Learning SAS: Understanding and Implementing DO Loops

Authored by

Mohammed looti

October 30, 2025

In data programming, repetitive tasks are common. The SAS language provides powerful mechanisms for handling such repetition, the most fundamental of which is the **DO loop**. A DO loop allows a sequence of statements to be executed repeatedly, typically within a DATA step, until a specified condition is met or a set number of iterations is completed.

Understanding the three primary types of **DO loops** available in SAS is essential for efficient data manipulation and simulation. These variations offer flexibility in controlling when and how the looping process terminates.

The Three Fundamental DO Loops in SAS

SAS recognizes three primary structures for iterative execution. While they share the common goal of repetition, their control mechanisms--specifically when the condition is evaluated--differentiate them significantly. We will explore each type below, focusing on its structure, execution logic, and precise termination criteria.

The three core types of **DO loops** are:

The Simple Iterative **DO Loop** (DO index = start TO end).
The Conditional **DO WHILE Loop** (Pre-check condition).
The Conditional **DO UNTIL Loop** (Post-check condition).

Type 1: The Basic Iterative DO Loop

The simple **DO loop** is used when the exact number of iterations is known beforehand. It executes a block of code a fixed number of times, incrementing an index variable with each pass. This is the most straightforward method for generating datasets of a specific size or performing calculations over a predefined range.

```
data data1;
x = 0;
do i = 1 to 10;
x = i*4;
output;
end;
run;
```

In this example, the **DO loop** executes exactly 10 iterations, starting with the index variable `i` set to 1 and ending when `i` reaches 10. Inside the loop, the variable `x` is calculated by multiplying the current index value by 4, and the `OUTPUT` statement writes the resultant observation to the SAS dataset named `data1`.

The loop's termination condition is purely count-based; it completes only after the specified number of cycles (10) has been fully executed. No other logical checks influence the flow of this loop.

Type 2: The Conditional DO WHILE Loop

The **DO WHILE loop** combines the fixed iteration structure with a dynamic conditional check. The key characteristic of the **DO WHILE loop** is that the condition is checked *before* the loop body is executed. If the condition is initially false, the loop body may never run. This structure is ideal when you need the loop to stop immediately once a logical criterion is violated, even if the count range has not been exhausted.

```
data data2;
x = 0;
do i = 1 to 10 while(x < 20);
x = i*4;
output;
end;
run;
```

In this structure, the **DO loop** attempts to run for 10 iterations. However, it includes the critical `while` clause. At the start of every cycle, SAS checks if the value of `x` is less than 20. If this condition evaluates to false, the loop terminates immediately, regardless of the index variable `i`'s current value.

The loop will cease execution when either `x` equals or exceeds 20, or when the 10th iteration is completed, whichever stopping condition is met first. This design ensures that the output data always adheres to the specified maximum value for `x`.

Type 3: The Conditional DO UNTIL Loop

The **DO UNTIL loop** also utilizes a logical condition for termination, but with a crucial difference from the DO WHILE structure: the condition is checked *at the end* of the loop. This guarantees that the statements within the loop body are executed at least once before the condition is evaluated. The loop continues executing *until* the condition becomes true.

```
data data3;
x = 0;
do i = 1 to 10 until(x > 30);
x = i*4;
output;
```

```
end;
run;
```

For this structure, the code will first calculate `x = i*4` and output the result. Only then will SAS check the condition `until(x > 30)`. The **DO loop** will continue running as long as `x` is less than or equal to 30. Once `x` exceeds 30, the loop condition is met, and no further iterations are executed.

The loop terminates when `x` exceeds 30 or when the maximum number of iterations (10) is reached, whichever is completed first. The key benefit of the DO UNTIL syntax is the guarantee of at least one execution cycle, which is vital for processes where initialization must occur within the first pass.

Practical Application: Using the Simple DO Loop (Example 1)

To illustrate the fixed-iteration behavior, we use the basic **DO loop** to construct a simple SAS dataset containing precisely 10 observations. This example demonstrates how the loop automatically manages the counter (`i`) and generates the data records without relying on external conditions.

```
/*use DO loop to create dataset*/
data data1;
x = 0;
do i = 1 to 10;
x = i*4;
output;
end;
run;

/*view dataset*/
proc print data=data1;
```

Executing this code produces the following output, confirming the creation of 10 rows. The index variable `i` serves as the observation counter (1 to 10), and `x` contains the resultant calculation (4 to 40).

Obs	x	i
1	4	1
2	8	2
3	12	3
4	16	4
5	20	5
6	24	6
7	28	7
8	32	8
9	36	9
10	40	10

Often, the counter variable used to control the loop (`i` in this case) is not needed in the final dataset. We can easily exclude this variable using the `DROP` statement within the DATA step. By adding `drop i;` before the `RUN` statement, the generated column `i` is suppressed from the output dataset, resulting in a cleaner final table.

```
/*use DO loop to create dataset*/
data data1;
x = 0;
do i = 1 to 10;
x = i*4;
output;
end;
drop i;
run;

/*view dataset*/
proc print data=data1;
```

The resulting dataset now contains only the calculated values of `x`.

Obs	x
1	4
2	8
3	12
4	16
5	20
6	24
7	28
8	32
9	36
10	40

Practical Application: Implementing the DO WHILE Loop (Example 2)

The **DO WHILE loop** is invaluable when you want a process to stop based on a calculated value, overriding the fixed iteration count. In this example, we limit the output based on the value of the calculated variable `x`, requiring it to remain strictly less than 20 at the initiation of each iteration.

```
/*use DO WHILE loop to create dataset*/
data data2;
x = 0;
do i = 1 to 10 while(x < 20);
x = i*4;
output;
end;
run;

/*view dataset*/
proc print data=data2;
```

The loop executes through the first five iterations. When `i=5`, the loop condition (x < 20) is still true (since `x` from the previous iteration was 16). Inside the loop, `x` is calculated as 20, and the observation is outputted. However, when the loop attempts to start iteration 6, the condition `while(x < 20)` is checked using the current value of `x` (which is 20). Since this condition fails, the loop terminates immediately, resulting in only five rows.

Obs	x	i
1	4	1
2	8	2
3	12	3
4	16	4
5	20	5

As observed in the output generated by PROC PRINT, the loop stopped executing rows as soon as the calculated value of `x` reached 20, demonstrating the conditional termination overriding the fixed count of 10 iterations.

Practical Application: Implementing the DO UNTIL Loop (Example 3)

The **DO UNTIL loop** structure guarantees execution of the code block before checking the termination condition. We utilize this loop below to ensure execution continues until the calculated value of `x` exceeds 30, producing the largest possible dataset before that condition is met.

```
/*use DO UNTIL loop to create dataset*/
data data3;
x = 0;
do i = 1 to 10 until(x > 30);
x = i*4;
output;
end;
run;

/*view dataset*/
proc print data=data3;
```

The loop executes until `x` becomes greater than 30. Iterations 1 through 7 result in `x` values from 4 to 28, and the condition `x > 30` remains false. When `i=8`, the code runs, `x` becomes 32, and the row is outputted. Since 32 is greater than 30, the loop condition is finally met (true) at the end of this iteration, and the loop terminates, preventing iteration 9 from starting.

Obs	x	i
1	4	1
2	8	2
3	12	3
4	16	4
5	20	5
6	24	6
7	28	7
8	32	8

The output confirms that the process halted precisely when `x` exceeded 30, resulting in a dataset with 8 rows, demonstrating how the conditional logic of the **DO UNTIL loop** efficiently controls the dataset creation process.

Conclusion and Further Exploration

Mastering the three variations of the **DO loop**--Simple DO, DO WHILE, and DO UNTIL--is fundamental for writing efficient and robust code in SAS. By correctly choosing the loop type, developers can precisely control the number of iterations and the exact moment termination occurs, whether based on a fixed count or a dynamic logical condition.

For those looking to deepen their expertise in SAS programming and advanced data manipulation techniques, we recommend exploring the following related resources: