# what is the syntax for the WeekdayName function in VBA?

Authored by

**stats writer**

November 18, 2025

## Understanding the WeekdayName Function in VBA

The **WeekdayName** function is a powerful tool within **VBA** (Visual Basic for Applications) designed specifically for text manipulation based on date data. Its core purpose is to retrieve the descriptive name of the day of the week--such as "Monday" or "Friday"--corresponding to a given numerical representation of that day. This function is indispensable when automating reports or generating user-friendly outputs in **Excel** where numerical date codes need to be translated into clear, natural language labels. Utilizing this function streamlines the process of data presentation, making complex datasets immediately accessible and understandable to end-users who may not be familiar with numerical date indexing systems.

Unlike standard Excel formulas that might require nested functions or conditional logic to achieve the same result, the **WeekdayName** function offers a single, concise command. This simplicity is a hallmark of efficient **VBA** programming, allowing developers to quickly integrate reliable date-to-text conversions into their macros. The function accepts a mandatory numeric argument representing the day index (typically 1 for Sunday through 7 for Saturday), and up to two optional arguments that control the formatting and starting day of the week. Mastering this function is essential for anyone developing sophisticated date manipulation **macros** within the Microsoft Office suite.

When integrated into a looping structure, such as a **VBA** `For...Next` loop, the **WeekdayName** function becomes highly scalable. This means it can efficiently process hundreds or even thousands of dates located across various rows or columns in a spreadsheet, converting them instantly. This automation capability saves significant time and drastically reduces the potential for manual data entry errors. The following example demonstrates a common way to employ this function within a procedure, showing how to derive and display weekday names across a predefined dataset, setting the stage for more complex applications.

```
Sub FindWeekdayName()

Dim i As Integer

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

End Sub
```

## Detailed Breakdown of the WeekdayName Function Arguments

To properly leverage the **WeekdayName** function, it is essential to understand its required and

optional parameters. The complete syntax is `WeekdayName(Weekday, , )`. The first argument, `Weekday`, is mandatory and represents the numeric day of the week (an **Integer** value from 1 to 7). Typically, this value is derived from another date function, such as the companion `Weekday` function, which extracts the day number from a full date serial number. Ensuring this primary argument is a valid number between 1 and 7 is crucial for the function to return a correct textual output. If the input is outside this range, the function may return an empty string or an error, highlighting the necessity of combining it correctly with preceding date calculations.

The first optional argument, , is a Boolean value that determines the length of the returned weekday name. If this argument is set to `True`, the function returns the abbreviated name (e.g., "Mon" instead of "Monday"). Conversely, setting it to `False` (or omitting it, as `False` is the default) results in the full name of the weekday. This optional parameter provides flexibility in report generation, allowing users to choose between compact data presentation and fully descriptive labeling. For large datasets or reports where space is a constraint, using the abbreviated form can significantly improve readability and layout efficiency without losing necessary information.

The second optional argument, , is vital for international compatibility and localized reporting. This argument specifies which day is considered the first day of the week, influencing how the preceding `Weekday` function calculates the day number. While the default setting is typically Sunday (VBA constant `vbSunday`, or 1), the user can specify constants like `vbMonday` (2) or `vbUseSystem` (0) to align the calculation with regional standards or specific project requirements. Although the `WeekdayName` function itself is only concerned with converting the final number (1-7) to a string, providing the correct `FirstDayOfWeek` argument to the inner `Weekday` function ensures that the number passed to `WeekdayName` accurately reflects the desired standard.

## The Relationship Between WeekdayName and the Weekday Function

A common point of confusion for new **VBA** developers is the interplay between the `WeekdayName` function and its close relative, the `Weekday` function. They are almost always used together, as demonstrated in the practical example code: `WeekdayName(Weekday(Range("A" & i)))`. The fundamental distinction lies in their output types. The inner `Weekday` function takes a date serial number (extracted from a cell **Range** in our example) and returns an **Integer** representing the day of the week (1 through 7). This **Integer** is then immediately fed as the mandatory input to the outer `WeekdayName` function.

The `WeekdayName` function, on the other hand, performs the conversion from the numeric index provided by `Weekday` into a formatted text string (e.g., "Sunday"). If your goal is simply to perform calculations based on the day of the week--such as counting all Sundays in a month--you would use the `Weekday` function alone, dealing only with **Integers**. However, if the requirement is to display the result in a readable format for reporting or visualization purposes, then the `Weekday`

function must be nested within the `WeekdayName` function. This nested structure is critical for achieving the desired textual output, illustrating a key pattern in date manipulation **VBA** routines.

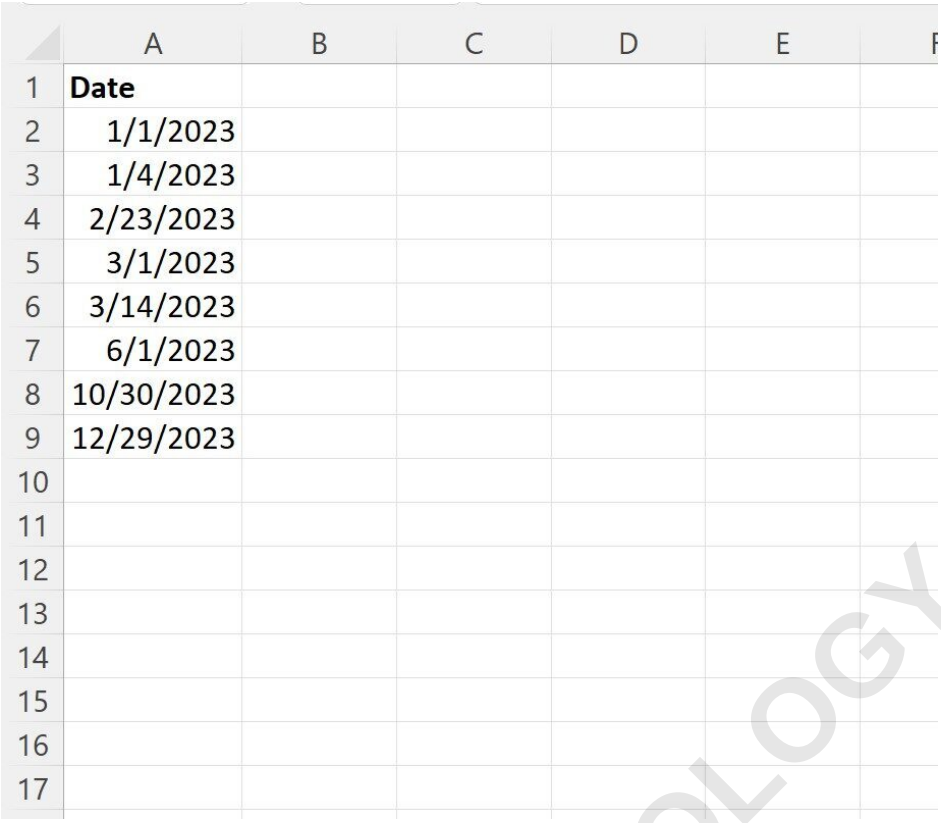
It is important to acknowledge that without the `Weekday` function, the `WeekdayName` function would require the programmer to manually supply the day number, which defeats the purpose of automation when dealing with dates stored in cells. If you were to attempt to pass a full date serial number directly to `WeekdayName`, **VBA** would likely interpret the large serial number as the day index, leading to an error or an incorrect textual output. Therefore, understanding this complementary relationship is key: `Weekday` extracts the numerical index, and `WeekdayName` translates that numerical index into a descriptive string.

**Note**: If you would rather return the day of the week as an **Integer**, suitable for mathematical operations or comparisons, then you should use the `Weekday` function instead of `WeekdayName`. The former returns a number, while the latter returns a string.

## Prerequisites: Setting Up Your Data in Microsoft Excel

Before executing any date-processing **macro**, proper data preparation in **Excel** is essential. For the `FindWeekdayName` macro to function correctly, the source data must be formatted as valid Excel dates in the specified input **Range**. Suppose we have a column of dates that need classification. This initial setup determines the efficiency and accuracy of the subsequent **VBA** operation. For our running example, we assume the date data resides in column A, specifically starting from cell A2 down to A9.

Suppose we have the following column of dates in **Excel**:



| | 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 | | | | | |



The crucial requirement is that Excel recognizes these entries as actual dates, converting them internally to serial numbers (the number of days elapsed since January 1, 1900). If the cells contain text strings that merely resemble dates (e.g., "01/01/2023" stored as text), the `Range("A" & i)` portion of the code might retrieve an unexpected value, potentially causing the `Weekday` function to return an error or an incorrect numerical index. Therefore, verifying the number format of the source column (A2:A9) is a non-negotiable step before running the automation script.

Our objective is to subsequently place the calculated weekday names into column B, adjacent to the source dates. This setup creates a clear mapping between the input date and the output weekday string. We must ensure that the output **Range** (B2:B9) is clear and formatted generally (or as text) to accept the strings returned by the **WeekdayName** function. This preparation allows the **macro** to execute seamlessly and provide the desired results efficiently across the specified data set.

## Step-by-Step VBA Implementation: Creating the FindWeekdayName Macro

To demonstrate the practical application of the **WeekdayName** function, we will construct a robust **macro** named `FindWeekdayName`. This procedure is designed to iterate through our specified input data (A2:A9) and dynamically populate the corresponding output cells (B2:B9) with the day names. Creating this macro involves opening the **VBA** editor (Alt + F11), inserting a new Module, and

pasting the necessary code structure. This isolation within a standard module ensures the macro is accessible and can be executed independent of specific sheet events.

The first key step in the code is the declaration of the loop counter: `Dim i As Integer`. This variable, `i`, will serve as the row index, allowing the macro to move sequentially from the starting row (2) to the ending row (9). Utilizing an **Integer** for the counter is appropriate given the relatively small **Range** of rows we are processing. The `For i = 2 To 9` loop structure initiates the iterative process, ensuring every date within the target range is processed exactly once, which is a fundamental practice in iterating over datasets in **VBA**.

Within the loop lies the core logic: `Range("B" & i) = WeekdayName(Weekday(Range("A" & i)))`. This single line performs the entire operation. On the right side, `Range("A" & i)` retrieves the date value from the current row in column A. This value is immediately passed to the `Weekday` function, which converts it to a day index (1-7). Finally, the `WeekdayName` function takes that index and returns the string name of the day. This resulting string is then assigned to the cell on the left side, `Range("B" & i)`, effectively writing the weekday name into the adjacent column. This entire structure encapsulates the elegance and efficiency of combining native **VBA** functions to achieve complex data transformation tasks.

We can create the following **macro** to do so:

**Sub FindWeekdayName()**

Dim i As Integer

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

End Sub

## Analyzing the VBA Code Structure and Execution Flow

The provided **VBA** snippet, `FindWeekdayName`, serves as a canonical example of automating sequential data processing in **Excel**. The structure begins with the standard `Sub` declaration, defining the scope of the procedure. The declaration of variable `i` as an **Integer** sets up the counter for the loop. Although modern **VBA** often favors `Long` for row counters to handle extremely large spreadsheets, `Integer` is perfectly adequate for this small eight-row example, reinforcing readability and simplicity.

The execution flow follows the `For...Next` loop iteration logic. The macro starts at row `i = 2`,

processes the calculation, assigns the result, and then increments `i` to 3, continuing until `i` reaches 9. For each iteration, the macro accesses the date in column A using dynamic cell referencing (e.g., `Range("A" & 2)`, then `Range("A" & 3)`, etc.). This dynamic referencing is a powerful concept in **VBA**, allowing a single line of code to apply logic across a vast array of cells without manual modification for each row.

Crucially, the nested function call ensures that the process is efficient. The `WeekdayName` function receives its input instantly from the `Weekday` function without needing any intermediary variable storage, which optimizes memory usage and speed. Once the loop concludes at `Next i` after processing row 9, the program terminates with `End Sub`, leaving the processed results directly in the Excel sheet. This entire methodology exemplifies how structured procedural programming can be used to solve repetitive data transformation tasks within a spreadsheet environment, fulfilling the initial goal of translating date serial numbers into descriptive weekday names across the specified **Range**.

## Interpreting the Output: Results and Validation

Upon successful execution of the `FindWeekdayName` **macro**, the user observes immediate changes in column B of the spreadsheet. The process transforms the numerical date data in column A into textual day names, significantly enhancing the readability and reporting quality of the sheet. The output confirms that the **WeekdayName** function performed its intended conversion accurately, mapping the day index returned by the `Weekday` function to the standard English weekday name.

When we run this **macro**, we receive the following output:

| | 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 | | | | | |



As clearly demonstrated in the resulting spreadsheet, Column B displays the name of the weekday for each corresponding date in column A. This immediate visual confirmation is key to validating the logic of the **VBA** macro. For instance, if a date in column A was a known Saturday, and the corresponding cell in column B displayed "Saturday," the function is working as expected. If the date input was correctly formatted and the system locale aligns with the default settings (Sunday being day 1), the output strings should align perfectly with calendar dates.

We can verify a few examples to confirm the precision of the function:

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

## Advanced Usage Considerations and Common Alternatives

While the direct application of `WeekdayName(Weekday(...))` is highly effective, advanced **VBA** developers often employ techniques to handle localization and dynamic range selection more effectively. When dealing with international data, explicitly setting the `FirstDayOfWeek` argument within the `Weekday` function is crucial to avoid mislabeling days. For instance, if the target region

uses Monday as the start of the week, failing to specify `vbMonday` could lead to off-by-one errors in the day index passed to `WeekdayName`, resulting in incorrect weekday names being displayed.

A primary alternative to using the **WeekdayName** function in **Excel** is utilizing the built-in `Format()` function. The `Format()` function is extremely versatile and can achieve the same textual result by applying specific formatting masks (e.g., `Format(DateValue, "dddd")`). While `WeekdayName` is more specialized and often easier to read in context when solely focused on day names, `Format()` provides greater flexibility for custom date and time formatting, including abbreviations and locale-specific outputs. Developers may choose `Format()` if they need to apply multiple formatting changes simultaneously, but `WeekdayName` remains the cleaner choice for simple day identification.

Another common improvement to the provided **macro** involves dynamically determining the last row of data instead of hardcoding the loop limits (e.g., `To 9`). By using constructs like `Cells(Rows.Count, "A").End(xlUp).Row`, the macro can automatically adjust to datasets of any size, significantly increasing its utility and robustness. When integrating the **WeekdayName** function into production-level code, always prioritize dynamic range determination to ensure future compatibility and prevent runtime errors when data sizes change.

**Note**: You can find the complete documentation for the **VBA** **WeekdayName** function and its parameters on the official Microsoft Developer Network website.

## Summary: Key Takeaways for Date Conversion

The **WeekdayName** function is a cornerstone of effective date handling in **VBA**, simplifying the process of converting numerical date indices into human-readable strings. The structure demonstrated in the `FindWeekdayName` **macro**--nesting the `Weekday` function within `WeekdayName`--is the standard practice for automating this conversion across large datasets in **Excel**.

Effective utilization relies on careful handling of the function's arguments, particularly the optional `Abbreviate` and `FirstDayOfWeek` parameters, which dictate the output format and adherence to regional standards, respectively. By understanding the distinction between the **Integer** output of `Weekday` and the String output of `WeekdayName`, developers can confidently select the correct function combination for their reporting needs, whether for calculations or textual display.

The application of this function, particularly within efficient looping structures targeting specific data **Ranges**, is fundamental to creating automated, error-free data processing solutions that enhance the utility of any spreadsheet application. Always ensure your source data is correctly formatted as dates to guarantee the numerical indices provided to `WeekdayName` are accurate.