Use IF-THEN-ELSE in SAS (With Examples)

Authored by

Mohammed looti

March 28, 2026

In **SAS**, a premier statistical software suite essential for advanced data analysis and management, the **IF-THEN-ELSE** statement stands as a foundational construct for executing **conditional logic**. This powerful statement provides the mechanism necessary to dictate the flow of data processing, enabling the system to execute different actions or assign specific values to a **variable** based on whether a defined condition is satisfied (true) or not (false). It is an indispensable part of creating dynamic and responsive routines, particularly within the crucial **SAS Data Step** environment.

The core utility of the **IF-THEN-ELSE** statement is its ability to precisely control the sequence of operations within your program. By implementing this statement, developers and analysts can define specific, measurable outcomes for a variety of scenarios, thereby making the code robust, adaptable, and highly accurate. Whether the task involves categorizing raw data into predefined groups, flagging specific observations that meet certain criteria, or performing complex calculations only under specific prerequisites, mastering conditional processing is paramount for effective SAS programming.

Understanding the IF-THEN-ELSE Statement

The principle governing the operation of the **IF-THEN-ELSE** statement is straightforward yet highly effective: if a condition is evaluated and found to be true, the code block immediately following the `THEN` keyword is executed; conversely, if the condition is false, the code following the optional `ELSE` keyword is executed. This system of binary decision-making provides the structural backbone for many essential data manipulation procedures in SAS, offering analysts the ability to exercise precise control over data transformations at the observational level.

One of the greatest advantages of this structure is its clear and intuitive **syntax**, which facilitates easy comprehension and implementation, even for those who are relatively new to the realm of computer programming. The inherent clarity of the IF-THEN-ELSE construction promotes the creation of readable and maintainable SAS code. In complex data analysis projects, where multiple transformations and logical checks are required, well-structured conditional statements are crucial for debugging and future auditing.

Basic Syntax and Components

The fundamental **syntax** required for implementing an **IF-THEN-ELSE** statement in SAS comprises three essential components. First, the `IF` keyword introduces the statement, followed by a **Boolean expression**--the condition itself. Second, the `THEN` keyword signals the action to be performed if the condition is true. Third, the optional `ELSE` keyword defines the alternative action if the condition is false. The condition typically involves comparing a **variable** against a specified value or another **variable**, utilizing standard comparison operators such as `>` (greater than), `<` (less than), `=` (equal to), and others.

Understanding this foundational structure is key to beginning conditional programming. While the statement can become complex with nested conditions, the basic form remains highly accessible.

Here is a concise representation of its fundamental structure:

```
if var1 > 30 then var2 = 'good';
else var2 = 'bad';
```

In this typical example, the code evaluates the value of `var1` for the current observation. If `var1` exceeds 30, the secondary variable `var2` is immediately assigned the string value 'good'. Conversely, if the condition is not satisfied (i.e., `var1` is 30 or less), the action defined by the `ELSE` clause executes, assigning the string 'bad' to `var2`. This simple but effective structure allows for clear, efficient, and unambiguous conditional processing within the SAS Data Step.

Chaining Conditions with ELSE IF

When dealing with situations that necessitate more intricate **conditional logic** and multiple potential outcomes, SAS offers the capability to chain several conditions using the `ELSE IF` construct. This chaining mechanism permits the sequential evaluation of a series of conditions. The process stops as soon as the first true condition is encountered, and the associated action is executed. If the initial `IF` condition and all subsequent `ELSE IF` conditions evaluate to false, the optional final `ELSE` statement will execute, handling all remaining cases.

This feature is especially valuable when the objective is to categorize data into multiple distinct tiers or apply different analytical rules across a range of values, such as grading scores or classifying risk levels. It is critically important to remember that the order in which these statements are arranged dictates the logic of the entire routine, as **SAS** processes them strictly sequentially and halts evaluation once a condition proves true.

Consider the following expanded **syntax** designed to handle multiple conditions:

```
if var1 > 35 then var2 = 'great';
else if var1 > 30 then var2 = 'good';
else var2 = 'bad';
```

In this hierarchical sequence, the program first checks if `var1` is greater than 35, assigning 'great' if true. If that initial check fails, SAS proceeds to the `ELSE IF`, checking if `var1` is greater than 30. Crucially, because the first condition failed, this second check only applies to values between 31 and 35, assigning 'good' if true. If both preceding conditions are false, the final catch-all `ELSE` clause executes, assigning 'bad'. This structured evaluation ensures that every value is

categorized precisely based on the established thresholds.

Practical Application: Setting Up Your Data

To provide a clear demonstration of the practical application of **IF-THEN-ELSE** and IF-THEN-ELSE IF statements, we will begin by constructing a necessary sample **dataset** in SAS. This dataset, which we will name `original_data`, will contain simple records concerning various sports teams and their corresponding point totals. This data will serve as the realistic foundation upon which all our subsequent conditional logic examples are built.

The process of creating and populating this dataset is conducted entirely within a **Data Step**. We initiate the step using the `DATA` statement to name the new dataset, and the `INPUT` statement to clearly define its constituent **variables**, which are `team` (a character variable denoted by `$`) and `points` (a numeric variable). The `DATALINES` statement then allows us to embed the raw observational data directly within the program code, ensuring the example is entirely self-contained, reproducible, and easy to follow.

Once the `original_data` **dataset** has been successfully generated, we immediately utilize the **PROC PRINT** procedure to display its contents in the output window. This vital step confirms that the data has been loaded without error and provides a necessary visual baseline of the initial data state before any conditional transformations are applied.

```
/*create dataset*/
data original_data;
input team $ points;
datalines;
Cavs 12
Cavs 14
Warriors 15
Hawks 18
Mavs 31
Mavs 32
Mavs 35
Celtics 36
Celtics 40
;
run;

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

Obs	team	points
1	Cavs	12
2	Cavs	14
3	Warriors	15
4	Hawks	18
5	Mavs	31
6	Mavs	32
7	Mavs	35
8	Celtics	36
9	Celtics	40

Example 1: IF-THEN-ELSE in SAS

Our first practical scenario involves applying a foundational **IF-THEN-ELSE** statement to the `original_data` **dataset** to perform a simple, binary categorization. The primary objective is to generate a new **variable**, named `rating`, which assigns a performance category based on the value in the `points` column. Specifically, if a team's `points` total is greater than 30, they are categorized as "good"; otherwise, they are assigned a "bad" rating.

This transformation is executed within a new Data Step, which reads the existing data using the `SET original_data` statement. This ensures that all existing observations and variables are carried forward into the new dataset. The **IF-THEN-ELSE** statement then systematically evaluates the `points` variable for every observation, applying the appropriate conditional assignment to the new `rating` variable. This structure perfectly illustrates the power of binary **conditional logic**.

The following code implements this straightforward categorization logic:

```
/*create new dataset with new variable called rating*/
data new_data;
set original_data;
if points > 30 then rating = 'good';
else rating = 'bad';
run;

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

Obs	team	points	rating
1	Cavs	12	bad
2	Cavs	14	bad
3	Warriors	15	bad
4	Hawks	18	bad
5	Mavs	31	good
6	Mavs	32	good
7	Mavs	35	good
8	Celtics	36	good
9	Celtics	40	good

As clearly demonstrated in the resulting output table, the newly generated `rating` column accurately reflects the applied **conditional logic**. Observations corresponding to teams like Mavs and Celtics, which have `points` values exceeding the threshold of 30, are correctly assigned a "good" rating. In contrast, teams such as Cavs, Warriors, and Hawks, whose point totals are 30 or less, receive a "bad" rating. This successful outcome provides a robust initial illustration of the fundamental **IF-THEN-ELSE** statement used for straightforward categorization.

Example 2: IF-THEN-ELSE IF in SAS

Expanding upon the binary classification of the previous example, this section illustrates the implementation of multi-tiered categorization using the advanced **IF-THEN-ELSE IF** statement in **SAS**. Our specific objective here is to refine the `rating` variable to include three distinct categories: "great," "good," and "bad." These categories are determined by evaluating different sequential thresholds established for the `points` variable.

The multi-level conditions are precisely defined as follows:

The highest rating, "great," is assigned if the `points` value is strictly greater than 35.
If the first condition is not met, the program checks the next threshold: "good" is assigned if `points` is greater than 30.
In all other remaining cases--that is, if the team has 30 points or fewer--the default "bad" rating is assigned by the final `ELSE` clause.

This hierarchical **conditional logic** is managed efficiently through the chaining of `ELSE IF` clauses. The strict sequential evaluation ensures that each observation is tested in the defined order, and only the action corresponding to the first true condition is executed, ensuring precise

and non-overlapping categorization.

The following **Data Step** code implements this refined, three-tiered categorization logic:

```
/*create new dataset with new variable called rating*/
data new_data;
set original_data;
if points > 35 then rating = 'great';
else if points > 30 then rating = 'good';
else rating = 'bad';
run;

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

Obs	team	points	rating
1	Cavs	12	bad
2	Cavs	14	bad
3	Warriors	15	bad
4	Hawks	18	bad
5	Mavs	31	good
6	Mavs	32	good
7	Mavs	35	good
8	Celtics	36	great
9	Celtics	40	great

By examining the output, we can verify that the `rating` variable successfully reflects the three-tiered **conditional logic**. For example, the Celtics observation with 40 points receives the "great" rating. Mavs observations, falling between 31 and 35 points, are correctly rated "good." Finally, all teams with 30 or fewer points remain categorized as "bad." This outcome powerfully illustrates the flexibility and precision gained by using chained **IF-THEN-ELSE IF** statements for complex data classification tasks.

Best Practices and Considerations

To maximize the effectiveness and maintainability of your code when utilizing **IF-THEN-ELSE** and

IF-THEN-ELSE IF statements in **SAS**, adhering to certain best practices is essential. The flexibility of these statements allows for the incorporation of numerous `ELSE IF` clauses to address a wide spectrum of conditions, making them adaptable to virtually any analytical requirement.

A critical consideration, particularly when chaining conditions, is the logical ordering of your statements. Conditions should typically be arranged from the most restrictive or specific criteria to the least restrictive or most general. Because SAS evaluates conditions sequentially and immediately stops upon finding the first true match, placing a broad condition too early in the sequence can preempt more specific conditions that should have taken precedence, leading to incorrect data assignments. Careful sequencing is the cornerstone of accurate multi-condition logic.

Furthermore, strive to use clear and concise **Boolean expressions**. Avoid constructing overly intricate or nested conditions that compromise readability and complicate the debugging process. If a condition must be complex, utilize parentheses to explicitly define the order of operations. Finally, always include a concluding `ELSE` statement, when appropriate, to serve as a safety net. This ensures that every observation is assigned a value, preventing the creation of undesirable missing values or unexpected results for data that failed to meet any of the preceding `IF` or `ELSE IF` criteria.

Conclusion and Further Learning

The **IF-THEN-ELSE** and IF-THEN-ELSE IF statements are fundamental components of **SAS** programming, providing the necessary tools to implement powerful **conditional logic** for sophisticated data manipulation and analysis. By thoroughly mastering these essential constructs, users can develop dynamic and intelligent SAS programs capable of responding effectively and accurately to diverse data characteristics and analytical demands.

Proficiency in using these statements to accurately categorize data, generate new variables, and precisely control the flow of data is a crucial skill set for any dedicated **SAS** professional. The practical examples demonstrated throughout this article illustrate both binary and multi-condition scenarios, providing a clear and actionable framework for implementing these essential techniques in your own statistical projects.

To further advance your SAS proficiency and build upon the principles of conditional logic discussed here, it is highly recommended to explore additional resources on related topics. The following tutorials offer guidance on performing other common data preparation and analysis tasks in SAS: