Iterative Data Processing

Statistics 426: SAS Programming

Module 6

2021

Interative data processing

1 1 2 3 5 8 13 21 . . .

Silly of me to ask. :-) It is the Fibonacci sequence, and we will create a loop to do calculations of the sequence in this module

Iterative refers to a systematic, repetitive, and recursive process in qualitative data analysis. An iterative approach involves a sequence of tasks carried out in exactly the same manner each time and executed multiple times

Using the IF family

IF-THEN (IF-THEN/ELSE) statement(s) can only have one executable statement.

```
IF expression THEN statement;
```

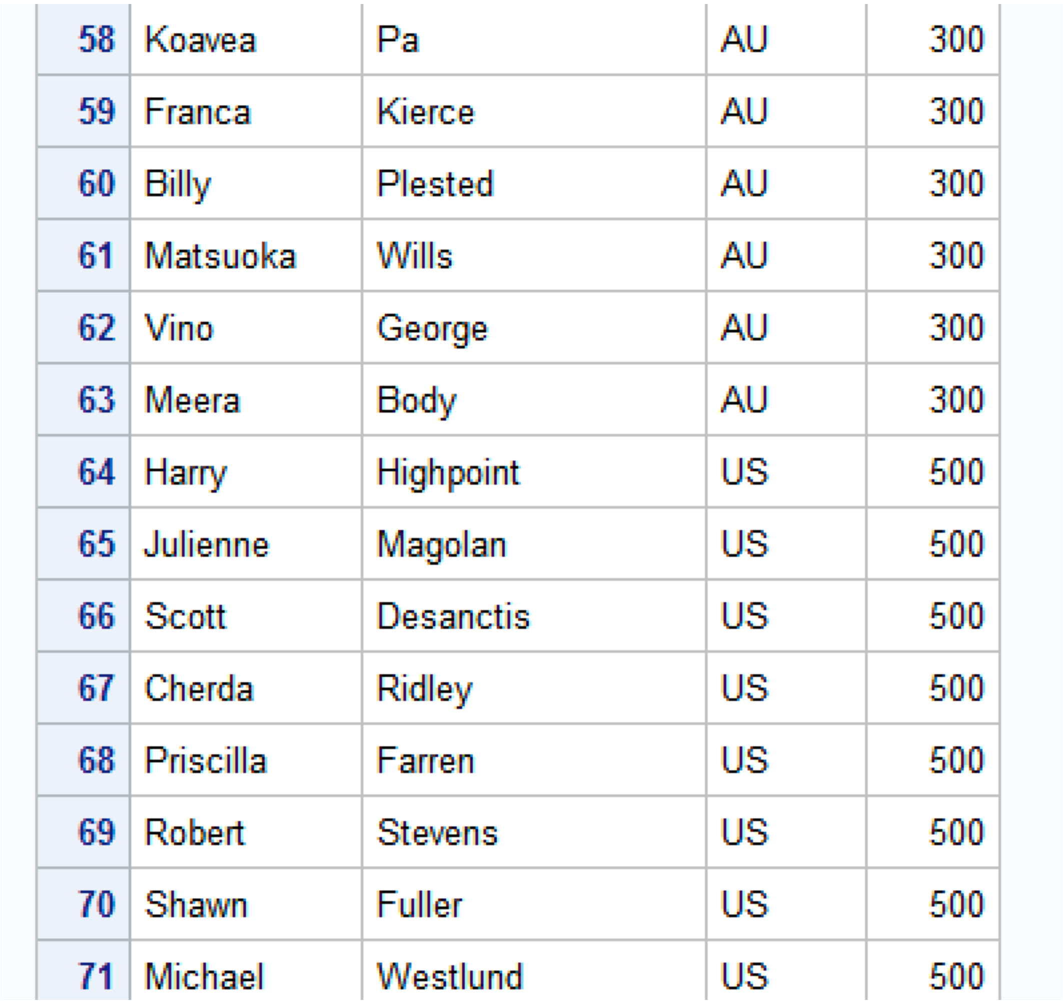
IF-THEN/DO statements are similar to the IF-THEN statements, but the IF-THEN/DO statements can have *multiple* executable statements unlike the IF-THEN statements

IF-THEN/ELSE I

Setting bonus based on country

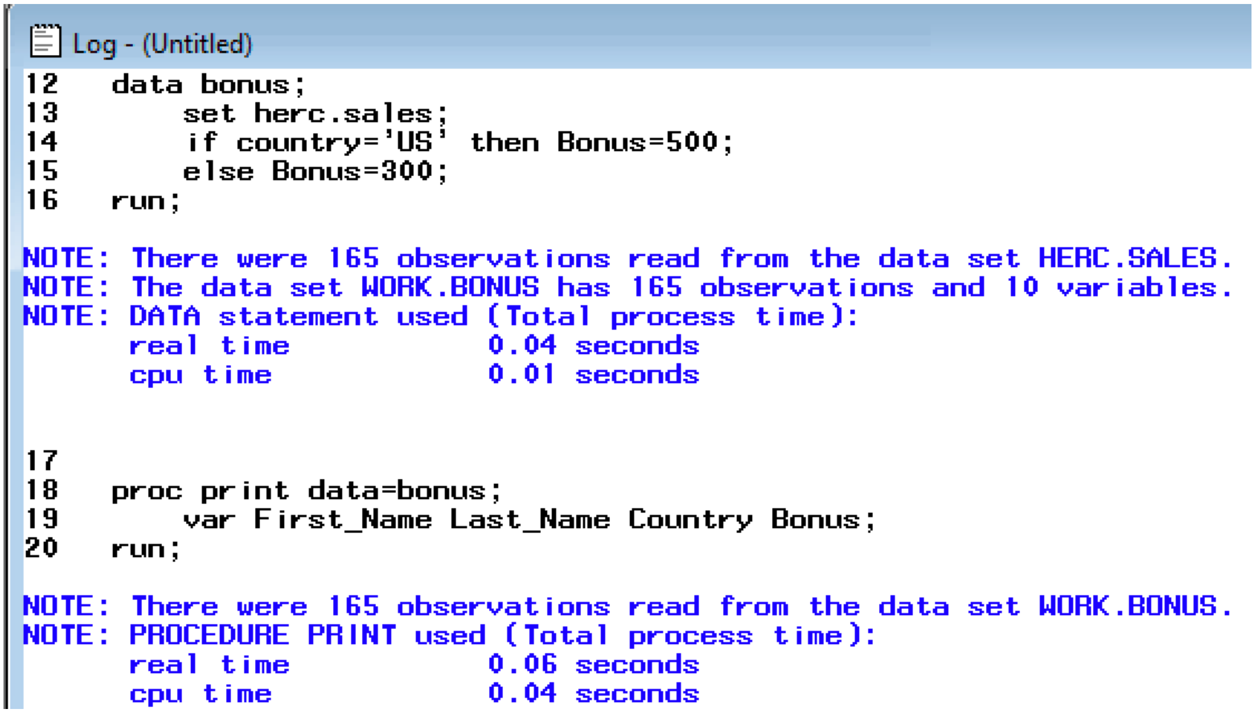
```
libname herc 'S:\Courses\stat-renaes\Stat426\sasdatafiles';

data bonus;
    set herc.sales;
    if country='US' then Bonus=500;
    else if country='AU' then Bonus=300;
run;

proc print data=bonus;
    var First_Name Last_Name Country Bonus;
run;
```

IF-THEN/ELSE I print



58	Koavea	Pa	AU	300
59	Franca	Kierce	AU	300
60	Billy	Plested	AU	300
61	Matsuoka	Wills	AU	300
62	Vino	George	AU	300
63	Meera	Body	AU	300
64	Harry	Highpoint	US	500
65	Julienne	Magolan	US	500
66	Scott	Desanctis	US	500
67	Cherda	Ridley	US	500
68	Priscilla	Farren	US	500
69	Robert	Stevens	US	500
70	Shawn	Fuller	US	500
71	Michael	Westlund	US	500

IF-THEN/ELSE I log

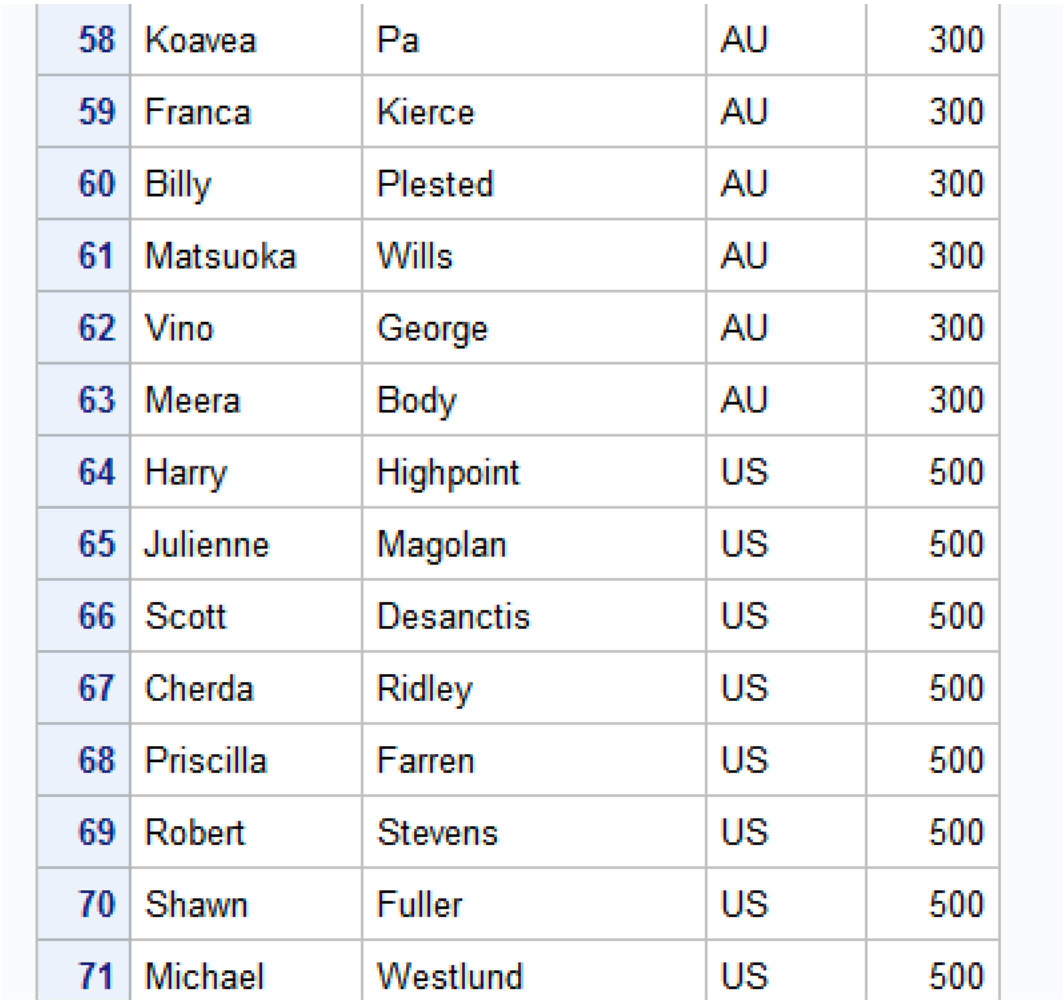


```
     data bonus;
         set herc.sales;
         if country='US' then Bonus=500;
         else Bonus=300;
     run;

NOTE: There were 165 observations read from the data set HERC.SALES.
NOTE: The data set WORK.BONUS has 165 observations and 10 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds



     proc print data=bonus;
         var First_Name Last_Name Country Bonus;
     run;

NOTE: There were 165 observations read from the data set WORK.BONUS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.06 seconds
      cpu time            0.04 seconds
```

IF-THEN/ELSE II

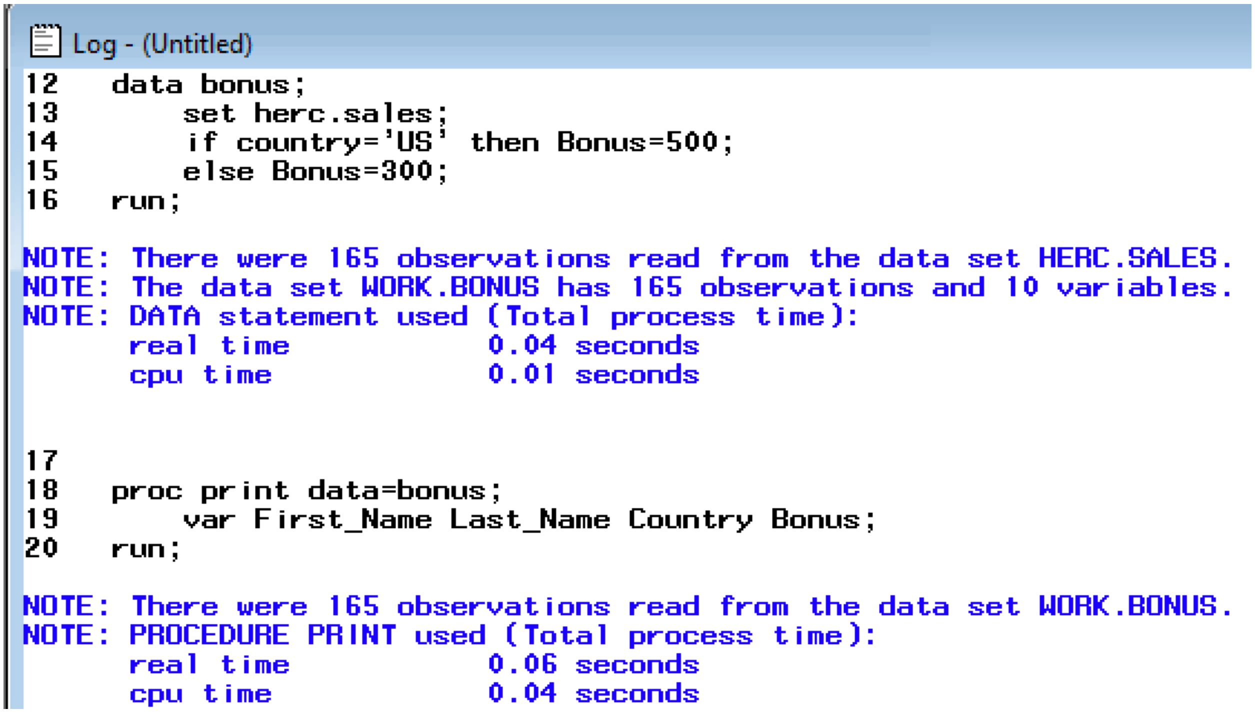
```
data bonus;
    set herc.sales;
    if country='US' then Bonus=500;
    else Bonus=300;
run;

proc print data=bonus;
    var First_Name Last_Name Country Bonus;
run;
```

IF-THEN/ELSE II print



58	Koavea	Pa	AU	300
59	Franca	Kierce	AU	300
60	Billy	Plested	AU	300
61	Matsuoka	Wills	AU	300
62	Vino	George	AU	300
63	Meera	Body	AU	300
64	Harry	Highpoint	US	500
65	Julienne	Magolan	US	500
66	Scott	Desanctis	US	500
67	Cherda	Ridley	US	500
68	Priscilla	Farren	US	500
69	Robert	Stevens	US	500
70	Shawn	Fuller	US	500
71	Michael	Westlund	US	500

IF-THEN/ELSE II log



```
12    data bonus;
13        set herc.sales;
14        if country='US' then Bonus=500;
15        else Bonus=300;
16    run;

NOTE: There were 165 observations read from the data set HERC.SALES.
NOTE: The data set WORK.BONUS has 165 observations and 10 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.01 seconds


17
18    proc print data=bonus;
19        var First_Name Last_Name Country Bonus;
20    run;

NOTE: There were 165 observations read from the data set WORK.BONUS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.06 seconds
      cpu time            0.04 seconds
```

General form of IF-THEN/DO

```
IF expression THEN DO;
<executable statements>;
END;

ELSE IF expression THEN DO;
<executable statements>;
END;
```

Each DO group can contain multiple executable statements that apply to each expression, and each DO group *must* end with an END statement

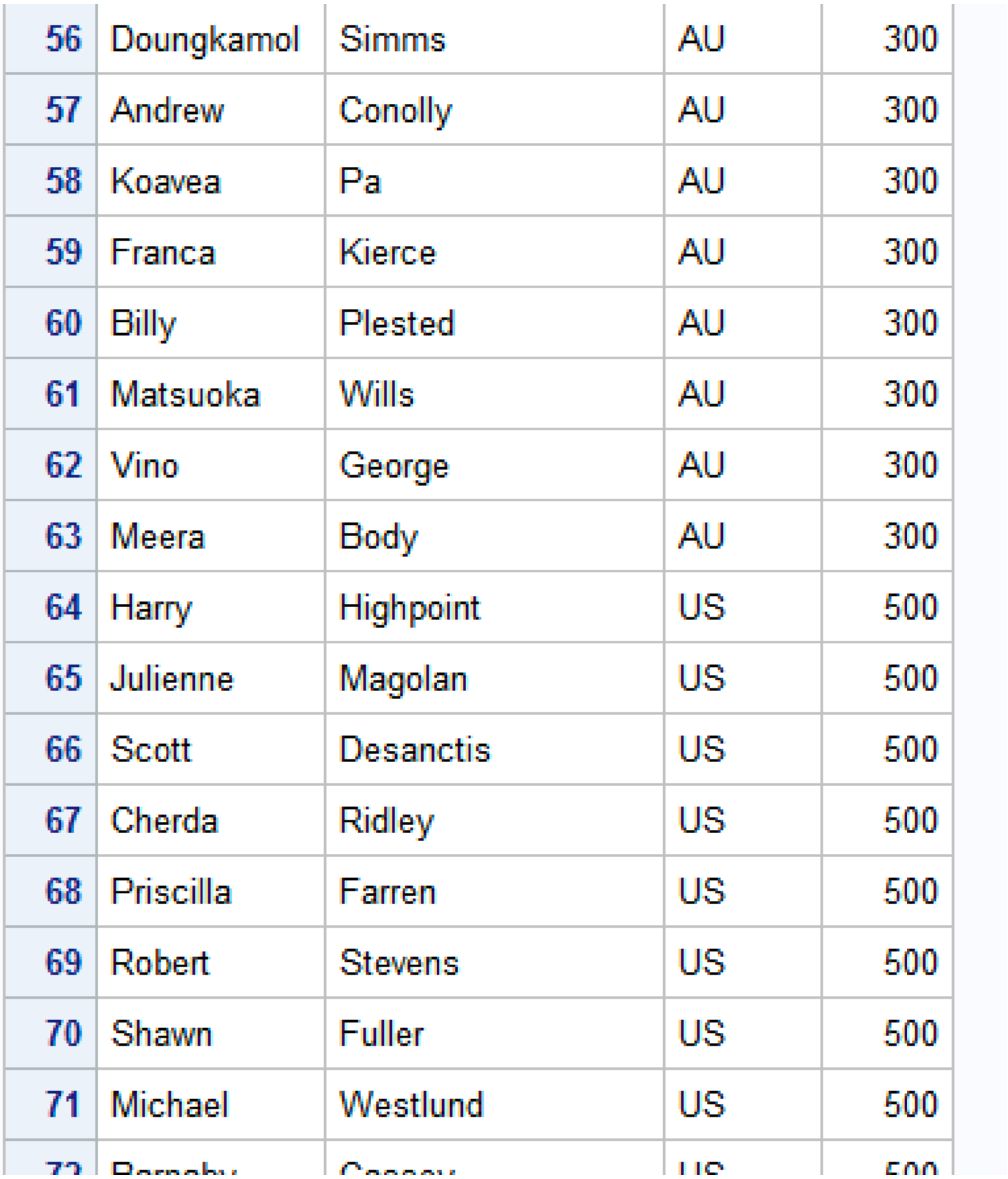
if then do

Setting bonus based on country and signifying occurrence

```
data bonus;
    set herc.sales;
    if country='US' then do;
       Bonus=500;
       Freq='Once a Year';
    end;

    else if country='AU' then do;
       Bonus=300;
       Freq='Twice a Year';
    end;
run;

proc print data=bonus;
```

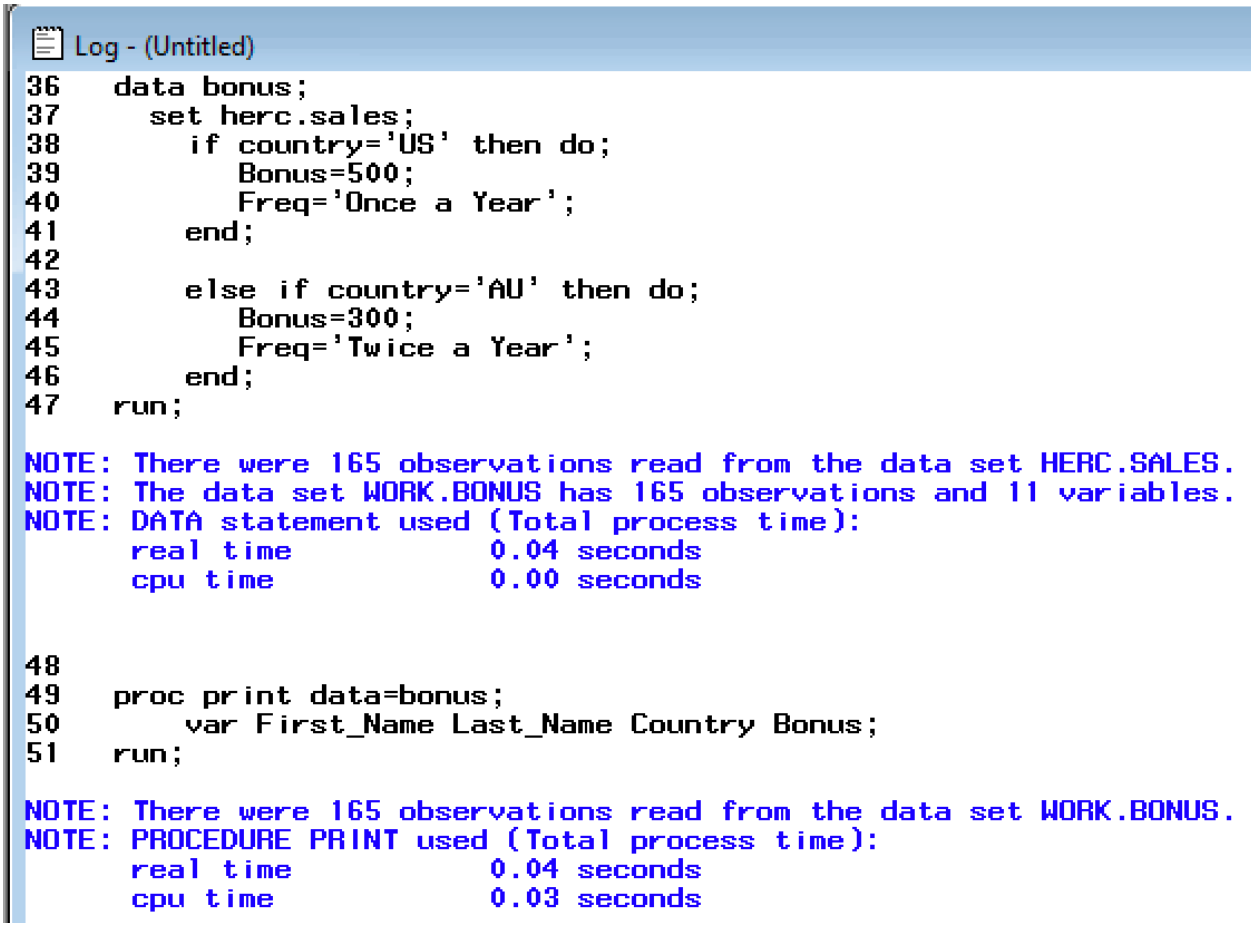
```
    var First_Name Last_Name Country Bonus;
run;
```

if then do print



56	Doungkamol	Simms	AU	300
57	Andrew	Conolly	AU	300
58	Koavea	Pa	AU	300
59	Franca	Kierce	AU	300
60	Billy	Plested	AU	300
61	Matsuoka	Wills	AU	300
62	Vino	George	AU	300
63	Meera	Body	AU	300
64	Harry	Highpoint	US	500
65	Julienne	Magolan	US	500
66	Scott	Desanctis	US	500
67	Cherda	Ridley	US	500
68	Priscilla	Farren	US	500
69	Robert	Stevens	US	500
70	Shawn	Fuller	US	500
71	Michael	Westlund	US	500
72	Barnaby	Cassey	US	500

if then do log



```
data bonus;
  set herc.sales;
    if country='US' then do;
       Bonus=500;
       Freq='Once a Year';
    end;

    else if country='AU' then do;
       Bonus=300;
       Freq='Twice a Year';
    end;
run;

NOTE: There were 165 observations read from the data set HERC.SALES.
NOTE: The data set WORK.BONUS has 165 observations and 11 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.00 seconds



proc print data=bonus;
    var First_Name Last_Name Country Bonus;
run;

NOTE: There were 165 observations read from the data set WORK.BONUS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.04 seconds
      cpu time            0.03 seconds
```

Constraints of IF-THEN/DO

If we need to run many of these statements at once, writing out many DO groups is time consuming.

Ex: Want to find annual and quarterly compounded interest for 20 years (80 quarters), with 100 total statements needed to compute this. Is there an easier way? (of course, or why would I ask, right?)

We can use a DO *loop* to take care of multiple DO statements.

General form of DO loops

```
DO index-variable = start TO stop <BY increments>;
Iterated SAS statements... ;
<OUTPUT>;
END;
```

`start`: specifies initial value of the index-variable
`stop`: specifies the ending value of the index-variable
`increment`: specifies a positive or a negative number to control the incrementing of the index-variable
`<output>`: an option to display all iterations of the index-variable

Executes statements between the DO and the END statements repetitively, based on the value of an index-variable

Start and stop logistics

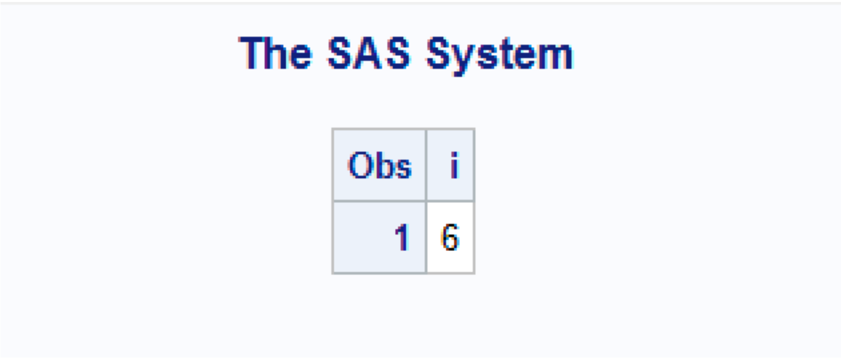
Start, stop and increment values: must be numbers or expressions that yield results, are established before executing the loop and if omitted, the increment defaults to 1
- When increment is positive, start must be the lower bound and stop must be the upper bound
- When increment is negative, start must be the upper bound and stop must be the lower bound

Index-variable details
- The index-variable is written, by default, to the output dataset
- At the termination of the loop, the value of index-variable is one increment beyond the stop value

Basic do loop forward I

```
data one;
do i=1 to 5;
end;
run;

proc print data=one;
run;
```

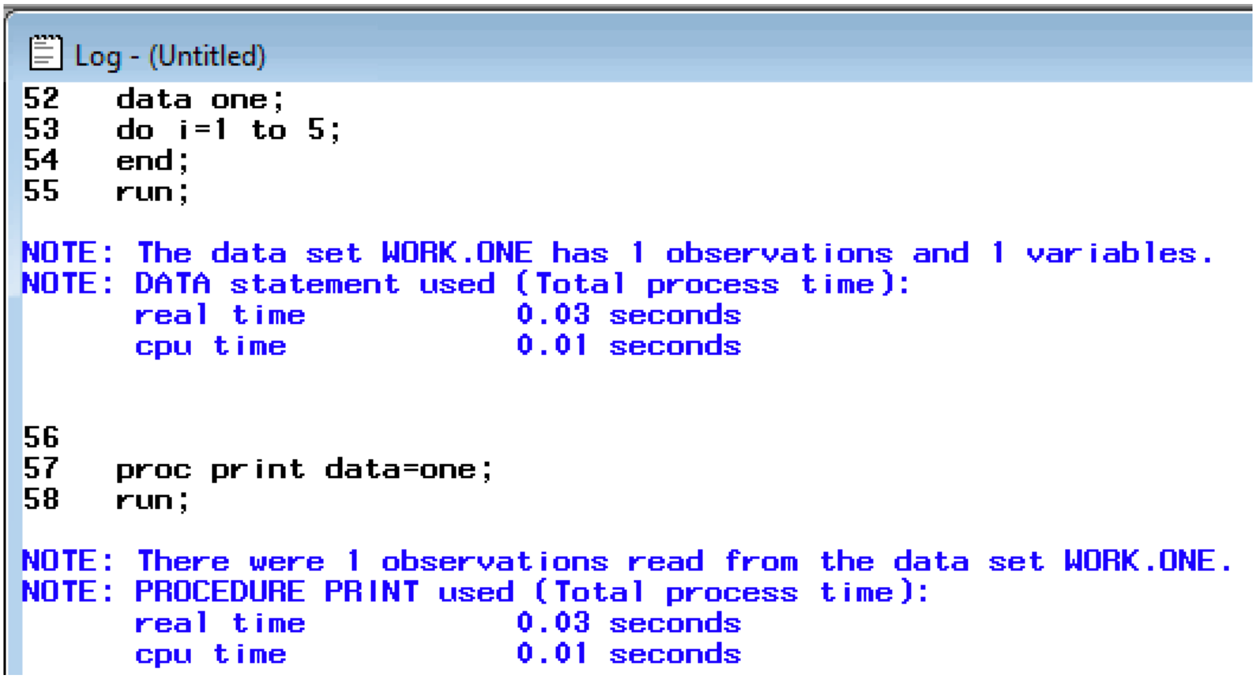
Basic do loop forward I print



The SAS System

Obs	i
1	6

Basic do loop forward I log

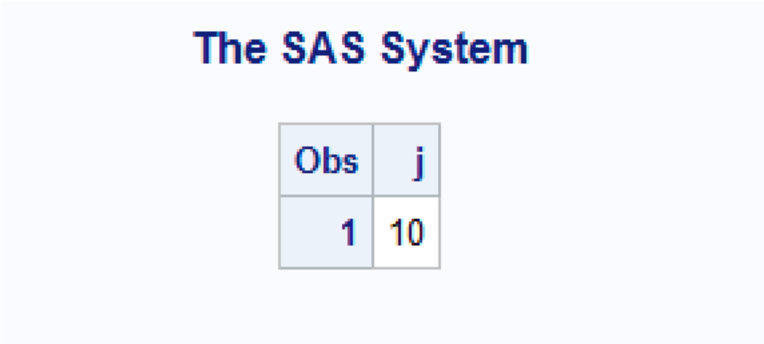


```
52    data one;
53    do i=1 to 5;
54    end;
55    run;

NOTE: The data set WORK.ONE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


56
57    proc print data=one;
58    run;

NOTE: There were 1 observations read from the data set WORK.ONE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

Basic do loop forward II

```
data two;
do j=2 to 8 by 2;
end;
run;

proc print data=two;
run;
```

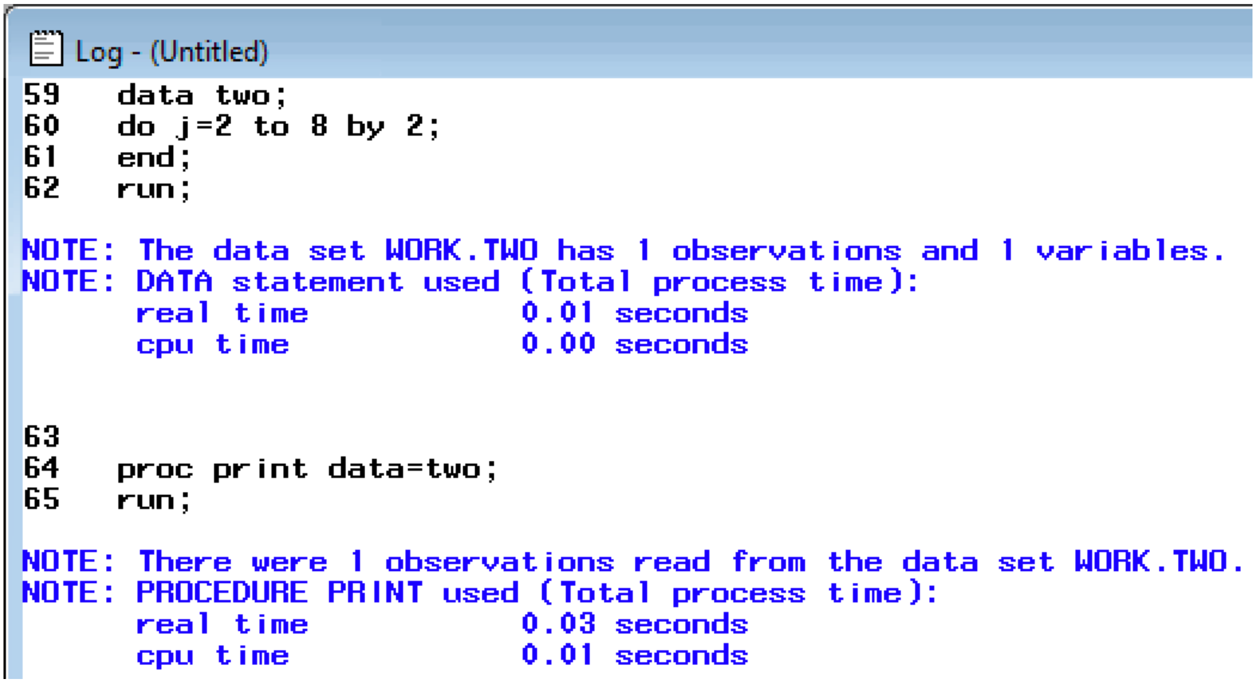
Basic do loop forward II print



The SAS System

Obs	j
1	10

Basic do loop forward II log



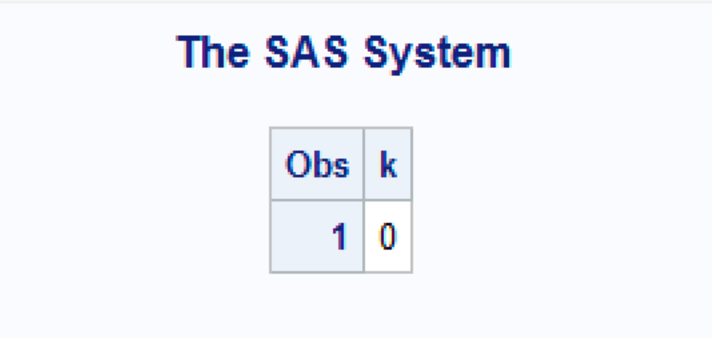
Log - (Untitled)

```
59     data two;
60     do j=2 to 8 by 2;
61     end;
62     run;

NOTE: The data set WORK.TWO has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


63
64     proc print data=two;
65     run;

NOTE: There were 1 observations read from the data set WORK.TWO.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

Basic do loop backward

```
data three;
do k=10 to 2 by -2;
end;
run;

proc print data=three;
run;
```

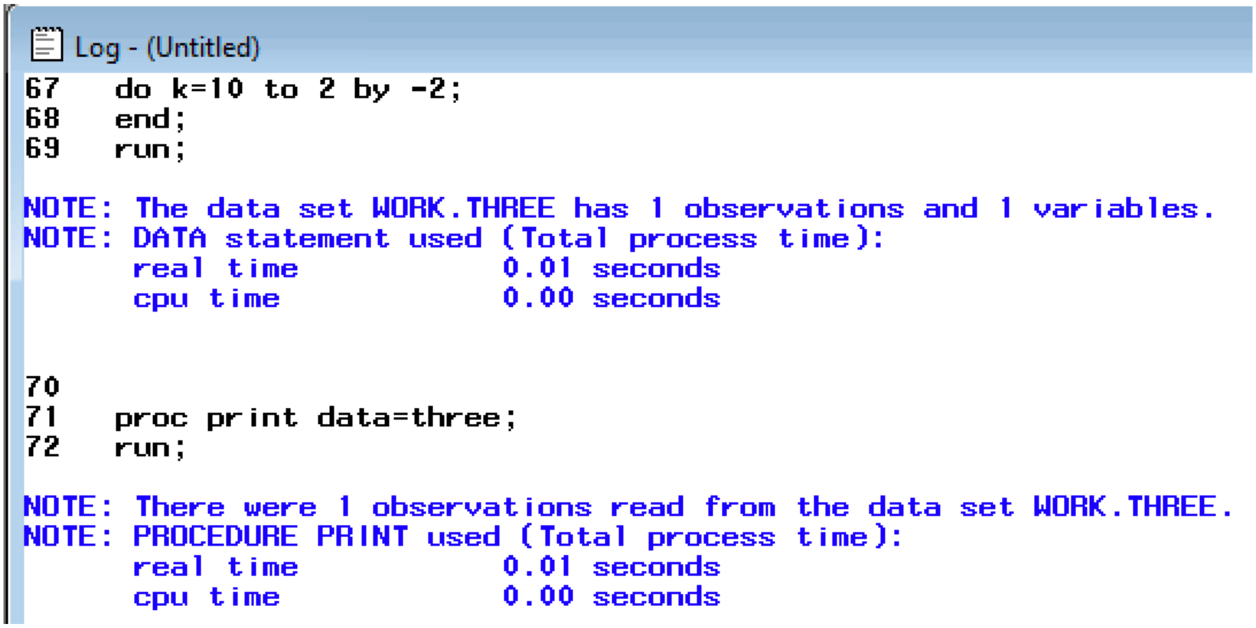
Basic do loop backward print



The SAS System

Obs	k
1	0

Basic do loop backward log



```
67     do k=10 to 2 by -2;
68     end;
69     run;

NOTE: The data set WORK.THREE has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


70
71     proc print data=three;
72     run;

NOTE: There were 1 observations read from the data set WORK.THREE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Writing loop for the Fibonacci sequence

First we need to identify the math equation we could use. So, the sequence starts at 1, then the second value is the sum of the two previous values, which is again 1. The third value of the sequence is the sum of the first two values (1+1=2), the fourth is the sum of the two preceding values (2+1=3), and so on.

There is a sum function and a lag function to use here. The sum function does exactly what it sounds like (summing a variable or specified list of numbers). Lag is similar, it will retrieve the last value.

Use sum to calculate the next value of the sequence to sum the current position's value and the previous value (with lag).

Let r_i be the i^{th} value of the Fibonacci sequence.

r_i of Fibonacci

$r_1 = 1, r_2 = 1, r_3 = 2, r_4 = 3, \ldots$

The sequence would look like:

$$r_{i+1} = r_i + r_{i-1}$$

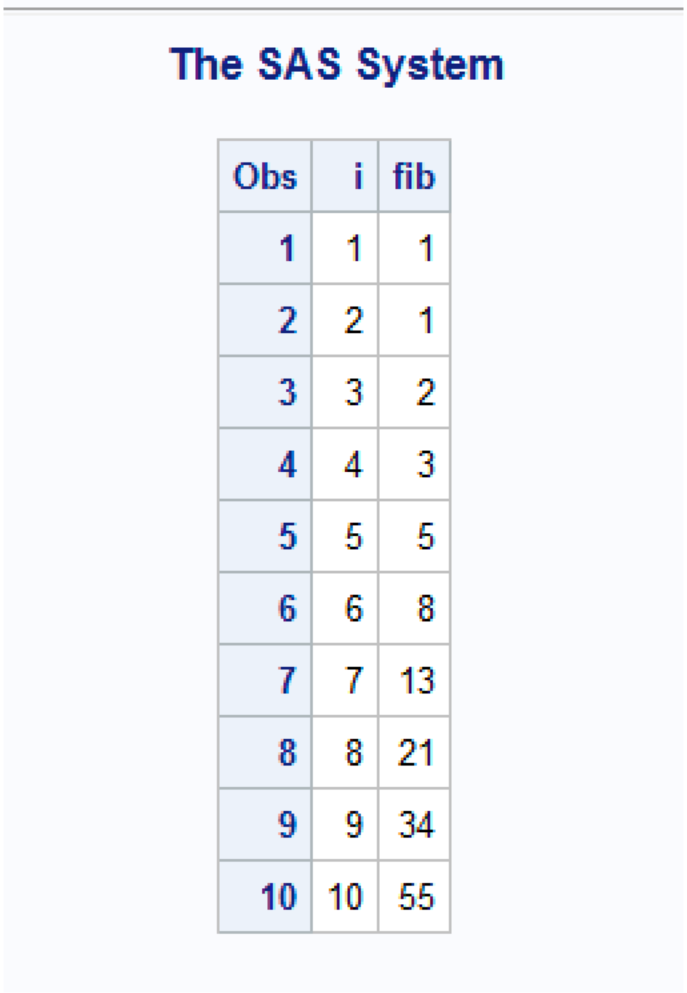
The next value of the sequence (r_{i+1}) is the sum of the two preceding values ($r_i + r_{i-1}$).

Fibonacci loop

```
data fseq;
do i = 1 to 10;
    fib = sum(fib, lag(fib));
    if i eq 1 then fib = 1;
    output;
    end;
run;
proc print data=fseq;
```

```
run;
```

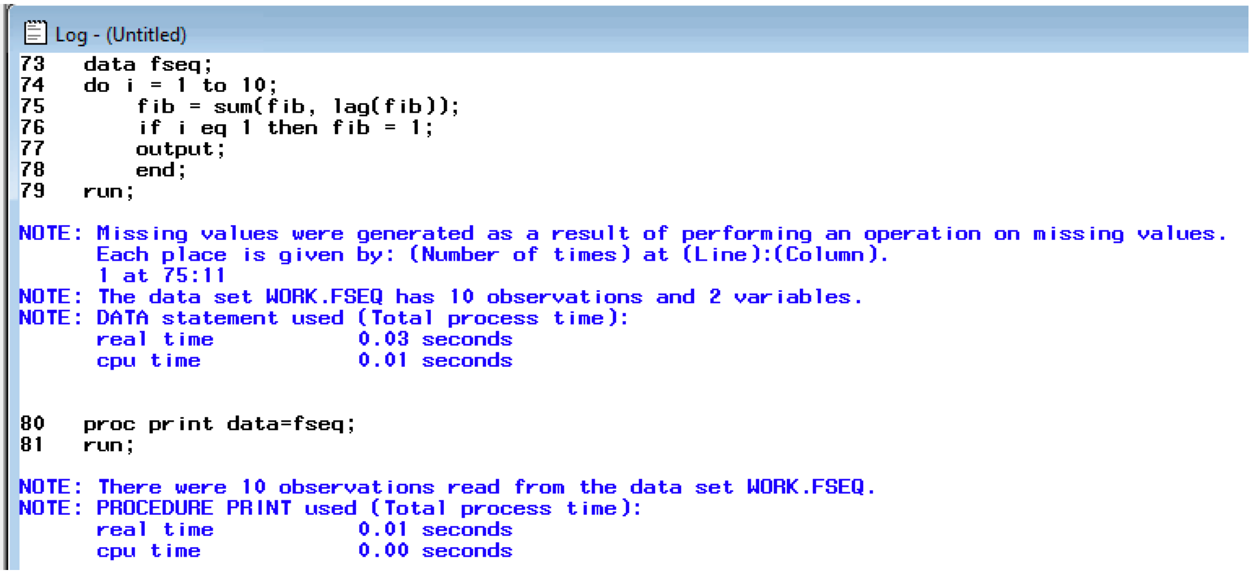
Fibonacci loop print



The SAS System

Obs	i	fib
1	1	1
2	2	1
3	3	2
4	4	3
5	5	5
6	6	8
7	7	13
8	8	21
9	9	34
10	10	55

Fibonacci loop log



Log - (Untitled)

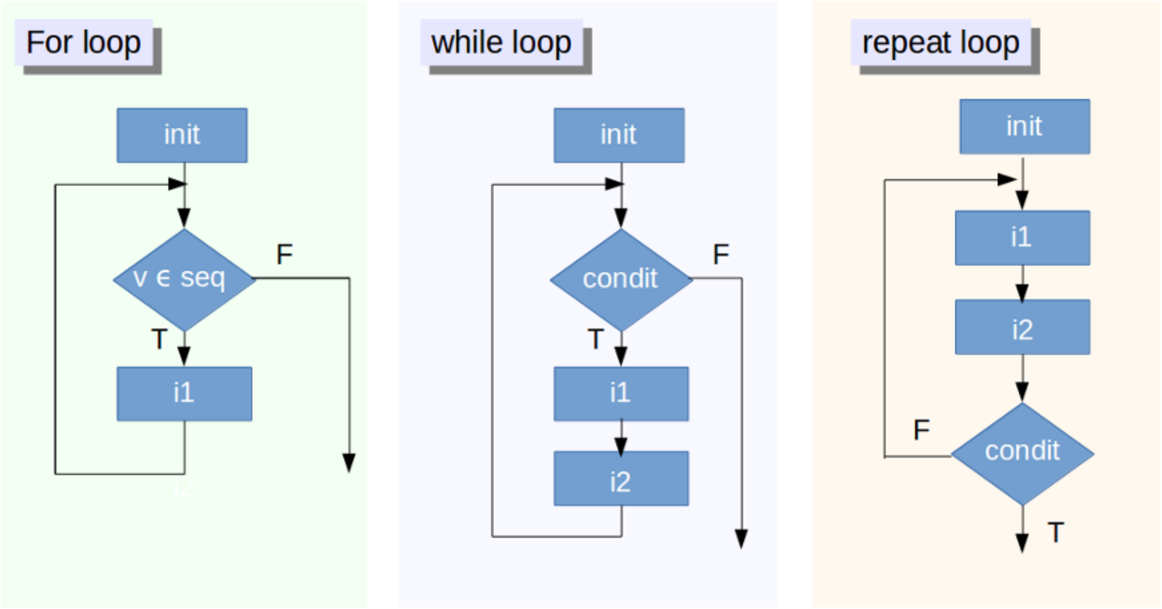
```
73   data fseq;
74   do i = 1 to 10;
75       fib = sum(fib, lag(fib));
76       if i eq 1 then fib = 1;
77       output;
78       end;
79   run;

NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 75:11
NOTE: The data set WORK.FSEQ has 10 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds


80   proc print data=fseq;
81   run;

NOTE: There were 10 observations read from the data set WORK.FSEQ.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Conditional iterative processing

You can use DO WHILE and DO UNTIL statements to stop the loop when a condition is met rather than when the loop is executed a specific number of times. To avoid infinite loops, be sure that the specified condition will be met.

Visualize loop process

The following diagram shows the logic and path the different loops take.



DO WHILE statement

DO WHILE executes statements in a DO loop repetitively *while* a condition is true

General form of DO WHILE

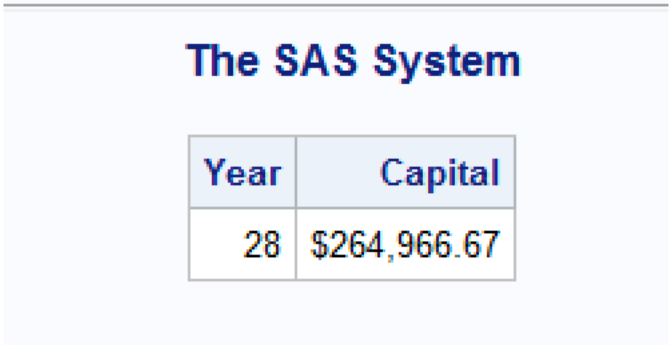
```
DO WHILE (expression);
  <additional SAS statements>;
END;
```

The value of expression is evaluated at the *top* of the loop and the statements in the loop never execute if expression is initially false

do while

```
data dowhile;
   do Year=1 to 30 while(Capital<=250000);
    Capital+5000;
     Capital+(Capital*.045);
   end;
run;

proc print data=dowhile noobs;
   format Capital dollar14.2;
run;
```

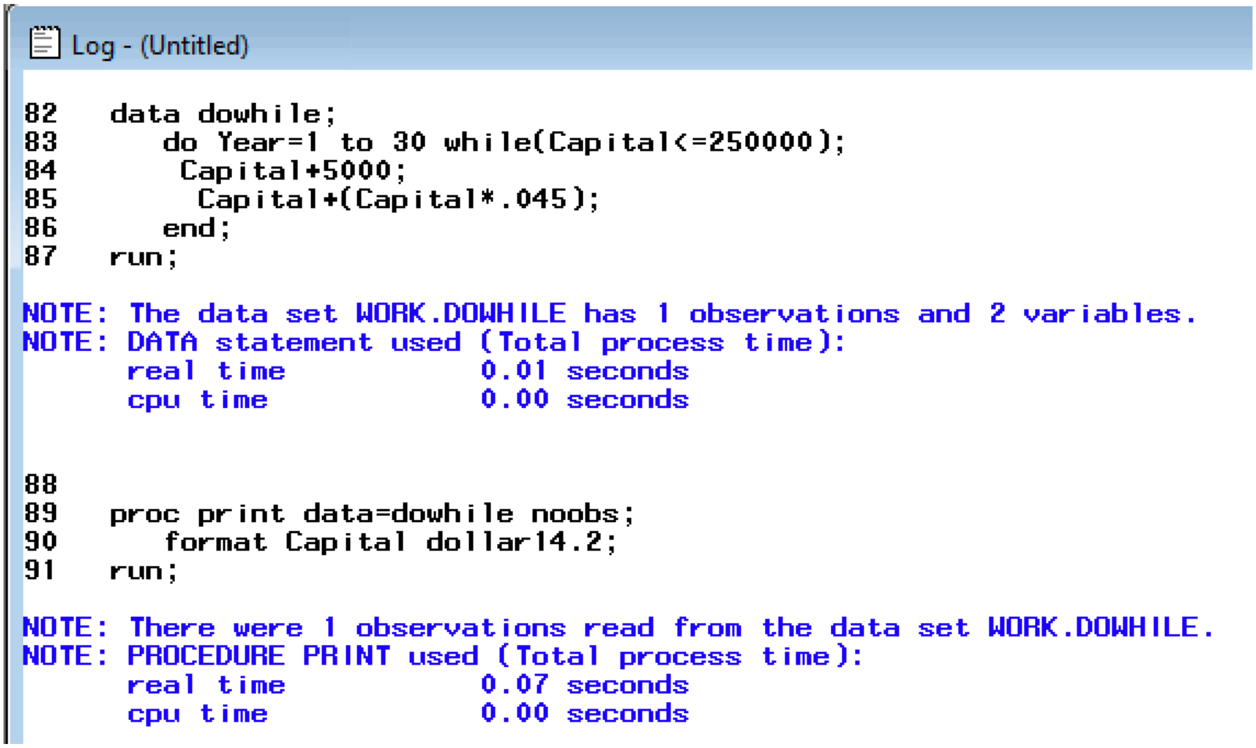
do while print



The SAS System

Year	Capital
28	$264,966.67

do while log



```
82    data dowhile;
83       do Year=1 to 30 while(Capital<=250000);
84        Capital+5000;
85         Capital+(Capital*.045);
86       end;
87    run;

NOTE: The data set WORK.DOWHILE has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


88
89    proc print data=dowhile noobs;
90       format Capital dollar14.2;
91    run;

NOTE: There were 1 observations read from the data set WORK.DOWHILE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.07 seconds
      cpu time            0.00 seconds
```

DO UNTIL statement

DO UNTIL executes statements in a DO loop repetitively *until* a condition is true. Once the condition is met, the loop is finished with its calculations

General form of DO UNTIL

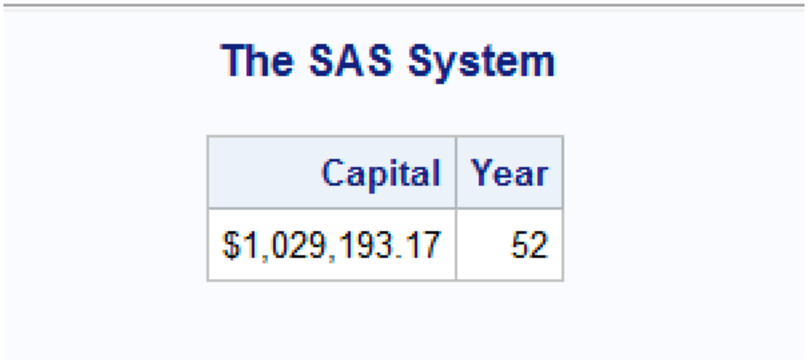
```
DO UNTIL (expression);
     <additional SAS statements>;
END;
```

The value of expression is evaluated at the *bottom* of the loop, the statements in the loop are executed at least one time, and though the condition is placed at the top of the statement, it is evaluated at the bottom of the loop

do until

```
data dountil;
  do until(Capital>1000000);
     Year+1;
     Capital+5000;
     Capital+(Capital*.045);
  end;
run;

proc print data=dountil noobs;
  format Capital dollar14.2;
run;
```

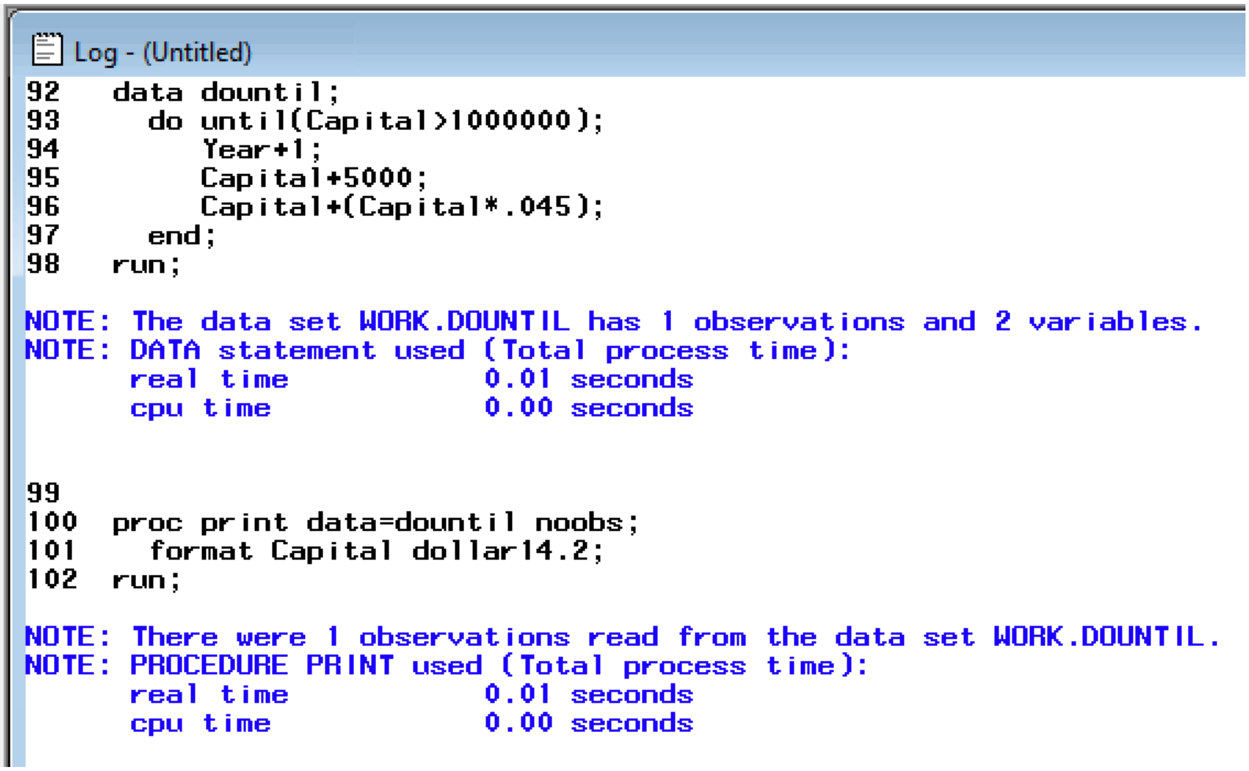
do until print



The SAS System

Capital	Year
$1,029,193.17	52

do until log



```
data dountil;
  do until(Capital>1000000);
     Year+1;
     Capital+5000;
     Capital+(Capital*.045);
  end;
run;

NOTE: The data set WORK.DOUNTIL has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds



proc print data=dountil noobs;
  format Capital dollar14.2;
run;

NOTE: There were 1 observations read from the data set WORK.DOUNTIL.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

Nesting loops

Nested DO loops: are DO loops within DO loops
- Be sure to use different index variables for each loop
- Each DO statement must have its own corresponding END statement
- The inner loop executes completely for each iteration of the outer loop

General form of nested DO loops

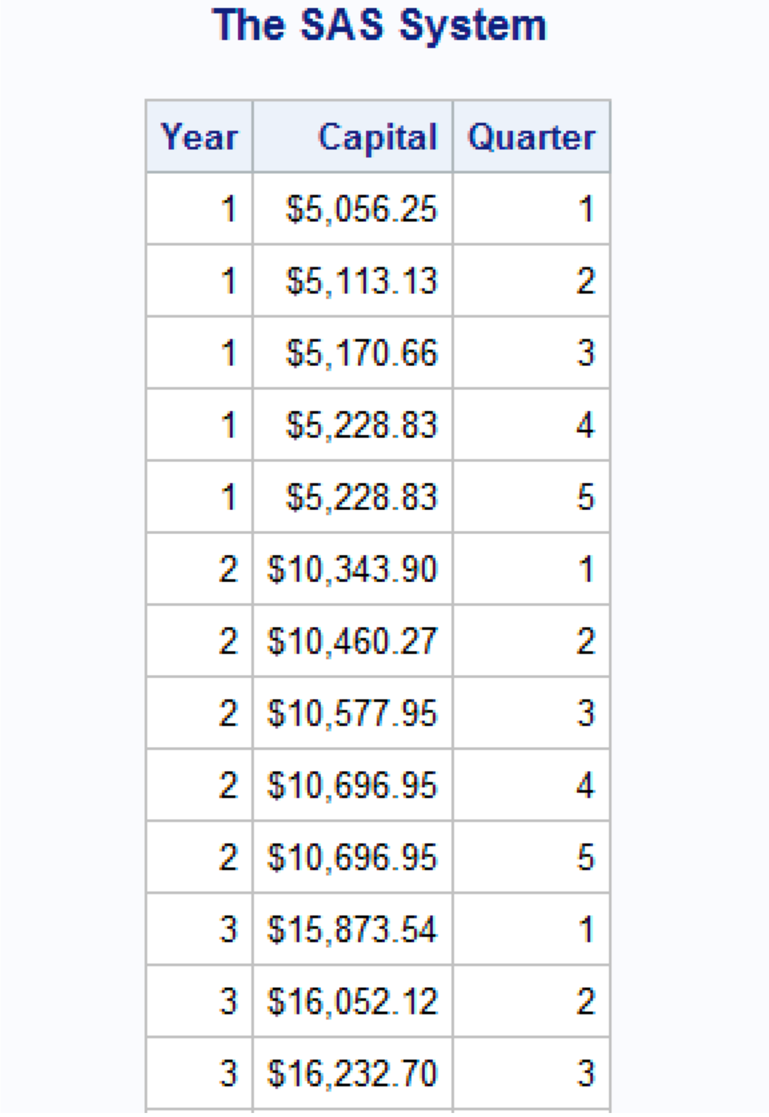
```
DO index-variable1=start TO stop
    <BY increment>;
      <OUTPUT>;
         DO index-variable2=start TO stop
         <BY increment>;
        <OUTPUT>;
         END;
END;
```

The OUTPUT statement can be used (and will be in our Fibonacci loop) to make sure we get a value output to the dataset for each iteration

nested do loop

```
data nested;
  do Year=1 to 5;
   Capital+5000;
   do Quarter=1 to 4;
    Capital+(Capital*(.045/4));
     output;
   end;
   output;
  end;
run;

proc print data=nested noobs;
   format Capital dollar14.2;
run;
```

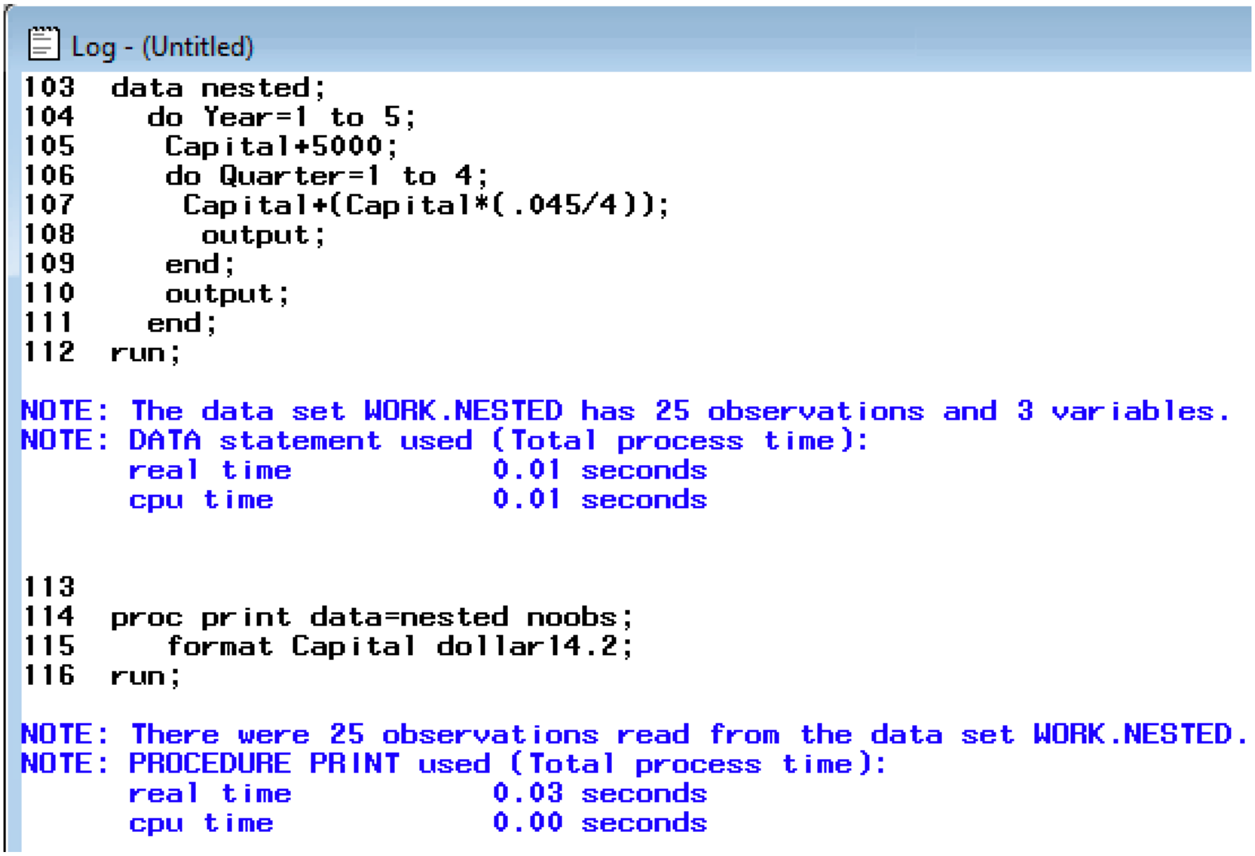
nested do loop print



The SAS System

Year	Capital	Quarter
1	$5,056.25	1
1	$5,113.13	2
1	$5,170.66	3
1	$5,228.83	4
1	$5,228.83	5
2	$10,343.90	1
2	$10,460.27	2
2	$10,577.95	3
2	$10,696.95	4
2	$10,696.95	5
3	$15,873.54	1
3	$16,052.12	2
3	$16,232.70	3

nested do loop log



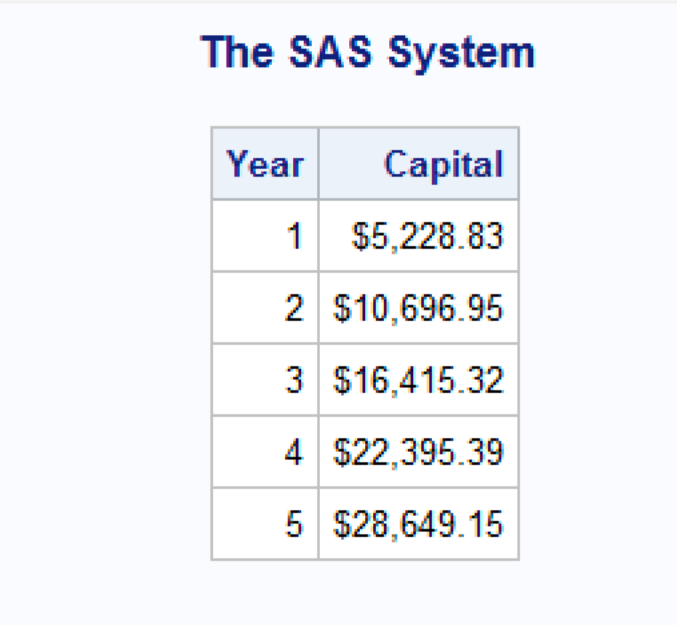
```
103   data nested;
104     do Year=1 to 5;
105      Capital+5000;
106      do Quarter=1 to 4;
107       Capital+(Capital*(.045/4));
108        output;
109      end;
110      output;
111     end;
112   run;

NOTE: The data set WORK.NESTED has 25 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


113
114   proc print data=nested noobs;
115      format Capital dollar14.2;
116   run;

NOTE: There were 25 observations read from the data set WORK.NESTED.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Alt nested

Alternative method not showing the quarter variable

```
data nested(drop=Quarter);
  do Year=1 to 5;
   Capital+5000;
   do Quarter=1 to 4;
    Capital+(Capital*(.045/4));
   end;
   output;
  end;
run;

proc print data=nested noobs;
   format Capital dollar14.2;
run;
```

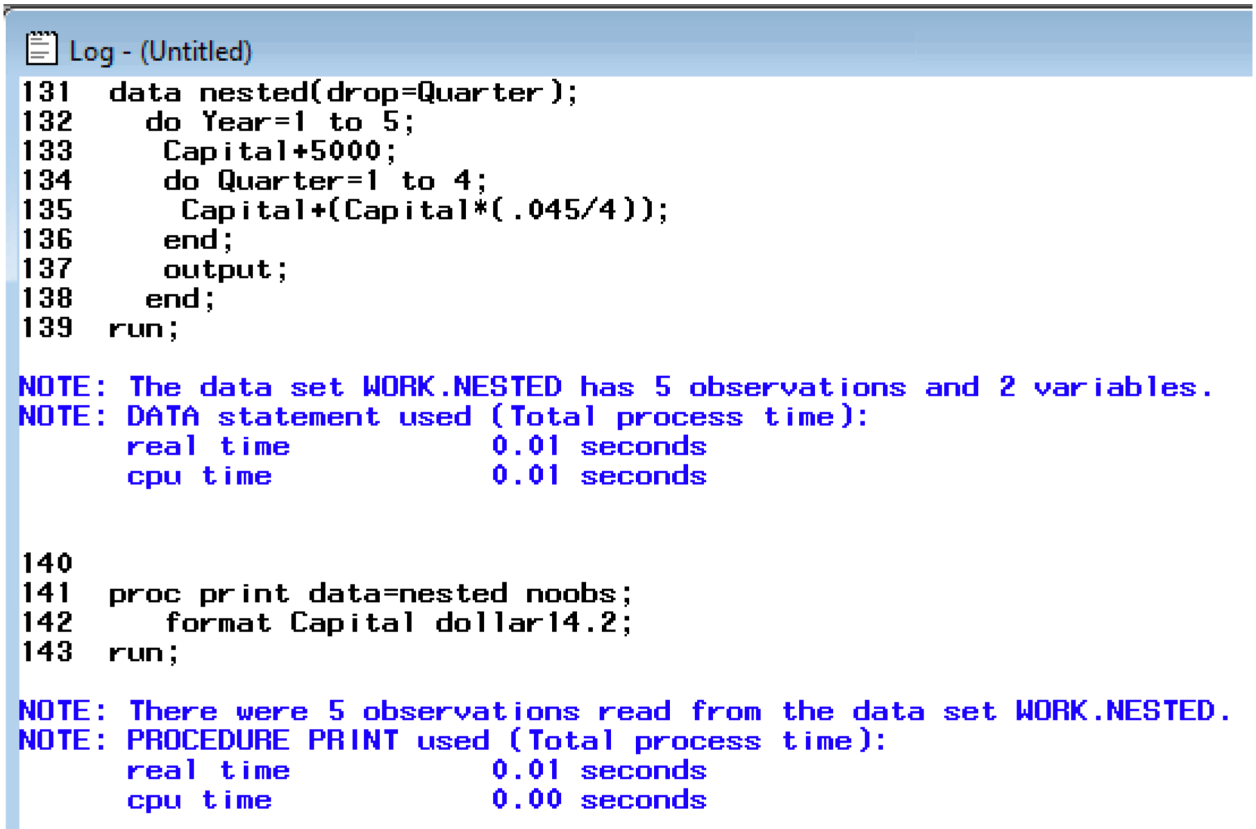
Alt nested print



The SAS System

Year	Capital
1	$5,228.83
2	$10,696.95
3	$16,415.32
4	$22,395.39
5	$28,649.15

Alt nested log



```
131   data nested(drop=Quarter);
132     do Year=1 to 5;
133       Capital+5000;
134       do Quarter=1 to 4;
135         Capital+(Capital*(.045/4));
136       end;
137       output;
138     end;
139   run;

NOTE: The data set WORK.NESTED has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


140
141   proc print data=nested noobs;
142      format Capital dollar14.2;
143   run;

NOTE: There were 5 observations read from the data set WORK.NESTED.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```