How to use a DO UNTIL Statement in SAS

Authored by

stats writer

November 19, 2025

The **DO UNTIL statement** in **SAS** is a foundational looping construct designed to execute a block of instructions repeatedly until a specified conditional test evaluates to true. Unlike the basic iterative DO loop, the DO UNTIL loop is condition-controlled, meaning the number of iterations is not predetermined. Crucially, the condition check occurs at the bottom of the loop, ensuring that the instructions within the loop are executed at least once, regardless of the initial state of the condition. This mechanism makes the DO UNTIL structure indispensable for scenarios where processing must occur before the termination criterion can be meaningfully assessed, providing great flexibility in data manipulation tasks.

This powerful looping structure consists primarily of the **DO statement**, which encapsulates the body of code to be repeatedly executed, and the UNTIL clause, which specifies the **logical expression** governing termination. The loop continues its cycle--executing the code block and then checking the condition--until the logical expression yields a true result. The inherent design of the DO UNTIL loop addresses programming situations where the precise iteration count is unknown beforehand, allowing analysts to iterate through data or calculations dynamically based on evolving variable values, thereby optimizing resource usage and increasing procedural efficiency within the **SAS** environment.

Utilizing the **DO UNTIL** statement in **SAS** allows programmers to execute a sequence of commands repeatedly until a specific end condition is satisfied. The following discussion and examples will illustrate how to effectively apply this crucial looping mechanism in practical data processing contexts.

We will explore two distinct methodologies for implementing the DO UNTIL statement, showcasing both simple conditional termination and advanced integration with iterative control mechanisms.

Understanding the Mechanics of DO UNTIL

The primary distinguishing feature of the **DO UNTIL statement** is its post-test execution model. Unlike its counterpart, the DO WHILE loop, which evaluates the condition before the block executes, DO UNTIL performs the instructions first, and only then checks the termination condition. This fundamental difference guarantees a minimum of one execution cycle, which is essential for tasks where the initial calculation or data initialization needs to be finalized before the stopping criterion is checked against the results. This bottom-check process ensures reliability when designing complex iterative algorithms.

The structure requires the programmer to define the looping condition immediately following the UNTIL keyword. This condition must be a valid **logical expression** that results in either true or false. Once the expression evaluates to true, the loop terminates immediately, and the SAS program continues to the statement following the **END** keyword. Careful construction of this logical

expression is paramount; if the condition is never met, an infinite loop may occur, leading to system resource depletion and program failure. Therefore, variables modified within the loop must influence the conditional test to ensure eventual convergence.

Furthermore, the **DO UNTIL** statement must be properly nested within a **DATA step** or a **PROC step** where iterative processing is necessary. It is typically used in conjunction with the **OUTPUT** statement within the DATA step when the goal is to create multiple observations from a single input observation or, more commonly, to generate a series of observations based on iterative calculations, as demonstrated in the examples that follow. The robustness of this structure allows for sophisticated data generation and simulation routines often required in statistical analysis.

When to Choose DO UNTIL Over Other Loops

Programmers often face a choice between the DO UNTIL, DO WHILE, and iterative DO I=1 TO N loops. The DO UNTIL loop is the optimal choice primarily when the process demands execution at least once. Consider scenarios involving user input validation, where the validation logic must run once before assessing whether the input is correct, or complex statistical simulations where initializing parameters is part of the first step before checking convergence. In these cases, the guarantee of initial execution simplifies the code structure significantly.

Another crucial differentiator is that the DO UNTIL loop operates purely based on a conditional test, making it ideal for processes where the number of required iterations is dynamic and dependent on internal calculations rather than a fixed count. For example, calculating mortgage payments until the principal reaches zero, or iterating a numerical method until the difference between successive estimates falls below a defined tolerance level. This reliance on a termination condition, rather than a fixed counter, provides flexibility unmatched by the standard iterative **DO** loop.

In contrast, if you need to iterate exactly N times, the simple DO I = 1 TO N structure is preferable. If the logic requires checking the condition before any execution occurs--such as ensuring data exists before attempting to process it--the DO WHILE loop is the appropriate choice. Understanding these nuances is critical for writing efficient and reliable **SAS** code, ensuring that the selected looping construct perfectly matches the algorithmic requirement. The subsequent examples highlight practical applications where the DO UNTIL mechanism proves most effective.

Example 1: Basic Conditional Looping Structure in SAS

This initial example demonstrates the simplest form of the **DO UNTIL statement**, where the loop generates new observations for a **dataset** based solely on a conditional variable threshold. We aim to create a dataset named `my_data` containing two variables, `var1` and `var2`. The loop will continue generating new values and outputting new observations until the value of `var1` finally

exceeds 100. This showcases the fundamental bottom-check mechanism, driving calculation until the terminal state is reached.

Within the **DATA step**, both variables are initialized to 1. Inside the **DO UNTIL** block, `var1` and `var2` are iteratively updated. Notice that the **DO UNTIL statement** is placed at the beginning of the iterative block, but the condition `(var1 > 100)` is checked only after the variables are calculated and the **OUTPUT** statement executes. This ensures the iteration where `var1` finally surpasses 100 is indeed captured as the last observation, demonstrating the inclusion of the boundary condition iteration.

The code structure below provides a clean template for dynamic data generation. The **OUTPUT** statement is essential here, as it writes the current values of the variables to the **dataset** on each iteration. If the **OUTPUT** statement were omitted, only the final calculated values (after loop termination) would be written as a single observation, potentially defeating the purpose of the iteration. This precise control over observation creation is a hallmark of efficient **SAS** programming.

Example 1: Generating Data Until a Threshold is Exceeded

The following code demonstrates how to utilize a **DO UNTIL** statement in **SAS** to create a **dataset** that contains two variables called **var1** and **var2**, generating new observations until the value of **var1** is greater than 100:

```
/*create dataset using DO UNTIL statement for dynamic iteration*/
data my_data;

var1 = 1;
var2 = 1;

do until(var1 > 100);
var1 = var1 + var2;
var2 = var1 * var2;
var1 + 1;

output;

end;

run;

/*view dataset using PROC PRINT*/
proc print data=my_data;
```

Obs	var1	var2
1	3	2
2	6	10
3	17	160
4	178	28320

The **DO UNTIL statement** continued to generate new values for **var1** and **var2 until** the value of **var1** was greater than 100.

Once the value of **var1** exceeded 100, the **DO UNTIL** statement stopped and new values stopped being added to the **dataset**.

Analyzing the Output of Conditional Looping

Upon execution, the **DO UNTIL statement** systematically generates new values for **var1** and **var2**. We trace the execution steps to confirm the post-test nature of the loop. The variables are calculated, and the observation is outputted, before the termination check (`var1 > 100`) occurs. Since the calculated value of **var1** crosses the 100 threshold in the final executed iteration (becoming 108), that boundary-crossing value is successfully written to the dataset.

The crucial observation is that the loop ensured that the iteration which finally caused **var1** to exceed 100 was executed entirely, and the resultant values were written to the dataset before the loop checked the condition and stopped. This confirms the post-test nature of the construct: calculations occur, output is generated, the condition is tested, and only then does termination happen. This characteristic is vital for understanding when a boundary-crossing value will be included in the final data output, which is a key difference from the DO WHILE loop.

Once the **logical expression** `(var1 > 100)` evaluates to true during the termination check, the flow of execution breaks out of the DO block. No further new observations are added to the dataset, resulting in the final output structure shown in the image. This precise control over the termination point allows for highly specific data creation based on dynamic computational goals, rather than fixed iteration counts.

Example 2: Combining DO UNTIL with the Iterative DO (DO TO)

The **DO UNTIL statement** can be seamlessly integrated with other iterative structures, such as the DO index variable loop (often referred to as the DO TO loop). This combination allows for a dual termination mechanism: the loop can stop either when the index variable reaches its limit, or when

the UNTIL condition is met, whichever occurs first. This is incredibly useful for imposing external constraints on a loop that is primarily driven by fixed iteration counts.

In the following example, we instruct the **DO statement** to iterate the variable **var2** from 1 to 5. However, the added UNTIL clause imposes an override: `UNTIL(var1 > 10)`. The loop will attempt to complete its 1-to-5 cycle, but if the condition `var1 > 10` becomes true prematurely, the loop must terminate immediately. This setup effectively establishes a conditional break within a fixed iterative sequence, optimizing the process by stopping when the required outcome is achieved.

This powerful technique ensures computational efficiency. If the condition for termination is met early--meaning the desired result or threshold has been reached--**SAS** avoids unnecessary processing of the remaining iterations. The loop becomes hybrid: iteration-controlled by **var2**, but condition-controlled by **var1**. This provides a safety measure and optimization layer, particularly beneficial when processing time is a concern or when iterating through a large data volume where early exit is a possibility.

Example 2: Applying Dual Termination Logic

The following code illustrates how to combine a **DO UNTIL** statement with a **TO** statement in **SAS** to create a **dataset** where iteration is controlled by an index (**var2**) but terminated prematurely if the calculated variable (**var1**) exceeds 10:

```
/*create dataset using DO UNTIL statement with TO statement*/
data my_data;

var1 = 0;

do var2 = 1 to 5 until(var1 > 10);
var1 = var2**2;

output;

end;

run;

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

Obs	var1	var2
1	1	1
2	4	2
3	9	3
4	16	4

The **TO** statement defined the nominal range for **var2** (1 to 5), but the **UNTIL** clause forced termination once the value of **var1** was greater than 10.

Once the value of **var1** exceeded 10, the **DO UNTIL** condition was met, and the loop stopped immediately, preventing further iterations from being added to the **dataset**.

Dissecting the Hybrid Termination Logic

In this second example, the iteration begins with **var2** starting at 1. The calculation within the loop sets **var1** equal to **var2** squared. We trace the execution steps to observe how the hybrid termination rule functions, noting that the condition check `UNTIL(var1 > 10)` occurs only after the calculation and output of the current iteration are complete. After the fourth iteration (where **var2**=4), **var1** becomes 16, and this observation is outputted. Only then is the condition checked: 16 > 10, which is true.

Because the **logical expression** evaluated to true following the fourth iteration, the loop terminates early. Although the **TO** statement specified iteration up to **var2**=5, the UNTIL condition acted as an overriding safety mechanism. This prevents the execution of the fifth iteration (where **var2** would equal 5), thus ensuring that the data generated adheres strictly to the imposed constraints. The resultant dataset contains only four observations, confirming the effective implementation of the conditional break.

The key takeaway from using the combined **DO TO UNTIL** statement is that it provides a controlled environment where a fixed number of iterations is the default, but a sudden or dynamic computational result can force an immediate stop. If the termination condition is never met (e.g., if the range were only 1 to 3), the loop would complete all iterations specified by the **TO** clause, stopping naturally at the end of the range. This powerful combination highlights the flexibility of the **DO statement** family in SAS programming.

Important Considerations: Infinite Loops and Initialization

When working with condition-controlled loops like **DO UNTIL**, careful attention must be paid to

variable initialization and the modification of variables within the loop. Since the loop relies on internal calculation to eventually make the logical expression true, forgetting to update the control variable or setting an impossible condition will inevitably lead to an **infinite loop**. An infinite loop consumes system resources indefinitely and usually requires manual intervention to stop the SAS session. Programmers must ensure that at least one variable involved in the UNTIL condition is modified in a way that moves it toward the termination threshold on every pass.

Another crucial point is the distinction between **DO UNTIL** and **DO WHILE**, specifically concerning the boundary condition. Because **DO UNTIL** checks the condition at the end, the iteration that causes the condition to become true is executed and outputted. If the requirement is to stop *before* the condition is met (i.e., excluding the boundary-crossing result), one might need to adjust the condition (e.g., use `UNTIL(var1 >= 100)` instead of `UNTIL(var1 > 100)`, depending on the exact calculation) or utilize the **DO WHILE** statement, which is a pre-test loop. Always test the loop boundary conditions rigorously to confirm the desired data points are included or excluded.

Finally, remember that variables initialized within the **DATA step** outside of the loop retain their values across loop iterations unless explicitly reset. When using the **OUTPUT** statement within the loop, **SAS** creates a new observation, but the variables maintain their current values for the start of the next iteration until the **END** statement is reached, which triggers the automatic return to the beginning of the DATA step (or loop start, if the loop is generating observations). Understanding this iterative behavior is vital for ensuring variable accumulation and calculation sequences behave as intended during the execution of the **DO statement** structure.

Summary of Looping Tasks in SAS

The **DO UNTIL** construct provides robust conditional control in SAS programming, serving as a powerful tool for generating data points and performing iterative calculations where the stopping point is defined by a result rather than a fixed count. Its post-test nature guarantees execution, making it uniquely suited for initialization or validation tasks where minimum one run is required.

By mastering the syntax and understanding the interaction between the DO block, the iterative calculations, and the termination clause, programmers can leverage this statement to create efficient, dynamic, and complex data generation processes within the SAS environment. Proper selection between the various DO loop types--DO UNTIL, DO WHILE, and the iterative DO TO--is a mark of an expert SAS programmer.

For further exploration of looping structures and common programming techniques in SAS, consider the following related tutorials: