Learning VBA: A Comprehensive Guide to the WeekdayName Function for Extracting Day Names

Authored by

Mohammed looti

November 9, 2025

The Power of the VBA WeekdayName Function

The ability to efficiently manipulate and analyze date serial numbers is fundamental to advanced spreadsheet operations and sophisticated data processing within Microsoft Excel. At the heart of this capability lies VBA (Visual Basic for Applications), which offers developers a suite of powerful tools for date handling. One of the most useful functions for presentation and reporting is the **WeekdayName** function. This utility is specifically engineered to translate the numerical representation of a day--a value typically produced by the related **Weekday** function--into its full, human-readable name, such as "Monday" or "Tuesday." Mastering the implementation of the WeekdayName function is essential for constructing professional, user-friendly reports and automating complex date-based systems within the Excel environment.

Unlike many standard, built-in Excel worksheet functions that handle date formatting, the **WeekdayName** function operates exclusively within the VBA execution environment. This distinction provides developers with precise control over date output presentation and formatting within custom macro scripts. By integrating this tool into your procedures, you can transform cryptic date data into clear, descriptive textual names, significantly enhancing the overall clarity and accessibility of your automated data analysis results. Furthermore, its native integration with other VBA date functions makes it highly flexible for large-scale data cleansing and reporting tasks.

To begin understanding its utility, consider the core logic required to process multiple dates simultaneously. Since **WeekdayName** requires a number (1-7) as input, it must be nested around the **Weekday** function, which first extracts the required numerical index from a standard date object. The following snippet demonstrates the concise implementation of this nesting within a simple loop structure designed to process a range of dates.

```
Sub FindWeekdayName()

Dim i As Integer

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

End Sub
```

This initial example illustrates a highly efficient approach to date processing. Specifically, this particular macro is designed to iterate seamlessly through a predefined list of dates located in the range **A2:A9** of the active worksheet. For every date encountered in column A, the script executes the nested function calculation, determining the corresponding day of the week name, and subsequently displaying the resultant text string in the corresponding cell within the range **B2:B9**.

This simple but powerful iteration loop forms the bedrock of most large-scale, automated date processing assignments within the VBA environment.

Deconstructing the Syntax and Arguments of WeekdayName

Effective utilization of the **WeekdayName** function requires a comprehensive understanding of its syntax and the arguments it accepts. The function's core objective is straightforward: receive a numerical value representing the day of the week (ranging from 1 through 7) and return the corresponding textual name. While the function may appear simple in its most basic form, its full syntax provides powerful customization options regarding the output format and the definition of the week's starting day, which is crucial in international or financial contexts.

The standard syntax structure is defined as: `WeekdayName(WeekdayNumber, Abbreviate, FirstDayOfWeek)`. The first argument, `WeekdayNumber`, is mandatory; this is the Integer value (by default, 1 represents Sunday, 2 is Monday, and so on) that must be supplied to the function. Because Excel stores dates internally as serial numbers, developers almost universally need to nest the Weekday function inside **WeekdayName**. This inner function is responsible for extracting the necessary numerical index from the standard date object before the outer function can perform the text conversion.

The second argument, `Abbreviate`, is optional but highly useful for generating concise reports. If this argument is set to `True`, the function returns a three-letter abbreviation (e.g., "Sun" instead of "Sunday"). If `False` or simply omitted, the full descriptive name is returned. The third argument, `FirstDayOfWeek`, is also optional but carries significant weight, particularly when managing data sets that adhere to specific international standards or unique business calendars. This argument allows the user to explicitly define which day is considered the start of the week, using predefined VBA constants such as `vbMonday` or `vbSunday`. If this argument is omitted, the function defaults to the system's regional settings, which typically define Sunday as the first day (1).

Careful and consistent consideration of the `FirstDayOfWeek` argument is paramount for ensuring calculation accuracy. It is vital to remember that if the input `WeekdayNumber` (derived from the inner Weekday function) is calculated based on a week starting on Monday (e.g., using `vbMonday`), then **WeekdayName** must also be explicitly informed of this convention through its own `FirstDayOfWeek` argument. Failure to align these optional parameters between the nested functions can result in systematic errors, where, for example, a date that is truly a Thursday might be incorrectly identified and displayed as a Friday, leading to flawed data analysis and reporting.

Automating Weekday Identification Across a Date Range

A frequent requirement in professional data analysis involves transforming a column filled with raw

date values into a corresponding descriptive column of weekday names. This transformation is invaluable when analyzing behavioral trends based on the day of the week--for instance, charting website traffic surges or determining sales volume fluctuations that correlate with specific weekdays. Our objective is to automate this entire process using a reliable macro capable of handling any range of dates, whether that range is explicitly specified by the user or hardcoded within the script.

The core automation mechanism provided in our examples utilizes the standard **For...Next** loop, a fundamental building block of procedural macro programming. We declare the loop counter variable `i` as an Integer to accurately represent the row index. The loop is systematically set to start at row 2 (assuming row 1 is reserved for a header label) and continues through row 9, precisely corresponding to the range **A2:A9** containing our sample date inputs. This structure ensures that every relevant row is processed exactly once.

The most critical line of code within the iteration is: `Range("B" & i) = WeekdayName(Weekday(Range("A" & i)))`. This single statement performs three essential data manipulation actions in rapid sequence, demonstrating the efficiency of function nesting:

Range(**"A" & i**): This initial step extracts the raw date value, which is stored as a serial number, from the current cell being processed in column A (e.g., A2, A3, etc.).

Weekday(...): The nested inner function receives the extracted date serial number and immediately returns a numerical equivalent for the day of the week, ranging from 1 (Sunday) to 7 (Saturday).

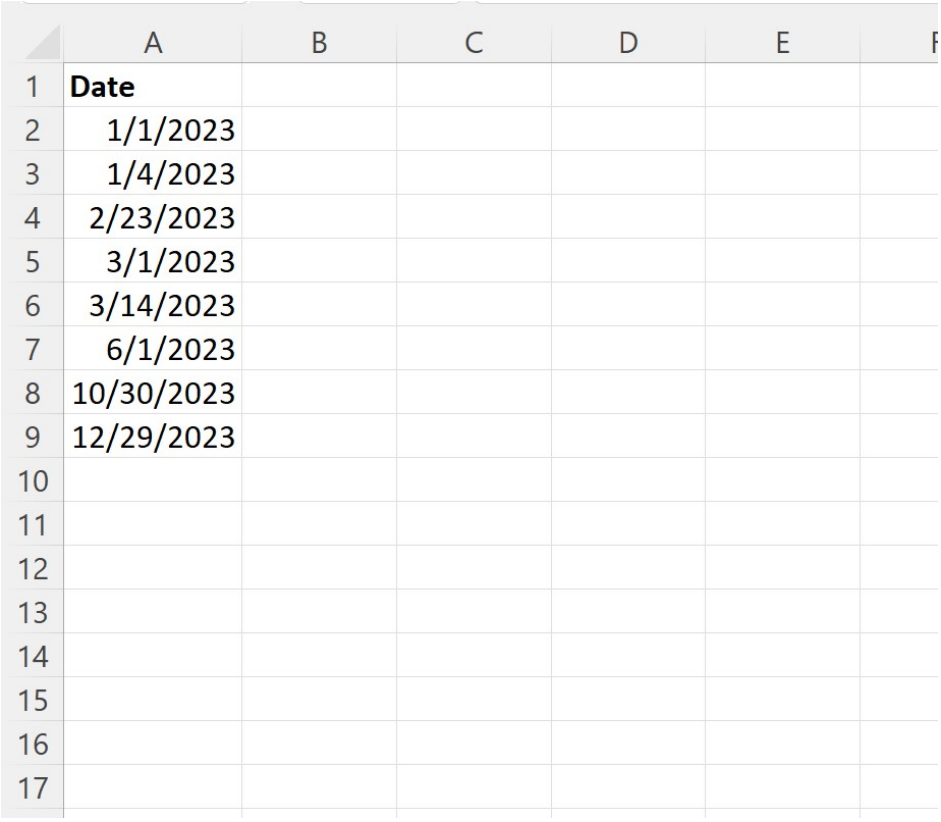
WeekdayName(...): Finally, the outer function accepts this numerical index output and converts it into the desired descriptive text string (e.g., "Sunday"). This resulting text is then assigned to the corresponding cell in column B (**Range("B" & i)**).

This elegant nesting technique ensures that the raw date data is processed entirely within the confines of the script, immediately yielding the desired text output without necessitating any manual data intervention or the use of intermediate calculation cells within the spreadsheet itself. This methodology powerfully underscores the efficiency achieved by combining native VBA date functions.

Step-by-Step Implementation and Code Execution

To solidify the understanding of this process, let us examine a concrete scenario involving a column of dates that require conversion. Assume we have compiled a small data set containing eight different dates, all maintained as standard Excel date values, located specifically in column A of the active worksheet. Our clear objective is to generate a parallel column, column B, which will contain the human-readable weekday name corresponding to each date entry.

The initial state of the worksheet, clearly displaying the raw date inputs, is shown in the image below. These dates intentionally span several months and years, providing a realistic and robust test case to verify the **WeekdayName** function's accuracy across varied calendar structures and date formats.



	A	B	C	D	E	F
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						

To execute the required transformation, we must first access the VBA Editor (typically Alt + F11), insert a new standard module, and define the procedure named `FindWeekdayName`. The macro snippet provided below contains the precise script necessary to perform this task based on our defined range of **A2:A9**. This structure is highly portable and can be easily adapted to much larger data sets simply by adjusting the loop boundaries.

This code block explicitly defines the scope of the iteration using the `For i = 2 To 9` statement. It ensures that the calculation is meticulously performed row-by-row, establishing a precise, one-to-one mapping between the input dates in column A and the output cells in column B. The use of the **Range** object dynamically concatenated with the loop counter `i` guarantees accurate cell referencing throughout the entire execution of the script.

Sub FindWeekdayName()

Dim i As Integer

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

End Sub
```

Once this script is placed into a standard module and executed (either via F5 in the VBA editor or through the Excel Macro interface), the program rapidly processes all eight date entries. This programmatic approach offers vastly superior efficiency compared to manually typing complex formulas or repeatedly using the built-in Excel `TEXT()` function, especially when dealing with data sets that span hundreds or thousands of rows. The immediate result is a clean, accurate, and automated translation of dates into their corresponding weekday names, making the data instantly suitable for further analytical work or reporting.

Verifying Accuracy: Analyzing the Script Output

Upon the successful execution of the `FindWeekdayName` macro, the resulting weekday names are immediately visible and populated in column B of the active worksheet. This output serves as validation that the nested structure--combining the Weekday function with the WeekdayName function--correctly interpreted the underlying serial date values and returned the appropriate day names as descriptive text strings.

The following image clearly displays the worksheet after the execution of the script, showing column B fully populated with the calculated weekday names:

	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 illustrated, column B now provides the unambiguous name of the weekday for each corresponding date entry in column A. This specific transformation is critically important for report generation and user comprehension, as it eliminates the need for users to manually cross-reference dates with a calendar. We can confirm the accuracy of the script's calculations by verifying a few specific data points against a standard Gregorian calendar:

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

These verification points confirm that the default system settings used by the Weekday function (where Sunday is designated as the first day of the week, coded as 1) were accurately communicated to and interpreted by the **WeekdayName** function. This successful alignment yields the accurate textual representation for every date within the analyzed range, demonstrating the inherent reliability of this combined function approach in VBA date handling.

Distinguishing WeekdayName from the Weekday Function

Although these two functions are frequently employed together through nesting, it is absolutely

essential for developers to clearly distinguish between the **WeekdayName** function and the standalone Weekday function. They serve fundamentally different purposes and return distinct data types. Misunderstanding this crucial distinction is a common source of runtime errors or flawed logical flow within custom macro development.

The **Weekday** function's exclusive purpose is to accept a date input and return a numerical value (an Integer) ranging from 1 to 7, which numerically identifies the day of the week. This numerical output is primarily useful for conditional logic within the script--for example, skipping processing steps on weekend dates or applying specialized calculations only to Mondays. Crucially, the output is a number, suitable for mathematical and logical comparisons, not text.

In sharp contrast, the **WeekdayName** function requires that numerical output as its input and solely returns a **String** (text). This function is dedicated purely to presentation, formatting, and readability for the end-user. If your application requires filtering, sorting, or numerical aggregation based on the day (e.g., summing all transactions that occurred on day 5 of the week), you must utilize the **Weekday** function directly. Conversely, if your primary goal is to populate a report column, label a chart axis, or display the result clearly in a user interface, then **WeekdayName** is the necessary tool.

The core principle governing the choice between them hinges entirely on the required output format: if you need a numerical index (1 through 7) for internal script processing, use **Weekday** alone. If you require a descriptive text string for display, use **WeekdayName**, often nested around **Weekday**. This distinction ensures that the data type matches the application's ultimate need, preventing logical errors in the script.

Conclusion and Expert Resources

The **WeekdayName** function stands as an indispensable component of the VBA toolkit for any developer or analyst who works extensively with date data within Microsoft Excel. By enabling the easy conversion of numerical weekday indexes into clear, descriptive names, the function dramatically improves the readability, professionalism, and utility of automated reports and data exports. Achieving mastery over the concept of nesting **Weekday** inside **WeekdayName**, as demonstrated through our comprehensive, practical example, ensures accurate and highly efficient date processing across any defined data range.

For developers seeking deeper technical insight into the function's optional arguments (such as `Abbreviate` or `FirstDayOfWeek`) and how to handle potential edge cases related to regional settings, the official Microsoft documentation remains the definitive and authoritative reference. The complete technical specifications for the VBA **WeekdayName** function are detailed thoroughly, offering critical context for advanced implementations.

To continue expanding your proficiency in automating sophisticated tasks within Microsoft Excel, we recommend exploring additional tutorials that cover related date manipulation, formatting techniques, and the use of other foundational macro objects.

Additional Resources for VBA Proficiency

The following resources explain how to perform other common, productivity-boosting tasks in VBA, building directly upon the foundational knowledge gained from mastering essential date functions: