How to get day of week from date in SAS?

Authored by

stats writer

November 19, 2025

Introduction to Date Management in SAS

The ability to accurately process and extract components from date variables is fundamental to effective data analysis using SAS (Statistical Analysis System). Often, analysts need to determine not just the year or month, but specifically the day of the week associated with a transaction, event, or birthdate. This information is crucial for time series analysis, scheduling optimization, and identifying weekly patterns within large datasets. Fortunately, SAS provides highly efficient built-in functions designed specifically for this purpose, allowing users to convert the raw numerical representation of a SAS date into a meaningful day-of-week indicator.

In SAS, a date is stored internally as the number of days since January 1, 1960. This internal structure, known as a SAS Date Value, requires special handling to translate it back into human-readable components. To retrieve the day of the week, we primarily rely on two powerful tools: the **WEEKDAY** function, which provides a numeric output, and the **PUT** function combined with the **DOWNAME** format, which yields the textual name of the day. Mastering these techniques is essential for any data professional working extensively with temporal variables.

This detailed guide will explore the mechanics behind these functions, demonstrating how to implement them in a practical data step environment. We will cover the specific outputs of each method, clarifying the necessary syntax and best practices for converting date values into reliable day-of-week data points. Understanding the difference between the numeric and character output options allows analysts to select the most appropriate method based on subsequent analytical needs, whether that involves filtering by day index or reporting descriptive statistics based on day names.

Understanding Date Values and Formats in SAS

Before diving into the extraction functions, it is vital to grasp how SAS manages date and time information. Unlike standard calendar representations, SAS stores dates internally as continuous integers. For instance, the date January 1, 1960, is represented as 0, and subsequent dates increment by one. This system ensures consistent handling of time differences and calendar complexities across various operating environments. When you input a date like '01JAN2021', SAS converts this external representation into its corresponding internal numerical value.

The formatting of these dates is handled through specialized formats, such as DATE9. or DDMMYY10., which dictate how the internal SAS Date Value is displayed to the user or stored in an output file. However, these formats only control presentation; they do not alter the underlying numeric value used for calculations. When applying functions like **WEEKDAY**, the function operates directly on this raw numerical date value, interpreting it relative to the established SAS calendar epoch. This separation between internal storage and external display is a cornerstone of SAS date handling efficiency.

To effectively work with temporal data, especially when determining the day of the week, ensuring that the input variable is recognized by SAS as a genuine date variable is critical. If the variable is stored as a character string, it must first be converted into a numerical SAS date using functions like **INPUT** or the **ANYDTTE** function before the day-of-week extraction can proceed. If this conversion step is skipped, the **WEEKDAY** function will produce erroneous or missing results, as it is expecting a numerical date integer as its input argument.

Primary Methods for Extracting the Day of the Week

Analysts have two primary, highly recommended methods for extracting the day of the week from a SAS Date Value. The choice between these methods depends entirely on the required output format: a numerical index for computational purposes (e.g., aggregating weekend data) or a descriptive string for reporting (e.g., displaying "Monday"). Both methods are implemented efficiently within the SAS DATA step and offer reliable results across all modern versions of the software.

The first method utilizes the **WEEKDAY** function, which is the most direct approach for obtaining a numeric representation of the day. This function follows a standardized convention used in many statistical environments: Sunday is indexed as 1, and the sequence progresses linearly through the week until Saturday, which is indexed as 7. This numerical output is ideal when the analyst needs to easily filter, sort, or perform mathematical operations based on the position of the day within the week.

The second method combines the **PUT** function with the specialized **DOWNAME** format. Unlike **WEEKDAY**, this combination generates a character string containing the full name of the day (e.g., "Wednesday"). While this approach is slightly more verbose in terms of syntax, it is invaluable for creating final, presentable datasets where the textual clarity of the day is paramount. We will explore each of these powerful tools in detail, providing the necessary syntax to implement them correctly.

Method 1: Using the WEEKDAY Function

The WEEKDAY function is arguably the simplest way to retrieve the day of the week numerically in SAS. Its syntax is straightforward, requiring only the numerical SAS Date Value as its sole argument. The function is designed to return an integer between 1 and 7, adhering to the convention where 1 is **Sunday** and 7 is **Saturday**. This fixed mapping ensures consistency regardless of regional settings or dataset properties.

When integrating this function into a **DATA step**, the calculated result can be assigned directly to a new numeric variable. For example, if your date variable is named `birth_date`, the syntax `weekday_number = WEEKDAY(birth_date);` immediately creates the required numerical day-of-

week variable. It is important to remember that the output is numeric, meaning the resulting variable will be treated as a number by SAS unless explicitly formatted otherwise for display.

The primary benefit of using the WEEKDAY function lies in its efficiency and direct application for sorting or grouping data. For instance, if an analyst wishes to filter all observations occurring on a weekend, they can simply check if the generated `weekday_number` is equal to 1 (Sunday) or 7 (Saturday). This makes complex conditional processing much simpler than trying to compare character strings like "Sunday" or "Saturday" later in the analysis process.

Method 2: Utilizing the PUT Function with DOWNAME Format

While the numeric output of **WEEKDAY** is analytically useful, reporting often requires the full textual name of the day. This is achieved by employing the versatile PUT function in conjunction with the **DOWNAME** format. The PUT function's fundamental role is to apply a specified format to a variable and return the result as a character string.

The DOWNAME format is specifically designed for date variables; when applied, it translates the numerical date value into the corresponding day name (e.g., Monday, Tuesday). The format takes an optional width parameter, though typically, a default width sufficient for the longest day name (Wednesday) is used. The syntax structure for this method is `new_variable = PUT(date_variable, DOWNAME.);`, ensuring the resulting variable is automatically created as a character type.

This approach is particularly valuable for generating reports and visualizations where clarity is key. It eliminates the need for manual look-up tables or custom format definitions to map the numbers 1 through 7 back to their respective day names. By using the PUT function with DOWNAME, the transformation is handled internally by SAS, minimizing the risk of errors associated with manual data manipulation and ensuring the textual output is correctly formatted for presentation.

Practical Example: Setting Up the Source Dataset

To illustrate the application of both the **WEEKDAY** and **PUT** functions, let us begin by creating a sample SAS dataset. This dataset, which we will call `original_data`, contains the birth dates for several individuals. Note that when defining the dataset, we use the **FORMAT** statement to ensure the `birth_date` variable is recognized and displayed correctly as a numerical date value using the DATE9. format. This setup is crucial for subsequent date calculations.

The following code snippet demonstrates the creation of this initial dataset using an in-line **DATALINES** block. The dates are entered in a standard SAS format (DDMMMYY), which the **INPUT** statement, along with the DATE9. informats, correctly interprets and stores as numerical SAS date values. Reviewing the initial dataset ensures that the foundation for our analysis is

correct before proceeding to the extraction phase.

Suppose we have the following dataset in SAS that shows the birth dates for seven individuals. The **PROC PRINT** step confirms the successful creation and formatting of the input table, displaying the dates in the chosen DATE9. format (e.g., 01JAN2021).

```
/*create dataset*/
data original_data;
format birth_date date9.;
input birth_date :date9.;
datalines;
01JAN2021
22FEB2022
14MAR2022
29MAY2022
14OCT2023
01NOV2024
26DEC2025
;
run;

/*view dataset*/
proc print data=original_data;
```

Obs	birth_date
1	01JAN2021
2	22FEB2022
3	14MAR2022
4	29MAY2022
5	14OCT2023
6	01NOV2024
7	26DEC2025

Executing the Day-of-Week Extraction

With the source data prepared, we can now execute the core logic required to derive the day of the week. This is accomplished within a new **DATA step**, which reads the `original_data` and creates a new dataset, `new_data`, containing the added day-of-week variables. This is where we apply the

two key functions discussed previously: WEEKDAY for the numeric index and PUT(..., DOWNAME.) for the character name.

The syntax is simple and efficient. For the numeric representation, we define `weekday_number` using the **WEEKDAY** function, feeding it the `birth_date` variable. For the textual representation, we define `weekday_name` using the **PUT** function combined with the **DOWNAME** format. It is crucial to use the trailing period (`.`) after `dowName` to correctly specify the format within the **PUT** function call.

This process generates two new variables that perfectly satisfy both analytical and reporting requirements. The resulting dataset, viewed using **PROC PRINT**, will clearly display the original date alongside its corresponding day index (1-7) and the day's full name. This combined output allows for immediate verification of the calculations and provides flexibility for subsequent steps in the data pipeline.

```
/*create new dataset*/
data new_data;
set original_data;
weekday_number = WEEKDAY(birth_date);
weekday_name = put(birth_date, dowName.);
run;

/*view new dataset*/
proc print data=new_data;
```

Obs	birth_date	weekday_number	weekday_name
1	01JAN2021	6	Friday
2	22FEB2022	3	Tuesday
3	14MAR2022	2	Monday
4	29MAY2022	1	Sunday
5	14OCT2023	7	Saturday
6	01NOV2024	6	Friday
7	26DEC2025	6	Friday

Interpreting the Results and Verification

Upon reviewing the output generated by the **PROC PRINT** statement on the `new_data` dataset, we

can clearly observe the successful creation of the two new variables: `weekday_number` and `weekday_name`. The results confirm that the **WEEKDAY** function provides the numerical index based on the Sunday=1 convention, and the **PUT** function provides the full textual label.

A systematic verification of a few dates ensures the accuracy of the extraction process. For example, consider the first date, January 1st, 2021. The output shows a `weekday_number` of 6 and a `weekday_name` of "Friday." This aligns perfectly with the calendar, confirming that the SAS date handling correctly calculated this date as the sixth day of the week. Similarly, February 22nd, 2022, is correctly identified as a Tuesday, corresponding to the numerical index 3.

This dual-variable approach offers robustness in data quality checks. If, for instance, a subsequent analysis step relies on the numeric index, the analyst can quickly cross-reference the index (e.g., 2) with the name ("Monday") displayed in the same row to spot any potential data entry or processing errors before proceeding. This confirmation step is critical in ensuring the integrity of time-based reporting, especially in highly regulated industries.

January 1st, 2021 is on a **Friday**, which is the 6th day of the week.

February 22nd, 2022 is on a **Tuesday**, which is the 3rd day of the week.

March 14th, 2022 is on a **Monday**, which is the 2nd day of the week.

Conclusion and Advanced Considerations

Mastering date component extraction, particularly determining the day of the week, is fundamental when working with temporal data in SAS. Whether you require a numeric index for computational speed using the **WEEKDAY** function or a descriptive string for reporting using **PUT(..., DOWNAME.)**, SAS offers clean and efficient solutions. These functions operate directly on the underlying SAS Date Value, ensuring accuracy and consistency across different datasets and projects.

While this tutorial focused on the primary methods, SAS offers related functions that might be useful in different contexts. For instance, the **DOWNAMEw.** format can be customized to abbreviate the day name if space is a concern (e.g., DOWNAME3. would return "Mon", "Tue", etc.). Understanding the range of date and time functions available allows analysts to select the precise tool necessary for nuanced analytical tasks, such as calculating the number of working days between two dates or adjusting calculations based on specific weekday rules.

To further enhance your skills in SAS date manipulation, consider exploring tutorials on how to calculate the week number of the year (using the **WEEK** function) or how to handle time zones effectively. These skills, built upon a solid foundation of date component extraction, prepare the

analyst for complex time series modeling and advanced data preparation required in enterprise-level statistical analysis.

The following tutorials explain how to perform other common tasks in SAS: