The Basics of DO Loops

The Basics of Using Grouping Statements and DO Groups

You can execute a group of statements as a unit by using DO groups.

To construct a DO group, you use the DO and END statements along with other SAS statements.

Syntax, DO group:

DO;

SAS statements

END;

- The DO statement begins DO-group processing.
- *SAS statements* between the DO and END statements are called a DO group and are executed as a unit.
- The END statement terminates DO-group processing.

Tip: You can nest DO statements within DO groups.

You can use DO groups in IF-THEN/ELSE statements and SELECT groups to execute many statements as part of the conditional action.

Example: DO and END Statements

In this simple DO group, the statements between DO and END are performed only when TotalTime is greater than 800. If TotalTime is less than or equal to 800, statements in the DO group are not executed, and the program continues with the assignment statement that follows the appropriate ELSE statement.

```
data work.stresstest;
  set cert.tests;
  TotalTime=(timemin*60)+timesec;
  retain SumSec 5400;
  sumsec+totaltime;
  length TestLength $6 Message $20;
  if totaltime>800 then
    do;
        TestLength='Long';
        message='Run blood panel';
    end;
    else if 750<=totaltime<=800 then TestLength='Normal';
    else if totaltime<750 then TestLength='Short';
run;
proc print data=work.stresstest;
run;
```

Output 11.1 *PROC PRINT Output Work.StressTest (partial output)*

Obs	ID	Name	TimeSec	Tolerance	TotalTime	SumSec	TestLength	Message
1	2458	Murray, W	38	D	758	6158	Normal	
			. . . *more observations*. . .					
7	2552	Reberson, P	41	D	941	10411	Long	Run blood panel
			. . . *more observations*. . .					
10	2568	Eberhardt, S	49	N	1009	12835	Long	Run blood panel
11	2571	Nunnelly, A	2	I	902	13737	Long	Run blood panel
			. . . *more observations*. . .					
13	2574	Peterson, V	9	D	849	15317	Long	Run blood panel
			. . . *more observations*. . .					
15	2578	Cameron, L	27	I	867	16870	Long	Run blood panel
			. . . *more observations*. . .					
17	2584	Takahashi, Y	7	D	967	18636	Long	Run blood panel
18	2586	Derber, B	35	N	1055	19691	Long	Run blood panel
19	2588	Ivan, H	41	N	941	20632	Long	Run blood panel
20	2589	Wilcox, E	57	I	897	21529	Long	Run blood panel

DO Statement, Iterative Syntax

The *iterative DO statement* executes statements between the DO and END statements repetitively, based on the value of an index variable.

Syntax, DO statement, iterative:

DO *index-variable=specification-1* <, *...specification-n*>;

...more SAS statements...

END;

- *index-variable* names a variable whose value governs execution of the DO group.

  CAUTION:
  Avoid changing the index variable within the DO group. If you modify the index variable within the iterative DO group, you might cause infinite looping.

  Note: Unless you specify to drop it, the index variable is included in the data set that is being created.

- *specification* denotes an expression or series of expressions such as these:

start <TO *stop*> <BY *increment*> <WHILE(*expression*) | UNTIL(*expression*)>

The DO group is executed first with *index-variable* equal to *start*. The value of *start* is evaluated before the first execution of the loop.

- *start* specifies the initial value of the index variable.
- TO *stop* specifies the ending value of the index variable.

  TIP Any changes to stop made within the DO group do not affect the number of iterations. To stop iteration of a loop before it finishes processing, change the value of *index-variable*, or use a LEAVE statement to go to a statement outside the loop.

- BY *increment* specifies a positive or negative number (or an expression that yields a number) to control the incrementing of *index-variable*.

  The value of *increment* is evaluated before the execution of the loop. If no increment is specified, the index variable is increased by 1. When *increment* is positive, *start* must be the lower bound, and *stop*, if present, must be the upper bound for the loop. If *increment* is negative, *start* must be the upper bound, and *stop*, if present, must be the lower bound for the loop.

- WHILE(*expression*) | UNTIL(*expression*) evaluates, either before or after execution of the DO group, any SAS expression that you specify. Enclose the expression in parentheses.

  A WHILE expression is evaluated before each execution of the loop, so that the statements inside the group are executed repetitively while the expression is true. An UNTIL expression is evaluated after each execution of the loop, so that the statements inside the group are executed repetitively until the expression is true.

Note: The order of the optional TO and BY clauses can be reversed.

Note: When you use more than one specification, each one is evaluated before its execution.

Example: Processing Iterative DO Loops

DO loops process a group of statements repeatedly rather than once. This can greatly reduce the number of statements required for a repetitive calculation. For example, these 12 sum statements compute a company's annual earnings from investments. Notice that all 12 statements are identical.

```
data work.earn (drop=month);
  set cert.master;
  Earned=0;
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
  earned+(amount+earned)*(rate/12);
run;
```

In this program, each sum statement accumulates the calculated interest that is earned for an investment for one month. The variable Earned is created in the DATA step to store the earned interest. The investment is compounded monthly, meaning that the value of the earned interest is cumulative.

By contrast, a DO loop enables you to achieve the same results with fewer statements. In this case, the sum statement executes 12 times within the DO loop during each iteration of the DATA step. In this example, the DO group Month is the index variable, 1 is the start-variable, and 12 is the stop variable.

```
data work.earnings (drop=month);
  set cert.master;
  Earned=0;
  do month=1 to 12;
    earned+(amount+earned)*(rate/12);
  end;
  Balance=Amount+Earned;
run;
proc print data=work.earnings;
run;
```

Output 11.2 *PROC PRINT Output of Work.Earnings*

Obs	Account	Amount	Rate	Earned	Balance
1	1025	9600	0.07	693.985	10293.98
2	1026	1500	0.05	76.743	1576.74
3	1027	2500	0.05	127.905	2627.90
4	1028	5000	0.08	414.998	5415.00
5	1029	6500	0.07	469.886	6969.89
6	1030	5000	0.07	361.450	5361.45
7	1031	4000	0.06	246.711	4246.71
8	1032	3000	0.01	30.138	3030.14
9	1033	2500	0.04	101.854	2601.85
10	1034	3500	0.04	142.595	3642.60
11	1035	1000	0.02	20.184	1020.18

Constructing DO Loops

DO Loop Execution

Here is how the DO loop executes in the DATA step. This example sums the interest that was earned each month for a one-year investment.

```
data work.earnings;
  Amount=1000;
  Rate=0.75/12;
  do month=1 to 12;
    Earned+(amount+earned)*rate;
  end;
run;
```

This DATA step does not read data from an external source. When submitted, it compiles and then executes only once to generate data. During compilation, the program data vector is created for the Work.Earnings data set.

Program Data Vector

_N_	Amount	Rate	month	Earned
•	•	•	•	•

When the DATA step executes, the values of Amount and Rate are assigned.

Program Data Vector

_N_	Amount	Rate	month	Earned
1	1000	0.0625	•	0

Next, the DO loop executes. During each execution of the DO loop, the value of Earned is calculated and is added to its previous value. Then the value of Month is incremented. On the 12th execution of the DO loop, the value of Month is incremented to 12 and the value of Earned is 1069.839.

Program Data Vector

_N_	Amount	Rate	month	Earned
1	1000	0.0625	12	1069.83

After the 12th execution of the DO loop, the value of Month is incremented to 13. Because 13 exceeds the stop value of the iterative DO statement, the DO loop stops executing, and processing continues to the next DATA step statement. The end of the DATA step is reached, the values are written to the Work.Earnings data set, and in this example, the DATA step ends. Only one observation is written to the data set.

Figure 11.1 *SAS Data Set Work.Earnings*

	Amount	Rate	month	Earned
1	1000	0.0625	13	1069.8899918

Notice that the index variable Month is also stored in the data set. In most cases, the index variable is needed only for processing the DO loop and can be dropped from the data set.

Using Explicit OUTPUT Statements

To create an observation for each iteration of the DO loop, place an OUTPUT statement inside the loop. By default, every DATA step contains an implicit OUTPUT statement at the end of the step. But placing an explicit OUTPUT statement in a DATA step overrides automatic output, causing SAS to add an observation to the data set only when the explicit OUTPUT statement is executed.

The previous example created one observation because it used automatic output at the end of the DATA step. In the following example, the OUTPUT statement overrides automatic output, so the DATA step writes 20 observations.

```
data work.earn;
  Value=2000;
  do Year=1 to 20;
    Interest=value*.075;
    value+interest;
    output;
  end;
run;
proc print data=work.earn;
run;
```

Figure 11.2 *HTML Output: OUTPUT Statement inside Each DO Loop (partial output)*

Obs	Value	Interest
1	2150.00	150.000
2	2311.25	161.250
3	2484.59	173.344
4	2670.94	186.345
5	2871.26	200.320
...more observations...		
15	5917.75	412.867
16	6361.59	443.832
17	6838.71	477.119
18	7351.61	512.903
19	7902.98	551.371
20	8495.70	592.723

Decrementing DO Loops

You can decrement a DO loop's index variable by specifying a negative value for the BY clause. For example, the specification in this iterative DO statement decreases the index variable by **1**, resulting in values of **5**, **4**, **3**, **2**, and **1**. The following brief examples show you the syntax.

```
DO index-variable=5 to 1 by -1;
   ...more SAS statements...
END;
```

When you use a negative BY clause value, the start value must always be greater than the stop value in order to decrease the index variable during each iteration.

```
DO index-variable=5 to 1 by -1;
   ...more SAS statements...
END;
```

Specifying a Series of Items

You can also specify how many times a DO loop executes by listing items in a series.

Syntax, DO loop with a variable list:

DO *index-variable=value1, value2, value3...* **;**

...more SAS statements...

END;

values can be character or numeric.

When the DO loop executes, it executes once for each item in the series. The index variable equals the value of the current item. You must use commas to separate items in the series.

To list items in a series, you must specify one of the following, as shown in the syntax:

- all numeric values.

  ```
  DO index-variable=2,5,9,13,27;
     ...more SAS statements...
  END;
  ```

- all character values, which are enclosed in quotation marks.

  ```
  DO index-variable='MON','TUE','WED','THR','FRI';
     ...more SAS statements...
  END;
  ```

- all variable names. The index variable takes on the values of the specified variables.

  ```
  DO index-variable=win,place,show;
     ...more SAS statements...
  END;
  ```

Variable names must represent either all numeric or all character values. Do not enclose variable names in quotation marks.

Nesting DO Loops

Indenting and Nesting DO Groups

You can nest DO groups to any level, just like you nest IF-THEN/ELSE statements.

Note: The memory capabilities of your system might limit the number of nested DO statements that you can use.

Here is an example structure of nested DO groups:

```
do;
    ...more SAS statements...;
        do;
            ...more SAS statements...;
                do;
                    ...more SAS statements...;
                end;
        end;
end;
```

TIP It is good practice to indent the statements in DO groups, as shown in the preceding statements, so that their position indicates the levels of nesting.

Examples: Nesting DO Loops

Iterative DO statements can be executed within a DO loop. Putting a DO loop within a DO loop is called nesting.

```
do i=1 to 20;
   ...more SAS statements...
   do j=1 to 10;
      ...more SAS statements...
   end;
   ...more SAS statements...
end;
```

The DATA step below computes the value of a one-year investment that earns 7.5% annual interest, compounded monthly.

```
data work.earn;
  Capital=2000;
  do month=1 to 12;
    Interest=capital*(.075/12);
    capital+interest;
  end;
run;
```

Assume that the same amount of capital is to be added to the investment each year for 20 years. The new program must perform the calculation for each month during each of the 20 years. To do this, you can include the monthly calculations within another DO loop that executes 20 times.

```
data work.earn;
  do year=1 to 20;
    Capital+2000;
      do month=1 to 12;
        Interest=capital*(.075/12);
        capital+interest;
      end;
  end;
run;
```

During each iteration of the outside DO loop, an additional 2,000 is added to the capital, and the nested DO loop executes 12 times.

```
data work.earn;
  do year=1 to 20;
    Capital+2000;
      do month=1 to 12;
        Interest=capital*(.075/12);
        capital+interest;
      end;
  end;
run;
```

Remember, in order for nested DO loops to execute correctly, you must do the following:

- Assign a unique index-variable name in each iterative DO statement.

```
data work.earn;
  do year=1 to 20;
    Capital+2000;
      do month=1 to 12;
        Interest=capital*(.075/12);
        capital+interest;
      end;
  end;
run;
```

- End each DO loop with an END statement.

```
data work.earn;
  do year=1 to 20;
    Capital+2000;
      do month=1 to 12;
        Interest=capital*(.075/12);
        capital+interest;
      end;
  end;
run;
```

It is easier to manage nested DO loops if you indent the statements in each DO loop as shown above.

Iteratively Processing Observations from a Data Set

Previous examples of DATA steps used DO loops to generate one or more observations from one iteration of the DATA step. It is also possible to write a DATA step that reads a data set and uses variables in the input data set to compute the value of a new variable.

The SAS data set Work.CDRates contains interest rates for certificates of deposit (CDs) that are available from several institutions.

Suppose you want to compare how much each CD earns at maturity with an investment of $5,000. The DATA step below creates a new data set, Work.Compare, that contains the added variable, Investment.

```
data work.compare(drop=i);
  set work.cdrates;
  Investment=5000;
  do i=1 to years;
```

```
    investment+rate*investment;
  end;
run;
proc print data=work.compare;
run;
```

The index variable is used only to execute the DO loop, so it is dropped from the new data set. Notice that the data set variable Years is used as the stop value in the iterative DO statement. As a result, the DO loop executes the number of times specified by the current value of Years.

Here is what happens during each iteration of the DATA step:

- An observation is read from Work.CDRates.
- The value 5000 is assigned to the variable Investment.
- The DO loop executes, based on the current value of Years.
- The value of Investment is incremented (each time that the DO loop executes), using the current value of Rate.

At the end of the first iteration of the DATA step, the first observation is written to the Work.Compare data set. Control returns to the top of the DATA step, and the next observation is read from Work.CDRates. These steps are repeated for each observation in Work.CDRates. The resulting data set contains the computed values of Investment for all observations that have been read from Work.CDRates.

Figure 11.3 *HTML Output: Work.Compare Data Set*

Obs	Institution	Rate	Years	Investment
1	MBNA America	0.0817	5	7404.64
2	Metropolitan Bank	0.0814	3	6323.09
3	Standard Pacific	0.0806	4	6817.57

Conditionally Executing DO Loops

Overview

The iterative DO statement specifies a fixed number of iterations for the DO loop. However, there are times when you want to execute a DO loop until a condition is reached or while a condition exists, but you do not know how many iterations are needed.

Suppose you want to calculate the number of years required for an investment to reach $50,000. In the DATA step below, using an iterative DO statement is inappropriate because you are trying to determine the number of iterations required for Capital to reach $50,000.

```
data work.invest;
  do year=1 to ? ;
    Capital+2000;
    capital+capital*.10;
  end;
run;
```

The DO WHILE and DO UNTIL statements enable you to execute DO loops based on whether a condition is true or false.

Using the DO UNTIL Statement

The DO UNTIL statement executes a DO loop until the expression becomes true.

Syntax, DO UNTIL statement:

DO UNTIL(*expression*)**;**

...more SAS statements...

END;

expression is a valid SAS expression enclosed in parentheses.

The expression is not evaluated until the bottom of the loop. Therefore, a DO UNTIL loop always executes at least once. When the expression is evaluated as true, the DO loop stops.

Assume you want to know how many years it takes to earn $50,000 if you deposit $2,000 each year into an account that earns 10% interest. The DATA step below uses a DO UNTIL statement to perform the calculation until $50,000 is reached. Each iteration of the DO loop represents one year.

```
data work.invest;
  do until(Capital>=50000);
    capital+2000;
    capital+capital*.10;
    Year+1;
  end;
run;
```

Here is what happens during each iteration of the DO loop:

- 2000 is added to the value of Capital to reflect the annual deposit of $2,000.
- 10% interest is added to Capital.
- The value of Year is incremented by 1.

Because there is no index variable in the DO UNTIL statement, the variable Year is created in a sum statement to count the number of iterations of the DO loop. This program produces a data set that contains the single observation shown below. To accumulate more than $50,000 in capital requires 13 years (and 13 iterations of the DO loop).

Figure 11.4 *SAS Data Set Work.Invest: Accumulation of More Than $50,000*

	Capital	Year
1	53949.97	13

Using the DO WHILE Statement

Like the DO UNTIL statement, the DO WHILE statement executes DO loops conditionally. You can use the DO WHILE statement to execute a DO loop while the expression is true.

Syntax, DO WHILE statement:

DO WHILE(*expression*)**;**

...more SAS statements...

END;

expression is a valid SAS expression enclosed in parentheses.

An important difference between the DO UNTIL and DO WHILE statements is that the DO WHILE expression is evaluated at the top of the DO loop. If the expression is false the first time it is evaluated, the DO loop never executes. For example, in the following program the DO loop does not execute because the value of Capital is initially zero, which is less than 50,000.

```
data work.invest;
  do while(Capital>=50000);
    capital+2000;
    capital+capital*.10;
    Year+1;
  end;
run;
```

Suppose you also want to limit the number of years you invest your capital to 10 years. You can add the UNTIL or WHILE expression to an iterative DO statement to further control the number of iterations. This iterative DO statement enables you to execute the DO loop until Capital is greater than or equal to 50000 or until the DO loop executes 10 times, whichever occurs first.

```
data work.invest;
  do year=1 to 10 until (Capital>=50000);
    capital+2000;
    capital+capital*.10;
  end;
run;
```

Figure 11.5 *SAS Data Set Work.Invest: Executing DO Loop until Capital >=$50,000*

	Year	Capital
1	10	35062.33

In this case, the DO loop stops executing after 10 iterations, and the value of Capital never reaches 50000. If you increase the amount added to Capital each year to 4000, the DO loop stops executing after the eighth iteration when the value of Capital exceeds 50000.

```
data work.invest;
  do year=1 to 10 until (Capital>=50000);
    capital+4000;
    capital+capital*.10;
  end;
run;
```

Figure 11.6 *SAS Data Set Work.Invest: Increase Amount Added to Capital Using a DO Loop*

	year	Capital
1	8	50317.91

The UNTIL and WHILE expressions in an iterative DO statement function similarly to the DO UNTIL and DO WHILE statements. As shown in the following syntax, both statements require a valid SAS expression that is enclosed in parentheses.

```
DO index-variable=start TO stop BY increment UNTIL(expression);
```

```
DO index-variable=start TO stop BY increment WHILE(expression);
```

The UNTIL expression is evaluated at the bottom of the DO loop. Therefore, the DO loop always executes at least once. The WHILE expression is evaluated before the execution of the DO loop. As a result, if the condition is initially false, the DO loop never executes.