

Do Loops in SAS Software

Prepared by



International SAS® Training and Consulting

Destiny Corporation - 1321 Silas Deane Highway #A - Wethersfield, CT 06109-4302

Phone: (860) 721-1684 - 1-800-7TRAINING - Fax: (860) 721-9784

Email: destinycorp@compuserve.com

Web: www.destinycorp.com

The Data Step Loop

A data step in the SAS programming language is an implied loop. It usually contains a read statement (INPUT or SET) that reads data from an external file or another SAS data set.

	Compile		Statements Checked For Syntax
	V		
	Create		Input Buffer, Program Data Vector Created, Descriptor Portion Written
	V		
>	Begin		Data Statement, counts iterations
	V		
	Assign		Assign some variables to missing
	V		
L	Test	>	Close Another Record?
O	V		
O	Read		Input Record
P	V		
	Execute		SAS Statements
	V		
	Write		Observation to data set
	V		
<	Return		Loop back to Top

The data step loop is implied by the RETURN statement present at the end of every data step:

Example:

PROGRAM EDITOR

```
data patients;
  input patient drugcost surgcost;
  datalines;
1 160 550
2 250 790
3 190 1800
4 45 440
5 300 4500
6 120 900
run;
```

The data step loop implies the following statements:

PROGRAM EDITOR

```
data patients;
  input patient drugcost surgcost;
  output;
  return;
  datalines;
1 160 550
2 250 790
3 190 1800
4 45 440
5 300 4500
6 120 900
run;
```

Any data step that READS data from an external file or SAS data set contains an implied loop. All the executable program statements are executed once for each input record or observation.

Example:

The patients data set would cause the following data step to execute 6 times:

PROGRAM EDITOR

```
data highcost lowcost;
  set patients;
  if sum(drugcost,surgcost) gt 1000
    then output highcost;
    else output lowcost;
run;
```

Both the SET and the IF statement are executed 6 times, once for each input observation.

Loops in Loops with Do; End;

Within the implied loop of the data step, you can program other loops. These can be useful in performing repeated actions, generating data, reading data and conditionally executing code.

In this example, we need to calculate the new concentration of several chemical solutions. A molar solution is diluted by successively dividing the concentration of the solution by 2. This is done 3 times to give the 8th molar solution:

PROGRAM EDITOR

```
data dilution;
  input @1 solution $20. stdconc;
  newconc=stdconc;
  newconc=newconc/2;
  newconc=newconc/2;
  newconc=newconc/2;
  datalines;
NaHC03          8.3
NaCl             5.85
KMn04           15.8
proc print data=dilution noobs;
run;
```

The value of 8.3 will be used: $8.3/2=4.15/2=2.075/2=1.0375$

Do Loops can be used to facilitate coding:

PROGRAM EDITOR

```
data dilution;
  input @1 solution $20. stdconc;
  newconc=stdconc;
  do i = 1 to 3;
    newconc=newconc/2;
  end;
cards;
NaHC03          8.3
NaCl             5.85
KMn04           15.8
proc print data=dilution noobs;
Title 'Grams of solute to achieve 8th molar solution';
run;
```

OUTPUT

Grams of solute to achieve 8th molar solution

SOLUTION	STDCONC	NEWCONC	I
NaHC03	8.30	1.03750	4
NaCl	5.85	0.73125	4
KMn04	15.80	1.97500	4

This makes use of the fact that you simply want to repeat program statements for each input record. Thus we have used a loop within the implied loop of the data step itself.

Hands-on Workshops

Loops in Loops:

```
>data .....;
  input .....;      * causes data step iterations;

  >do index = start to finish; * causes do loop iterations;

      SAS statements calculating values;

  >end;

>run;
```

Notes on DO Loops

Syntax:

```
DO indexvariable=start TO stop BY increment;

    statements using index variable as counter
                or
    performing repetitive calculations
```

END;

Details:

- 1 The indexvariable (I in our chemistry experiment) is NOT automatically dropped from the data set.
- 2 Start, stop and increment are:
 - set on entry to the loop
 - cannot be changed during the processing of the loop
 - numbers, variables or expressions
- 3 The BY clause is optional; the default value of increment is 1
- 4 Indexvariable is incremented at the bottom of the loop
- 5 Comparison of the values of indexvariable and stop is done at the top of the loop.

Bounded Loops

The following 8 examples are all valid statements:

● DO LOOP

- 1) do x = 4 to 40 by 4;
- 2) do y = .07 to .1 by 0.003;
- 3) do z = 20 to 2 by -4;
- 4) do a= 1,2,3,5,7,11,13,17,23;
- 5) do day= 'MON', 'TUE', 'WED', 'THU', 'FRI';
- 6) do var1= var2 to var3 by var4;
- 7) do var5= var2/12 to 100;
- 8) do date='01JAN96'D, '15JAN96'D, '31JAN96'D;

The common feature about all these loops is that they are bounded. The number of times the loop will iterate is fixed, even in cases where the start, stop and increment values are determined by variable values.

Generating Data with Do Loops:**PROGRAM EDITOR**

```
data squares;
  do x=-6 to 6;
    y=x*x;
    output;
  end;
proc print data = squares noobs;
run;
```

OUTPUT

X	Y
-6	36
-5	25
-4	16
-3	9
-2	4
-1	1
0	0
1	1
2	4
3	9
4	16
5	25
6	36

How many times does the data step execute?

Hands-on Workshops

Conditional Termination

The loops we have seen have a fixed number of iterations defined by their *start*, *stop* and *by* values. Suppose we do not know how many times the loop executes and we just want the loop to continue until a condition is met, or while a condition remains true. To do this requires the use of the DO WHILE, or DO UNTIL structures.

Do While

```
DO WHILE (expression);  
    further SAS statements;  
  
END;
```

- The loop continues **as long as the condition is true**.
- The expression is evaluated at the top of the loop so the code in the loop is not necessarily executed at all.

PROGRAM EDITOR

```
data dowhile;  
    x=3;  
    do while(x<4);  
        x=x*2;  
        output;  
    end;  
run;  
proc print data=dowhile;  
run;
```

OUTPUT

OBS	X
1	6

1. x=3
2. x=6
3. output
4. go to top of loop
5. test 6<4, yes so
6. stop loop

Do Until

```
DO UNTIL (expression);  
    further SAS statements;  
  
END;
```

- The loop executes until the condition becomes true.
- The expression is evaluated at the bottom of the loop, so the code within the loop must be executed at least once.

PROGRAM EDITOR

```
data dountil;
  x=3;
  do until(x<4);
    x=x*2;
    output;
  end;
run;
proc print data=dountil;
run;
```

1. x=3
2. x=6
3. output
4. test 6<4? no, so continue
4. x=6
5. x=12
6. output...etc...

LOG: data step stops when number (X) is too large, at OBS 1023

OUTPUT

OBS	X
1	6
2	12
3	24
4	48
5	96
1020	3.3707E307
1021	6.7413E307
1022	1.3483E308
1023	.

Correcting the Do Until

PROGRAM EDITOR

```
data dountil;
  x=3;
  do until(x > 4);
    x=x*2;
    output;
  end;
run;
proc print data=dountil;
run;
```

Hands-on Workshops

1. $x=3$
2. $x=6$
3. output
4. test $6>4$, yes so stop.

Example:

Here we have a series of bacteria and their growth rate. We wish to calculate the number of days it will take for a given bacterium to double in sample size.

PROGRAM EDITOR

```
data growth;
  input bug $ g_rate;
  amount=100;
  days=0;
  do while(amount le 200);
    amount=amount*g_rate;
    days+1;
  end;
  output;
  cards;
ECOLI 1.1
STREPC 1.3
STAPHC 1.42
HAEMO 1.05
run;
proc print;
run;
```

This gives the following output:

OUTPUT

OBS	BUG	G_RATE	AMOUNT	DAYS
1	ECOLI	1.10	214.359	8
2	STREPC	1.30	219.700	3
3	STAPHC	1.42	201.640	2
4	HAEMO	1.05	207.893	15

Questions

1. What would be the corresponding DO UNTIL statement?
2. What would be the effect of moving the output statement after statement 5 or 7?

Answers

1. Corresponding Do Until:

PROGRAM EDITOR

```

data growth;
  input bug $ g_rate;
  amount=100;
  days=0;
  do until(amount gt 200);
    amount=amount*g_rate;
    days+1;
  end;
  output;
  cards;
ECOLI 1.1
STREPC 1.3
STAPHC 1.42
HAEMO 1.05
run;
proc print;
run;

```

OUTPUT

OBS	BUG	G_RATE	AMOUNT	DAYS
1	ECOLI	1.10	214.359	8
2	STREPC	1.30	219.700	3
3	STAPHC	1.42	201.640	2
4	HAEMO	1.05	207.893	15

2. Moving the Output Statement after the calculation

PROGRAM EDITOR

```

data growth;
  input bug $ g_rate;
  amount=100;
  days=0;
  do until(amount gt 200);
    amount=amount*g_rate;
    output;
    days+1;
  end;
  cards;
ECOLI 1.1
STREPC 1.3
STAPHC 1.42
HAEMO 1.05
run;
proc print;
run;

```

OUTPUT

OBS	BUG	G_RATE	AMOUNT	DAYS
1	ECOLI	1.10	110.000	0
2	ECOLI	1.10	121.000	1
3	ECOLI	1.10	133.100	2
4	ECOLI	1.10	146.410	3
5	ECOLI	1.10	161.051	4
6	ECOLI	1.10	177.156	5
7	ECOLI	1.10	194.872	6
8	ECOLI	1.10	214.359	7
9	STREPC	1.30	130.000	0
10	STREPC	1.30	169.000	1
11	STREPC	1.30	219.700	2
12	STAPHC	1.42	142.000	0
13	STAPHC	1.42	201.640	1
14	HAEMO	1.05	105.000	0
15	HAEMO	1.05	110.250	1
16	HAEMO	1.05	115.763	2
17	HAEMO	1.05	121.551	3
18	HAEMO	1.05	127.628	4
19	HAEMO	1.05	134.010	5
20	HAEMO	1.05	140.710	6
21	HAEMO	1.05	147.746	7
22	HAEMO	1.05	155.133	8
23	HAEMO	1.05	162.889	9
24	HAEMO	1.05	171.034	10
25	HAEMO	1.05	179.586	11
26	HAEMO	1.05	188.565	12
27	HAEMO	1.05	197.993	13
28	HAEMO	1.05	207.893	14

3. Moving the Output Statement before the calculation

PROGRAM EDITOR

```

data growth;
  input bug $ g_rate;
  amount=100;
  days=0;
  do until(amount gt 200);
    output;
    amount=amount*g_rate;
    days+1;
  end;
  cards;
ECOLI 1.1
STREPC 1.3
STAPHC 1.42
HAEMO 1.05
run;
proc print;
run;

```

OUTPUT

OBS	BUG	G_RATE	AMOUNT	DAYS
1	ECOLI	1.10	100.000	0
2	ECOLI	1.10	110.000	1
3	ECOLI	1.10	121.000	2
4	ECOLI	1.10	133.100	3
5	ECOLI	1.10	146.410	4
6	ECOLI	1.10	161.051	5
7	ECOLI	1.10	177.156	6
8	ECOLI	1.10	194.872	7
9	STREPC	1.30	100.000	0
10	STREPC	1.30	130.000	1
11	STREPC	1.30	169.000	2
12	STAPHC	1.42	100.000	0
13	STAPHC	1.42	142.000	1
14	HAEMO	1.05	100.000	0
15	HAEMO	1.05	105.000	1
16	HAEMO	1.05	110.250	2
17	HAEMO	1.05	115.763	3
18	HAEMO	1.05	121.551	4
19	HAEMO	1.05	127.628	5
20	HAEMO	1.05	134.010	6
21	HAEMO	1.05	140.710	7
22	HAEMO	1.05	147.746	8
23	HAEMO	1.05	155.133	9
24	HAEMO	1.05	162.889	10
25	HAEMO	1.05	171.034	11
26	HAEMO	1.05	179.586	12
27	HAEMO	1.05	188.565	13
28	HAEMO	1.05	197.993	14