Learning outcomes:

1. DO loop processing
2. DO loop with WHILE|UNTIL clause
3. Using SAS Array
4. Debugging for logic errors (interactive interface)

SAS components learnt:

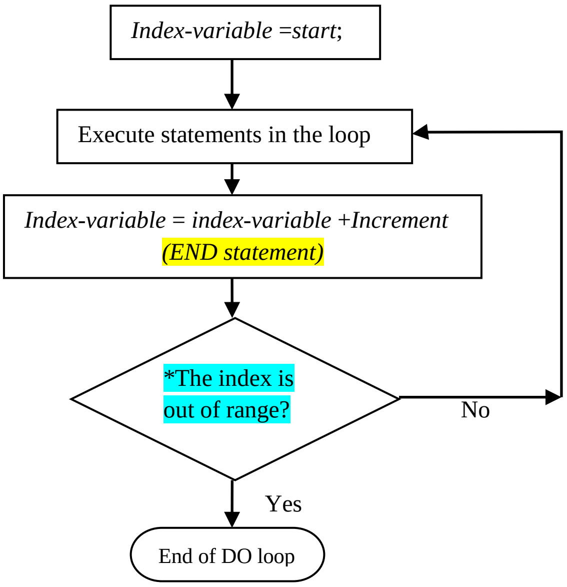
1. DO statement
2. DO WHILE statement
3. DO UNTIL statement
4. DO loop with conditional clause
5. NESTED DO loops
6. SAS Array
7. PUTLOG
8. END= *variable*
9. DEBUG option

9.1 DO loop processing

- Purpose for using DO loop:
  - Use Do loop to perform the repetitive calculations
  - To eliminate redundant code
- Various forms of iterative DO loops
  - The iterative DO statement
    - with integer as index variable
    - with an item list
  - Nested Do loops
  - Conditional iterative DO loops
    - DO WHILE statement
    - DO UNTIL statement
- The iterative DO statement
  - Syntax with integer as index variable

```
DO index-variable = start TO stop <BY increment>;
   <iterated SAS statements>
END;
```

    - *Start* specifies the initial value of the index variable.
    - *Stop* specifies the ending value of the index variable.
    - *Increment* specifies a POSITIVE or NEGATIVE number to control the incrementing of *index-variable*.
    - The values of start, stop and increment
      (a) must be numbers or expressions that yield numbers
      (b) are established before executing the loop
      (c) if omitted, increment defaults to 1
    - The index-variable is written to the output data set.
    - At the termination of the loop, the value of index-variable is one increment beyond the *stop* value.
  - DO Loop logic:


Index-variable =start;
Execute statements in the loop
Index-variable = index-variable +Increment
(END statement)
*The index is out of range?
No
Yes
End of DO loop

* When *Increment >0*, check *Index-variable > Stop?*
When *Increment <0*, check *Index-variable < Stop?*

Ex. 9.1: What are the values of the index variables after the executions of the following DO loops end?

(a)

```
do i=1 to 9 by 3;
end;
```

(b)

```
do j=1 to 10;
end;
```

(c)

```
do k=6 to -1 by -5;
end;
```

Ans:

- Syntax for the DO statement with an item list

```
DO index-variable = item-1 <,… item-n>;
   <iterated SAS statements>
END;
```

- Items are character constant, numeric constants or variables.
- The DO loops is executed once for each item in the list
- The list must be comma separated.

Ex. 9.2: What are the values of the index variables after the executions of the following DO loops end?

(a)

```
do month = 'March', 'June', 'Decemeber';
end;
```

(b)

```
do m5=5,10,15,20;
end;
```

Ans:

Ex. 9.3:(a) Write a DATA step to find out the answers for Ex.9.1 and Ex.9.2.

(b) Modify the program to add one extra variable for each DO-loop for recording the number of times of the loop has been run.

Use of SUM statements. The final output of Work.abc for the variables used in the SUM statements:

i1_times	i2_times	k_times	month_times	m5_times

- An example - **Planned Saving in the Rich Bank**
  Suppose the fixed annual interest rate is 1.5% for the 6-month-maturity planned saving. If the monthly deposit amount is $1000, how much principal and interest will be got on 6-month-maturity? Write a SAS code for the total saving after 6 months.
  - Mathematically:
    (a) monthly interest = total saving *1.5%/12
    (b) monthly deposit = 1000
    (c) total saving per month = monthly deposit + saving of last month + monthly interest
    (d) Number of months = 6

```
*p9_planned_saving.sas;
data PI6;
do i = 1 to 6;
deposit +1000;
deposit+(deposit*1.5/100/12);
end;
run;
options nocenter;
proc print data=PI6;
format deposit Dollar7.2;
run;
```

- The output from PROC PRINT:

```
Obs    i     deposit

 1     7     6026.30
```

- Deposit=6026.30 after 6 runs.
- Program Data Vector (PDV) processing for partial run:

Stage	Variables			
Compilation		**PDV**		
	i	deposit	_N_	
		Retain	Drop	
Execution		**PDV**		
Iteration	i	deposit	_N_	
1: DATA Statement, initialize PDV	.	0	1	
1: DO statement	1	0	1	
1: Deposit +1000	1	1000	1	
1: deposit+(deposit*1.5/100/12);	1	1001.25	1	
1:end;	2	1001.25	1	
1: DO statement	2	1001.25	1	*i* out of range?
1: Deposit +1000	2	2001.25	1	
1: deposit+(deposit*1.5/100/12);	2	2003.752	1	
1:end;	3	2003.752	1	
1: DO statement	3	2003.752	1	*i* out of range?
1: Deposit +1000	3	3003.752	1	
1: deposit+(deposit*1.5/100/12);	3	3007.506	1	
1:end;	4	3007.506	1	
1: DO statement	4	3007.506	1	*i* out of range?
1: Deposit +1000	4	4007.506	1	
1: deposit+(deposit*1.5/100/12);	4	4012.516	1	
1:end;	5	4012.516	1	
1: DO statement	5	4012.516	1	*i* out of range?
1: Deposit +1000	5	5012.516	1	
1: deposit+(deposit*1.5/100/12);	5	5018.781	1	
1:end;	6	5018.781	1	
1: DO statement	6	5018.781	1	*i* out of range?
1: Deposit +1000	6	6018.781	1	
1: deposit+(deposit*1.5/100/12);	6	6026.305	1	
1:end; Index out of range!	7	6026.305	1	
1:run;	7	6026.305		Output!
2: DATA Statement, re-initialize PDV	*.*	*6026.305*	*2*	
2: DO statement	*1*	*6026.305*	*2*	

The second iteration will only be carried out when the input-data-set has more than one observation! In this example, no input SAS dataset is given in the DATA step and therefore, the last two iterations shown in above table is not required for this example.

Ex. 9.4: (a) Modify the SAS program for the planned saving scheme so that the month and the saving for each month are listed.

```
     i     saving

     1     $1001.25
     2     $2003.75
     3     $3007.51
     4     $4012.52
     5     $5018.78
     6     $6026.30
```

(b) Suppose the planned saving scheme starts from March. Modify the SAS program for the planned saving scheme so that the month name can be listed.

Answers:
(a)

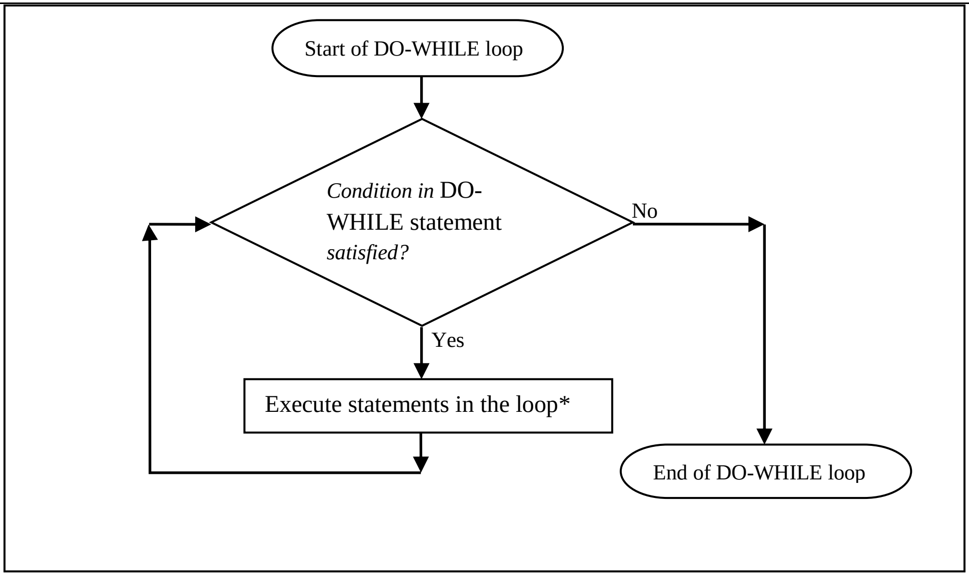
(b)

Conditional iterative DO loops

- Syntax of the DO WHILE statement

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

- The DO WHILE statement executes statements in a DO loop repetitively while a condition (expression) is true.
- DO WHILE loop logic:

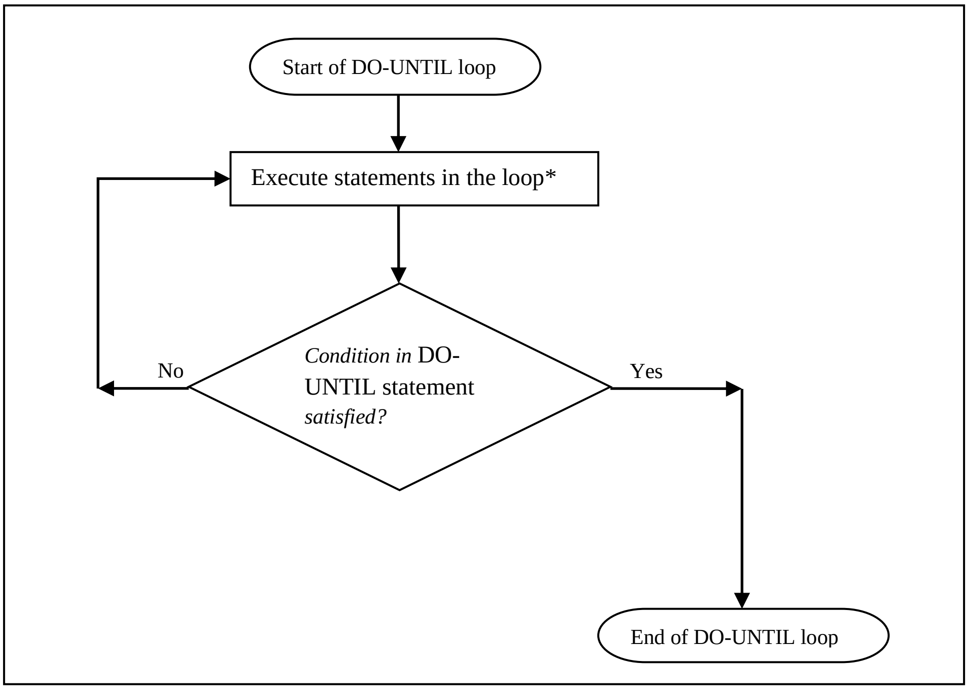

Start of DO-WHILE loop
Condition in DO-WHILE statement satisfied?
No
Yes
Execute statements in the loop*
End of DO-WHILE loop

- The value of expression is evaluated at the top of the loop.
- When the value of expression is initially false, the statements in the loop would never be executed.
- *Normally include statement for updating of expression in DO-WHILE statement. Make sure the condition specified would be met or there will be an INFINITE LOOP! (no ending of the program run!)

- Syntax of the DO UNTIL statement

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

- The DO UNTIL statement executes statements in a DO loop repetitively until a condition (i.e. expression) is true.
- The DO UNTIL loop logic:


Start of DO-UNTIL loop
Execute statements in the loop*
No
Condition in DO-UNTIL statement satisfied?
Yes
End of DO-UNTIL loop

- The value of expression is evaluated at the bottom of the loop. It can be called as stopping criterion.
- The statements in the loop are executed at least once
- Statements for update the value of expression are required to avoid INFINITE LOOP!

- Planned saving scheme in the Rich Bank revisit:
  - Rewrite the SAS program using DO-WHILE statement and DO-UNTIL statement

```
* p9_do_while.sas;
data PI6;
do while (i <6);
  i+1;
  deposit+1000;
  deposit +(deposit*1.5/100/12);
  output;
end;
run;

proc print data=PI6 label noobs;
format deposit Dollar8.2;
label deposit='saving';
run;
```

```
* p9_do_until.sas;
data PI6;
do until ( i >=6);
  i+1;
  deposit+1000;
  deposit +(deposit*1.5/100/12);
  output;
end;
run;

proc print data=PI6 label noobs;
format deposit Dollar8.2;
label deposit='saving';
run;
```

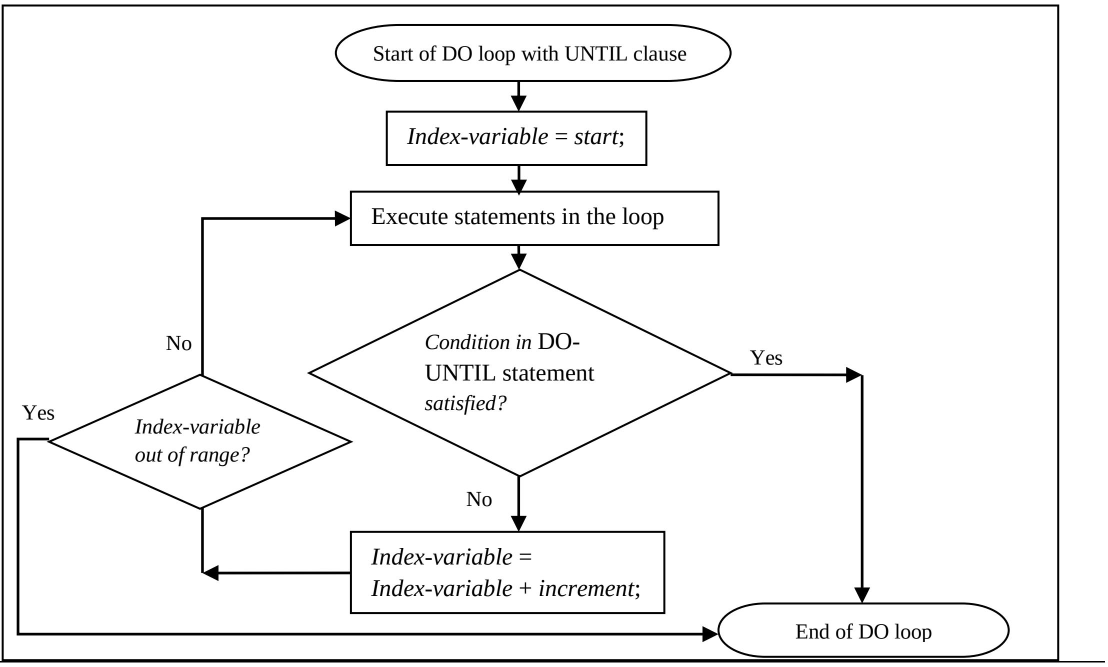
  - Compare the conditions: The expression used in DO-WHILE statement is the **negation** of the expression used in DO-UNTIL statement.
  - Both produce the same results.

Iterative DO Loops with a conditional clause

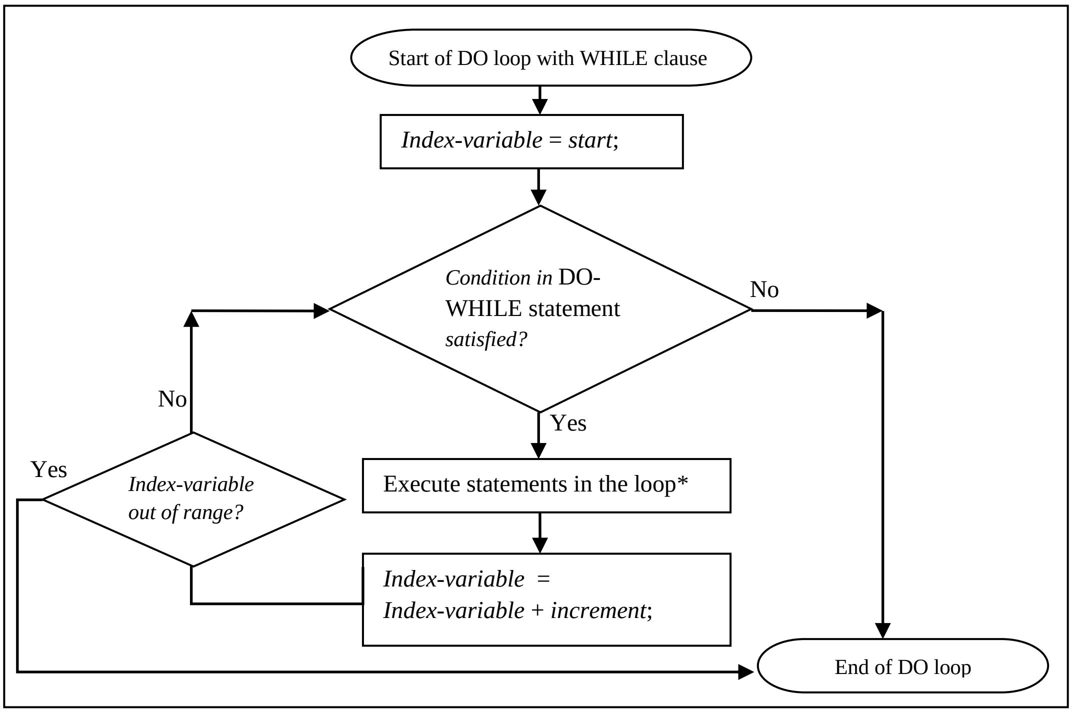
- Combine DO WHILE statement and DO UNTIL statement with the iterative DO statement.
- Syntax

```
DO index-variable = start TO stop <BY increment>
   WHILE | UNTIL (expression);
   <additional SAS statements>
END;
```

- This is one method of avoiding an infinite loop in a DO WHILE or DO UNTIL statements.
- Logic for DO loop with UNTIL clause:


Start of DO loop with UNTIL clause
Index-variable = start;
Execute statements in the loop
Condition in DO-UNTIL statement satisfied?
Yes
No
Index-variable = Index-variable + increment;
Index-variable out of range?
No
Yes
End of DO loop

  - In a DO UNTIL loop, the condition is checked before the index variable is incremented.
- Logic for DO loop with WHILE clause:


Start of DO loop with WHILE clause
Index-variable = start;
Condition in DO-WHILE statement *satisfied?*
No
Yes
Execute statements in the loop*
Index-variable = Index-variable + increment;
No
Yes
Index-variable out of range?
End of DO loop

  - In a DO WHILE loop, the condition is checked after the index variable is incremented.
- Planned saving scheme in the Rich Bank revisit:
  - Suppose David would like to use the scheme to save $10,000. How many months is required to achieve his target under the same interest rate 1.5% per annual and the monthly deposit is $1000. Use DO WHILE and DO UNTIL statements:

```
*p9_david_problem.sas;
* do with while clause;
data PI_david;
do i = 1 to 20 while (deposit
<10000);
deposit +1000;
deposit+(deposit*1.5/100/12);
output;
end;
run;
```

```
* do with until clause;
data PI_david;
do i = 1 to 20 until (deposit
>=10000);
deposit +1000;
deposit+(deposit*1.5/100/12);
output;
end;
run;
```

NESTED DO loops

- Syntax

```
DO index-variable-1 = start-1 TO stop-1 <BY increment-1>;
  DO index-variable-2 = start-2 TO stop-2 <BY increment-2>
     <additional SAS statements>
  END;
END;
```

- Nested DO loops are loops within loops. The above syntax shows only two levels nested DO loops but it can be up to *n*-levels (*n* > 1).
- The index variables are different for each loop.

- Each one inner DO loop must have a corresponding END statement so that the inner DO loop is complete. The inner loop executes completely for each iteration of the outer loop.
- An example on planned saving scheme in the Rich Bank for David and Mary. David and Mary would like to join the planned saving scheme for 2 and 3 years respectively. As before, David deposit $1000 monthly while Mary would like to deposit $1200 per month. The planned saving scheme information are stored in worksheet 'Customers' of EXCEL file 'planned_saving.xlsx'. How much saving will David and Mary have yearly?

```
*p9_nested_do_loop.sas;
Libname xlsdata 'd:\SAS_datasets\planned_saving.xlsx';
data PI3yr (drop=month);
   set xlsdata.'Customers$'n;
   saving =0;
do year=1 to no_of_year;
    do month = 1 to 12;
       saving +Monthly_deposit;
       saving+(saving*1.5/100/12);
    end;
    output;
end;
run;

proc print data=PI3yr label;
var name no_of_year monthly_deposit year saving;
label no_of_year =' No. of years'
        year = 'After year '
        monthly_deposit='Monthly deposit'
        saving='Total saving';
format deposit saving Dollar10.3;
run;
```

- Note that the SAVING variable must be initially set to zero for each customer.
- OUTPUT statement is used to list the yearly saving records.
- The output:

```
                     No. of    Monthly    After          Total
Obs    Name          years     deposit     year          saving

 1     David           2         1000        1      $12097.948
 2     David           2         1000        2      $24378.619
 3     Mary            3         1200        1      $14517.538
 4     Mary            3         1200        2      $29254.342
 5     Mary            3         1200        3      $44213.725
```

9.2 SAS Array processing

- SAS Array is used to simplify the programs which involve
  - repetitive calculations
  - use of many variables with the same attributes (can be viewed by PROC CONTENTS)
  - variables comparison
  - a table lookup
- A SAS array
  - a temporary grouping of SAS variables which are arranged in a particular order
  - identified by an array name
  - must contain ALL numeric or ALL character variables
  - exist only for the duration of the current DATA step, i.e. Once the DATA step is ended, the SAS array does not exist.
  - a reference to the memory for processing, NOT a variable
- Declaration
  - ARRAY statement

```
ARRAY array-name {subscript} <$> <length> <array-elements>;
```

- {*subscript*} represents the number of elements. Brackets [ ] and parentheses ( ) are also allowed to enclose the *subscript*.
- $ indicates the array is character type, i.e. all elements in the array are character. It is not necessary if the elements in the array were previously defined as character elements
- *length* specifies the length of elements in the array which were no previously assigned a length
- *array-elements* refer to the names of elements (variable).
- **Compile time only** statement

- Defining an ARRAY with similar element names

- Declare the array M:

```
ARRAY M {12} Mon1 Mon2 Mon3 Mon4 Mon5 Mon6 Mon7 Mon8 Mon9
Mon10 Mon11 Mon12;
```

	M											
	M{1}	M{2}	M{3}	M{4}	M{5}	M{6}	M{7}	M{8}	M{9}	M{10}	M{11}	M{12}
PDV												
Name	Mon1	Mon2	Mon3	Mon4	Mon5	Mon6	Mon7	Mon8	Mon9	Mon10	Mon11	Mon12

(a) The name 'M' is only a name, NOT variable but can be used in the way as variable for programming.
(b) Mon1 to Mon12 are called elements of array M.

- Other ways for the same array M declaration:
  (a) use of '*'

```
ARRAY M {*} Mon1 Mon2 Mon3 Mon4 Mon5 Mon6 Mon7
Mon8 Mon9 Mon10 Mon11 Mon12;
```

  (b) Use of '-'

```
ARRAY M {*} Mon1-Mon12;
```

  (c) Use of ':'

```
ARRAY M {*} Mon:;
```

- One can define the array with lower bound index not from 1. For example:

```
ARRAY Y {6:12} Mon6-Mon12;
```

- Defining an ARRAY with different element names

- In the PDV, FHY is an array representing the saving of the first half year.

	FHY							
	FHY{1}	FHY{2}		FHY{4}	FHY{5}	FHY{3}	FHY{6}	
PDV								
Name	Mon1	Feb	~~Total~~	Mon4	May	Mar	Mon6	~~Mon8~~

- The declaration:

```
ARRAY FHY {*} Mon1 Feb Mar Mon4 May Mon6;
```

- Note that an array does not need:

(a) to have similar, related or numbered names.
(b) to be stored sequentially.
(c) to be adjacent.

- Using ARRAY in a DO loop
  - To reference an element, the index-variable is often used as a subscript.
  - It is often that the DO loop form is:

```
DO index-variable = 1 TO number-of-elements-in-array;
   <additional SAS statements>
END;
```

  - In the most of the case the index of the first array element is 1. If not, LBOUND( ) and HBOUND( ) is used for finding out the smallest index and the highest index for the element. For example:

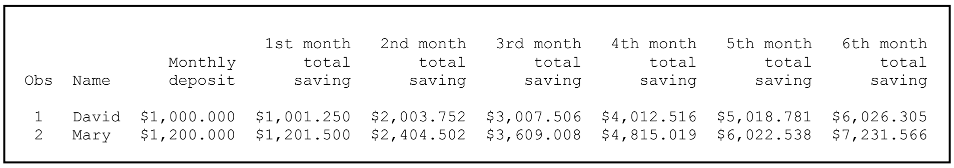
```
DO index-variable = LBOUND(array-name) TO HBOUND(array-name);
   <additional SAS statements>
END;
```

  - Planned saving scheme for David and Mary revisit:
    - David and Mary would like to see the first six months saving under the planned saving scheme. Write a SAS program to store the six months saving into a data set.

```
*p9_planned_saving_array.sas;
Libname xlsdata 'd:\SAS_datasets\planned_saving.xlsx';
data Planned_Saving;
array savm {6} sav1-sav6;
  set xlsdata.'Customers$'n;
  saving=0;
  do i = 1 to 6;
   saving +monthly_deposit;
   saving+(saving*1.5/100/12);
   savm{i}=saving;
end;
run;

proc print data=Planned_saving label;
var Name monthly_deposit sav1-sav6;
format monthly_deposit sav1-sav6 Dollar10.3;
label monthly_deposit = ' Monthly deposit'
             sav1='1st month total saving'
             sav2='2nd month total saving'
             sav3='3rd month total saving'
             sav4='4th month total saving'
             sav5='5th month total saving'
             sav6='6th month total saving';
run;
```

    - The output:



```
                                1st month   2nd month   3rd month   4th month   5th month   6th month
                     Monthly      total       total       total       total       total       total
Obs   Name           deposit      saving      saving      saving      saving      saving      saving

 1    David       $1,000.000  $1,001.250  $2,003.752  $3,007.506  $4,012.516  $5,018.781  $6,026.305
 2    Mary        $1,200.000  $1,201.500  $2,404.502  $3,609.008  $4,815.019  $6,022.538  $7,231.566
```

- DIM function

```
DIM(array-name)
```

  - The dimension function gives the number of elements in the array.

- It can be used in a SAS program, as an expression.

- Using an array
  - as a function argument
    - For example,

```
ARRAY Quartersum{4} Qtr1- Qtr4;
…
TOT1=sum(Qtr1-Qtr4);
TOT2=sum(of Quartersum{*});
```

    - The above portion is extracted from a DATA step.
    - Note that TOT1=TOT2.
    - The array is passed as if it were a variable list.
  - Create numeric variable
    - Syntax

```
ARRAY array-name{subscript} <array-elements>;
```

    - For example,

```
ARRAY Qtr{4} Quarter1-Quarter4;
ARRAY Sales{4};
```

(a) No '$' symbol is used for numeric variables declaration
(b) If the *array-elements* do not exist, SAS will create them and use them in the PDV. In the above example, if *Quarter1* to *Quarter4* do not exist, SAS will create the variables and use them in the PDV.
(c) If the names of *array-elements* are not supplied, SAS will create variables names as *array-name-1*,*array-name-2*,…,*array-name-n*. In the above examples, SAS will create *Sales1*, *Sales2*, *Sales3* and *Sales4* as variables and use them in the PDV.

PDV

Sales1 N 8	Sales2 N 8	Sales3 N 8	Sales4 N 8

  - Create character variable
    - Syntax

```
ARRAY array-name{subscript} $ <length>;
```

    - For example,

```
ARRAY Month{12} $10;
```

(a) '$' is used to represent the character array
(b) In the PDV, variables *Month1*, *Month2*, … , *Month12* are created.

PDV

Month1 $ 10	Month2 $ 10	…..	Month10 $ 10	Month11 $ 10	Month12 $ 10

- Assigning initial values to an array
  - Syntax

```
ARRAY array-name {subscript} <$> <length> <(initial-value-list)>;
```

  - The initial values of an array can be assigned at the array declaration. The values are separated by spaces or commas in the list.
- Create a temporary lookup table
  - Syntax

```
ARRAY array-name {subscript} _temporary_ <(initial-value-list)>;
```

  - For example,

```
ARRAY New_Target{4} _temporary_ (10,10,14,19);
```

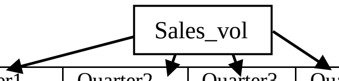
- for example, in BMW car sales volume:
  - The quarterly sales volumes of each salesperson are recorded. The BMW company has a target sales volumes for the salesperson which are 10, 10, 10 and 15 cars in first, second, third and fourth quarter. The manager would like to evaluate their performance by calculating the difference between two consecutive quarterly sales volumes and the difference between the quarterly sales volumes and quarterly company target volumes.
  - SAS codes:

```
*p9_sales_vol.sas;
Data Salesvol;
   length name $15;
    infile 'd:\SAS_datasets\sales_vol.csv' dlm=',';
    input name Quarter1-Quarter4;
   array Sales_vol{*} Quarter:;
   array Diff{3};
   array SalesMinTarget{4} _temporary_ (10,10,10,15);
   array Diff_Target{4};
   do i=1 to dim(Sales_vol)-1;
     Diff{i}=Sales_vol{i+1}-Sales_vol{i};
   end;
   do i =1 to dim(Sales_vol);
     Diff_Target{i}=Sales_vol{i} - SalesMinTarget{i};
   end;
   Drop i;
run;
options ls=100;
proc print data=SalesVol;
var name Quarter1-Quarter4 Diff1-Diff3 Diff_target1-Diff_target4;
format Diff1-Diff3 Diff_target1-Diff_target4 Quarter1-Quarter4 3.;
run;
```

(a) See SAS list for Quarter1-Quarter4 in the INPUT statement and Quarter: in the next ARRAY statement.
(b) SAS will create Diff1, Diff2, Diff3 for the array of Diff{3}.
(c) Use a temporary array SalesMinTarget{4} for the company quarterly target.
(d) The index variable is dropped before outputting to SAS data set.
(e) When DO loop is executed, SAS treats the ARRAY as variables for computation.

- Program Data Vector Processing

(a) Compilation stage



PDV

Name	Quarter1	Quarter2	Quarter3	Quarter4	Diff1	Diff2	Diff3

SalesMin Target1	SalesMin Target2	SalesMin Target3	SalesMin Target4	Diff_Target1	Diff_Target2	Diff_Target3	Diff_Target4
Drop* & Retain	Drop & Retain	Drop & Retain	Drop & Retain				

i
Drop

*If the array "SalesMinTarget{1:4}" is not temporary, only "Retain" flag are found in PDV.

(b) Execution stage (1st iteration only)

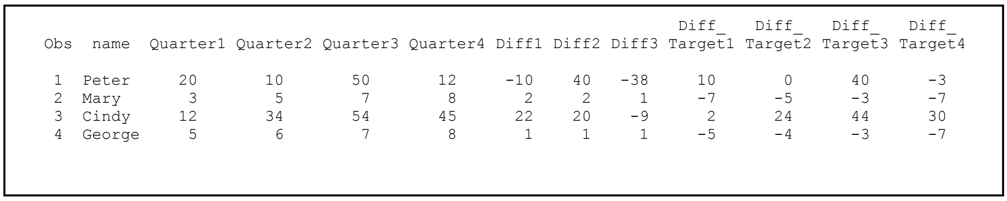
(i) During initialization: All variables except SalesMinTarget{1:4} are set to missing and the values of the array SalesMinTarget:

SalesMinTarget1	SalesMinTarget2	SalesMinTarget3	SalesMinTarget4
Drop & Retain	Drop & Retain	Drop & Retain	Drop & Retain
10	10	10	15

(ii) DO loop processing: Before running the DO loop, DIM(Sales_vol) is calculated and equal to 4.

	1st DO loop	2nd DO loop
i	`Diff{i}=Sales_vol{i+1}-Sales_vol{i}`	`Diff_Target{i}=Sales_vol{i} - SalesMinTarget{i}`
1	Diff{1}=Sales_vol{2}-Sales_vol{1} → Diff{1}=Quarter2-Quarter1	Diff_Target{1}=Sales_vol{1}-SalesMinTarget{1} → Diff_Target{1}=Quarter1- SalesMinTarget1
2	Diff{2}=Sales_vol{3}-Sales_vol{2} → Diff{2}=Quarter3-Quarter2	Diff_Target{2}=Sales_vol{2}-SalesMinTarget{2} → Diff_Target{2}=Quarter2- SalesMinTarget2
3	Diff{3}=Sales_vol{4}-Sales_vol{3} → Diff{3}=Quarter4-Quarter3	Diff_Target{3}=Sales_vol{3}-SalesMinTarget{3} → Diff_Target{3}=Quarter3- SalesMinTarget3
4	-	Diff_Target{4}=Sales_vol{4}-SalesMinTarget{4} → Diff_Target{4}=Quarter4- SalesMinTarget4

- The output



```
                                                                 Diff_   Diff_   Diff_   Diff_
Obs  name   Quarter1 Quarter2 Quarter3 Quarter4 Diff1 Diff2 Diff3 Target1 Target2 Target3 Target4

 1  Peter      20       10       50       12     -10    40   -38     10       0      40      -3
 2  Mary        3        5        7        8       2     2     1     -7      -5      -3      -7
 3  Cindy      12       34       54       45      22    20    -9      2      24      44      30
 4  George      5        6        7        8       1     1     1     -5      -4      -3      -7
```

Ex. 9.5: Define a SAS array called TopUni to store the following **PDV**:

Variable	TopUni1	TopUni2	TopUni3	TopUni4	TopUni5
Content	Cambridge	Stanford	Harvard	MIT	Oxford

Ans:

9.3 Tools for debugging

- Type of errors
  - A syntax error occurs when program statements do not conform to the rules of the SAS language. An error message is written to the log.
  - A logic error occurs when the program statements follow the rules but the results are not correct. No notes are written to the log, so logic errors are often **difficult** to detect.
  - This section focuses the detection of logic errors.

- Special variables useful for debugging in DATA step

Variable	Description	Debugging use
_N_	The number of times which the DATA step iterated	Display debugging messages for some number of iterations of the DATA step
ERROR	Initialized to 0, set to 1 when an error occurs	Display debugging messages when an error occurs

- PUTLOG statement in DATA step
  - Write the value of a variable
    - Syntax

```
PUTLOG variable-name=;
```

    - For example, “PUTLOG name=;”
  - Write formatted values
    - Syntax

```
PUTLOG variable-name format-namew.;
```

    - For example, “PUTLOG name $quote22.;”
    - The $quote*w*. Format can be used to see if there are leading spaces in a variable’s value.
  - Write a text line
    - Syntax

```
PUTLOG ‘Text’;
```

    - For example, “PUTLOG ‘This is last record’;”
- The END= Option
  - Create a temporary variable which is initialized to 0 and changed to 1 for encountering of end-of-file.
  - It is normally combined with “IF-THEN statement” and/or with “DO-END statement”.
  - Syntax

```
SET SAS-data-set END=variable <options>;
```

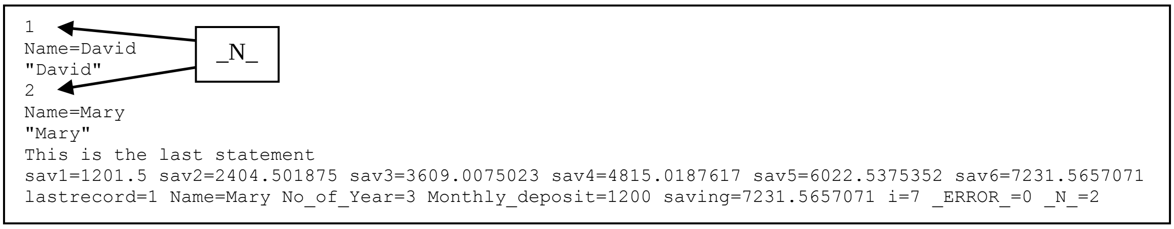
```
INFILE ‘raw-data-file’ END=variable <options>;
```

- Planned saving example revisit for demonstration the use of PUTLOG and END = *variable* option:
  - PUTLOG and END= lastrecord are put into the SAS program

```
*P9_PUTLOG.sas;
Libname xlsdata
'd:\SAS_datasets\planned_saving.xlsx';
data Planned_Saving;
array savm {6} sav1-sav6;
  set xlsdata.'Customers$'n end=lastrecord;
  PUTLOG _N_;
  PUTLOG name=;
  PUTLOG name $quote22.;
  saving=0;
  do i = 1 to 6;
   saving +monthly_deposit;
   saving+(saving*1.5/100/12);
   savm{i}=saving;
end;
if lastrecord then do;
  PUTLOG 'This is the last statement';
  PUTLOG _ALL_;
end;
run;
```

i.e. lastrecord =1 for Mary's record.

  - The output log window shows

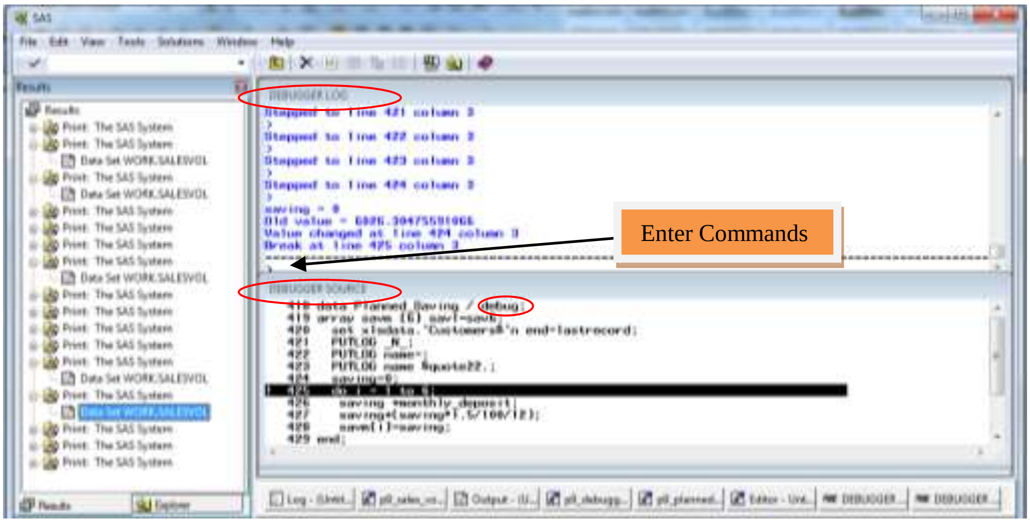


```
1
Name=David
"David"
2
Name=Mary
"Mary"
This is the last statement
sav1=1201.5 sav2=2404.501875 sav3=3609.0075023 sav4=4815.0187617 sav5=6022.5375352 sav6=7231.5657071
lastrecord=1 Name=Mary No_of_Year=3 Monthly_deposit=1200 saving=7231.5657071 i=7 _ERROR_=0 _N_=2
```

- DEBUG option in DATA step
  - An interactive interface to the DATA step, not for PROC step.
  - Syntax

```
DATA SAS-data-set /DEBUG;
```

  - With DEBUG option, commands are entered in PROC DEBUGGER LOG window interactively. Commands are available to
    - execute a DATA step one statement at a time
    - examine the value of one or more variables
    - watch one or more variables as they change in value.


SAS
File Edit View Tools Solutions Window Help
Results
DEBUGGER LOG
DEBUGGER SOURCE
Log - (Untit... | pll_sales_vs... | Output - (U... | pll_debugg... | pll_planned... | Editor - Unt... | DEBUGGER... | DEBUGGER...

```
Stepped to line 421 column 3
>
Stepped to line 422 column 3
>
Stepped to line 423 column 3
>
Stepped to line 424 column 3
>
saving = 0
Old value = 6826.30475591066
Value changed at line 424 column 3
Break at line 425 column 3
```

Enter Commands

```
418 data Planned_Saving / debug;
419 array save {6} sav1-sav6;
420   set xlsdata.'Customers$'n end=lastrecord;
421   PUTLOG _N_;
422   PUTLOG name=;
423   PUTLOG name $quote22.;
424   saving=0;
425   do i = 1 to 6;
426     saving +monthly_deposit;
427     saving+(saving*1.5/100/12);
428     save[i]=saving;
429 end;
```

- Common commands used with the DEBUG option:

Command	Alias	Syntax	Action
STEP	ENTER key		Execute statements one at a time
EXAMINE	E *variable(s)*		Display the value of one or more variables
WATCH	W *variable(s)*		Suspend execution when the value of a watched variable changes
QUIT	Q		Terminate a debugger session
LIST	L *argument*	LIST _ALL_ | WATCH	Display all occurrences of the item listed in the argument L W – list watched variables
DELETE Watch	D W *variable(s)*	DELETE WATCH *variable(s)* | _ALL_	Delete the watch status of the listed variables
SET	SET	SET *variable* = *expression*	Assign a new value to the specified variable

- Setting breaking point
  - A breakpoint can be set to suspend the execution of the program so that the user can submit debugger commands.
  - Properties of a breakpoint:
    (a) Conditional or unconditional
    (b) In effect on every iteration of the DATA step or after a set number of iterations
    (c) With an exclamation mark appearing to the left of the line in the Debugger Source window.
  - Syntax of BREAK Command

```
BREAK location <AFTER count><WHEN expression> <DO group>;
```

    (a) Alias B.
    (b) *Location*: The line number in the debugger window at which to suspend execution
    (c) AFTER *count*: Break after the line has been executed *count* times.
    (d) WHEN expression: Break when expression evaluate to TRUE.
    (e) DO *group*: One or more debugger commands enclosed by a DO and an END statement.
  - Examples
    (a) "B 7 AFTER 3 WHEN num=0": Break on line 7 after it executed three times only if num=0
    (b) "D B 9": Delete the breakpoint on line 9
  - Syntax of GO Command

```
GO <line number>;
```

Line number: The number of the program line at which execution is to be suspended

- Examples

  (a) "G": Resume execution at the next executable line

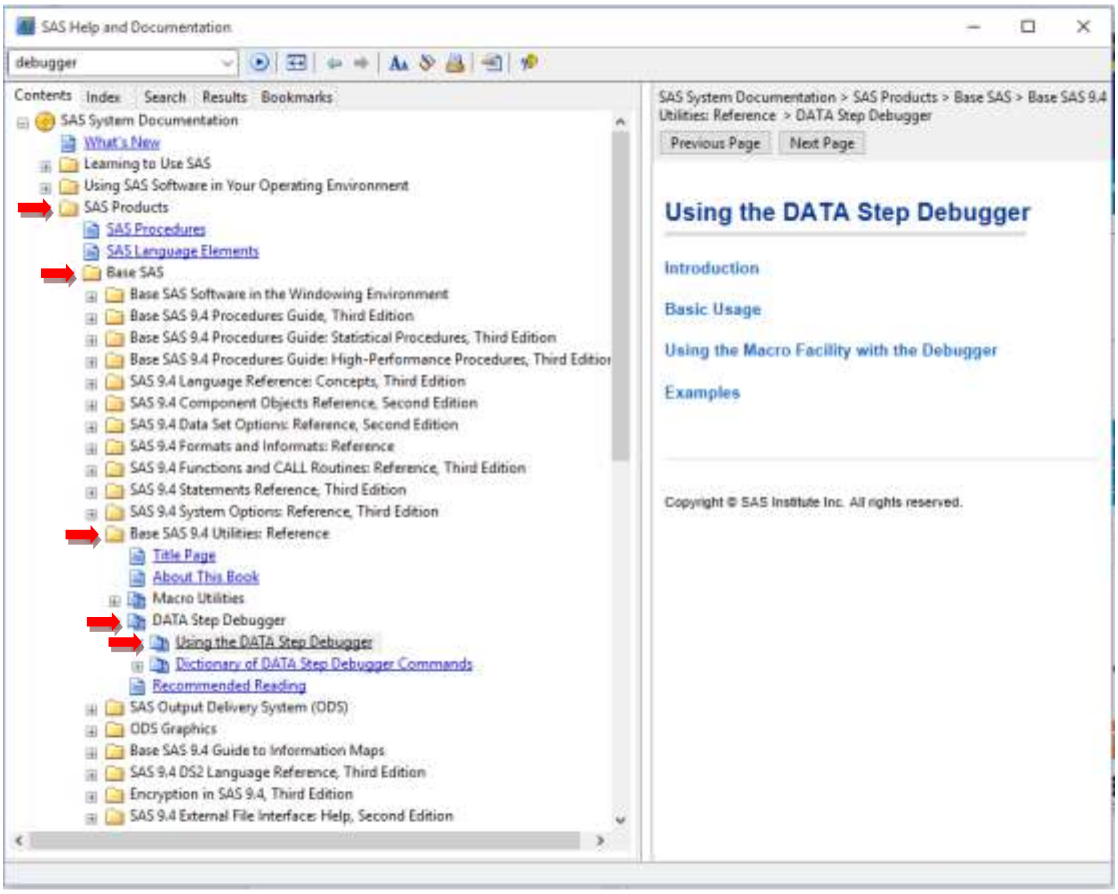
  (b) 'G 25": Resume execution and suspend execution at line 25

- An example

```
*p9_debugger.sas;
Libname xlsdata 'd:\SAS_datasets\planned_saving.xlsx';
data Planned_Saving / debug;
array savm {6} sav1-sav6;
  set xlsdata.'Customers$'n end=lastrecord;
   saving=0;
  do i = 1 to 6;
   saving +monthly_deposit;
   saving+(saving*1.5/100/12);
   savm{i}=saving;
end;
run;
```

Submit this program and test the commands given above.

- DATA Step Debugger Documentation
  - For help on the DATA Step Debugger, select **SAS Products** →**Base SAS** → **SAS 9.4 Utilities: Reference** → **DATA Step Debugger**.



- SAS System Documentation
  - What's New
  - Learning to Use SAS
  - Using SAS Software in Your Operating Environment
  - SAS Products
    - SAS Procedures
    - SAS Language Elements
    - Base SAS
      - Base SAS Software in the Windowing Environment
      - Base SAS 9.4 Procedures Guide, Third Edition
      - Base SAS 9.4 Procedures Guide: Statistical Procedures, Third Edition
      - Base SAS 9.4 Procedures Guide: High-Performance Procedures, Third Editio
      - SAS 9.4 Language Reference: Concepts, Third Edition
      - SAS 9.4 Component Objects Reference, Second Edition
      - SAS 9.4 Data Set Options: Reference, Second Edition
      - SAS 9.4 Formats and Informats: Reference
      - SAS 9.4 Functions and CALL Routines: Reference, Third Edition
      - SAS 9.4 Statements Reference, Third Edition
      - SAS 9.4 System Options: Reference, Third Edition
      - Base SAS 9.4 Utilities: Reference
        - Title Page
        - About This Book
        - Macro Utilities
        - DATA Step Debugger
          - Using the DATA Step Debugger
          - Dictionary of DATA Step Debugger Commands
        - Recommended Reading
      - SAS Output Delivery System (ODS)
      - ODS Graphics
      - Base SAS 9.4 Guide to Information Maps
      - SAS 9.4 DS2 Language Reference, Third Edition
      - Encryption in SAS 9.4, Third Edition
      - SAS 9.4 External File Interface: Help, Second Edition

SAS System Documentation > SAS Products > Base SAS > Base SAS 9.4 Utilities: Reference > DATA Step Debugger

Using the DATA Step Debugger

Introduction

Basic Usage

Using the Macro Facility with the Debugger

Examples

Copyright © SAS Institute Inc. All rights reserved.