How to Determine the Day of the Week in VBA Using the Weekday Function

Authored by

stats writer

February 24, 2026

Overview of the VBA Weekday Functionality

In the sophisticated world of **Microsoft Excel** automation, the ability to parse and manipulate chronological data is a fundamental skill for any developer. The **Visual Basic for Applications** (VBA) environment provides a variety of built-in tools to handle these tasks, with the **Weekday** function standing out as one of the most versatile. This function is specifically designed to evaluate a given **date** and return a numerical value that corresponds to its day of the week, allowing for seamless integration into larger data processing workflows.

Utilizing the **Weekday** function is essential for tasks that require the categorization of records based on the day they occurred. For instance, financial analysts might use this function to distinguish between weekday transactions and weekend activity, while project managers might use it to automate the calculation of deadlines, ensuring that non-working days are excluded. By converting a complex **date** string into a simple **integer**, the function simplifies conditional logic within a **programming** script, making it easier to write and maintain.

The standard behavior of this function is to map the days of the week to a scale of 1 through 7. By default, the system recognizes Sunday as the first day of the week, assigning it the value of 1, while Saturday is designated as 7. This standardized numerical output provides a consistent framework for **macro** logic, enabling developers to create triggers and filters that respond dynamically to the calendar. For example, a report that should only execute on a Monday can be programmed to check if the current **Weekday** value is equal to 2.

Beyond its basic utility, the **Weekday** function is highly valued for its role in data analysis and organizational efficiency. Whether you are managing a simple personal budget or a massive enterprise-level database, understanding how to programmatically identify the day of the week is a critical step toward full automation. The following sections will explore the technical syntax, practical examples, and advanced implementation strategies for using this function effectively within your **Excel** projects.

Technical Syntax and Optional Arguments

To implement the **Weekday** function correctly, it is important to understand its underlying **syntax** and the various parameters it accepts. At its core, the function requires a **date** expression as an input, which can be a literal date, a cell reference containing a date, or a variable of the **Date** type. While the primary argument is straightforward, **VBA** also allows for an optional second argument known as `FirstDayOfWeek`, which permits the user to redefine the starting point of the weekly cycle based on regional or professional requirements.

The flexibility of the `FirstDayOfWeek` argument is particularly useful in international contexts where the start of the workweek varies. While the default is `vbSunday` (1), users can specify

`vbMonday` (2), `vbTuesday` (3), and so forth. This level of customization ensures that the **integer** output remains relevant to the specific calendar system being used. Furthermore, **VBA** provides the `vbUseSystemDayOfWeek` constant, which automatically adopts the settings defined in the user's regional operating system configuration.

In many professional scripts, developers prefer to access this tool through the **WorksheetFunction** **object**. By calling `WorksheetFunction.Weekday`, the **macro** leverages the same logic found in the standard **Excel** worksheet formulas. This approach is often favored for its reliability and its ability to handle **Excel** range objects directly, as seen in many common automation patterns. Understanding these subtle differences in invocation is key to writing robust and error-free code.

Finally, it is worth noting that the **Weekday** function is strictly numerical in its output. It does not return the name of the day (e.g., "Monday") but rather the **integer** representing that day. This design choice is intentional, as integers are much more efficient for comparison operations and mathematical calculations within a **programming** environment. For those who require the text-based name of the day, **VBA** offers the complementary `WeekdayName` function, which can be used in tandem with the results of the **Weekday** function.

Practical Example: Automating Weekday Calculations

To illustrate the practical application of this function, let us consider a scenario where an **Excel** user needs to process a long list of dates and determine which day of the week each one falls on. Manually calculating this for hundreds of rows would be inefficient and prone to human error. By utilizing a **VBA** **macro**, we can automate this entire process, populating an adjacent column with the correct **integer** values in a matter of milliseconds.

The following code snippet demonstrates a common method for achieving this task. This **Sub** procedure, titled `FindWeekday`, utilizes a **For** loop to iterate through a specific range of cells. In this instance, the code is designed to examine dates in column A and output the corresponding weekday number into column B. This type of automation is highly effective for preparing datasets for further analysis, such as pivot tables or custom reporting dashboards.

Sub FindWeekday()

Dim i As Integer

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

```
End Sub
```

As shown in the logic above, the variable `i` serves as a row counter, beginning at row 2 and continuing through row 9. For each iteration of the loop, the **macro** reads the value in column A, passes it into the `WorksheetFunction.Weekday` method, and then writes the resulting value into the corresponding cell in column B. This approach is scalable; the range can easily be expanded to accommodate thousands of rows by simply adjusting the loop parameters or making them dynamic based on the actual data volume.

By implementing such a routine, users can ensure that their data is consistently tagged with the correct day of the week. This is particularly beneficial when dealing with large datasets where manual entry is not feasible. Furthermore, because the **Weekday** function is part of the core **Excel library**, it performs exceptionally well even when processing significant amounts of information, making it a reliable choice for high-performance **VBA** development.

Interpreting the Numerical Results

Once the **Weekday** function has been executed and the results are populated in your worksheet, it is crucial to understand how to interpret the numerical values returned. Since the function utilizes a specific mapping system, knowing which **integer** corresponds to which day is vital for any subsequent logic or data visualization. The default **Excel** configuration uses a 1-to-7 scale that is widely recognized in many Western business environments.

The following list outlines the standard mapping used by **VBA** when the default settings are applied:

1: Indicates that the date falls on a **Sunday**.
2: Indicates that the date falls on a **Monday**.
3: Indicates that the date falls on a **Tuesday**.
4: Indicates that the date falls on a **Wednesday**.
5: Indicates that the date falls on a **Thursday**.
6: Indicates that the date falls on a **Friday**.
7: Indicates that the date falls on a **Saturday**.

Understanding this mapping is essential when writing conditional statements. For example, if you wanted to highlight all weekend dates in a spreadsheet, you would look for values of 1 (Sunday) and 7 (Saturday). Alternatively, if your business logic focuses on the workweek, you might create a filter that only includes records where the **Weekday** value is between 2 and 6. This numerical representation allows for very efficient filtering and sorting compared to string-based day names.

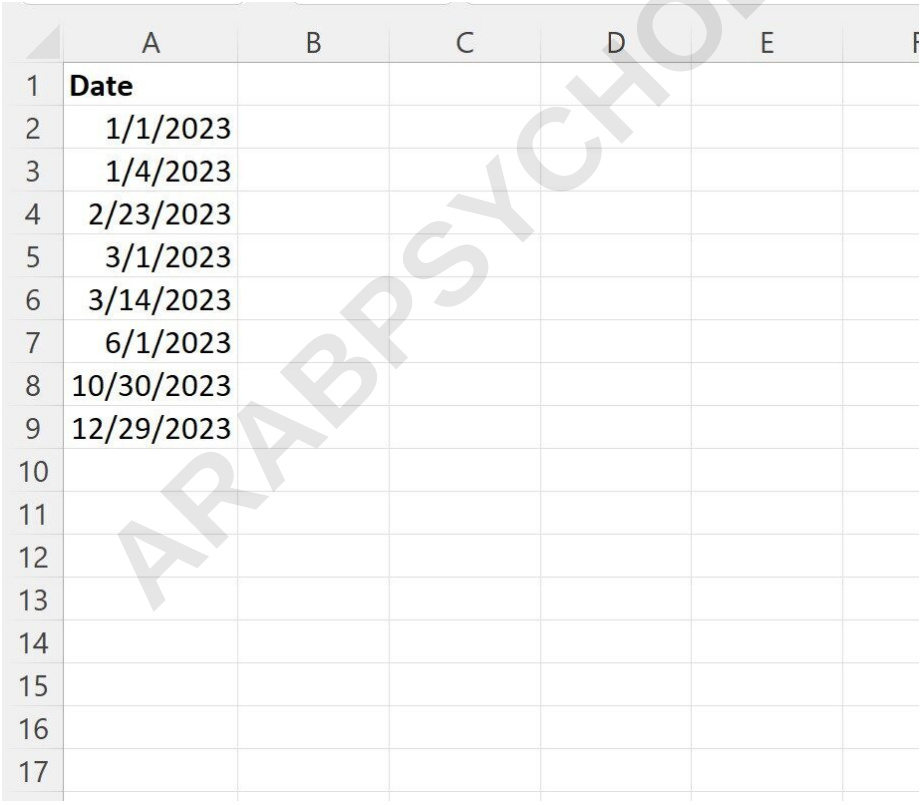
It is important to remember that if you have modified the `FirstDayOfWeek` argument, this mapping

will shift accordingly. For instance, if you set Monday as the first day of the week, Monday would return 1 and Sunday would return 7. Consistency is key here; always ensure that your interpretation logic matches the parameters you have passed to the function in your **VBA** code to avoid logical errors in your final report.

Step-by-Step Walkthrough: Finding Day of Week

To visualize how this function works within a real **Microsoft Excel** environment, let us examine a specific example involving a set of disparate dates. Suppose we have a column of dates that represent various company milestones or transaction dates. The goal is to identify the day of the week for each entry and display that information clearly in an adjacent column for easier sorting and categorization.

The image below illustrates the starting point for our dataset, where column A contains several dates formatted in the standard MM/DD/YYYY format. Without the **Weekday** function, determining the specific day for each of these would require checking a calendar, which is inefficient for large datasets.



	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 solve this, we implement our **macro** which will programmatically calculate the weekday for each row. The code below is the exact procedure used to perform this calculation. It demonstrates the use of variables and the **WorksheetFunction** to bridge the gap between **VBA** logic and

spreadsheet data.

```
Sub FindWeekday()

Dim i As Integer

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

End Sub
```

After executing the **macro**, the results are instantly populated in column B. Each row now contains an **integer** that corresponds to the day of the week for the date in column A. This transformation turns static date information into dynamic, actionable data that can be used for further **programming** logic or user-facing reports.

The resulting output in the spreadsheet will look like the following image, showing the clear relationship between the **date** input and the numerical output:

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

In this specific example, the function correctly identifies that 1/1/2023 is a Sunday, returning the value of 1. Similarly, 1/4/2023 is identified as a Wednesday (returning 4), and 2/23/2023 is identified as a Thursday (returning 5). This automated verification process ensures that all records are accurately tagged without the need for manual oversight.

Differentiating Weekday and WeekdayName

While the **Weekday** function is incredibly useful for logical operations, there are many instances where a developer might need to display the actual name of the day to the end-user. In these cases, **VBA** provides a separate but related function called `WeekdayName`. Understanding the distinction between these two functions is vital for creating professional and user-friendly **Excel** applications.

The **Weekday** function is primarily used for calculation and comparison. Because it returns an **integer**, it is the preferred choice for `If...Then` statements or `Select Case` blocks. For example, if you want to run a specific piece of code only on weekends, it is much faster and more reliable to check if `Weekday(dt) = 1 Or Weekday(dt) = 7` than it is to check for the strings "Sunday" or "Saturday," which might be subject to spelling errors or localization issues.

On the other hand, the `WeekdayName` function takes an **integer** (usually provided by the **Weekday** function) and converts it into a string representing the day's name. This is ideal for generating headers, titles, or messages that will be read by human users. For instance, a message box that says "The report was generated on a Wednesday" is much more informative than one that says "The report was generated on day 4."

By combining these two functions, you can achieve both technical efficiency and clarity. A common pattern in **macro** development is to use **Weekday** to perform the logic and `WeekdayName` to display the results. This dual-layered approach ensures that your code remains robust while still being accessible to users who may not be familiar with the numerical mapping of the calendar week.

Best Practices for Date Manipulation in VBA

Working with dates in **VBA** requires a disciplined approach to ensure that your scripts are both accurate and performant. One of the most important best practices is to always declare your variables with specific data types. Using `Dim dte As Date` instead of a generic `Variant` helps prevent type-mismatch errors and allows the **VBA** compiler to optimize the code. This is especially true when passing values to the **Weekday** function, as it expects a valid date format to function correctly.

Another key consideration is the handling of different **date** formats across various regions. **Excel** stores dates as serial numbers, but they are often displayed in formats like DD/MM/YYYY or

MM/DD/YYYY. To avoid confusion, it is best to use the `DateSerial` function when hardcoding dates into your **macro**, or to ensure that the source data in your cells is properly formatted as a **Date** type rather than a simple text string.

Error handling is also a critical component of professional **programming**. If the **Weekday** function encounters a cell that contains text instead of a date, it will trigger a runtime error. To make your **macros** more resilient, you should include checks such as `IsDate()` before calling the function. This allows the script to skip invalid entries or notify the user of the error without crashing the entire application.

Finally, keep in mind the scope of your automation. For small ranges, the `For...Next` loop shown in our example is perfectly adequate. However, for datasets with tens of thousands of rows, you might consider reading the range into an array first, processing the data in memory, and then writing the array back to the sheet. This technique significantly reduces the number of interactions with the **Excel** worksheet, leading to much faster execution times and a smoother user experience.

Advanced Use Cases and Further Learning

Once you have mastered the basics of the **Weekday** function, you can begin to explore more advanced use cases that leverage this tool for complex business intelligence. For instance, you could use it to create a dynamic calendar within **Excel** that automatically updates based on the current month. By calculating the weekday of the first day of the month, you can determine where the dates should be placed within a grid, allowing for the creation of custom scheduling tools.

Another advanced application involves integrating **Weekday** with other **VBA** functions like `DateAdd` or `DateDiff`. This allows you to perform sophisticated calculations, such as finding the date of the next occurring Friday or determining the number of business days between two specific dates. These types of calculations are invaluable for HR departments calculating leave time or logistics companies managing delivery schedules across different time zones.

For those interested in diving deeper into the technical specifications of **VBA**, the official Microsoft documentation is an excellent resource. It provides comprehensive details on the **Weekday** function, including all available constants and detailed descriptions of how the function interacts with different **Microsoft Excel** versions. Exploring these resources will help you expand your toolkit and become a more effective **VBA** developer.

In conclusion, the **Weekday** function is a powerful and essential tool for anyone looking to automate **Excel** tasks. By providing a simple way to convert dates into numerical values, it opens the door to a wide range of analytical and organizational possibilities. Whether you are building simple **macros** to tag dates or developing complex systems for data management, the **Weekday** function will undoubtedly be a central part of your development process.