# How to Easily Calculate Date Differences in SAS Using INTNX()

Authored by

**stats writer**

December 1, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Calculate Date Differences in SAS Using INTNX()*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=103089

ARABPSYCHOLOGY.COM

The calculation of time differences is a fundamental requirement in data analysis, particularly when working with transactional, longitudinal, or financial data. While the SAS programming language offers various tools for date manipulation, the **INTCK()** (Interval Count) function stands out as the most efficient and reliable method for determining the difference between two SAS date values. This powerful function accurately computes the number of specified time units--whether days, weeks, months, or years--that elapse between a start date and an end date.

It is important to note a common point of confusion: while the **INTNX()** function is used to advance a date by a specific interval (e.g., finding the date 3 months after a start date), the **INTCK()** function is specifically designed for **counting** the number of intervals that occur between two specified points in time. Understanding the correct application of **INTCK()** is crucial for ensuring accuracy in time-series analysis and reporting within the SAS environment.

## The Essential Tool: Understanding the INTCK Function

The **INTCK** (Interval Count) function is the cornerstone for date arithmetic in SAS when you need to count how many periods fall between two dates. Unlike simple subtraction, which only yields the difference in days, **INTCK()** handles the complexities of time, such as varying month lengths, leap years, and defining boundaries like week start days. By abstracting these complexities, it allows the user to focus purely on the required temporal unit.

You can use the **INTCK** function in SAS to quickly calculate the difference between two dates. This function is extremely flexible, supporting a vast library of intervals, from standard ones like DAY and MONTH to more specialized ones like QTR (quarter) or DTDAY (day of the week). Proper use of this function ensures your calculations align perfectly with the defined temporal boundaries that SAS uses internally.

## Deep Dive into INTCK Syntax and Arguments

The basic structure of the INTCK Function is straightforward, yet highly adaptable. Mastering its four primary arguments unlocks precise control over how intervals are counted, which is critical when dealing with diverse analytical requirements. The function's syntax provides the necessary framework for defining the exact temporal unit and counting methodology.

This function uses the following basic syntax:

**INTCK(interval, start date, end date, method)**

Here is a detailed breakdown of each component:

**interval**: Specifies the unit of time to be counted (e.g., 'DAY', 'WEEK', 'MONTH', 'YEAR'). The choice of interval determines the boundaries used during the calculation.

ARABPSYCHOLOGY.COM

**start date**: The beginning SAS date value for the calculation. This must be a valid numeric date representation.
**end date**: The ending SAS date value for the calculation.
**method**: This optional argument controls the counting methodology. It determines whether to count complete intervals ('D' = No (Default), 'C' = Yes).

## Practical Application: Setting Up the Sample Dataset

To demonstrate the utility of the INTCK Function, we will begin by creating a simple dataset containing pairs of start and end dates. This setup simulates a common scenario in data analysis where you need to derive temporal metrics from existing records. It is crucial to use the appropriate date format (such as `DATE9.`) when inputting dates to ensure SAS correctly interprets the date strings as numeric date values.

Suppose we have the following dataset in SAS that contains two date variables, `start_date` and `end_date`:

```
/*create dataset*/
data original_data;
format start_date end_date date9.;
input start_date :date9. end_date :date9.;
datalines;
01JAN2022 09JAN2022
01FEB2022 22FEB2022
14MAR2022 04APR2022
01MAY2022 14AUG2022
;
run;

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

Executing this code generates the initial table, confirming that our date variables are correctly stored and formatted. This foundational step ensures that subsequent calculations using the INTCK Function will operate on valid numeric date references.

ARABPSYCHOLOGY.COM

| Obs | start_date | end_date |
|---|---|---|
| 1 | 01JAN2022 | 09JAN2022 |
| 2 | 01FEB2022 | 22FEB2022 |
| 3 | 14MAR2022 | 04APR2022 |
| 4 | 01MAY2022 | 14AUG2022 |

## Calculating Simple Intervals (Days, Weeks, Months)

The primary use case for **INTCK()** is calculating the difference across multiple interval types simultaneously. By calling the function multiple times within a single **DATA Step**, we can create new variables representing the elapsed time in days, weeks, and months, offering a comprehensive view of the time duration for each record.

We will now use the following code block to apply the **INTCK** function to our `original_data`. Notice that we are utilizing three different intervals ('day', 'weeks', and 'months') without specifying the optional method argument, thus utilizing the default method ('D' for discrete counting).

```
/*create new dataset*/
data new_data;
set original_data;
days_diff = intck('day', start_date, end_date);
weeks_diff = intck('weeks', start_date, end_date);
months_diff = intck('months', start_date, end_date);
run;

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

The resulting table clearly demonstrates the power of **INTCK()**. The three new variables--`days_diff`, `weeks_diff`, and `months_diff`--provide the calculated difference between `start_date` and `end_date` based on the specified unit. For instance, the difference between '01JAN2022' and '09JAN2022' is 8 days, but 2 weeks, because the default method counts the number of week boundaries crossed, treating partial weeks as full transitions.

ARABPSYCHOLOGY.COM

| Obs | start_date | end_date | days_diff | weeks_diff | months_diff |
|---|---|---|---|---|---|
| 1 | 01JAN2022 | 09JAN2022 | 8 | 2 | 0 |
| 2 | 01FEB2022 | 22FEB2022 | 21 | 3 | 0 |
| 3 | 14MAR2022 | 04APR2022 | 21 | 3 | 1 |
| 4 | 01MAY2022 | 14AUG2022 | 105 | 15 | 3 |

## The Importance of the 'Method' Argument: Counting Complete Intervals

One of the most frequent errors in date difference calculation involves misinterpreting how partial intervals are counted. By default, **INTCK()** uses the 'D' (Discrete) method, which counts every time a boundary (like the first day of a month or the start of a week) is crossed. However, in scenarios requiring only full, completed periods (e.g., calculating full months of service), the 'C' (Continuous) method is essential.

The 'C' argument instructs the INTCK Function to only count the number of complete, non-overlapping intervals that fit entirely within the specified start and end dates. This leads to a more conservative count, often resulting in lower values for week and month differences compared to the default 'D' method, especially when the duration is short or spans only part of an interval.

Note that we can use the '**C**' argument in the **INTCK** function to calculate the difference only in complete days, weeks, and months:

```
/*create new dataset*/
data new_data_complete;
set original_data;
days_diff_c = intck('day', start_date, end_date, 'c');
weeks_diff_c = intck('weeks', start_date, end_date, 'c');
months_diff_c = intck('months', start_date, end_date, 'c');
run;

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

ARABPSYCHOLOGY.COM

| Obs | start_date | end_date | days_diff | weeks_diff | months_diff |
|---|---|---|---|---|---|
| 1 | 01JAN2022 | 09JAN2022 | 8 | 1 | 0 |
| 2 | 01FEB2022 | 22FEB2022 | 21 | 3 | 0 |
| 3 | 14MAR2022 | 04APR2022 | 21 | 3 | 0 |
| 4 | 01MAY2022 | 14AUG2022 | 105 | 15 | 3 |

## Comparing Discrete ('D') vs. Continuous ('C') Counting

The differences observed between the two resulting tables highlight the crucial distinction between the 'D' and 'C' methods. When using the default ('D') method, the function counts the number of interval boundaries crossed, providing a measure of how many periods were touched during the duration.

Conversely, in the second table, where the 'C' method was used, the calculation is strictly limited to completed intervals. For the first row (Jan 1st to Jan 9th), the difference in weeks is calculated as **1** (using 'C'), whereas it was **2** in the previous table (using 'D'). This is because only one whole, seven-day week can fit entirely between Jan 1st and Jan 9th, whereas the discrete method counts the week boundaries crossed at the beginning and the end.

Choosing the correct method depends entirely on the analytical question. If you are calculating the exact number of calendar months elapsed (e.g., for age or tenure), 'D' is usually sufficient. If you need to verify that a minimum number of full time periods were completed (e.g., contract fulfillment or full pay periods), then 'C' provides the necessary precision.

## Advanced Interval Types and Customization

While we demonstrated 'DAY', 'WEEK', and 'MONTH', the **INTCK()** function supports dozens of specialized intervals, allowing for highly specific temporal calculations. For example, using 'MONTH' counts the number of times the first day of a month is crossed, while 'MONTHS' (with an 'S' suffix) handles specific alignment requirements. Additionally, many intervals can be customized using a multiplier or a start point.

For instance, if you needed to count the number of 10-day periods, you would use `'DAY10'` as the interval. If your fiscal week starts on a Tuesday instead of the default Sunday, you could specify `'WEEK.3'` (where 1=Sunday, 2=Monday, 3=Tuesday). This level of customization makes INTCK() an indispensable function for complex analytical environments that require non-standard calendars or reporting cycles.

ARABPSYCHOLOGY.COM

## Conclusion: Leveraging INTCK for Robust Time Series Analysis

The INTCK Function is far more than a simple date subtraction tool; it is a critical component of robust time series analysis in SAS. By providing a standardized, reliable method for counting elapsed time units--and offering control over whether to count boundaries crossed ('D') or complete intervals ('C')--it ensures that temporal metrics derived from your data are accurate and consistent, regardless of the complexity of the underlying date ranges.

Mastering the four arguments of the **INTCK()** function and understanding the impact of the 'method' parameter allows SAS programmers and analysts to handle virtually any requirement related to calculating duration. This precision is essential for generating accurate reports, calculating tenure, managing cohorts, and supporting financial analysis where exact time metrics are paramount.

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

ARABPSYCHOLOGY.COM