# How to use the MDY function in SAS?

Authored by

**stats writer**

November 19, 2025

Working with temporal data is a fundamental aspect of statistical programming and data analysis. In SAS, dates are internally managed as numeric values—specifically, the number of days since January 1, 1960. When source data provides date components (month, day, year) as separate numeric variables rather than a single recognized date string, analysts require a specific tool to harmonize these elements into a single, usable SAS date value. This critical transformation ensures accurate calculations, sorting, and reporting based on time.

The MDY function is precisely engineered for this purpose. It serves as a powerful utility for converting three independent numeric arguments representing the month, day, and year into the cohesive internal date format required by SAS. The function guarantees that the resultant numeric value accurately reflects the calendar date specified by the inputs, thereby simplifying subsequent data manipulation tasks that depend on chronological ordering.

By employing the **MDY** function, developers avoid manual concatenation errors and ensure computational accuracy. This function is essential when importing legacy data files, working with database extracts that split date components, or generating dates dynamically within a DATA step. Understanding its structure and application is key to mastering date and time processing within the SAS environment.

## Understanding the Syntax and Arguments of the MDY Function

To effectively utilize this tool, one must first grasp its fundamental syntax and the specific requirements for each input argument. The **MDY** function is designed for simplicity, requiring only the three constituent numeric parts of the date in a specific order: month, day, and year. This structure is intuitive and mirrors standard calendar notation, though the strictness of the input types is crucial for successful execution.

The syntax for invoking the MDY function is straightforward:

**MDY(month, day, year)**

Each argument passed to the function must adhere to specific data type and range constraints to ensure a valid output date. If the inputs result in an impossible date (e.g., February 30th), the function will typically return a missing value, which is represented internally by a period (`.`) in SAS. Therefore, validating input integrity before calling the function is often a recommended practice for robust programming.

Detailed definitions for the arguments are provided below:

**month:** This argument requires an integer value corresponding to the month of the year, ranging strictly from 1 (January) to 12 (December). Non-integer or out-of-range values will result in errors or missing date outputs.

**day:** This argument requires an integer value representing the day of the month. The valid range is 1 to 31, but the function automatically checks for calendar validity based on the provided month and year (e.g., limiting February days to 28 or 29).
**year:** This argument specifies the year and must be a numeric integer. SAS accepts both two-digit years (e.g., 22) and four-digit years (e.g., 2022). However, using four-digit years is highly recommended to prevent potential ambiguity issues related to the 100-year window reference (which defaults depend on the SAS system options, such as YEARCUTOFF).

## Practical Applications and Context for Using MDY

While SAS offers several methods for date handling, the **MDY** function shines particularly bright when dealing with input data that is fundamentally atomic. Many legacy systems or raw data exports store date components separately—perhaps due to database schema limitations or organizational conventions—where the month, day, and year reside in distinct numeric fields. Attempting to concatenate these fields using simple string manipulation or arithmetic operations would invariably lead to incorrect internal SAS date value representations, making chronological analysis impossible.

The primary utility of **MDY** is ensuring that the combination of these three numeric inputs is correctly translated into the standardized numeric count of days elapsed since the SAS epoch (January 1, 1960). Once this conversion is complete, the resulting variable can be correctly formatted, compared, and used in date arithmetic functions like **INTNX** or **INTCK**. Without this critical step, calculations such as measuring the difference between two dates would yield meaningless results, underscoring the function's foundational role in data preparation.

Consider scenarios in financial reporting, epidemiological studies, or inventory management where precise date calculation is paramount. If a transactional database provides `Tran_Month`, `Tran_Day`, and `Tran_Year`, the MDY function becomes the most efficient and reliable method within the DATA step to synthesize these parts into a chronological key. This process not only validates the date components implicitly but also prepares the data for advanced analytics requiring temporal integrity.

## Detailed Example: Setting up the Sample Data in SAS

To illustrate the practical application of **MDY**, let us construct a hypothetical dataset. Suppose a retail store tracks daily sales, and the data logger records the date of the transaction by separating the month, day, and year into distinct columns. Our goal is to consolidate this information into a single, cohesive date variable that SAS can recognize and use for time-series analysis or aggregating sales by quarter.

We begin by creating a sample dataset using the DATA step and **DATALINES** statement. This

setup mimics reading raw input where the date elements are numeric inputs corresponding to `month`, `day`, and `year`, followed by a numeric field for `sales` volume. This structure highlights the common challenge that **MDY** is designed to solve.

The following code block demonstrates the initial creation and structure of the `my_data` dataset. Notice how the date components are stored purely as integers before any transformation is applied:

```
/*create dataset*/
data my_data;
input month day year sales;
datalines;
4 15 2022 94
6 17 2022 88
7 25 2022 90
8 14 2022 105
10 13 2022 119
12 15 2022 100
1 4 2023 87
3 15 2023 90
5 29 2023 130
;
run;

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

After executing this code, the resulting output shows the raw data exactly as entered. The `month`, `day`, and `year` columns are visible as standard numbers. It is important to note that, at this stage, SAS treats the `year` column, for example, as just the number 2022, not as a component of a specific date. The visual confirmation of this initial dataset is provided below, demonstrating the raw, unconverted structure:

| Obs | month | day | year | sales |
|---|---|---|---|---|
| 1 | 4 | 15 | 2022 | 94 |
| 2 | 6 | 17 | 2022 | 88 |
| 3 | 7 | 25 | 2022 | 90 |
| 4 | 8 | 14 | 2022 | 105 |
| 5 | 10 | 13 | 2022 | 119 |
| 6 | 12 | 15 | 2022 | 100 |
| 7 | 1 | 4 | 2023 | 87 |
| 8 | 3 | 15 | 2023 | 90 |
| 9 | 5 | 29 | 2023 | 130 |

## Applying MDY and Formatting the Output Dates

With the source data prepared, the next step involves applying the **MDY** function within a new DATA step to generate the SAS date value. We iterate through the existing `my_data` and create several new variables, demonstrating how the output of **MDY** can then be manipulated using different date formats for presentation purposes.

The core transformation happens in the line: `date_numeric = mdy(month, day, year);`. This single statement converts the separate numeric fields into a single variable, `date_numeric`, which holds the standardized SAS numeric date. Although this variable is now functionally a date, when printed without a format applied, it appears as a large integer (the raw count of days since 1960), which is not user-friendly. This leads directly to the importance of the **PUT** function.

The **PUT** function, combined with specific SAS date formats, is used immediately after the **MDY** conversion to display the resulting date in a recognizable calendar format. We create four distinct date columns to showcase the versatility of date presentation. Crucially, the **MDY** function is only responsible for the internal conversion; the **PUT** function dictates how that internal value is rendered for viewing and reporting.

Observe the powerful combination of functions in the following code block, where the MDY function is used repeatedly within the **PUT** function to generate various formatted outputs:

```
/*create new dataset*/
data new_data;
set my_data;
date_numeric = mdy(month, day, year);
```

```
date_worddate = put(mdy(month, day, year), worddate.);
date_date9 = put(mdy(month, day, year), date9.);
date_mmddyy10 = put(mdy(month, day, year), mmddyy10.);
run;

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

## Exploring Different SAS Date Formats

The previous example clearly demonstrated that the true power of the **MDY** function is realized when its output is paired with appropriate SAS date formats. Since the internal SAS date value is merely a number, the format determines its human-readable appearance. By applying formats like **WORDDATE.**, **DATE9.**, and **MMDDYY10.**, analysts can tailor the output to meet specific reporting or presentation requirements without altering the underlying numerical value.

Let's examine the different outcomes generated in the `new_data` dataset:

**date_numeric:** This column holds the raw numeric output of the **MDY** function (e.g., 22758 for April 15, 2022). It is unformatted and primarily used for calculations.
**date_worddate:** This column uses the **WORDDATE.** format, which spells out the month and includes the full year (e.g., "April 15, 2022"). This is excellent for formal reports and publications where clarity is prioritized over brevity.
**date_date9:** This column applies the **DATE9.** format, displaying the date as DDMMMYY (e.g., "15APR22"). This format is concise and standardized, making it a favorite for intermediate reporting views.
**date_mmddyy10:** This column uses the **MMDDYY10.** format, which provides a familiar numeric layout separated by slashes or dashes, depending on locale (e.g., "04/15/2022"). This format is often preferred for data entry or when integrating with non-SAS systems.

The resulting dataset, after applying these transformations and formats, showcases a comprehensive view of the sales data, now indexed by a correct and flexible date variable. This transformation is key to moving beyond raw data input and into functional data analysis:

| Obs | month | day | year | sales | date_numeric | date_worddate | date_date9 | date_mmddyy10 |
|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 15 | 2022 | 94 | 22750 | April 15, 2022 | 15APR2022 | 04/15/2022 |
| 2 | 6 | 17 | 2022 | 88 | 22813 | June 17, 2022 | 17JUN2022 | 06/17/2022 |
| 3 | 7 | 25 | 2022 | 90 | 22851 | July 25, 2022 | 25JUL2022 | 07/25/2022 |
| 4 | 8 | 14 | 2022 | 105 | 22871 | August 14, 2022 | 14AUG2022 | 08/14/2022 |
| 5 | 10 | 13 | 2022 | 119 | 22931 | October 13, 2022 | 13OCT2022 | 10/13/2022 |
| 6 | 12 | 15 | 2022 | 100 | 22994 | December 15, 2022 | 15DEC2022 | 12/15/2022 |
| 7 | 1 | 4 | 2023 | 87 | 23014 | January 4, 2023 | 04JAN2023 | 01/04/2023 |
| 8 | 3 | 15 | 2023 | 90 | 23084 | March 15, 2023 | 15MAR2023 | 03/15/2023 |
| 9 | 5 | 29 | 2023 | 130 | 23159 | May 29, 2023 | 29MAY2023 | 05/29/2023 |

The example confirms that the **MDY** function successfully synthesized the disparate month, day, and year columns into a single temporal variable, which we then formatted in various ways for improved readability and analytical utility. This flexibility is a hallmark of effective data handling in SAS.

## Conclusion and Further Resources

The **MDY** function is an indispensable component of the SAS programming toolkit, especially when dealing with data sources that fragment date information into separate numeric fields. By efficiently converting these components into a standardized SAS date value, it ensures that all subsequent temporal analyses are accurate, robust, and compliant with SAS internal data rules. Mastery of this function, coupled with an understanding of date formats, allows analysts to handle complex chronological data challenges with ease.

For programmers seeking to further enhance their date handling capabilities, it is highly beneficial to explore the full range of formatting options available. The appropriate format choice can drastically improve report clarity and user comprehension. Furthermore, familiarization with other date functions (like `DATEPART`, `TODAY`, or the input functions like `YYMMDD`) will provide a holistic approach to temporal data management within the DATA step environment.

For comprehensive reference and advanced techniques, the official documentation remains the ultimate source of truth. Analysts should consult these resources to stay updated on best practices and potential function nuances.

**Note #1**: You can find a complete list of potential date formats in SAS.

**Note #2**: You can find the complete documentation for the SAS **MDY** function here.