How to Extract Day, Month, and Year from Dates in SAS (Easy Guide)

Authored by

stats writer

December 1, 2025

The ability to manipulate and analyze time-series data is fundamental in statistical programming, and the SAS system provides robust tools for this purpose. Central to date manipulation are the intrinsic functions **DAY**, **MONTH**, and **YEAR**. These essential functions are specifically designed to dissect a numerical SAS date value, extracting its constituent parts--the day, the month, or the year--and returning them as independent numeric variables. Mastering these functions is crucial for any data analyst working with temporal data in SAS, allowing for granular control over time periods.

While the initial purpose of these functions is straightforward extraction, their utility extends far beyond simple variable creation. They can be employed in sophisticated ways, such as generating custom date formats, calculating temporal differences (e.g., age in years or duration in months), or facilitating complex conditional processing based on specific time components. For instance, extracting the year enables cohort analysis, while extracting the month is vital for seasonal adjustments or reporting. This guide will explore the syntax, practical applications, and best practices associated with using the **DAY**, **MONTH**, and **YEAR** functions, ensuring you can efficiently manage and transform date data within your SAS programming environment.

Understanding SAS Date Values and Syntax

Before implementing the date extraction functions, it is imperative to understand how SAS stores and handles date and time information. Unlike many other programming languages that store dates as character strings or complex objects, SAS utilizes a numerical system. A SAS date value is defined as the number of days between January 1, 1960, and a specific date. If a date is prior to January 1, 1960, the numerical value is negative. This standardized numerical representation is what allows the **DAY**, **MONTH**, and **YEAR** functions to operate efficiently, as they are essentially performing numerical decomposition and calculation rather than string parsing.

It is important to differentiate between a raw SAS date value (a number) and its formatted appearance. When a date variable is created or loaded into a dataset, it is often assigned a format (such as DATE9. or MMDDYY10.) to make it human-readable. However, the underlying data remains a numerical count. The **DAY**, **MONTH**, and **YEAR** functions require this underlying numerical date value as their single argument. If you attempt to pass a character variable representing a date to these functions, SAS will return an error or a missing value unless the character variable is first converted into a numeric SAS date value using the INPUT function with an appropriate informat.

The syntax for these three SAS functions is remarkably simple and uniform, requiring only one argument: the SAS date variable from which the component should be extracted. All three functions return a standard numeric value corresponding to the extracted component. Specifically, the **DAY** function returns a numeric value between 1 and 31, representing the day of the month;

the **MONTH** function returns a numeric value between 1 (January) and 12 (December); and the **YEAR** function returns a four-digit numeric value representing the calendar year. This ease of use ensures rapid incorporation into any data step.

Example 1: Extracting Day, Month, Year into Separate Variables

To demonstrate the utility of these functions, consider a common scenario where a data analyst must separate a single date field into its constituent parts for reporting purposes. This process often involves creating new, dedicated variables for the day, month, and year, enabling easier grouping or cross-tabulation in subsequent procedures. The following code sets up an initial dataset containing various birth dates, correctly formatted using the DATE9. format to ensure they are stored as valid SAS date values.

We begin by defining the source data, which includes the birth date for several individuals. Note the use of the `format` statement to ensure that the input dates are read and displayed correctly, and the subsequent use of PROC PRINT to visualize the initial data structure before transformation. This initial step is critical for verifying that the data is prepared correctly for the extraction functions.

```
/*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;
```

The output of the initial data step confirms the structure of our source data. This visual confirmation is a necessary step in the data preparation workflow, ensuring the input variable `birth_date` is properly recognized as a SAS date format, which is a prerequisite for successful extraction using

the **DAY**, **MONTH**, and **YEAR** functions.

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

Applying the Extraction Functions in a Data Step

To perform the extraction, we now introduce a new data step, `new_data`, where we apply the functions directly to the `birth_date` variable. The simplicity of the function call--`day = DAY(birth_date);`--belies the power of the operation, instantly generating three new numeric variables (`day`, `month`, and `year`) that isolate the respective components. These new variables are ready for any further analytical processing, such as calculating the frequency of birthdays per month or filtering records based on specific years.

This method ensures that the original date variable remains intact while providing the necessary granular numeric variables for subsequent calculations. Since the output is numeric, these variables can be immediately used in arithmetic operations, filtering conditions, or aggregation procedures, which is vital for efficient data manipulation in SAS.

```
/*create new dataset*/
data new_data;
set original_data;
day = DAY(birth_date);
month = MONTH(birth_date);
year = YEAR(birth_date);
run;

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

The resulting dataset, displayed below, clearly shows the original `birth_date` alongside the three newly derived numeric variables. This confirms that the functions have successfully parsed the date data, providing discrete numeric values that are now easily accessible for reporting, modeling,

or data visualization tasks. The three new variables show the day, month, and year of the **birth_date** variable, respectively, stored as standard numeric values.

Obs	birth_date	day	month	year
1	01JAN2021	1	1	2021
2	22FEB2022	22	2	2022
3	14MAR2022	14	3	2022
4	29MAY2022	29	5	2022
5	14OCT2023	14	10	2023
6	01NOV2024	1	11	2024
7	26DEC2025	26	12	2025

Practical Applications of Date Component Extraction

The extraction of date components is rarely an end in itself; typically, it serves as a critical preparatory step for deeper analysis. One of the most common applications is using the extracted month or year variables for data aggregation. For instance, if you have transactional data, you can use the `YEAR` and `MONTH` variables in a `GROUP BY` clause within PROC MEANS or PROC SQL to calculate monthly or yearly totals, averages, or counts. This transforms raw, daily-level data into actionable time-series summaries, which are essential for business intelligence and forecasting.

Furthermore, these variables are indispensable for conditional processing. Suppose an analyst needs to apply a specific discount or rule only to transactions occurring on a specific day of the month or during a particular season. By using the **DAY**, **MONTH**, or **YEAR** outputs, complex conditional logic can be implemented within the SAS data step using `IF-THEN/ELSE` structures. This level of granularity ensures that business rules tied to the calendar structure are accurately reflected in the data transformations, facilitating accurate segmentation and analysis.

Another powerful use case involves creating custom temporal identifiers. Sometimes, analysts require a unique identifier that combines year and month (e.g., '202205') to serve as a period key for merging different data sources. While simple concatenation of the numeric `YEAR` and `MONTH` values can achieve this, careful use of the PUT function is often safer to ensure proper zero-padding for the month (e.g., month '3' becoming '03'). By transforming the numeric outputs of **DAY**, **MONTH**, and **YEAR** into character strings and combining them, analysts can create highly customized temporal keys perfectly suited for their analytical framework or cross-system integration.

Example 2: Extracting Only Month & Year Using Date Formats

While the previous examples focused on extracting components into separate numeric variables, analysts often need to display the date itself in a highly specific aggregated format, such as showing only the month and year. This process relies on utilizing SAS formats rather than the extraction functions themselves, although the core purpose--isolating specific date components--remains the same. SAS formats act as instructions on how to display the underlying numeric date value without altering the value itself.

The following code shows how to create a new variable that holds the original numerical date value but applies the **MMYYN6.** format. This particular format instructs SAS to display the date using a two-digit month and four-digit year, thus condensing the date display to the desired level of detail. It is critical to copy the original date variable (`month_year = birth_date;`) before applying the new format to preserve the original date variable while creating the new formatted representation.

```
/*create new dataset*/
data new_data;
set original_data;
month_year = birth_date;
format month_year mmyyn6.;
run;

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

Upon reviewing the output via PROC PRINT, it becomes evident that the new variable **month_year** only displays the month and year, effectively masking the day component. This technique is often preferred over using extraction functions when the desired outcome is merely a visual aggregation or a shorter display of the date, as it retains the full numerical SAS date value in the variable, which is crucial if future calculations requiring the full date (like calculating the number of days between two events) are necessary.

Obs	birth_date	month_year
1	01JAN2021	012021
2	22FEB2022	022022
3	14MAR2022	032022
4	29MAY2022	052022
5	14OCT2023	102023
6	01NOV2024	112024
7	26DEC2025	122025

Reversing the Format Order: YYMMN6.

If your specific reporting standards require the year to appear before the month (e.g., YYYYMM format), the SAS formatting system allows for a simple substitution of the format name. By replacing **MMYYN6.** with **YYMMN6.**, the output presentation is instantly reordered. This flexibility highlights that SAS date formats provide superior, display-only control compared to manually constructing character strings using the **YEAR** and **MONTH** extraction functions.

Using formats is especially advantageous because the underlying data remains a valid date number, meaning you can change the visual presentation at any point without re-running complex data transformation steps. The format essentially acts as a lens through which the numeric date value is viewed. The following code demonstrates this change, ensuring the output aligns with a standard YYYYMM temporal key display.

```
/*create new dataset with YYYYMM format*/
data new_data;
set original_data;
month_year = birth_date;
format month_year yymmn6.;
run;

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

Advanced Considerations for Date Handling

When working with these intrinsic functions, analysts must be mindful of potential data quality issues, particularly missing values. If the input variable passed to the **DAY**, **MONTH**, or **YEAR** function is a missing SAS date value (represented numerically by a period '.'), the resulting

extracted component will also be a missing numeric value. Analysts must account for this behavior, especially when performing aggregations or filtering, as missing values can skew results or cause unexpected behavior in statistical procedures.

For example, if you are calculating the average number of transactions per month, you must ensure that records with missing dates are appropriately handled--either excluded from the calculation or categorized separately. A robust approach involves using the `MISSING` function in an `IF-THEN` block to check for missing input dates before attempting extraction. This proactive measure ensures data integrity and prevents downstream errors in your SAS processes.

In summary, the **DAY**, **MONTH**, and **YEAR** functions are fundamental components of date manipulation within the SAS programming environment. They offer a simple yet powerful mechanism for decomposing a numerical SAS date value into its core numerical components, facilitating everything from basic reporting to complex time-series analysis and conditional data processing. Mastering these three functions is essential for efficient and effective data management in SAS.

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