, the **WeekdayName** function cannot process a date directly. If you attempt to pass a full date like "1/1/2023" into **WeekdayName**, the code will likely result in an error because the function expects a single digit between 1 and 7. Therefore, to get the name of a day from a specific date, you must nest these functions. You first use the **Weekday** function to extract the numerical index from the date, and then you pass that index into the **WeekdayName** function to retrieve the string. This hierarchical approach is a standard practice in **computer programming**, where data is transformed through multiple stages to reach the desired output.

Choosing the right tool depends on the logic of your **macro**. If your script needs to perform calculations--such as checking if a date falls on a weekend--the numerical output of the **Weekday** function is more efficient for comparison operators. However, if the end goal is data visualization or reporting for a human audience, the **WeekdayName** function is the superior choice. Understanding this relationship prevents common bugs and ensures that your **source code** remains clean, readable, and logically sound.

Practical Implementation: The FindWeekdayName Macro

To see the **WeekdayName** function in a real-world scenario, we can examine a common task:

iterating through a list of dates in an **Excel** worksheet and populating the adjacent column with the corresponding day names. This requires an **iterative** approach, typically using a **For** loop. The following code demonstrates how to automate this process effectively, ensuring that each date in a specified range is evaluated and its weekday name is printed in the neighboring cell.

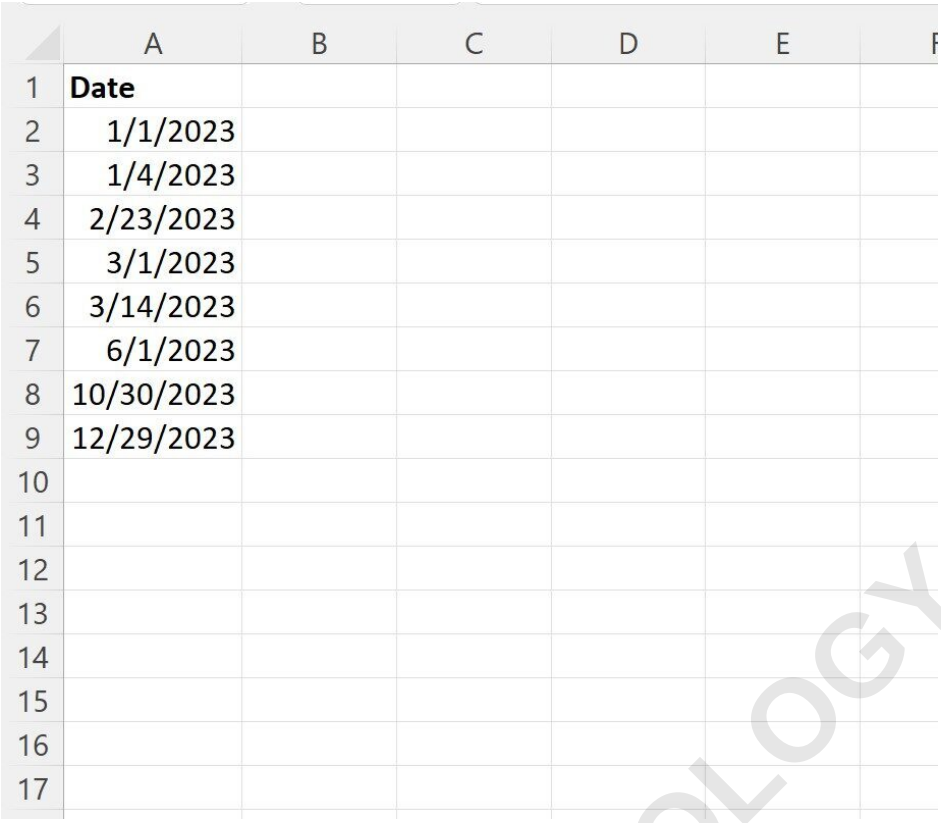
```
Sub FindWeekdayName()

Dim i As Integer

For i = 2 To 9
Range("B" & i) = WeekdayName(Weekday(Range("A" & i)))
Next i

End Sub
```

In this specific **subroutine**, we define a variable `i` as an **integer** to serve as our loop counter. The loop is set to run from row 2 to row 9, which corresponds to the location of our data. Inside the loop, the code targets column B and assigns it a value based on the date found in column A. Note how the `Weekday` function is nested inside the `WeekdayName` function. This captures the numerical day value from the cell and immediately converts it into a name string. This pattern is highly efficient for processing batches of information without manual intervention.

This macro is a perfect example of how **VBA** can bridge the gap between raw data and actionable information. By running this script, an end-user transforms a column of ambiguous dates into a structured list that clearly indicates the day of the week, which is vital for identifying trends, such as peak activity on weekends or scheduling conflicts. The logic is scalable; you could easily modify the range from `2 To 9` to `2 To 10000` to handle massive datasets in seconds, demonstrating the power of **automation** in productivity software.

Visualizing the Transformation in Excel

Before applying any **macro**, it is helpful to visualize the starting point of your data. Suppose we have a column of dates in Excel that represents various transactions or milestones. At first glance, these dates are just numbers in a sequence, and it is not immediately obvious which ones fall on a Monday or a Friday. This lack of clarity can hinder quick decision-making. The image below represents a typical starting **data set** where dates are listed in column A, awaiting processing.



	A	B	C	D	E
1	**Date**				
2	1/1/2023				
3	1/4/2023				
4	2/23/2023				
5	3/1/2023				
6	3/14/2023				
7	6/1/2023				
8	10/30/2023				
9	12/29/2023				
10					
11					
12					
13					
14					
15					
16					
17					

Once the **FindWeekdayName** macro is executed, the transformation is instantaneous. The script systematically reads each cell in the designated range, calculates the weekday, and writes the name into column B. This results in a much more readable and organized spreadsheet. Below is the **source code** again for reference, which you can copy directly into your VBA editor (accessible by pressing ALT + F11 in Excel).

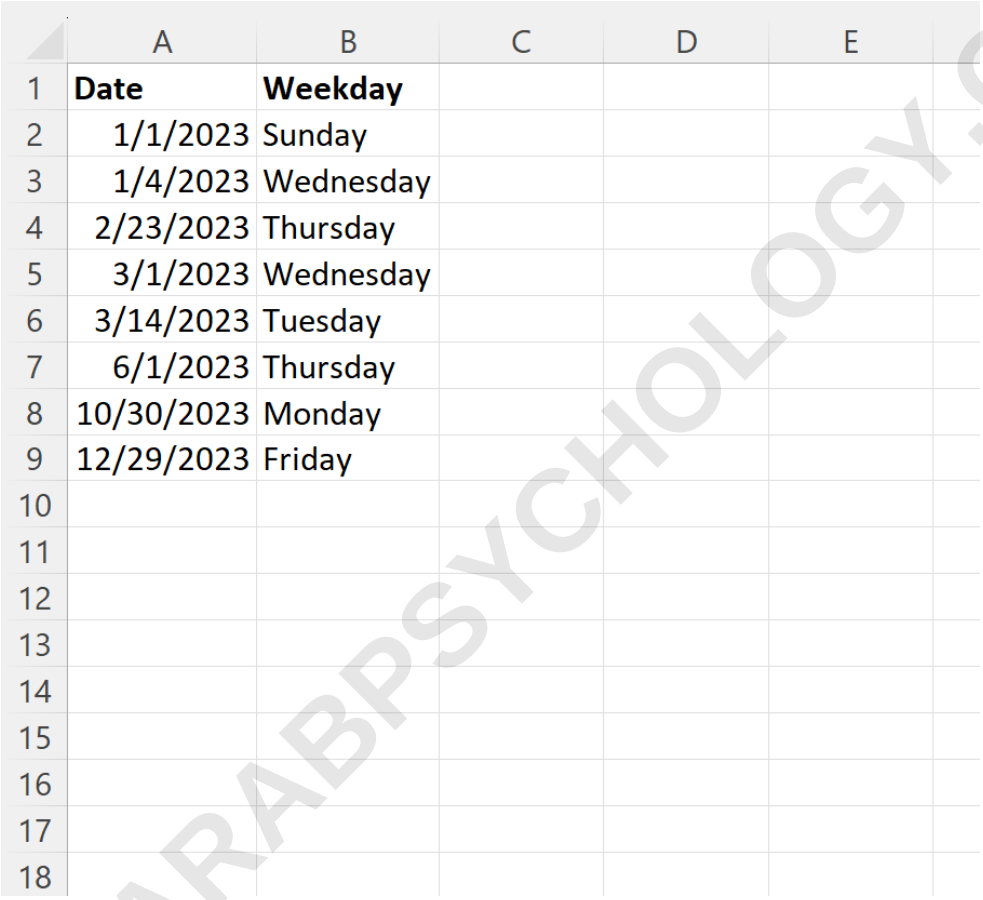
```
Sub FindWeekdayName()

Dim i As Integer

For i = 2 To 9
Range("B" & i) = WeekdayName(Weekday(Range("A" & i)))
Next i

End Sub
```

The output of this operation provides a clear mapping that enriches the original data. By looking at the results, you can quickly verify the accuracy of the function. For instance, if the date 1/1/2023 is present, the macro will correctly identify it as a Sunday. This visual confirmation is a crucial step in **software testing**, ensuring that the logic applied by the **function** aligns with the expected real-

world values.

Analyzing the Final Output and Results

After running the provided **macro**, the Excel worksheet is updated to reflect the weekday names. The image below illustrates the final state of the spreadsheet. As you can see, column B is now populated with the full names of the days corresponding to the dates in column A. This conversion makes the data significantly more accessible for non-technical users who may need to review the logs or reports.



	A	B	C	D	E
1	**Date**	**Weekday**			
2	1/1/2023	Sunday			
3	1/4/2023	Wednesday			
4	2/23/2023	Thursday			
5	3/1/2023	Wednesday			
6	3/14/2023	Tuesday			
7	6/1/2023	Thursday			
8	10/30/2023	Monday			
9	12/29/2023	Friday			
10					
11					
12					
13					
14					
15					
16					
17					
18					

The results demonstrate the consistency of the **WeekdayName** function. By examining specific entries, we can observe the following mappings:

The date **1/1/2023** is correctly identified as a **Sunday**.
The date **1/4/2023** is identified as a **Wednesday**.
The date **2/23/2023** is identified as a **Thursday**.

This level of detail is invaluable when performing time-series analysis or auditing financial records. Knowing the day of the week can help explain variance in data, such as lower sales on a Monday

or higher traffic on a Saturday. By integrating the **WeekdayName function** into your **VBA** toolkit, you gain the ability to add this layer of analytical depth to any project with minimal effort.

Best Practices and Advanced Considerations

When working with date functions in **Visual Basic for Applications**, it is essential to consider the **locale** settings of the host system. The **WeekdayName** function is inherently sensitive to the language and regional settings of the user's operating system. This means that a user in France will see "Lundi" while a user in the United States will see "Monday." While this is usually desirable for localized reporting, developers should be cautious when writing code that depends on specific string comparisons, as the hardcoded string "Monday" will fail if the system returns a name in another language.

Another best practice involves **error handling**. If a cell in your range is empty or contains a non-date value, the `Weekday` function will trigger a "Type Mismatch" error, halting your **macro**. To prevent this, it is advisable to include a check using the `IsDate` function before attempting to process the cell. This ensures that your script is resilient and can handle imperfect **data sets** without crashing, which is a hallmark of professional-grade **software development**.

Finally, remember that the **WeekdayName** function is just one part of a larger suite of date and time utilities. For more complex requirements, such as calculating the number of working days between two dates or adjusting for holidays, you may need to combine this function with others like `DateDiff`, `DateAdd`, or even custom **algorithms**. You can find the complete documentation for the VBA **WeekdayName** function on official resources like Microsoft Learn to explore even more advanced configurations and parameters.