TATA CONSULTANCY SERVICES
Experience certainty.

SAS® Training

Basic SAS Procedures

Topics covered…

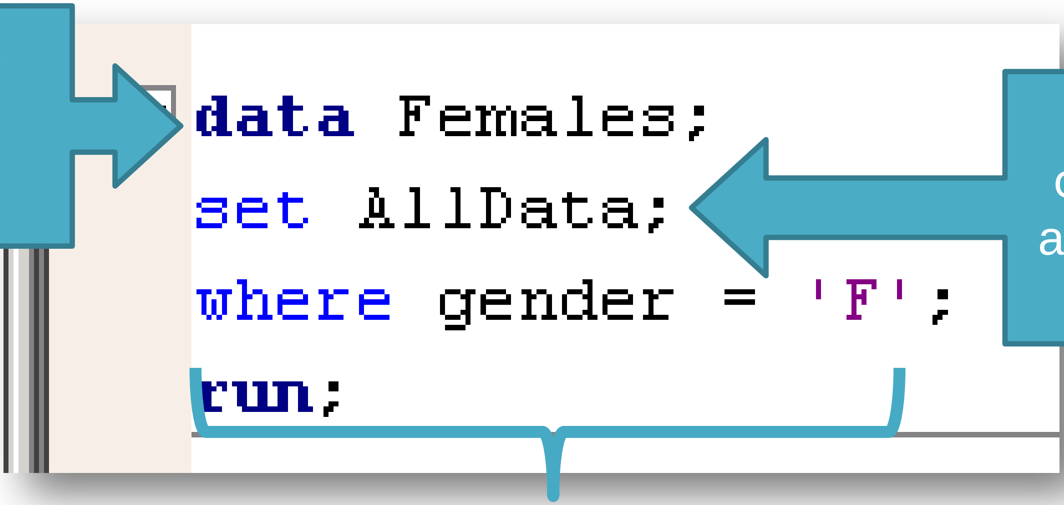
- SET statement
- Basic PROC Print
- Basic PROC Contents
- SORT
- Conditional logic
  - If, Then, Else
- Subsetting datasets
  - IF and WHERE
- Selecting variables
  - DROP and KEEP
- Appending datasets
  - Joins and merges
  - MERGE
- Processing Data Iteratively
  - Do Loop processing
  - SAS Array processing
  - Using SAS Arrays

SET Statement

SET statement

- After you’ve brought your data into a SAS dataset, most of your DATA steps will look like this:

Creates a new dataset called “Females”



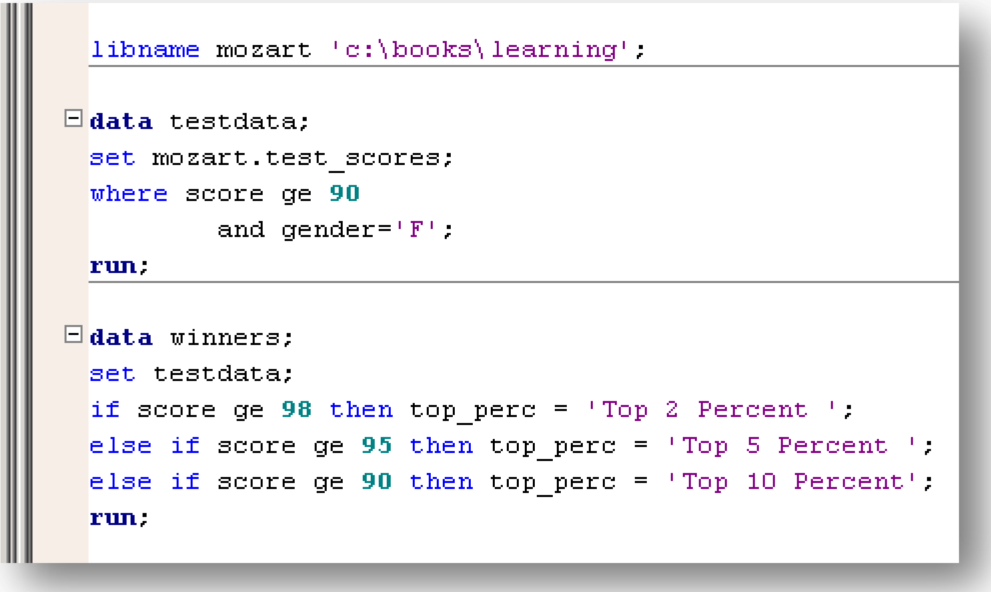
```
data Females;
set AllData;
where gender = 'F';
run;
```

Uses previous dataset “AllData” as the basis of the new dataset

Applies these modifications to the new dataset

SET statement

- The SET statement is similar to an INPUT statement
  - Except instead of a raw data file, you are reading observations from a SAS dataset
- Can read in temporary or permanent SAS datasets



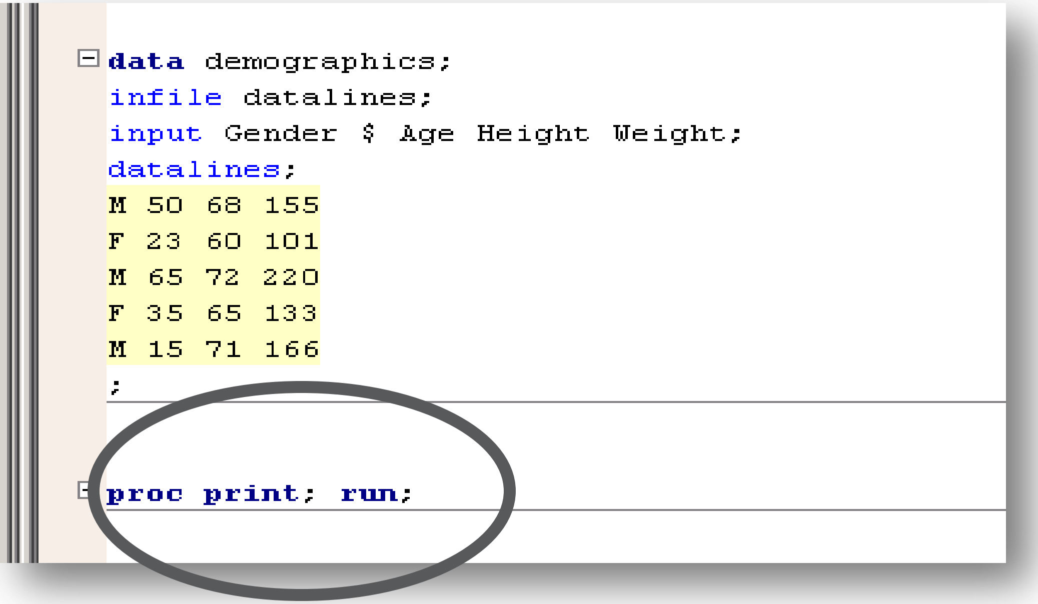
```
libname mozart 'c:\books\learning';

data testdata;
set mozart.test_scores;
where score ge 90
        and gender='F';
run;

data winners;
set testdata;
if score ge 98 then top_perc = 'Top 2 Percent ';
else if score ge 95 then top_perc = 'Top 5 Percent ';
else if score ge 90 then top_perc = 'Top 10 Percent';
run;
```

TATA CONSULTANCY SERVICES
Experience certainty.

PROC Print

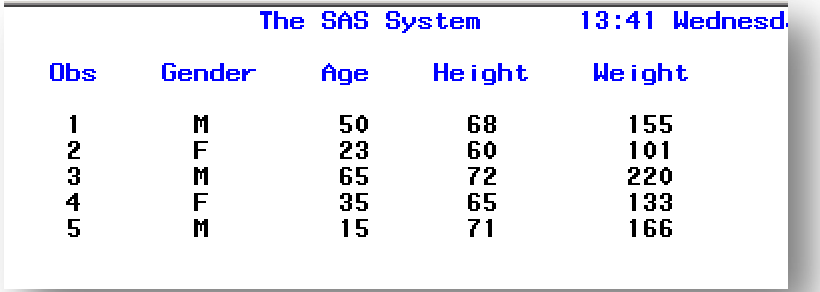
PROC Print

- PROC Print can be used to list the data in a SAS dataset



```
data demographics;
infile datalines;
input Gender $ Age Height Weight;
datalines;
M 50 68 155
F 23 60 101
M 65 72 220
F 35 65 133
M 15 71 166
;

proc print; run;
```

PROC Print

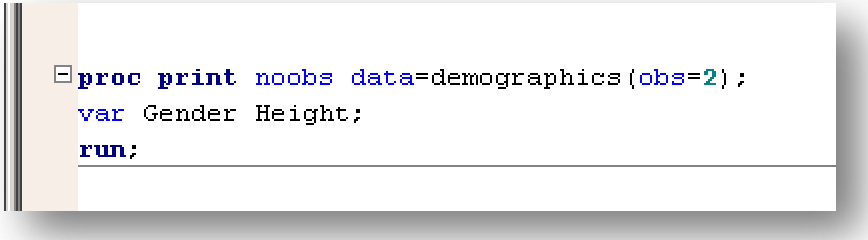


```
                  The SAS System        13:41 Wednesd

   Obs      Gender      Age      Height      Weight

    1         M          50        68          155
    2         F          23        60          101
    3         M          65        72          220
    4         F          35        65          133
    5         M          15        71          166
```

Results of PROC Print of “Demographics”

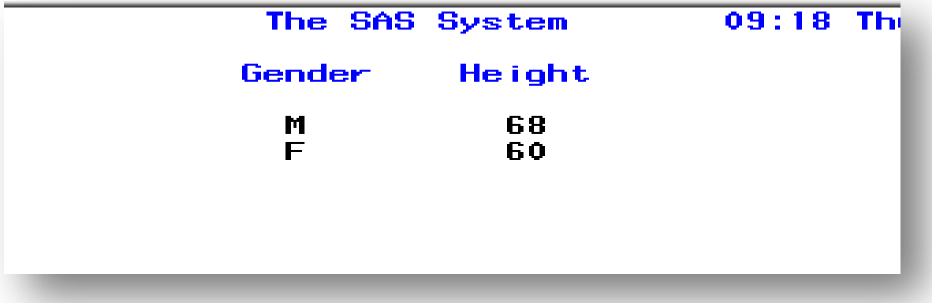
PROC Print

- Many options to control output of PROC Print
  - noobs – Suppresses "OBS" column in output
  - (obs=2) – Only prints the first two observations
    - Can put in any number: 1 through N
    - Must be placed in parentheses after **data=** option
  - var statement – Only prints listed variables



```
proc print noobs data=demographics(obs=2);
var Gender Height;
run;
```

PROC Print



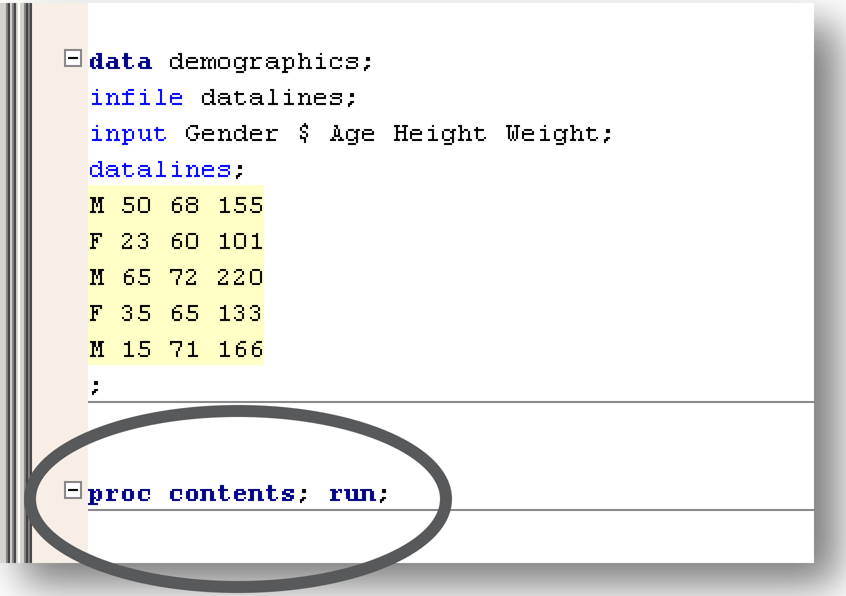
```
              The SAS System          09:18 Th

             Gender        Height

               M             68
               F             60
```

We’ll discuss other PROC Print options in later chapters

PROC Contents

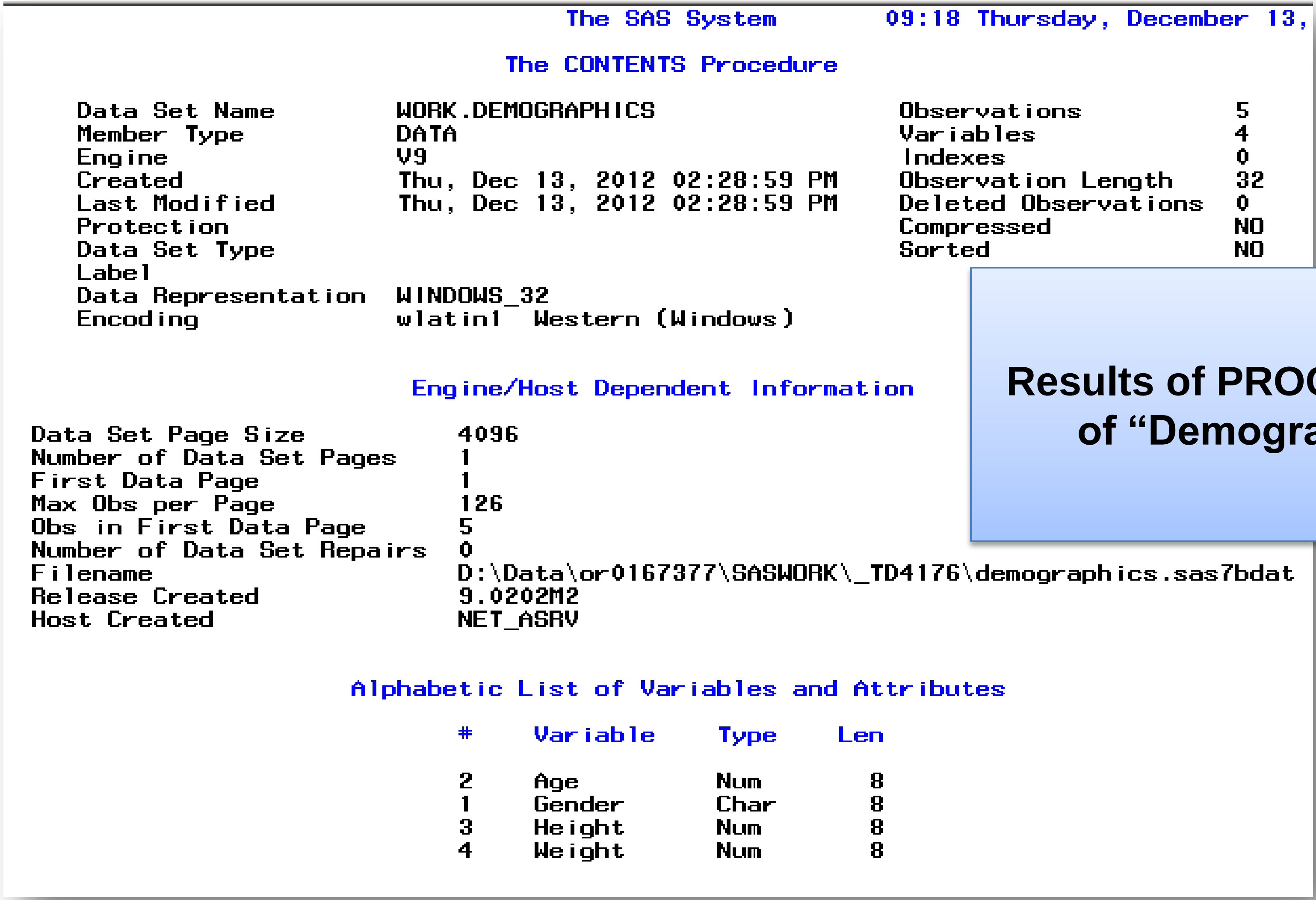
PROC Contents

- PROC Contents can be used to display the metadata (descriptor portion) of the SAS dataset



```
data demographics;
infile datalines;
input Gender $ Age Height Weight;
datalines;
M 50 68 155
F 23 60 101
M 65 72 220
F 35 65 133
M 15 71 166
;

proc contents; run;
```

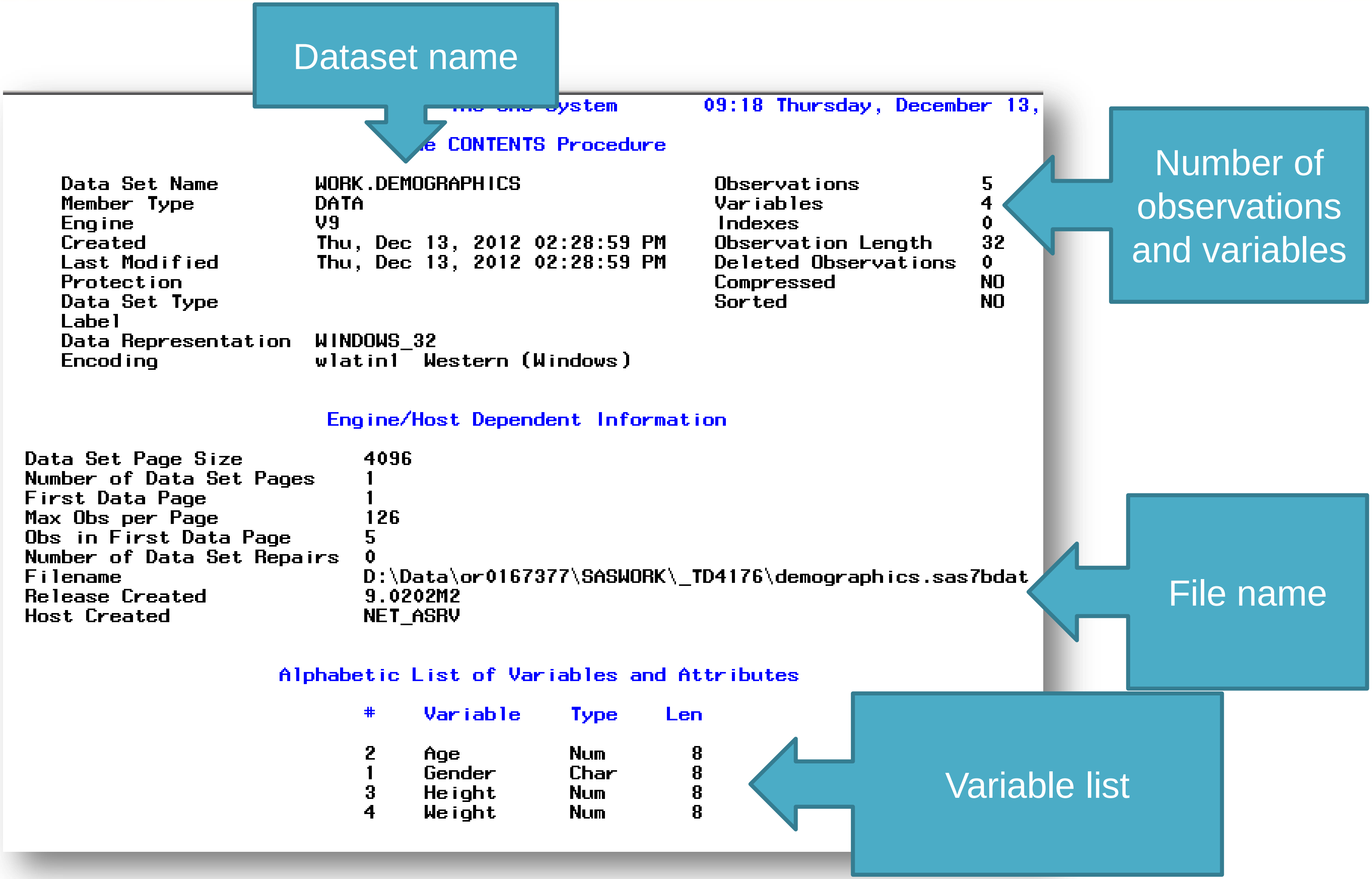
PROC Contents



```
                                   The SAS System       09:18 Thursday, December 13,

                               The CONTENTS Procedure

    Data Set Name          WORK.DEMOGRAPHICS                Observations          5
    Member Type            DATA                             Variables             4
    Engine                 V9                               Indexes               0
    Created                Thu, Dec 13, 2012 02:28:59 PM    Observation Length    32
    Last Modified          Thu, Dec 13, 2012 02:28:59 PM    Deleted Observations  0
    Protection                                              Compressed            NO
    Data Set Type                                           Sorted                NO
    Label
    Data Representation    WINDOWS_32
    Encoding               wlatin1  Western (Windows)

                          Engine/Host Dependent Information

 Data Set Page Size          4096
 Number of Data Set Pages    1
 First Data Page             1
 Max Obs per Page            126
 Obs in First Data Page      5
 Number of Data Set Repairs  0
 Filename                    D:\Data\or0167377\SASWORK\_TD4176\demographics.sas7bdat
 Release Created             9.0202M2
 Host Created                NET_ASRV

                      Alphabetic List of Variables and Attributes

                           #    Variable    Type    Len

                           2    Age         Num       8
                           1    Gender      Char      8
                           3    Height      Num       8
                           4    Weight      Num       8
```

Results of PROC Contents of “Demographics”

PROC Contents



```
                    The SAS System        09:18 Thursday, December 13,

                          e CONTENTS Procedure

   Data Set Name            WORK.DEMOGRAPHICS                Observations          5
   Member Type              DATA                             Variables             4
   Engine                   V9                               Indexes               0
   Created                  Thu, Dec 13, 2012 02:28:59 PM    Observation Length    32
   Last Modified            Thu, Dec 13, 2012 02:28:59 PM    Deleted Observations  0
   Protection                                                Compressed            NO
   Data Set Type                                             Sorted                NO
   Label
   Data Representation      WINDOWS_32
   Encoding                 wlatin1  Western (Windows)


                    Engine/Host Dependent Information

 Data Set Page Size          4096
 Number of Data Set Pages    1
 First Data Page             1
 Max Obs per Page            126
 Obs in First Data Page      5
 Number of Data Set Repairs  0
 Filename                    D:\Data\or0167377\SASWORK\_TD4176\demographics.sas7bdat
 Release Created             9.0202M2
 Host Created                NET_ASRV


                 Alphabetic List of Variables and Attributes

                          #    Variable    Type    Len

                          2    Age         Num       8
                          1    Gender      Char      8
                          3    Height      Num       8
                          4    Weight      Num       8
```

Dataset name

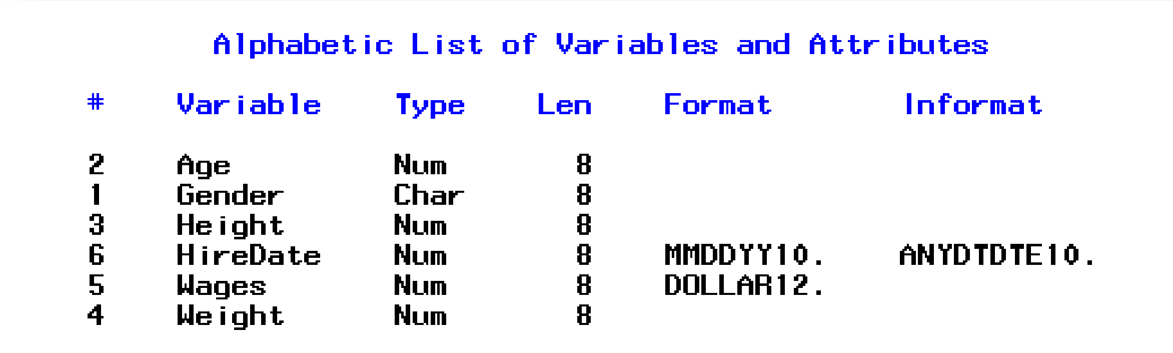
Number of observations and variables

File name

Variable list

PROC Contents variable list

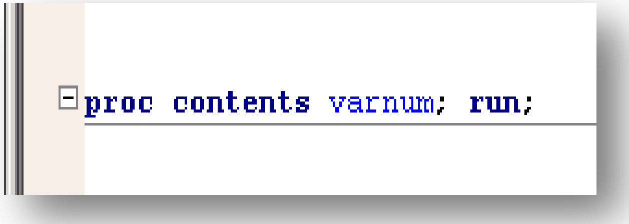
- # - Variable number (varnum)
- **Variable** – Name of variable
- **Type** – Numeric or Character
- **Len** – Variable length
- **Format** – How the data is displayed
- **Informat** – How the data was read by SAS



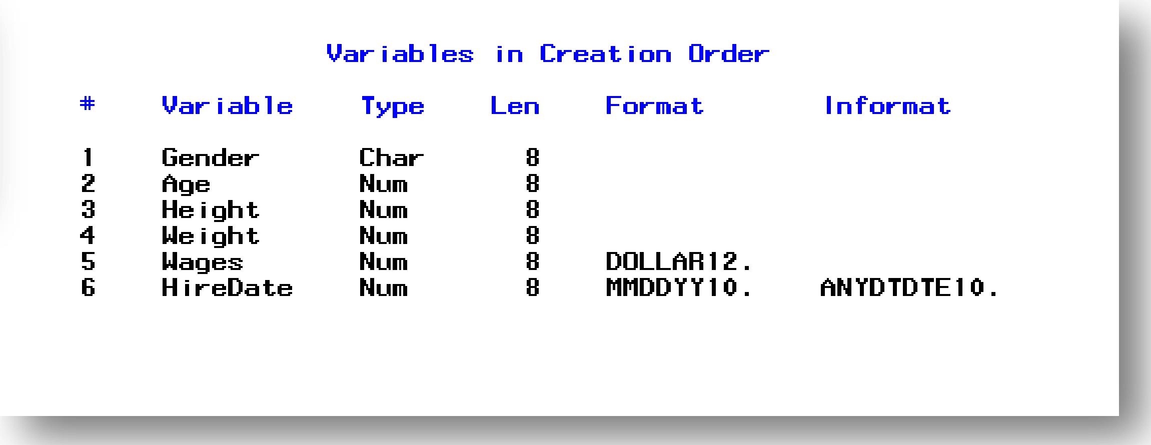
```
                Alphabetic List of Variables and Attributes

   #    Variable     Type    Len    Format         Informat

   2    Age          Num       8
   1    Gender       Char      8
   3    Height       Num       8
   6    HireDate     Num       8    MMDDYY10.     ANYDTDTE10.
   5    Wages        Num       8    DOLLAR12.
   4    Weight       Num       8
```

PROC Contents variable list

- Variables listed in alphabetical order by default
  - Uppercase alphabetized before lowercase (e.g., “ZZTOP” would be alphabetized before “aerosmith”)
- Use the `varnum` option to list variables in order they were created in



```
proc contents varnum; run;
```

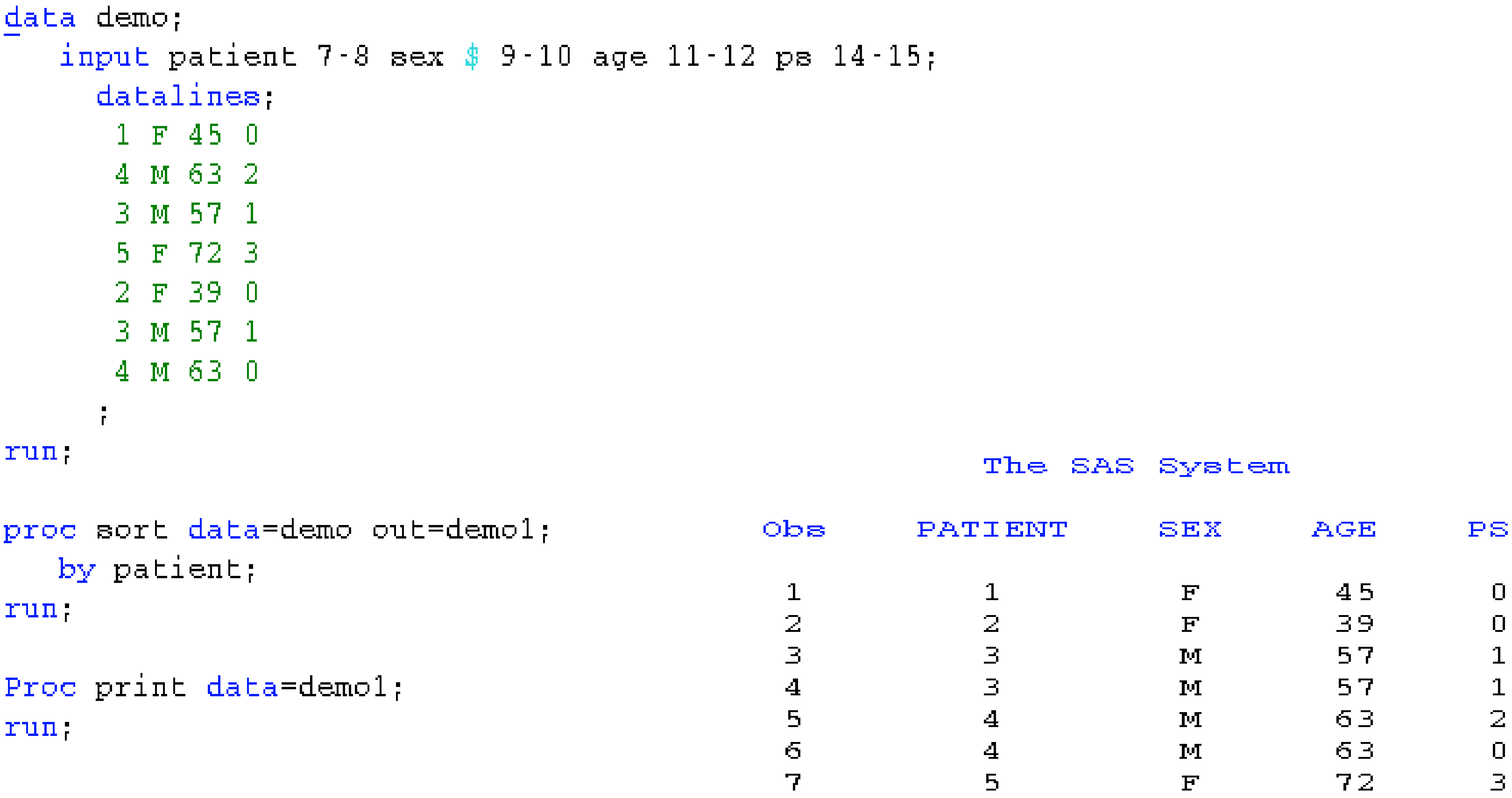


```
                       Variables in Creation Order

#     Variable     Type     Len     Format          Informat

1     Gender       Char       8
2     Age          Num        8
3     Height       Num        8
4     Weight       Num        8
5     Wages        Num        8     DOLLAR12.
6     HireDate     Num        8     MMDDYY10.       ANYDTDTE10.
```

PROC SORT

SORT

- The SORT procedure sorts observations in a SAS data set by one or more character or numeric variables, either replacing the original data set or creating a new, sorted data set.

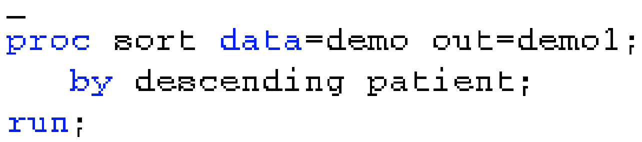


```
data demo;
   input patient 7-8 sex $ 9-10 age 11-12 ps 14-15;
     datalines;
      1 F 45 0
      4 M 63 2
      3 M 57 1
      5 F 72 3
      2 F 39 0
      3 M 57 1
      4 M 63 0
     ;
run;

proc sort data=demo out=demo1;
   by patient;
run;

Proc print data=demo1;
run;
```

```
                   The SAS System

 Obs     PATIENT     SEX     AGE     PS

   1        1         F       45      0
   2        2         F       39      0
   3        3         M       57      1
   4        3         M       57      1
   5        4         M       63      2
   6        4         M       63      0
   7        5         F       72      3
```

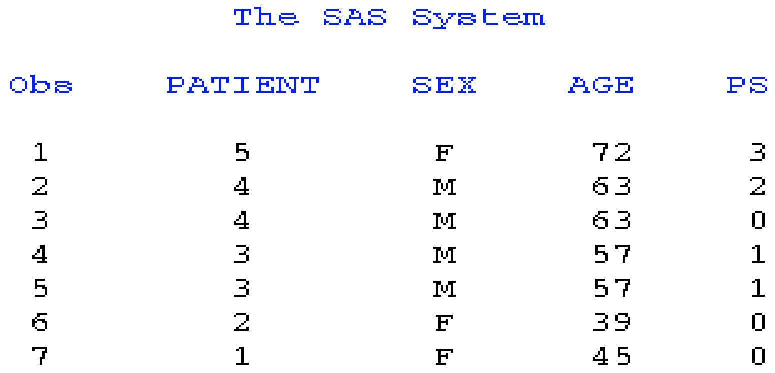
SORT – Descending Option

- By default SAS sorts the data with the BY variables in *ascending* order. If we want the data to be sorted by the BY variables in *descending* order then we can use the *descending* option in SAS.



```
_
proc sort data=demo out=demo1;
   by descending patient;
run;
```

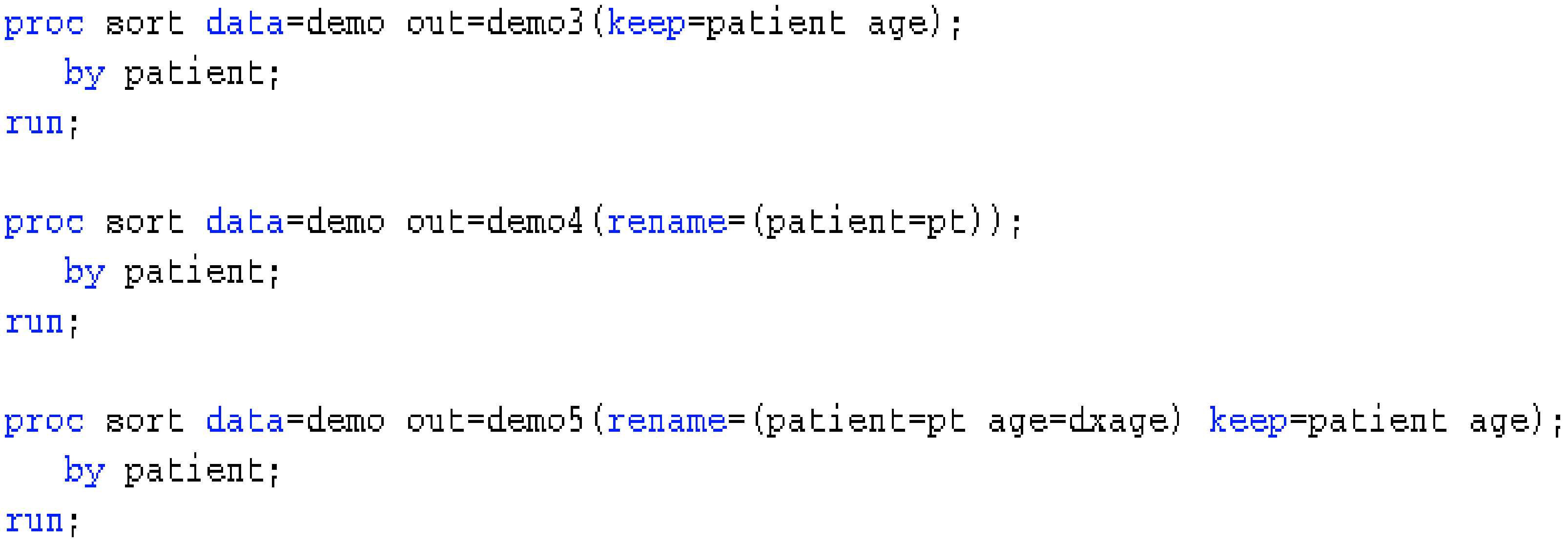
- SAS Output



```
                  The SAS System

Obs     PATIENT     SEX     AGE     PS

 1         5         F       72      3
 2         4         M       63      2
 3         4         M       63      0
 4         3         M       57      1
 5         3         M       57      1
 6         2         F       39      0
 7         1         F       45      0
```

DROP=, KEEP=, AND RENAME= OPTIONS

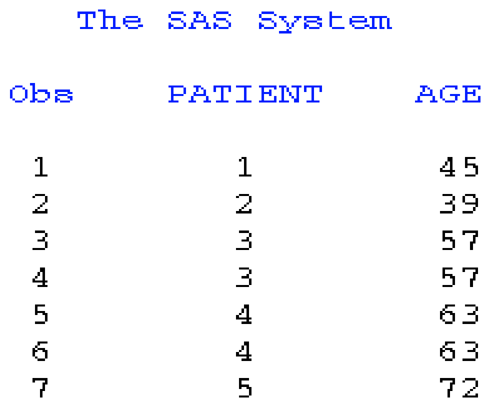
- You can use the DROP=, KEEP=, and RENAME= options within the SORT procedure just as you can within a DATA step. Here are some examples using these options along with how the output data sets look:



```
proc sort data=demo out=demo3(keep=patient age);
   by patient;
run;

proc sort data=demo out=demo4(rename=(patient=pt));
   by patient;
run;

proc sort data=demo out=demo5(rename=(patient=pt age=dxage) keep=patient age);
   by patient;
run;
```

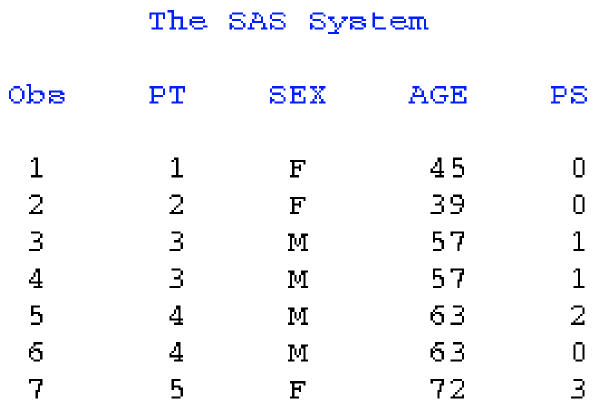
Examples:

- DEMO3 dataset:



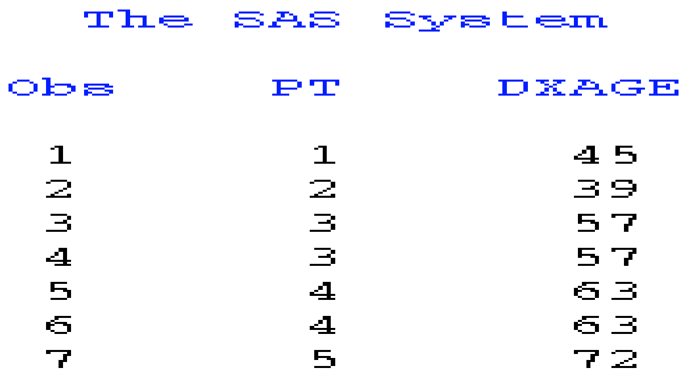
```
       The SAS System

 Obs      PATIENT     AGE

  1          1         45
  2          2         39
  3          3         57
  4          3         57
  5          4         63
  6          4         63
  7          5         72
```

DEMO4 Dataset:



```
         The SAS System

 Obs    PT    SEX    AGE    PS

  1      1     F      45     0
  2      2     F      39     0
  3      3     M      57     1
  4      3     M      57     1
  5      4     M      63     2
  6      4     M      63     0
  7      5     F      72     3
```

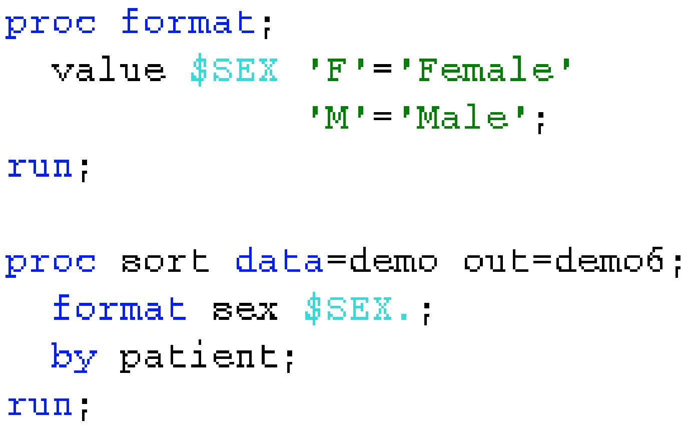
- DEMO5 Dataset:



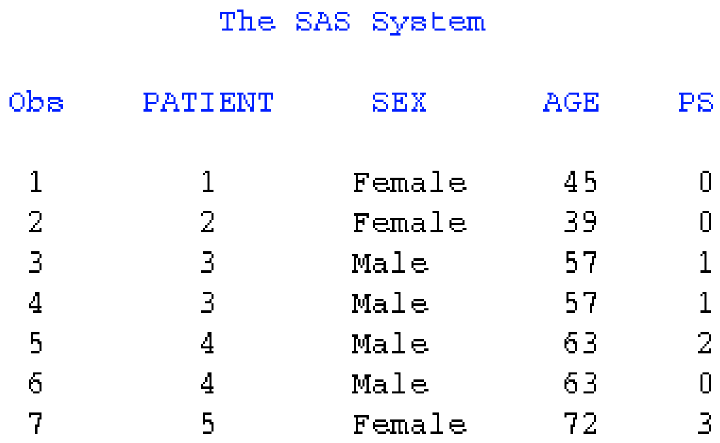
```
      The SAS System

 Obs       PT       DXAGE

  1         1         45
  2         2         39
  3         3         57
  4         3         57
  5         4         63
  6         4         63
  7         5         72
```

FORMAT AND LABEL STATEMENTS

- Other statements that are the same in the SORT procedure as in a DATA step are the FORMAT and LABEL statements. You can apply a variable format or create variable labels within PROC SORT. Example:

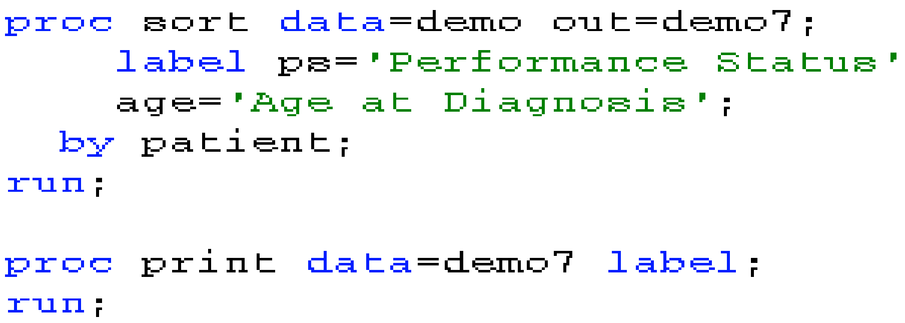


```
proc format;
   value $SEX 'F'='Female'
              'M'='Male';
run;

proc sort data=demo out=demo6;
   format sex $SEX.;
   by patient;
run;
```

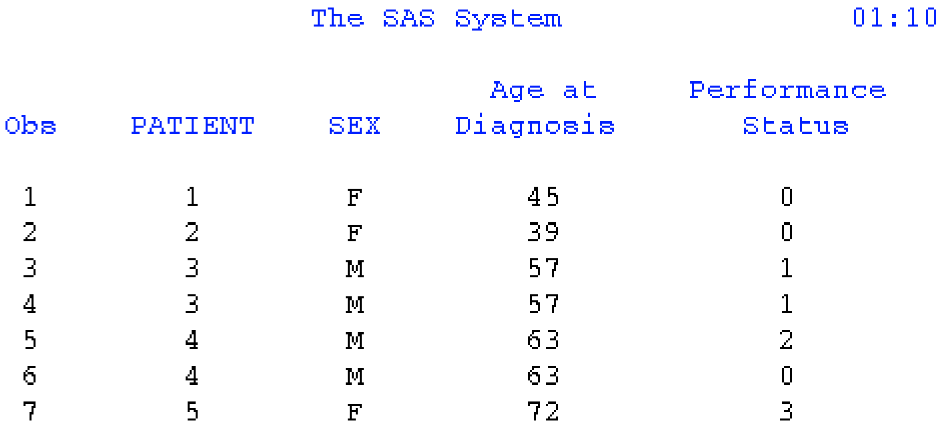


```
                    The SAS System

Obs     PATIENT     SEX       AGE     PS

 1         1        Female     45      0
 2         2        Female     39      0
 3         3        Male       57      1
 4         3        Male       57      1
 5         4        Male       63      2
 6         4        Male       63      0
 7         5        Female     72      3
```

Example Using Labels



```
proc sort data=demo out=demo7;
    label ps='Performance Status'
    age='Age at Diagnosis';
  by patient;
run;

proc print data=demo7 label;
run;
```

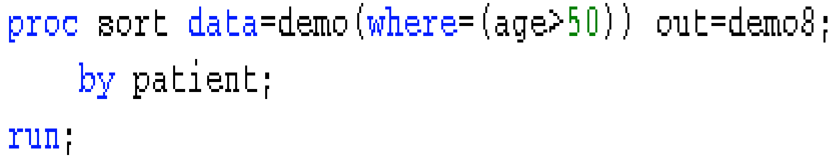


```
                      The SAS System               01:10

                                 Age at      Performance
Obs     PATIENT     SEX       Diagnosis        Status

 1         1         F            45              0
 2         2         F            39              0
 3         3         M            57              1
 4         3         M            57              1
 5         4         M            63              2
 6         4         M            63              0
 7         5         F            72              3
```

NOTE: By using the PRINT procedure with the label option following the PROC SORT statement, you can see the labels created for the variables PS and AGE. Like the FORMAT statement, the LABEL statement does not permanently alter the variables in the input data set

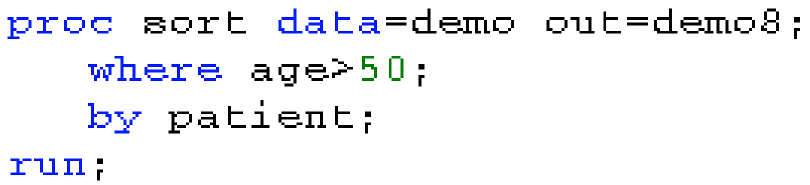
WHERE= OPTION OR WHERE STATEMENT

- SORT procedure allows you to subset your data by using the WHERE= option or WHERE STATEMENT.
  - ➢ Example using the WHERE= option:

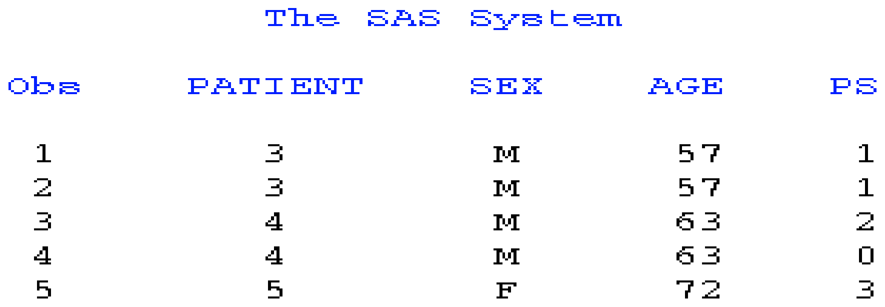


```
proc sort data=demo(where=(age>50)) out=demo8;
    by patient;
run;
```

  - ➢ Example using WHERE statement:



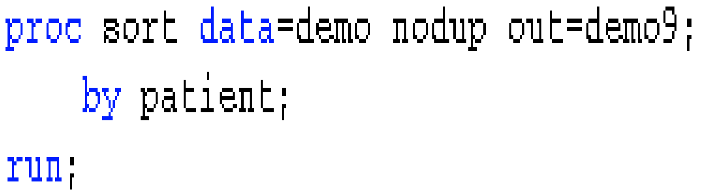
```
proc sort data=demo out=demo8;
    where age>50;
    by patient;
run;
```



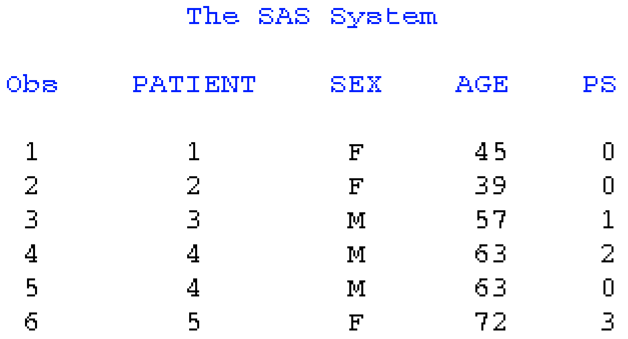
```
                     The SAS System

  Obs      PATIENT      SEX      AGE      PS

   1          3          M        57       1
   2          3          M        57       1
   3          4          M        63       2
   4          4          M        63       0
   5          5          F        72       3
```

NODUPRECS AND NODUPKEY OPTIONS

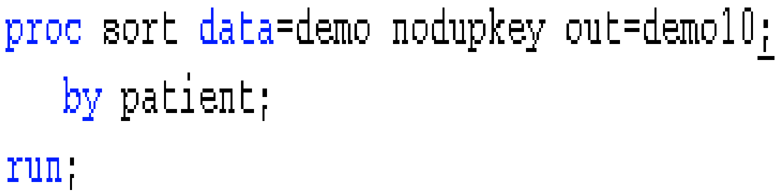
- The NODUPRECS (or NODUP) and NODUPKEY options work similarly in that they both can eliminate unwanted observations, but NODUP compares *all* the variables in your data set while NODUPKEY compares just the *BY* variables.



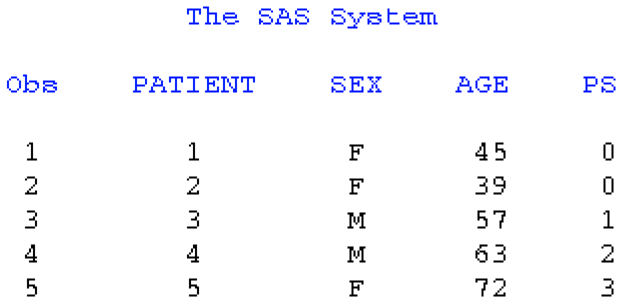
```
proc sort data=demo nodup out=demo9;
    by patient;
run;
```



```
                      The SAS System

Obs      PATIENT      SEX      AGE      PS

 1          1          F        45       0
 2          2          F        39       0
 3          3          M        57       1
 4          4          M        63       2
 5          4          M        63       0
 6          5          F        72       3
```



```
proc sort data=demo nodupkey out=demo10;
   by patient;
run;
```



```
                      The SAS System

Obs      PATIENT      SEX      AGE      PS

 1          1          F        45       0
 2          2          F        39       0
 3          3          M        57       1
 4          4          M        63       2
 5          5          F        72       3
```

Conditional Logic

If, Then, Else

If, Then, Else

```
IF <condition> THEN <X>;
ELSE <Y>;
```

```
If Score >= 7 Then Response = 'Passing Grade';
Else Grade = Failing Grade';
```

Student	Score	Grade
Jane	75	Passing Grade
Dave	56	Failing Grade
Jack	90	Passing Grade
Sue	68	Failing Grade

If, Then, Else

```
IF <condition> THEN <X>;
ELSE <Y>;
```

```
If Score >= 70 Then Grade = 'Passing Grade';
Else Grade = 'Failing Grade';
```

Assignment statement
Used to create new variables

Student	Score	Grade
Jane	75	Passing Grade
Dave	56	Failing Grade
Jack	90	Passing Grade
Sue	68	Failing Grade

If, Then, Else

```
IF <condition> THEN <X>;
ELSE IF <condition2> THEN <Y>;
ELSE <Z>;
```

```
If Score >= 7 Then Grade = 'Passing Grade';
Else If 60 <= Score <= 69 Then Grade = 'Incomplete';
Else Grade = 'Failing Grade';
```

Student	Score	Grade
Jane	75	Passing Grade
Dave	56	Failing Grade
Jack	90	Passing Grade
Sue	68	Incomplete

If, Then, Else

- When using ELSE IF:
  - Processes IF-THEN conditions until first true statement is met, then it moves on to the next observation
  - Once a condition is met, the observation is not reevaluated

If, Then, Else

```
If Score >= 70 Then Grade = 'Passing Grade';
Else If 60 <= Score <= 69 Then Grade = 'Incomplete';
Else Grade = 'Failing Grade';
```

Student	Score	Grade
Jane	75	Passing Grade
Dave	56	Failing Grade
Jack	90	Passing Grade
Sue	68	Incomplete

If, Then, Else

```
If Score >= 90 Then Grade = 'A';
If Score >= 80 Then Grade = 'B';
If Score >= 70 Then Grade = 'C';
If Score >= 60 Then Grade = 'D';
If Score < 60 Then Grade = 'F';
```

Student	Score	Grade
Jane	75	D
Dave	56	F
Jack	90	D
Sue	68	D

If, Then, Else

```
If Score >= 90 Then Grade = 'A';
If Score >= 80 Then Grade = 'B';
If Score >= 70 Then Grade = 'C';
If Score >= 60 Then Grade = 'D';
If Score < 60 Then Grade = 'F';
```

Common mistake: Not using **ELSE IF**

Each subsequent **IF** re-evaluated every observation

Student	Score	Grade
Jane	75	D
Dave	56	F
Jack	90	D
Sue	68	D

If, Then, Else

```
If Score >= 90 Then Grade = 'A';   ✓
If Score >= 80 Then Grade = 'B';   ✓
If Score >= 70 Then Grade = 'C';   ✓
If Score >= 60 Then Grade = 'D';   ✓
If Score < 60 Then Grade = 'F';    ✗
```

Student	Score	Grade
Jack	90	~~A~~ ~~B~~ ~~C~~ D

If, Then, Else

```
  If Score >= 90 Then Grade = 'A';
ELSE If Score >= 80 Then Grade = 'B';
ELSE If Score >= 70 Then Grade = 'C';
ELSE If Score >= 60 Then Grade = 'D';
ELSE If Score < 60 Then Grade = 'F';
```

Student	Score	Grade
Jack	90	A

If, Then, Else

```
If Score >= 90 Then Grade = 'A';
ELSE If Score >= 80 Then Grade = 'B';
ELSE If Score >= 70 Then Grade = 'C';
ELSE If Score >= 60 Then Grade = 'D';
ELSE If Score < 60 Then Grade = 'F';
```

Student	Score	Grade
Jane	75	C
Dave	56	F
Jack	90	A
Sue	68	D

Operators

Arithmetic operators

Arithmetic	Symbol	Example
Addition	+	`Xplus = 4+2;`
Subtraction	–	`Xminus = 4-2;`
Multiplication	*	`Xmult = 4*2;`
Division	/	`Xdiv = 4/2;`
Exponents	**	`Xexp = 4**2;`
Negative numbers	–	`Xneg = -2;`

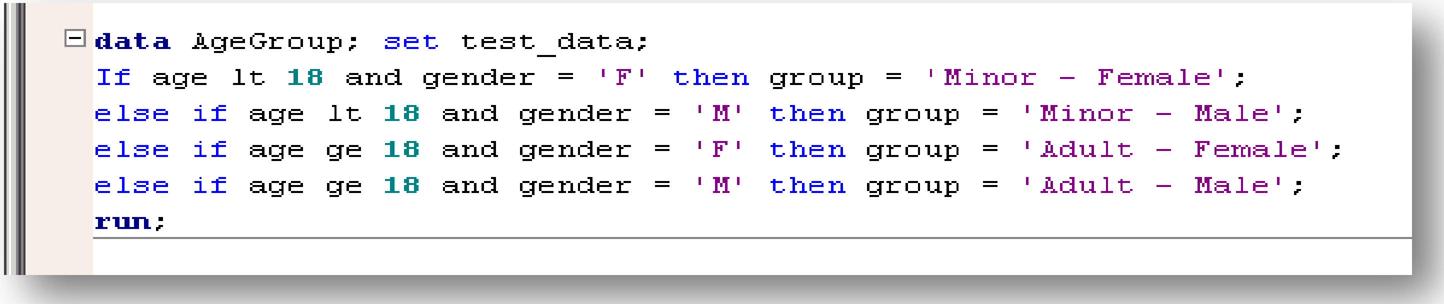
Comparison operators

Logical comparison	Mnemonic	Symbol
Equal to	EQ	=
Not equal to	NE	^= or ~=
Less than	LT	<
Less than or equal to	LE	<=
Greater than	GT	>
Greater than or equal to	GE	>=
Equal to one in a list	IN	
Not equal to any in a list	NOT IN	

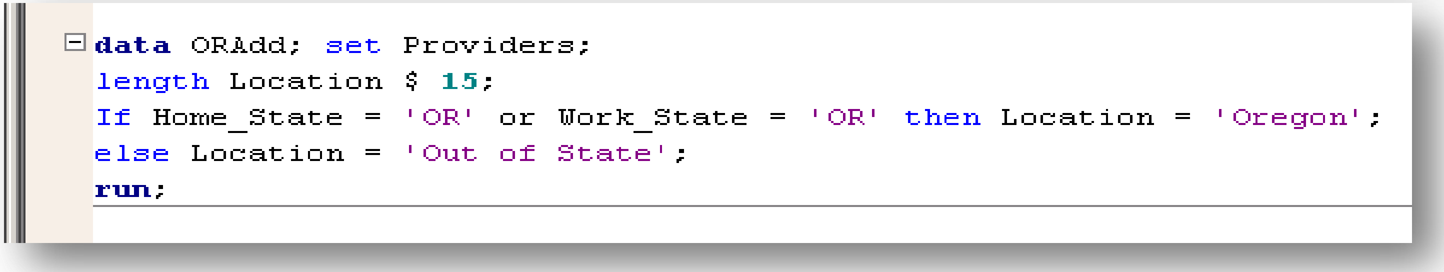
Note: <> also used for not equal to, but only in PROC SQL

Logical operators

Boolean operator
And
Or
Not



```
data AgeGroup; set test_data;
If age lt 18 and gender = 'F' then group = 'Minor - Female';
else if age lt 18 and gender = 'M' then group = 'Minor - Male';
else if age ge 18 and gender = 'F' then group = 'Adult - Female';
else if age ge 18 and gender = 'M' then group = 'Adult - Male';
run;
```



```
data ORAdd; set Providers;
length Location $ 15;
If Home_State = 'OR' or Work_State = 'OR' then Location = 'Oregon';
else Location = 'Out of State';
run;
```

Logical operators

- POP QUIZ:

```
If A or B and C;
```

is the same as…

```
If (A or B) and C;
If A or (B and C);
```

Logical operators

- POP QUIZ:

```
If A or B and C;
```

is the same as…

```
If (A or B) and C;
If A or (B and C);
```

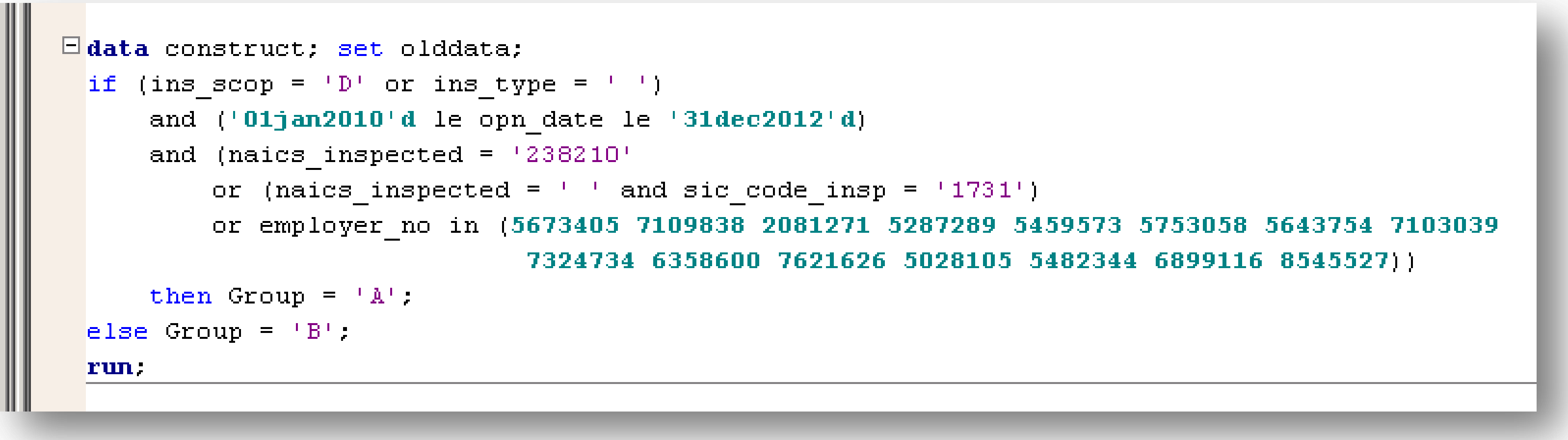
Order of operations

1. Arithmetic operators
2. Comparison operators (<, >, =, LIKE, etc.)
3. Logical operators
    a. NOT
    b. AND
    c. OR

Use parentheses to control the order of operations

If, Then, Else

- Logical conditions can be as complicated as you need them to be
  - Just make sure your order of operations is correct



```
data construct; set olddata;
if (ins_scop = 'D' or ins_type = ' ')
    and ('01jan2010'd le opn_date le '31dec2012'd)
    and (naics_inspected = '238210'
        or (naics_inspected = ' ' and sic_code_insp = '1731')
        or employer_no in (5673405 7109838 2081271 5287289 5459573 5753058 5643754 7103039
                            7324734 6358600 7621626 5028105 5482344 6899116 8545527))
    then Group = 'A';
else Group = 'B';
run;
```

Subsetting Datasets

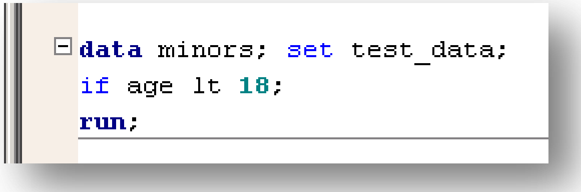
Using IF and Where

Subsetting datasets

- Can use IF or WHERE statements to only include observations you need
  - Both IF and WHERE statements can be used within DATA step if using SET statement to read in SAS data
  - IF statement must be used within DATA step if using INPUT statement to read raw data
  - WHERE statement must be used within PROC step

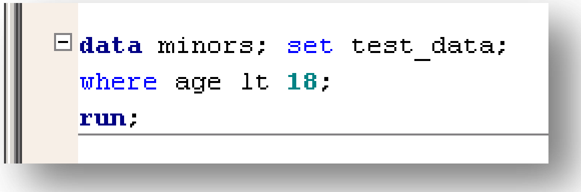
Subsetting datasets

- Can use either IF or WHERE in a DATA step with SET statement



```
data minors; set test_data;
if age lt 18;
run;
```

=



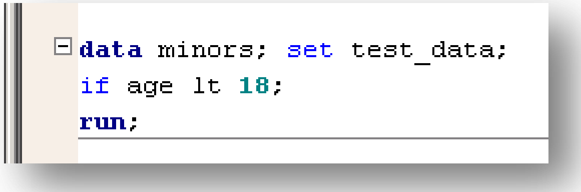
```
data minors; set test_data;
where age lt 18;
run;
```

- In both examples “Minors” dataset will only include observations where age is less than 18

Subsetting datasets

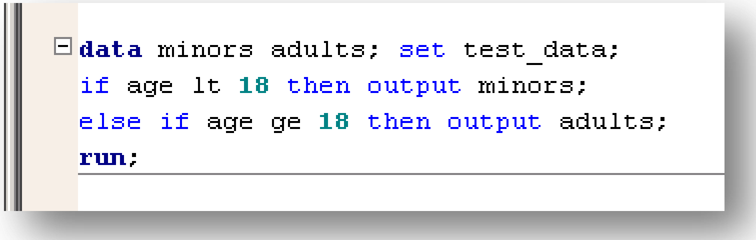
- Think of `if age lt 18;` as short for

  `if age lt 18 then output;`



```
data minors; set test_data;
if age lt 18;
run;
```

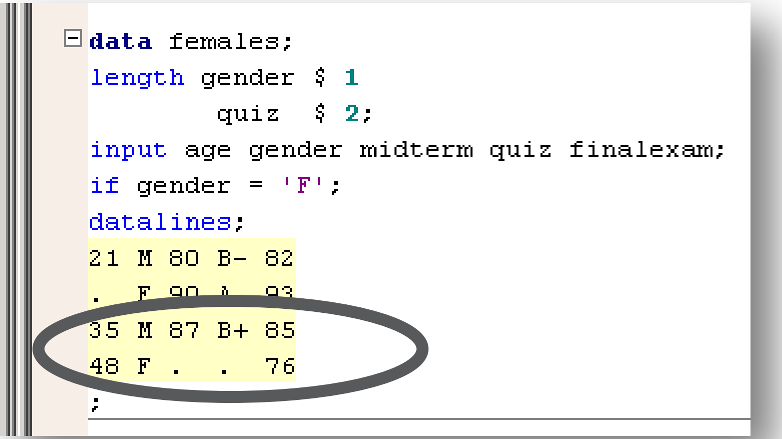
- Can output to multiple datasets using IF/THEN logic



```
data minors adults; set test_data;
if age lt 18 then output minors;
else if age ge 18 then output adults;
run;
```

Subsetting IF

- Use IF in the DATA step to bring in only the selected observations when using an INPUT statement

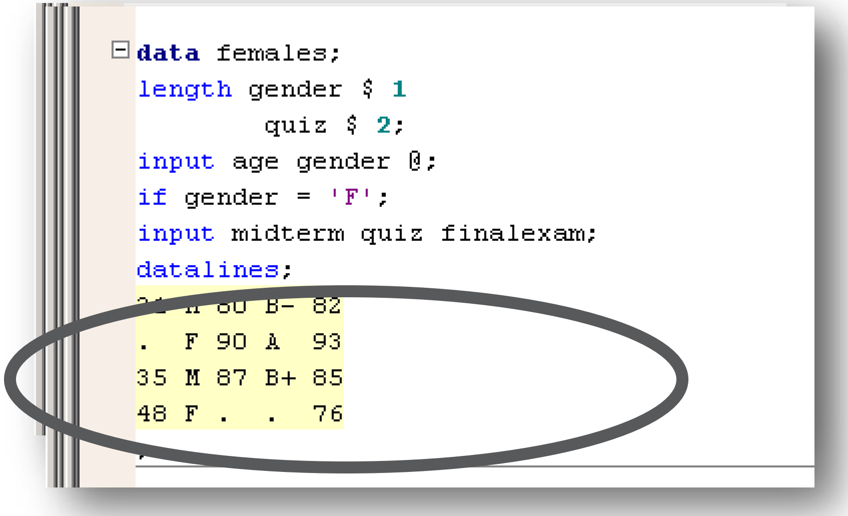


```
data females;
length gender $ 1
       quiz   $ 2;
input age gender midterm quiz finalexam;
if gender = 'F';
datalines;
21 M 80 B- 82
.  F 90 A  93
35 M 87 B+ 85
48 F .  .  76
;
```

- “Females” dataset will only include observations where gender = ‘F’

Subsetting IF

- To improve efficiency, read the value of gender before using it to subset the data

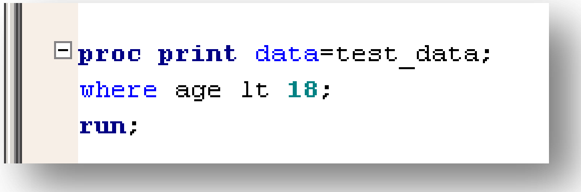


```
data females;
length gender $ 1
       quiz $ 2;
input age gender @;
if gender = 'F';
input midterm quiz finalexam;
datalines;
21 M 80 B- 82
.  F 90 A  93
35 M 87 B+ 85
48 F .  .  76
;
```

- Must use a trailing @ sign
  - (See chapter 21, section 11)

WHERE statement

- Use a WHERE statement in a PROC step to only include selected observations



```
proc print data=test_data;
where age lt 18;
run;
```

- “Test_data” dataset still includes all observations
- Only observations where age is less than 18 will be included in the calculations and output of the procedure

WHERE statement

- □ WHERE statement allows for additional operators

Operator	Description	Example	Matches
IS MISSING	Matches a missing value	`where gender is missing;`	A missing character value
IS NULL	Equivalent to IS MISSING	`where age is null;`	A missing numeric value
BETWEEN AND	An inclusive range	`where age between 20 and 40;`	All values between 20 and 40, including 20 and 40
CONTAINS	Matches a substring	`where name contains 'MAC';`	MACON, IMMACULATE
LIKE	Matching with wildcards	`where name like 'R_N%';`	RON, RONALD, RUN, RUNNING
=*	Phonetic matching	`where name =* 'NICK';`	NICK, NACK, NIKKI

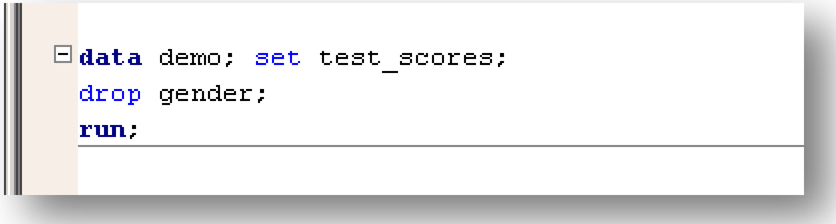
Selecting variables
Using DROP and KEEP

Selecting variables

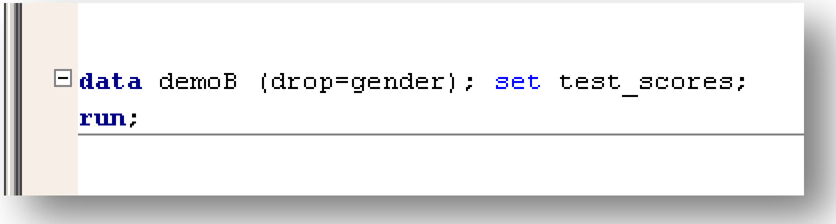
- By default, SAS will keep all variables of the input dataset
- Use DROP to exclude certain variables from the output dataset
- Use KEEP to include only certain variables from the output dataset

Selecting variables

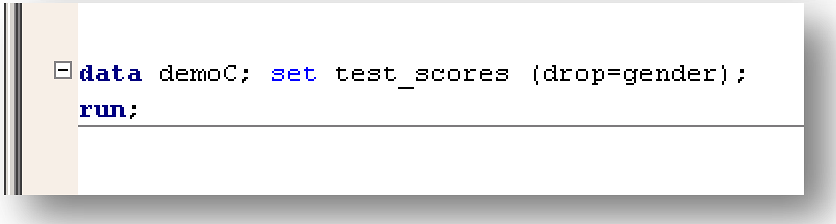
- Can be written as statements in a DATA step or as DROP= / KEEP= DATA step options



```
data demo; set test_scores;
drop gender;
run;
```



```
data demoB (drop=gender); set test_scores;
run;
```

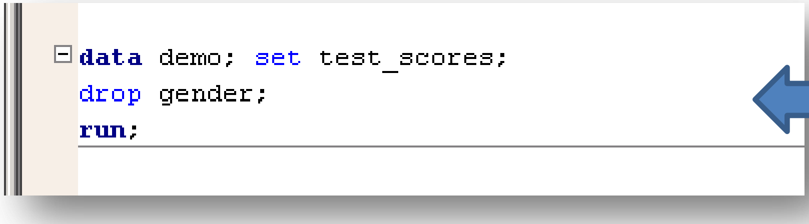


```
data demoC; set test_scores (drop=gender);
run;
```

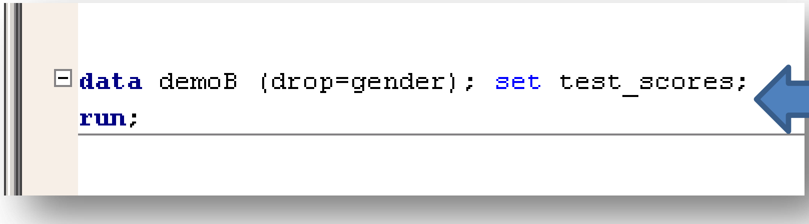
Which method you use will affect which variables are available within the DATA step as well as processing efficiency.

Selecting variables

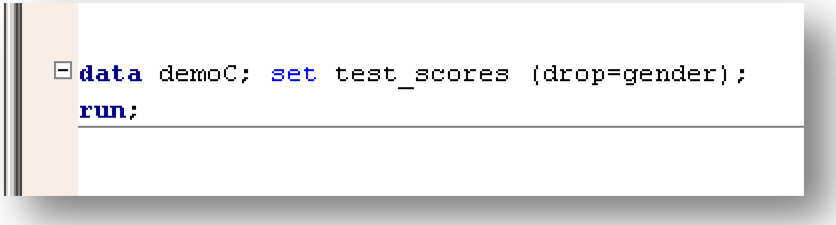
- Can be written as statements in a DATA step or as DROP= / KEEP= DATA step options



```
data demo; set test_scores;
drop gender;
run;
```



```
data demoB (drop=gender); set test_scores;
run;
```



```
data demoC; set test_scores (drop=gender);
run;
```

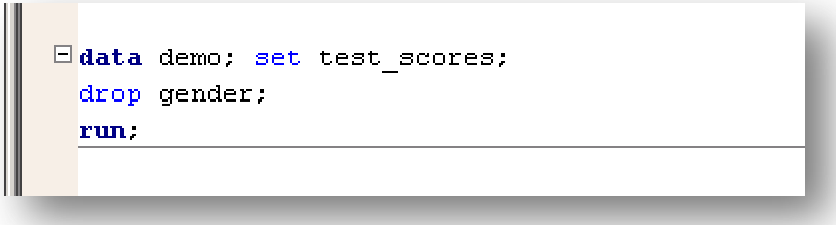
All variables (including gender) will be read to into working memory. Gender will be excluded when ‘DEMO’ dataset is written .

Gender available for use in the DATA step

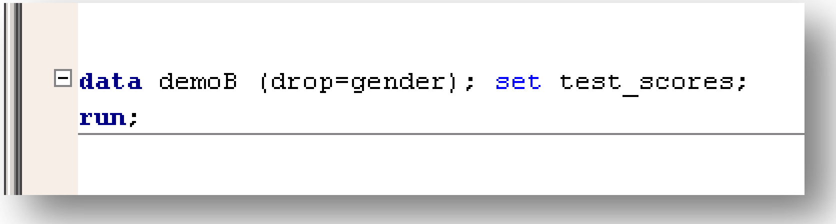
Less processing efficiency

Selecting variables

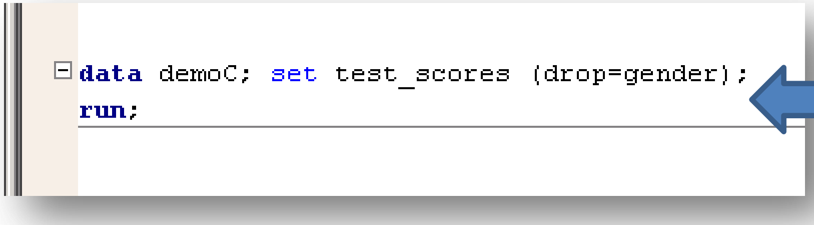
- Can be written as statements in a DATA step or as DROP= / KEEP= DATA step options



```
data demo; set test_scores;
drop gender;
run;
```



```
data demoB (drop=gender); set test_scores;
run;
```



```
data demoC; set test_scores (drop=gender);
run;
```

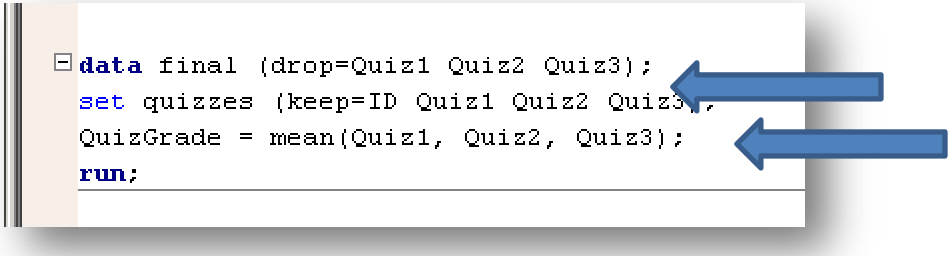
Gender will **not** be read to into working memory (and will be excluded when 'DEMO' dataset is written)

Gender **not** available for use in the DATA step

More processing efficiency

Selecting variables

- Can use DROP= and KEEP= together



```
data final (drop=Quiz1 Quiz2 Quiz3);
set quizzes (keep=ID Quiz1 Quiz2 Quiz3);
QuizGrade = mean(Quiz1, Quiz2, Quiz3);
run;
```

ID, Quiz1, Quiz2, and Quiz3 will be read into working memory (and available for use in the DATA step)

Only ID and QuizGrade will be written to the "Final" dataset

Processing Data Iteratively

Processing Data Iteratively

1. Do Loop Processing

2. SAS Array Processing

3. Using SAS Arrays

Example

An Study employee wants to compare the interest for yearly versus quarterly compounding on a $50,000 investment made for one year at 4.5 percent interest.

How much money will the employee accrue in each situation?

Repetitive Coding

```
data compound;
   Amount=50000;
   Rate=.045;
   Yearly=Amount*Rate;
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
   Quarterly+((Quarterly+Amount)*Rate/4);
run;
proc print data=compound noobs;
run;
```

PROC PRINT Output

```
Amount      Rate     Yearly    Quarterly

 50000     0.045      2250      2288.25
```

p207d01

Repetitive Coding

What if the employee wants to determine annual and quarterly compounded interest for a period of 20 years (80 quarters)?

```
data compound;
    Amount=50000;
    Rate=.045;
    Yearly +(Yearly+Amount)*Rate;
    .
    .
    .
    Yearly +(Yearly+Amount)*Rate;
    Quarterly+((Quarterly+Amount)*Rate/4);
    .
    .
    .
    Quarterly+((Quarterly+Amount)*Rate/4);
run;
```

20x

80x

DO Loop Processing

Use DO loops to perform the repetitive calculations.

```
data compound(drop=i);
   Amount=50000;
   Rate=.045;
   do i=1 to 20;
      Yearly +(Yearly+Amount)*Rate;
   end;
   do i=1 to 80;
      Quarterly+((Quarterly+Amount)*Rate/4);
   end;
run;
```

p207d02

Various Forms of Iterative DO Loops

There are several forms of iterative DO loops that execute the statements between the DO and END statements repetitively.

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

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

The Iterative DO Statement

General form of an iterative DO statement:

DO *index-variable*= *start* TO *stop* <BY *increment*>**;**

The values of *start*, *stop*, and *increment*

- must be numbers or expressions that yield numbers
- are established before executing the loop
- if omitted, *increment* defaults to 1.

The Iterative DO Statement

Index-variable details:

- The *index-variable* is written to the output data set by default.
- At the termination of the loop, the value of *index-variable* is one *increment* beyond the *stop* value.

⚠ Modifying the value of *index-variable* affects the number of iterations, and might cause infinite looping or early loop termination.

Poll

Quiz

Quiz

What are the final values of the index variables after the following DO loops execute?

```
do i=1 to 5;
   …
end;                    1 2 3

do j=2 to 8 by 2;
   …
end;

do k=10 to 2 by -2;
    …
end;
```

The final values are highlighted.

Quiz – Correct Answer

What are the final values of the index variables after the following DO statements execute?

The final values are highlighted.

```
do i=1 to 5;
   …
end;                    1 2 3 4 5 6

do j=2 to 8 by 2;
   …
end;                    2 4 6 8 10

do k=10 to 2 by -2;
    …
end;                    10 8 6 4 2 0
```

The Iterative DO Statement

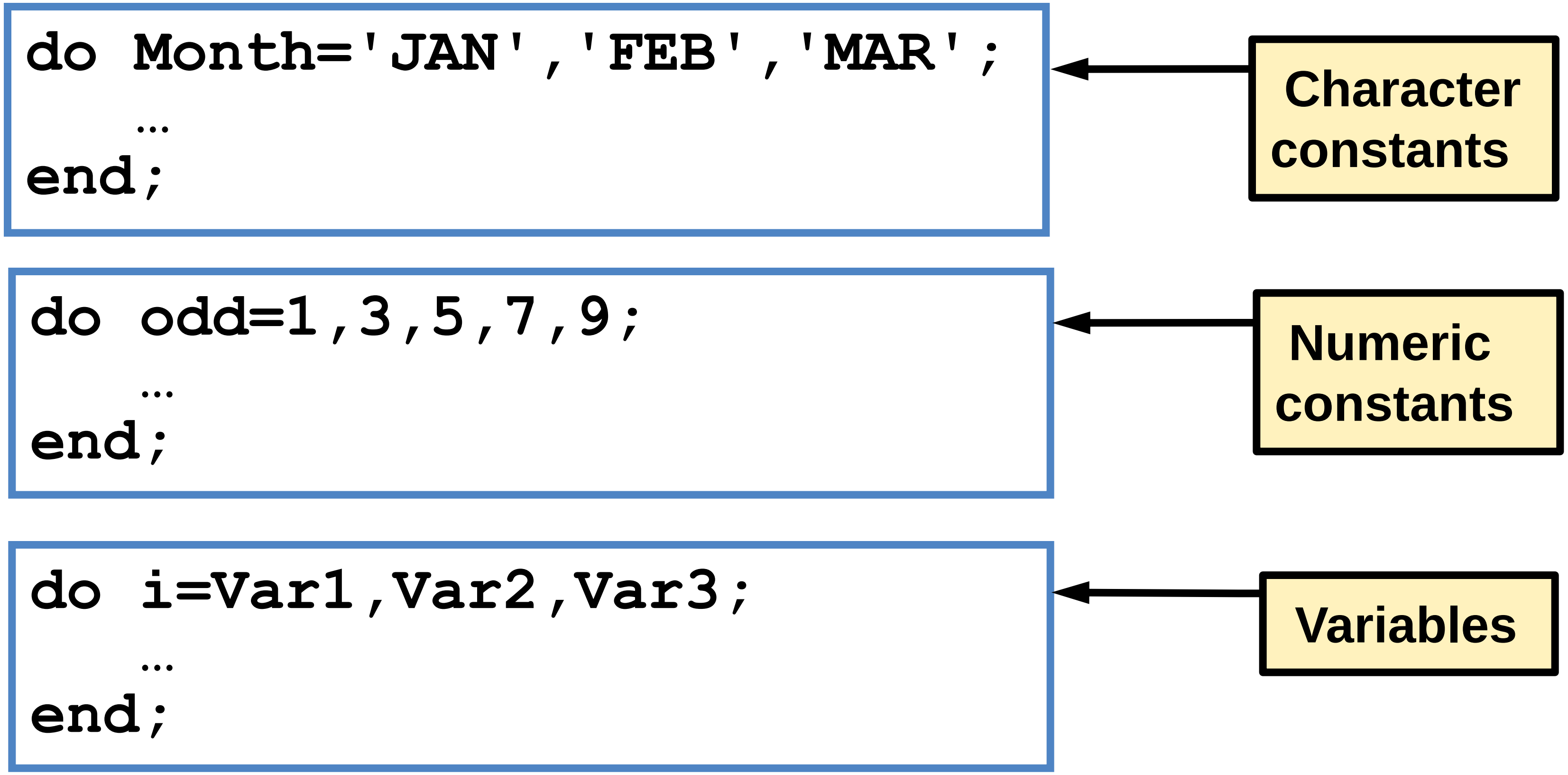
General form of an iterative DO statement with an *item-list*:

```
DO index-variable=item-1 <,…item-n>;
```

- The DO loop is executed once for each item in the list.
- The list must be comma separated.

Sample DO Loops with Item Lists

Items in the list can be all numeric or all character constants, or they can be variables.



```
do Month='JAN','FEB','MAR';
   …
end;
```

Character constants

```
do odd=1,3,5,7,9;
   …
end;
```

Numeric constants

```
do i=Var1,Var2,Var3;
   …
end;
```

Variables

Example

On January 1 of each year, an Study   employee invests $5,000 in an account. Determine the value
of the account after three years based on a constant annual interest rate of 4.5 percent, ting in 2008.

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Initialize PDV

PDV

Year	R Capital	D _N_
.	0	1

`Capital` is used in a sum statement, so it is automatically initialized to zero and retained.

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

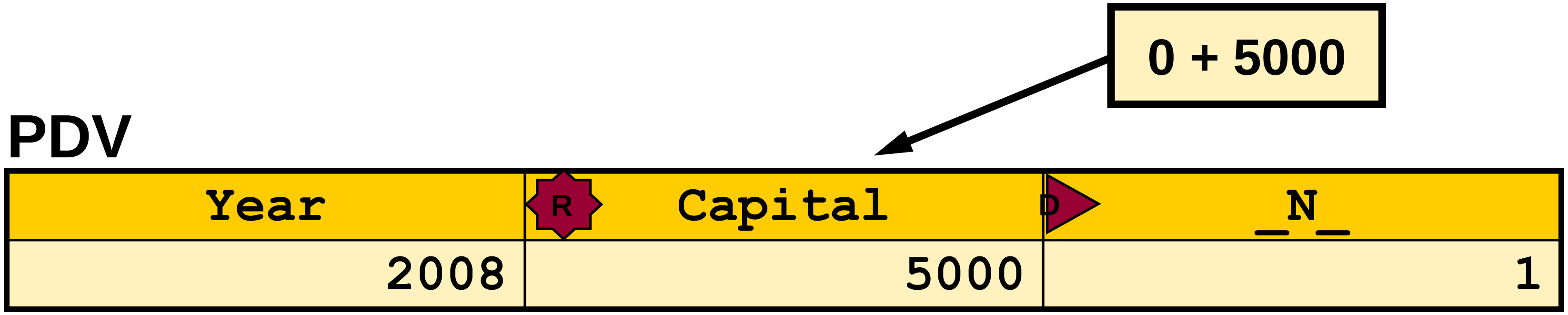
Is Year out of range?

PDV

Year	R Capital	D _N_
2008	0	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



PDV

Year	Capital	_N_
2008	5000	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

5000 + (5000 * .045)

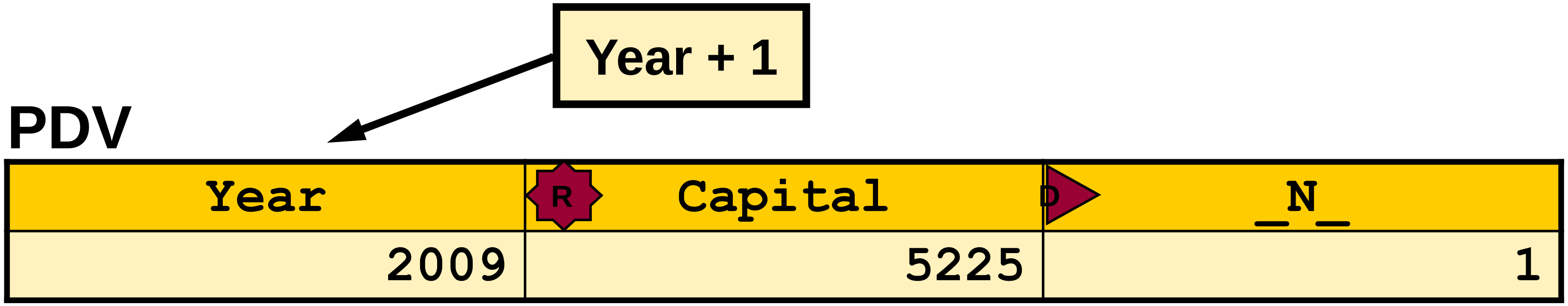
PDV

Year	R Capital	D _N_
2008	5225	1

...

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



Year	Capital	_N_
2009	5225	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Is Year out of range?

PDV

Year	R Capital	D _N_
2009	5225	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

5225 + 5000

PDV

Year	R Capital	D _N_
2009	10225	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

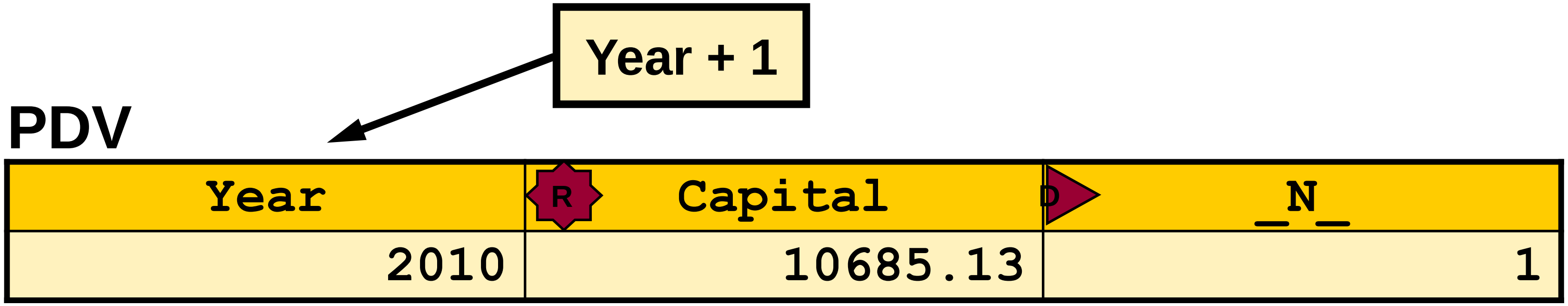
10225 + (10225 * .045)

PDV

Year	R Capital	D _N_
2009	10685.13	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```



Year + 1

PDV

Year	R Capital	D _N_
2010	10685.13	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Is Year out of range?

PDV

Year	R Capital	D _N_
2010	10685.13	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

10685.13 + 5000

PDV

Year	R Capital	D _N_
2010	15685.13	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

15685.13 + (15685.13 * .045)

PDV

Year	R Capital	D _N_
2010	16390.96	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Year + 1

PDV

Year	R Capital	D _N_
2011	16390.96	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Is Year out of range?

PDV

Year	R Capital	D _N_
2011	16390.96	1

Execution: Performing Repetitive Calculations

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
```

Implicit OUTPUT;
No Implicit RETURN;

PDV

Year	R Capital	D _N_
2011	16390.96	1

Output: Performing Repetitive Calculations

```
proc print data=invest noobs;
run;
```

PROC PRINT Output

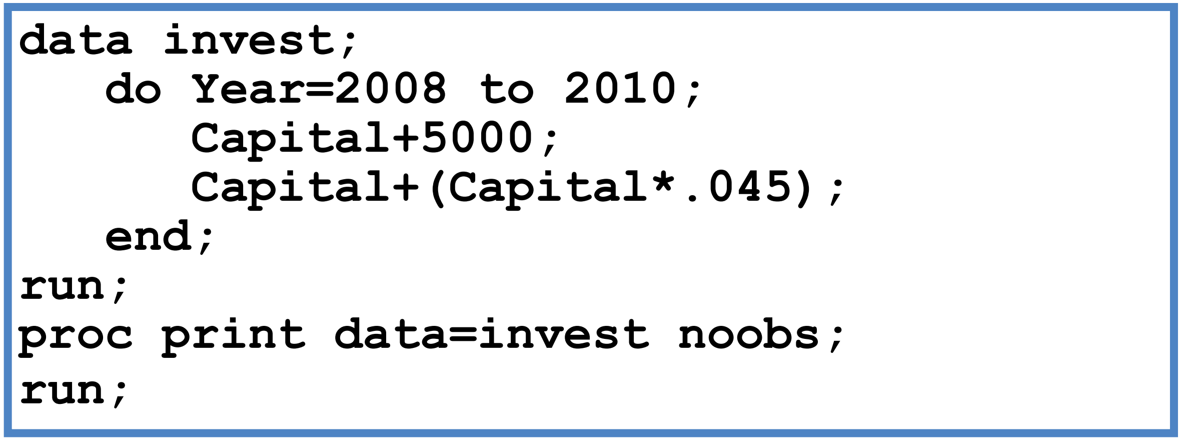
```
        Year      Capital

        2011     16390.96
```

Poll

Quiz

Quiz

How can you generate a separate observation for each year?



```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
run;
```

Quiz – Correct Answer

How can you generate a separate observation for each year? **Place an explicit OUTPUT statement inside the DO loop.**

```
data invest;
   do Year=2008 to 2010;
      Capital+5000;
      Capital+(Capital*.045);
        output;
   end;
run;
proc print data=invest noobs;
run;
```

PROC PRINT Output

```
Year      Capital

2008      5225.00
2009     10685.13
2010     16390.96
```

There is no observation for 2011.

Question
&
Answer

Example

Recall the example that forecasts the growth of several departments at Study . Modify the forecasting application to use a DO loop to eliminate redundant code.

Listing of **`Study.growth`**

```
                                              Total_
Department              Employees          Increase

Administration              34               0.25
Engineering                  9               0.30
IS                          25               0.10
Marketing                   20               0.20
Sales                      201               0.30
Sales Management            11               0.10
```

A Forecasting Application (Review)

```
data forecast;
   set Study.growth;
   Year=1;
   Total_Employees=Total_Employees*(1+Increase);
   output;
   Year=2;
   Total_Employees=Total_Employees*(1+Increase);
   output;
run;
proc print data=forecast noobs;
run;
```

What if you want to forecast growth over the next six years?

Use a DO Loop to Reduce Redundant Code

```
data forecast;
   set Study.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;

proc print data=forecast noobs;
run;
```

p207d05

Output

Partial PROC PRINT Output

```
                          Total_
Department                Employees    Increase    Year

Administration              42.500       0.25        1
Administration              53.125       0.25        2
Administration              66.406       0.25        3
Administration              83.008       0.25        4
Administration             103.760       0.25        5
Administration             129.700       0.25        6
Engineering                 11.700       0.30        1
Engineering                 15.210       0.30        2
Engineering                 19.773       0.30        3
Engineering                 25.705       0.30        4
Engineering                 33.416       0.30        5
Engineering                 43.441       0.30        6
IS                          27.500       0.10        1
```

Poll

Quiz ○ ☑ ○

Quiz

What stop value would you use in the DO loop to determine the number of years that it would take for the Engineering department to exceed 75 people?

```
data forecast;
   set Study.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;
proc print data=forecast noobs;
run;
```

p207d05

Quiz – Correct Answer

What stop value would you use in the DO loop to determine the number of years it would take for the Engineering department to exceed 75 people? **Unknown.**

```
data forecast;
   set Study.growth;
   do Year=1 to 6;
      Total_Employees=
         Total_Employees*(1+Increase);
      output;
   end;
run;
proc print data=forecast noobs;
run;
```

Us

a (

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 executed a specific number of times.

⚠ To avoid infinite loops, be sure that the specified condition will be met.

The DO WHILE Statement

The DO WHILE statement executes statements in a DO loop repetitively while a condition is true.

General form of the DO WHILE loop:

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

The value of *expression* is evaluated at the **top** of the loop.

The statements in the loop never execute if *expression* is initially false.

The DO UNTIL Statement

The DO UNTIL statement executes statements in a DO loop repetitively until a condition is true.

General form of the DO UNTIL loop:

```
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 once.

Although the condition is placed at the top of the loop, it is evaluated at the bottom of the loop.

Example

Determine the number of years that it would take for an account to exceed $1,000,000 if $5,000 is invested annually at 4.5 percent.

Using the DO UNTIL Statement

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

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

PROC PRINT Output

```
      Capital     Year

$1,029,193.17      52
```

Poll

Quiz

Quiz

How can you generate the same result with a DO WHILE statement?

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

proc print data=invest noobs;
   format capital dollar14.2;
run;
```

Quiz – Correct Answer

How could you generate the same result with a DO WHILE statement? **Change the DO UNTIL statement to a DO WHILE statement and modify the condition.**

```
data invest;
   do while(Capital<=1000000);
      Year+1;
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;

proc print data=invest noobs;
   format capital dollar14.2;
run;
```

Iterative DO Loop with a Conditional Clause

You can combine DO WHILE and DO UNTIL statements with the iterative DO statement.

General form of the iterative DO loop with a conditional clause:

```
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.

Using DO UNTIL with an Iterative DO Loop

Determine the value of the account again. Stop the loop if 30 years is reached or more than $250,000 is accumulated.

```
data invest;
   do Year=1 to 30 until(Capital>250000);
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
   format capital dollar14.2;
run;
```

In a DO UNTIL loop, the condition is checked **before** the index variable is incremented.

PROC PRINT Output

```
Year             Capital

 27          $264,966.67
```

p207d07

Using DO WHILE with an Iterative DO Loop

Determine the value of the account again, but this time use a DO WHILE statement.

```
data invest;
   do Year=1 to 30 while(Capital<=250000);
      Capital+5000;
      Capital+(Capital*.045);
   end;
run;
proc print data=invest noobs;
   format capital dollar14.2;
run;
```

PROC PRINT Output

```
Year            Capital

 28         $264,966.67
```

In a DO WHILE loop, the condition is checked **after** the index variable is incremented.

p207d07

Nested DO Loops

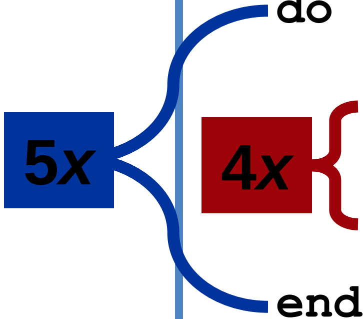
Nested DO loops are loops within loops.

- Be sure to use different index variables for each loop.
- Each DO statement must have a corresponding END statement.
- The inner loop executes completely for each iteration of the outer loop.

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

Example

Create one observation per year for five years, and show the earnings if you invest $5,000 per year with 4.5 percent annual interest compounded quarterly.



```
data invest(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=invest noobs;
run;
```

Output: Nested DO Loops

PROC PRINT Output

```
Year       Capital

  1        5228.83
  2       10696.95
  3       16415.32
  4       22395.39
  5       28649.15
```

Poll

Quiz

Quiz

How can you generate one observation for each quarterly amount?

```
data invest(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=invest noobs;
run;
```

Quiz – Correct Answer

How can you generate one observation for each quarterly amount?

Move the OUTPUT statement

to the inner loop and do not drop `Quarter`.

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

Partial PROC PRINT Output

```
 Year       Capital      Quarter

   1        5056.25         1
   1        5113.13         2
   1        5170.66         3
   1        5228.83         4
   2       10343.90         1
   2       10460.27         2
```

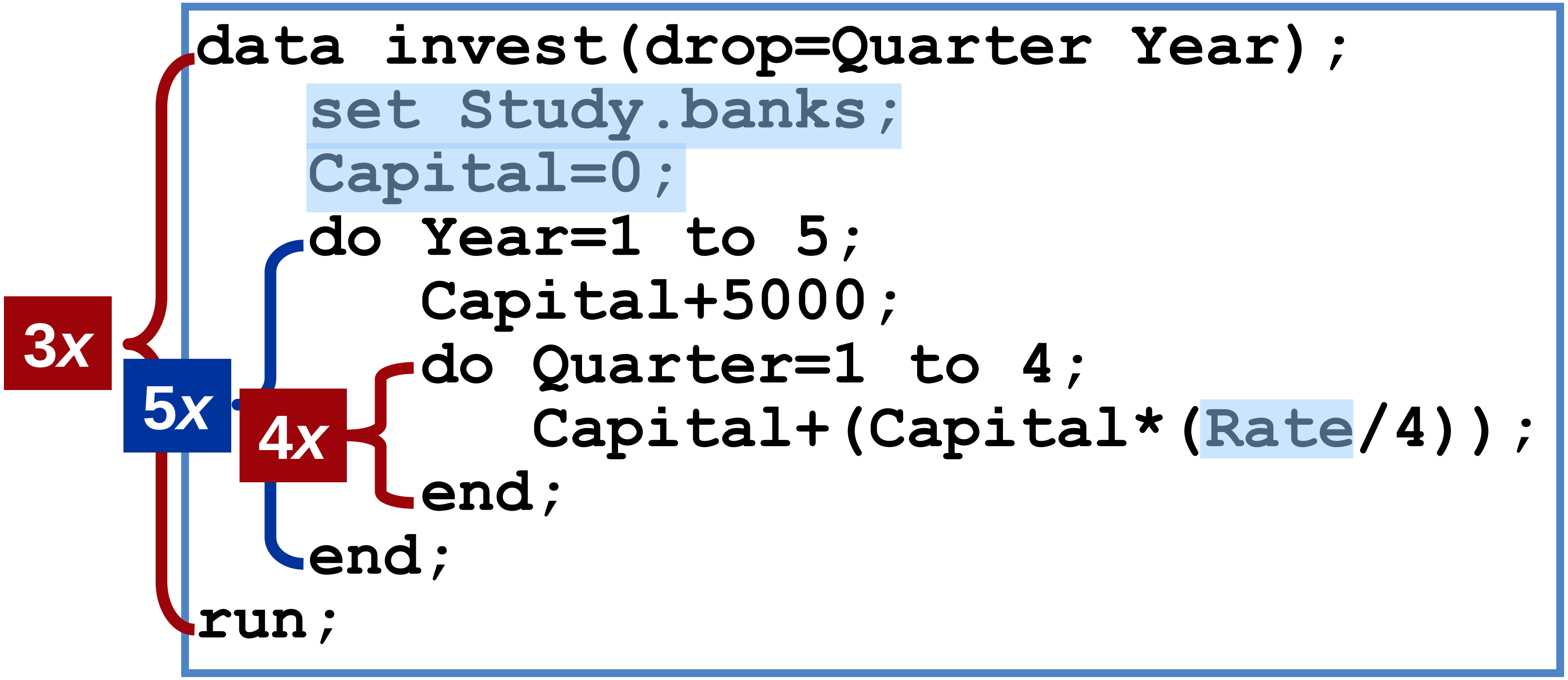
Example

Compare the final results of investing $5,000 a year for five years in three different banks that compound interest quarterly. Assume that each bank has a fixed interest rate, stored in the **`Study.banks`** data set.

Listing of **`Study.banks`**

```
           Name                       Rate

Carolina Bank and Trust             0.0318
State Savings Bank                  0.0321
National Savings and Trust          0.0328
```

Using Nested DO Loops with a SET Statement



```
data invest(drop=Quarter Year);
   set Study.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

There are three observations in **Study.banks**. Therefore, there will be three iterations of the DATA step. **Capital** must be set to zero on each iteration of the DATA step.

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set Study.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

First Iteration

0.0318

Partial PDV

Name	Rate	_N_
Carolina Bank and Trust	0.0318	1

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set Study.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

Second Iteration

0.0321

Partial PDV

Name	Rate	_N_
State Savings Bank	0.0321	2

Execution: Nested DO Loops

```
data invest(drop=Quarter Year);
   set Study.banks;
   Capital=0;
   do Year=1 to 5;
      Capital+5000;
      do Quarter=1 to 4;
         Capital+(Capital*(Rate/4));
      end;
   end;
run;
```

Third Iteration

0.0328

Partial PDV

Name	Rate	_N_
National Savings and Trust	0.0328	3

Output: Nested DO Loops

```
proc print data=invest noobs;
run;
```

PROC PRINT Output

```
Name                          Rate       Capital

Carolina Bank and Trust       0.0318    27519.69
State Savings Bank            0.0321    27544.79
National Savings and Trust    0.0328    27603.47
```

Question
&
Answer

- This exercise reinforces the concepts discussed previously.

SAS Array Processing

Objectives

- Explain the concepts of SAS arrays.
- Use SAS arrays to perform repetitive calculations.

Array Processing

You can use arrays to simplify programs that do the following:

- perform repetitive calculations
- create many variables with the same attributes
- read data
- compare variables
- perform a table lookup

Poll

Quiz

Quiz

Do you have experience with arrays in a programming language?
If so, which languages?

Example

The **`Study.employee_donations`** data set contains quarterly contribution data for each employee. Study management is considering a 25 percent matching program. Calculate each employee’s quarterly contribution, including the proposed company supplement.

Partial Listing of **`Study.employee_donations`**

```
Employee_ID  Qtr1     Qtr2     Qtr3     Qtr4
   120265       .        .        .      25
   120267      15       15       15      15
   120269      20       20       20      20
   120270      20       10        5       .
   120271      20       20       20      20
   120272      10       10       10      10
```

Performing Repetitive Calculations

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
proc print data=charity noobs;
run;
```

Partial PROC PRINT Output

```
Employee_ID      Qtr1       Qtr2       Qtr3       Qtr4

     120265        .          .          .       31.25
     120267     18.75      18.75      18.75      18.75
     120269     25.00      25.00      25.00      25.00
     120270     25.00      12.50       6.25        .
```

p207d10

Performing Repetitive Calculations

The four calculations cannot be replaced by a single calculation inside a DO loop because they are not identical.

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
proc print data=charity noobs;
run;
```

```
do i=1 to 4;
      ?
end;
```

A SAS array can be used to simplify this code.

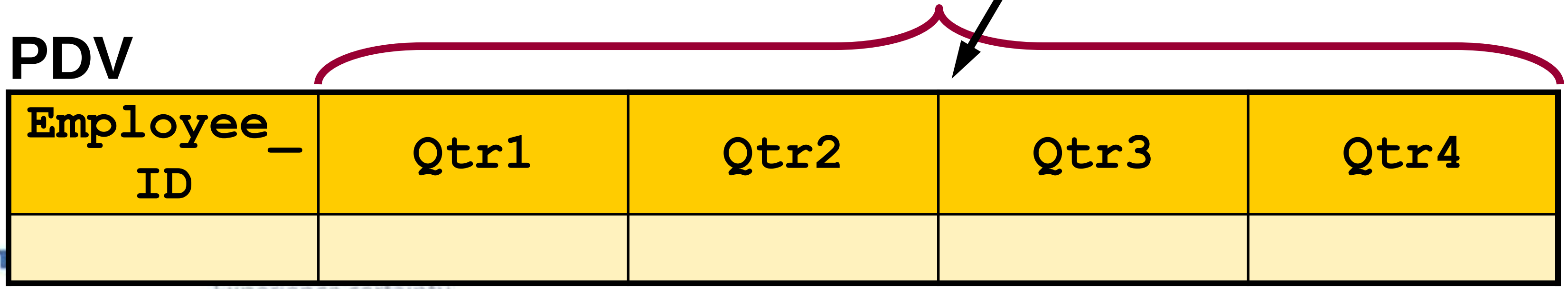
Use Arrays to Simplify Repetitive Calculations

An array provides an alternate way to access values in the PDV, which simplifies repetitive calculations.

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   Qtr1=Qtr1*1.25;
   Qtr2=Qtr2*1.25;
   Qtr3=Qtr3*1.25;
   Qtr4=Qtr4*1.25;
run;
proc print data=charity noobs;
run;
```

An array can be used to access `Qtr1`-`Qtr4`.

PDV



Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4

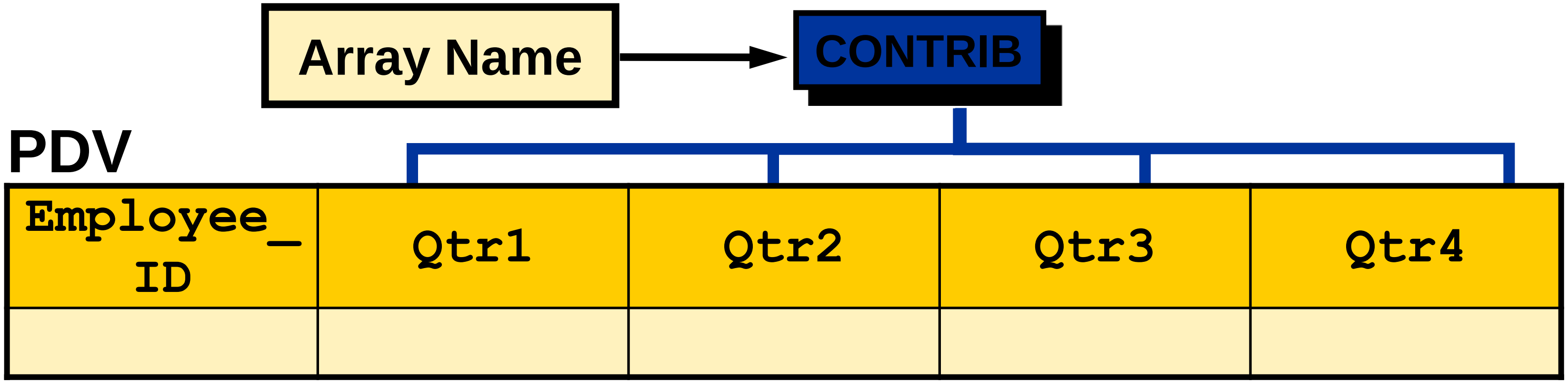
What is a SAS Array?

A *SAS array*

- is a temporary grouping of SAS variables that are arranged in a particular order
- is identified by an *array name*
- must contain all numeric or all character variables
- exists only for the duration of the current DATA step
- is **not** a variable.

Why Use a SAS Array?

Create an array named **`Contrib`** and use it to access the four numeric variables, **`Qtr1`** – **`Qtr4`**.



Array Name

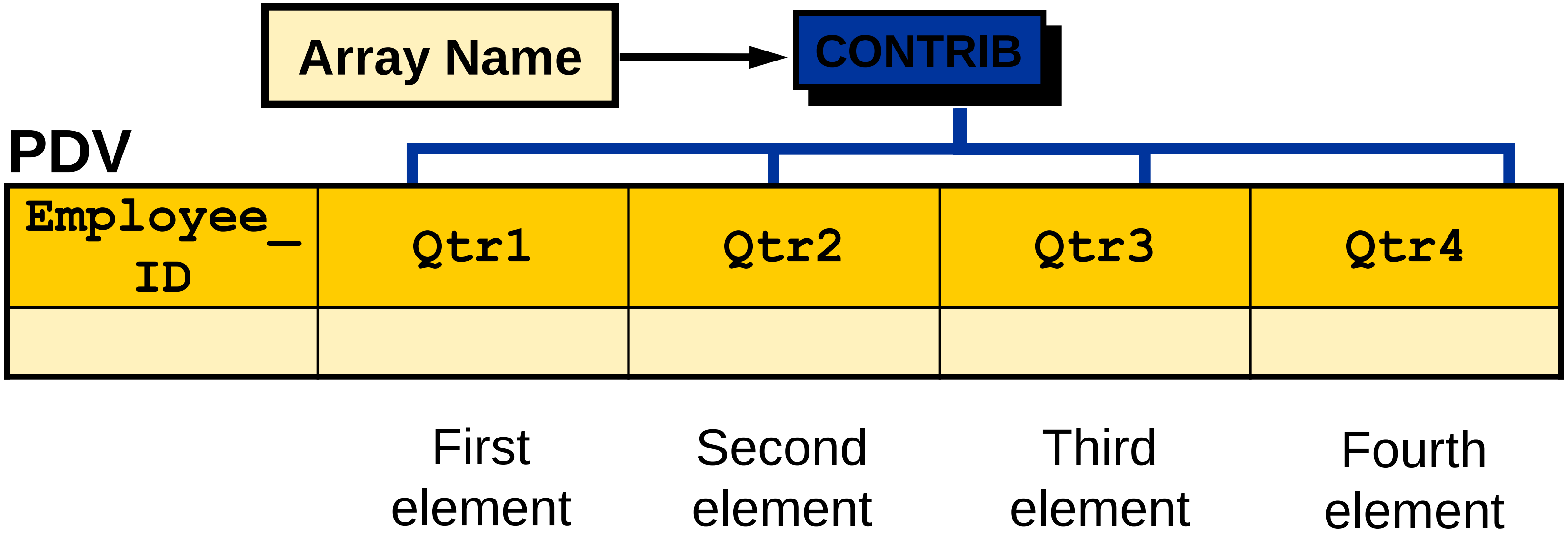
CONTRIB

PDV

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4

Array Elements

Each value in an array is called an *element.*



Array Name

CONTRIB

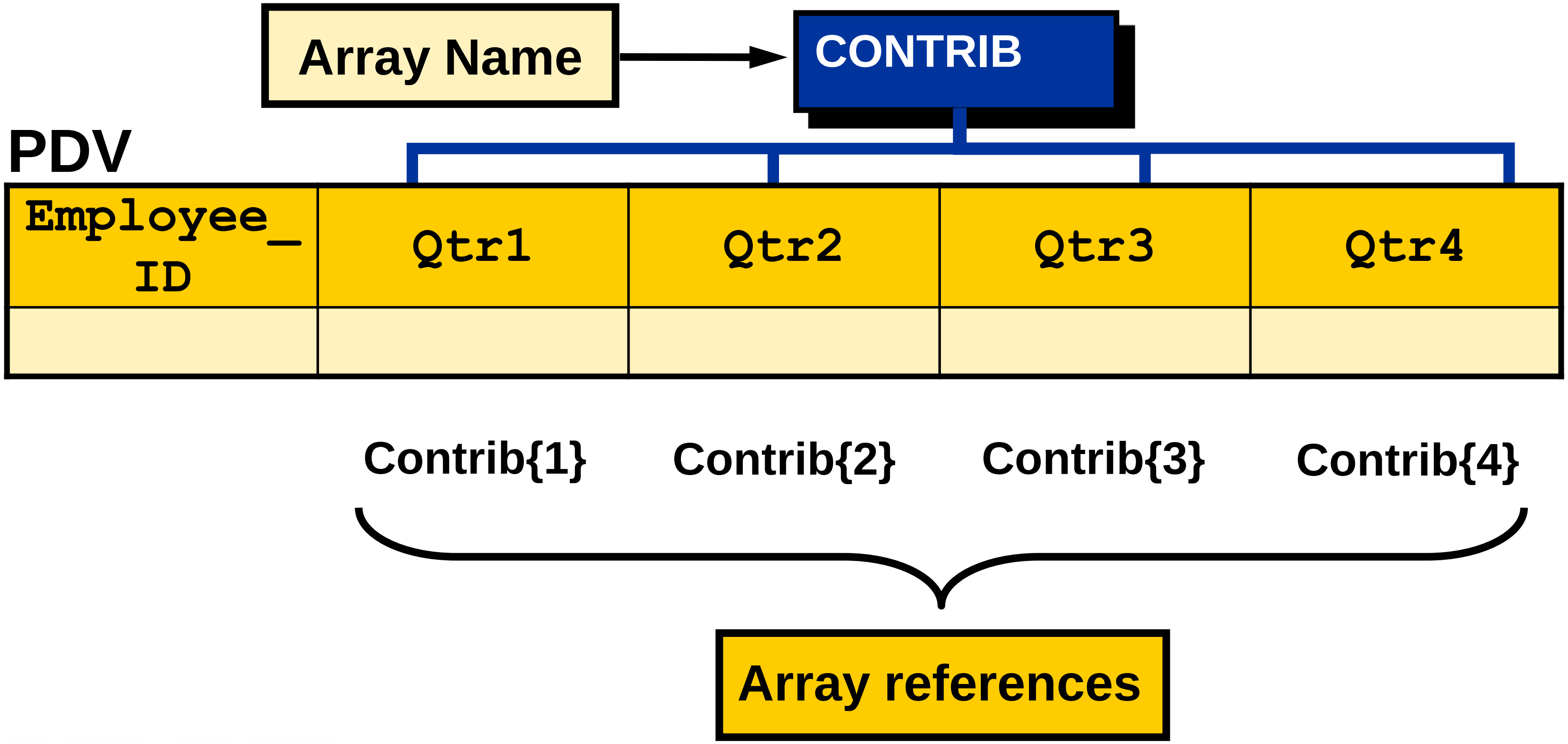
PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4

	First element	Second element	Third element	Fourth element

Referencing Array Elements

Each element is identified by a *subscript* that represents its position in the array. When you use an *array reference*, the corresponding value is substituted for the reference.



Array Name

CONTRIB

PDV

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4

Contrib{1} Contrib{2} Contrib{3} Contrib{4}

Array references

The ARRAY Statement

The ARRAY statement is a compile-time statement that defines the elements in an array. The elements are created if they do not already exist in the PDV.

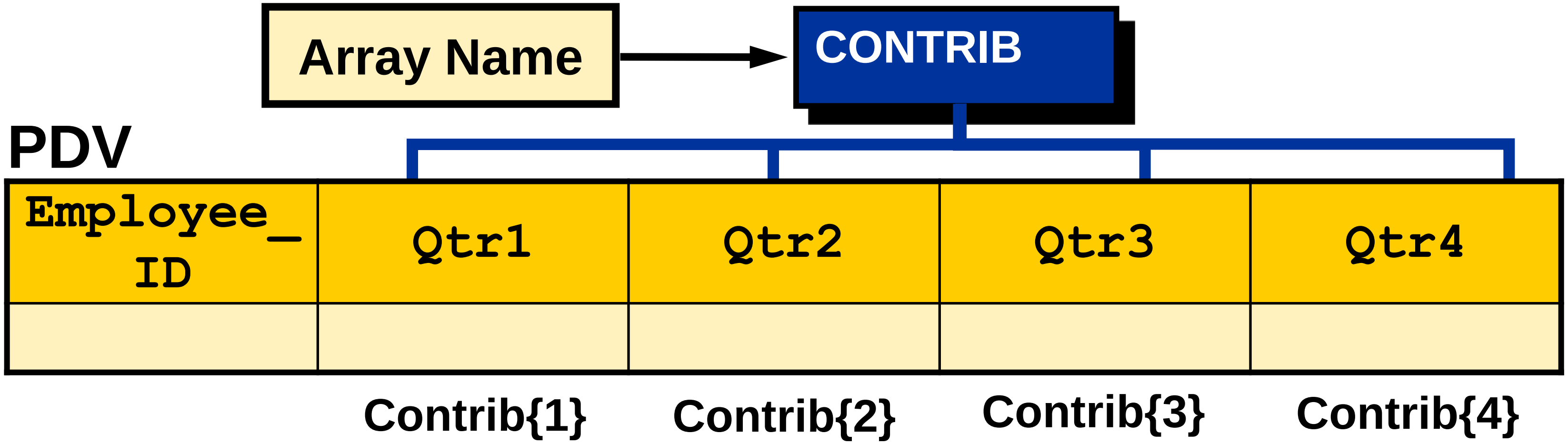
ARRAY *array-name* {*subscript*} <$> <*length*>
<*array-elements*>**;**

{subscript}	the number of elements
$	indicates character elements
length	the length of elements
array-elements	the names of elements

Defining an Array

The following ARRAY statement defines an array, **Contrib**, to access the four quarterly contribution variables.

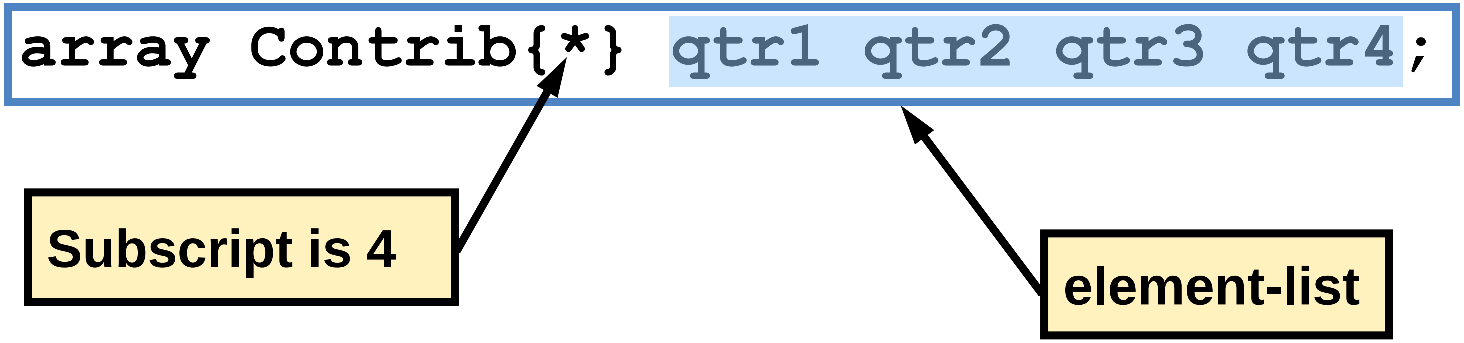
```
array Contrib{4} qtr1 qtr2 qtr3 qtr4;
```



Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4
	Contrib{1}	Contrib{2}	Contrib{3}	Contrib{4}

Defining an Array

An alternate syntax uses an asterisk instead of a subscript. SAS determines the subscript by counting the variables in the element-list. The element-list must be included.



```
array Contrib{*} qtr1 qtr2 qtr3 qtr4;
```

Subscript is 4

element-list

The alternate syntax is often used when the array elements are defined with a SAS variable list.

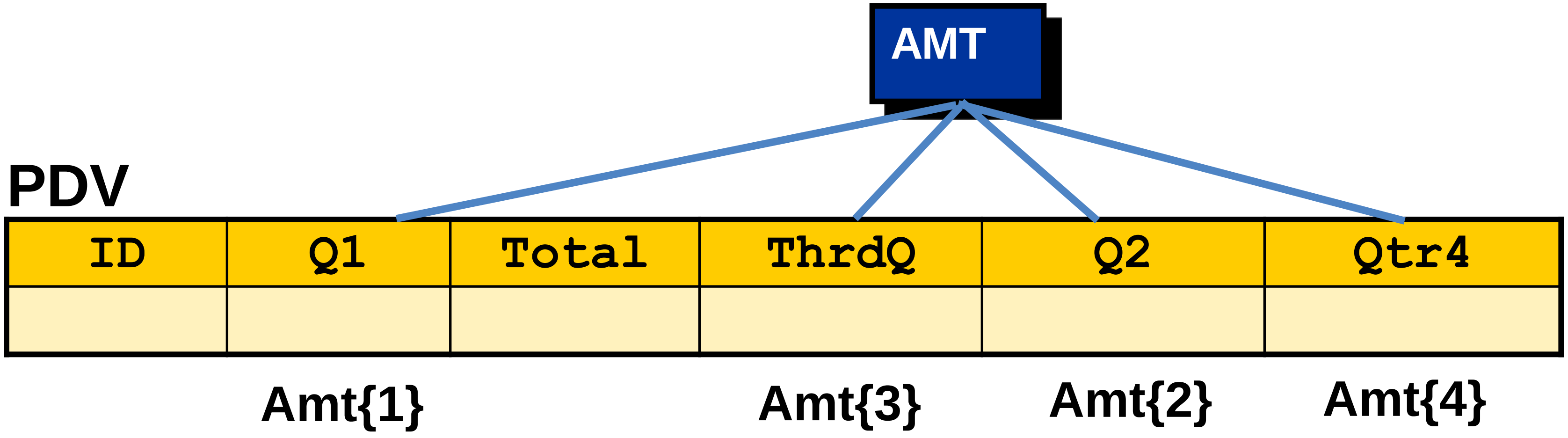
```
array Contrib{*} qtr:;
```

Defining an Array

Variables that are elements of an array do not need the following:

- to have similar, related, or numbered names
- to be stored sequentially
- to be adjacent

```
array Amt{*} Q1 Q2 ThrdQ Qtr4;
```



PDV

ID	Q1	Total	ThrdQ	Q2	Qtr4

Poll

Quiz

Quiz

Determine the cause of the error in the log.

```
data charity(keep=employee_id qtr1-qtr4);
   set Study.employee_donations;
   array Contrib1{3} qtr1-qtr4;
   array Contrib2{5} qtr:;
   /* additional SAS statements */
run;
```

Quiz – Correct Answer

Open and submit **p207a04**. View the log to determine the cause of the error. **The subscript and the number of elements in the list do not agree.**

```
data charity(keep=employee_id qtr1-qtr4);
   set Study.employee_donations;
   array Contrib1{3} qtr1-qtr4;
   array Contrib2{5} qtr:;
   /* additional SAS statements */
run;
```

The subscript and element-list must agree.

Partial SAS Log

```
177      array Contrib1{3} qtr1-qtr4;
ERROR: Too many variables defined for the dimension(s) specified
for the array Contrib1.
178      array Contrib2{5} qtr:;
ERROR: Too few variables defined for the dimension(s) specified
for the array Contrib2.
```

Using a DO Loop to Process an Array

Array processing often occurs within an iterative DO loop in the following form:

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

To reference an element, the *index-variable* is often used as a subscript:

```
array-name{index-variable}
```

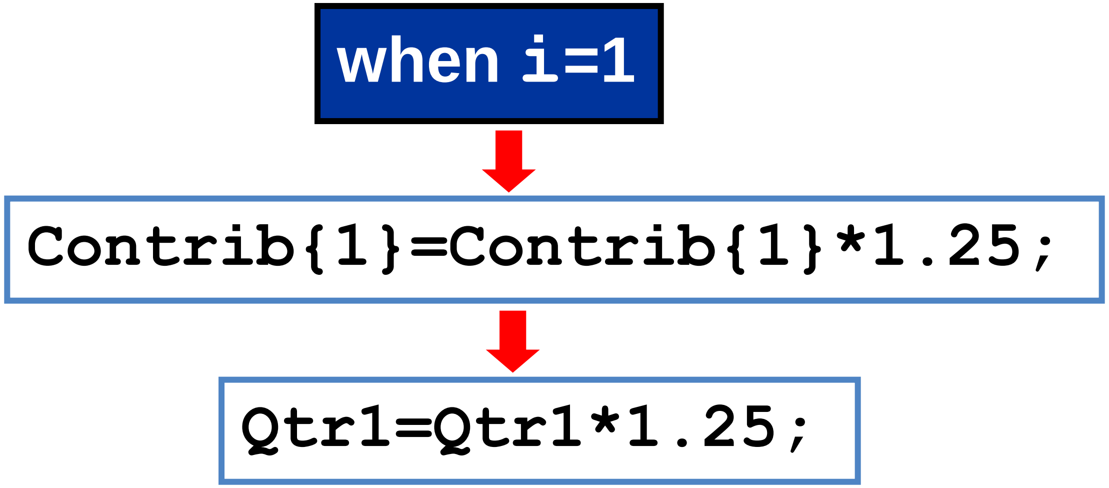
Using a DO Loop to Process an Array

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

The index variable, **i**, is not written to the output data set because it is not listed in the KEEP statement.

First Iteration of the DO Loop

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



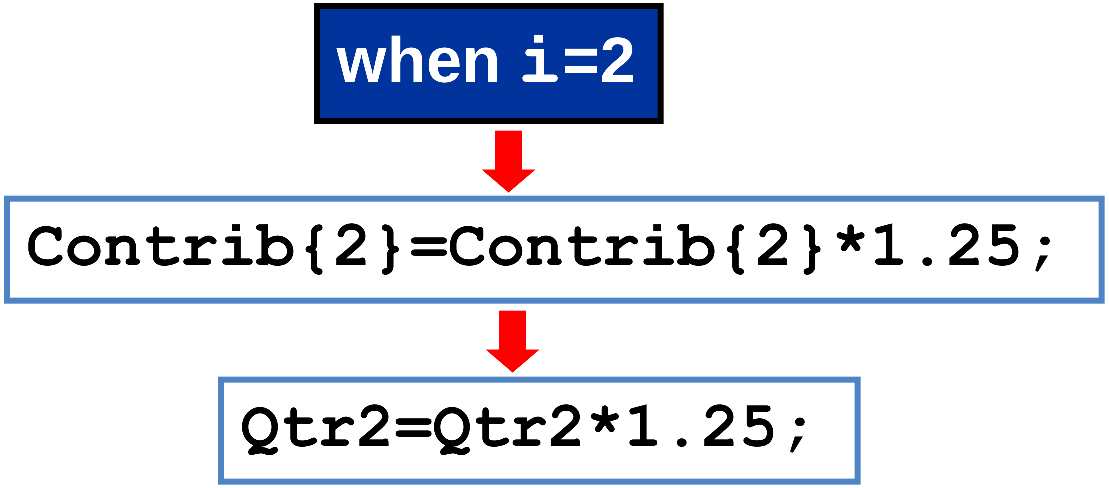
when i=1

```
Contrib{1}=Contrib{1}*1.25;
```

```
Qtr1=Qtr1*1.25;
```

Second Iteration of the DO Loop

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



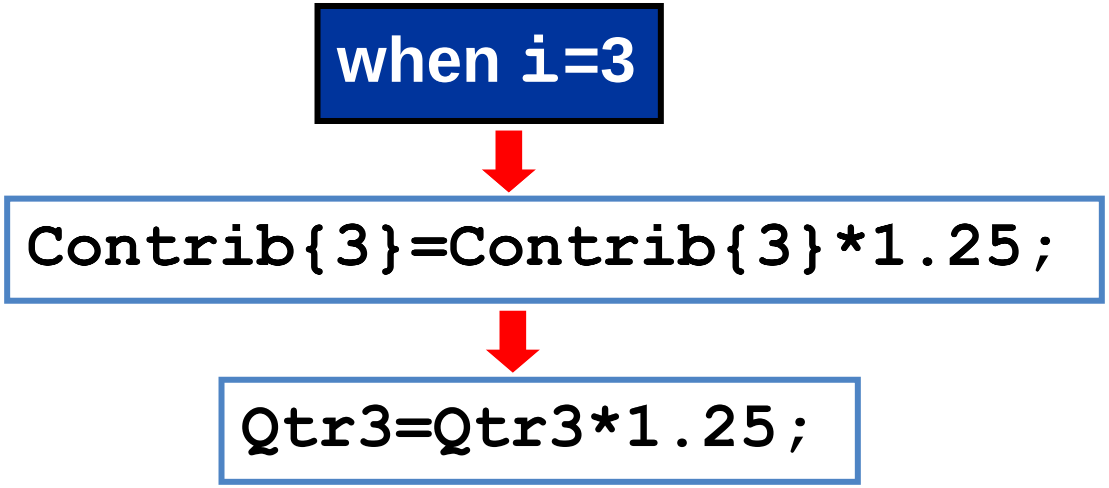
when i=2

```
Contrib{2}=Contrib{2}*1.25;
```

```
Qtr2=Qtr2*1.25;
```

Third Iteration of the DO Loop

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



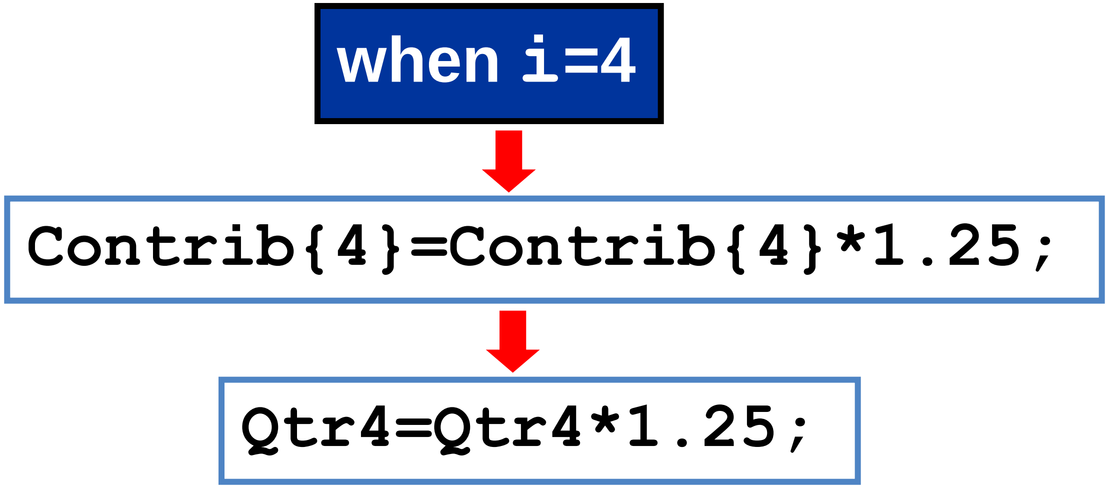
when i=3

```
Contrib{3}=Contrib{3}*1.25;
```

```
Qtr3=Qtr3*1.25;
```

Fourth Iteration of the DO Loop

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{4} qtr1-qtr4;
   do i=1 to 4;
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```



when i=4

```
Contrib{4}=Contrib{4}*1.25;
```

```
Qtr4=Qtr4*1.25;
```

Output: Using a Do Loop to Process an Array

```
proc print data=charity noobs;
run;
```

Partial PROC PRINT Output

```
Employee_ID     Qtr1     Qtr2     Qtr3     Qtr4

     120265        .        .        .    31.25
     120267    18.75    18.75    18.75    18.75
     120269    25.00    25.00    25.00    25.00
     120270    25.00    12.50     6.25        .
     120271    25.00    25.00    25.00    25.00
     120272    12.50    12.50    12.50    12.50
     120275    18.75    18.75    18.75    18.75
     120660    31.25    31.25    31.25    31.25
     120662    12.50        .     6.25     6.25
```

Question & Answer

Using SAS Arrays

Objectives

- Use arrays as arguments to SAS functions.
- Explain array functions.
- Use arrays to create new variables.
- Use arrays to perform a table lookup.

Using an Array as a Function Argument

The program below passes an array to the SUM function.

```
data test;
   set Study.employee_donations;
   array val{4} qtr1-qtr4;
   Tot1=sum(of qtr1-qtr4);
   Tot2=sum(of val{*});
run;
proc print data=test;
   var employee_id tot1 tot2;
run;
```

The array is passed as if it were a variable list.

Partial PROC PRINT Output

```
Obs    Employee_ID    Tot1    Tot2

  1       120265        25      25
  2       120267        60      60
  3       120269        80      80
```

The DIM Function

The DIM function returns the number of elements in an array. This value is often used as the stop value in a DO loop.

General form of the DIM function:

DIM(*array_name*)

```
array Contrib{*} qtr:;
num_elements=dim(Contrib);

do i=1 to num_elements;
   Contrib{i}=Contrib{i}*1.25;
end;
run;
```

The DIM Function

A call to the DIM function can be used in place of the stop value in the DO loop.

```
data charity;
   set Study.employee_donations;
   keep employee_id qtr1-qtr4;
   array Contrib{*} qtr:;
   do i=1 to dim(Contrib);
      Contrib{i}=Contrib{i}*1.25;
   end;
run;
```

Question & Answer

Using an Array to Create Numeric Variables

An ARRAY statement can be used to create new variables in the program data vector.

```
array discount{4} discount1-discount4;
```

If **`discount1-discount4`** do not exist in the PDV, they are created.

```
array Pct{4};
```

Four new variables are created:

PDV

Pct1 N 8	Pct2 N 8	Pct3 N 8	Pct4 N 8

Using an Array to Create Character Variables

Define an array named **Month** to create six variables to hold character values with a length of 10.

```
array Month{6} $ 10;
```

PDV

Month1 $ 10	Month2 $ 10	Month3 $ 10	Month4 $ 10	Month5 $ 10	Month6 $ 10

Example

Using **`Study.employee_donations`** as input, calculate the percentage that each quarterly contribution represents of the employee's total annual contribution. Create four new variables to hold the percentages.

Partial Listing of S**`tudy.employee_donations`**

```
Employee_ID   Qtr1      Qtr2      Qtr3      Qtr4

   120265        .         .         .        25
   120267       15        15        15        15
   120269       20        20        20        20
   120270       20        10         5         .
   120271       20        20        20        20
   120272       10        10        10        10
```

Creating Variables with Arrays

```
data percent(drop=i);
   set Study.employee_donations;
   array Contrib{4} qtr1-qtr4;
   array Percent{4};
   Total=sum(of contrib{*});
   do i=1 to 4;
      percent{i}=contrib{i}/total;
   end;
run;
```

The second ARRAY statement creates four numeric variables: **Percent1**, **Percent2**, **Percent3**, and **Percent4**.

Output: Creating Variables with Arrays

```
proc print data=percent noobs;
   var Employee_ID percent1-percent4;
   format percent1-percent4 percent6.;
run;
```

Partial PROC PRINT Output

```
Employee_ID    Percent1    Percent2    Percent3    Percent4

     120265        .           .           .         100%
     120267       25%         25%         25%         25%
     120269       25%         25%         25%         25%
     120270       57%         29%         14%          .
     120271       25%         25%         25%         25%
     120272       25%         25%         25%         25%
     120275       25%         25%         25%         25%
     120660       25%         25%         25%         25%
     120662       50%          .          25%         25%
     120663        .           .         100%          .
     120668       25%         25%         25%         25%
```

Example

Using **`Study.employee_donations`** as input, calculate the difference in each employee’s contribution from one quarter to the next.

Partial Listing of **`Study.employee_donations`**

```
  Employee_ID  Qtr1     Qtr2     Qtr3     Qtr4

     120265       .        .        .      25
     120267      15       15       15      15
     120269      20       20       20      20
     120270      20       10        5       .
     120271      20       20       20      20
     120272      10       10       10      10
```

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

Poll

Quiz

Quiz

How many ARRAY statements would you use to calculate the difference in each employee’s contribution from one quarter to the next?

Partial Listing of **`Study.employee_donations`**

```
  Employee_ID   Qtr1     Qtr2     Qtr3     Qtr4

     120265       .        .        .       25
     120267      15       15       15       15
     120269      20       20
```

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

Quiz – Correct Answer

How many ARRAY statements would you use to calculate the difference in each employee's contribution from one quarter to the next? **Answers can vary, but one solution is to use two arrays.**

Partial Listing of **`Study.employee_donations`**

Use one array to refer to the existing variables and a second array to create the three `Difference` variables.

```
Employee_ID   Qtr1     Qtr2     Qtr3     Qtr4

   120265        .        .        .      25
   120267       15       15       15      15
   120269       20       20
```

First difference: Qtr2 – Qtr1
Second difference: Qtr3 – Qtr2
Third difference: Qtr4 – Qtr3

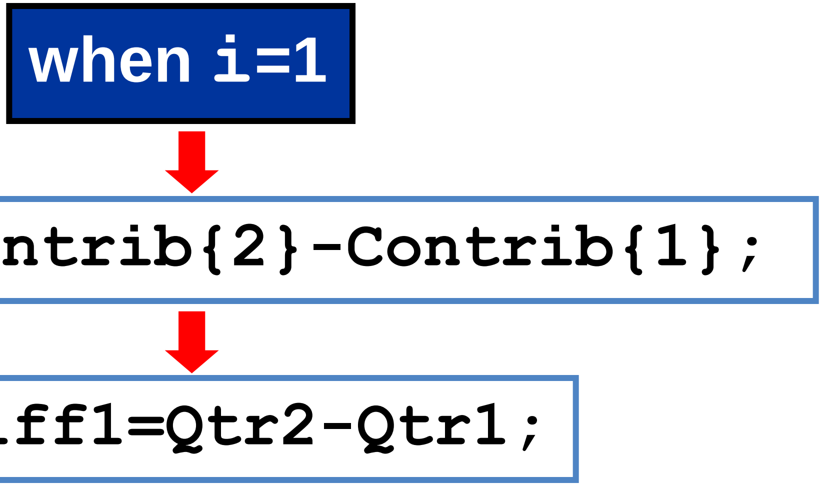
Creating Variables with Arrays

```
data change;
   set Study.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```

The **Contrib** array refers to existing variables. The **Diff** array creates three variables: **Diff1**, **Diff2**, and **Diff3**.

Creating Variables with Arrays

```
data change;
   set Study.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



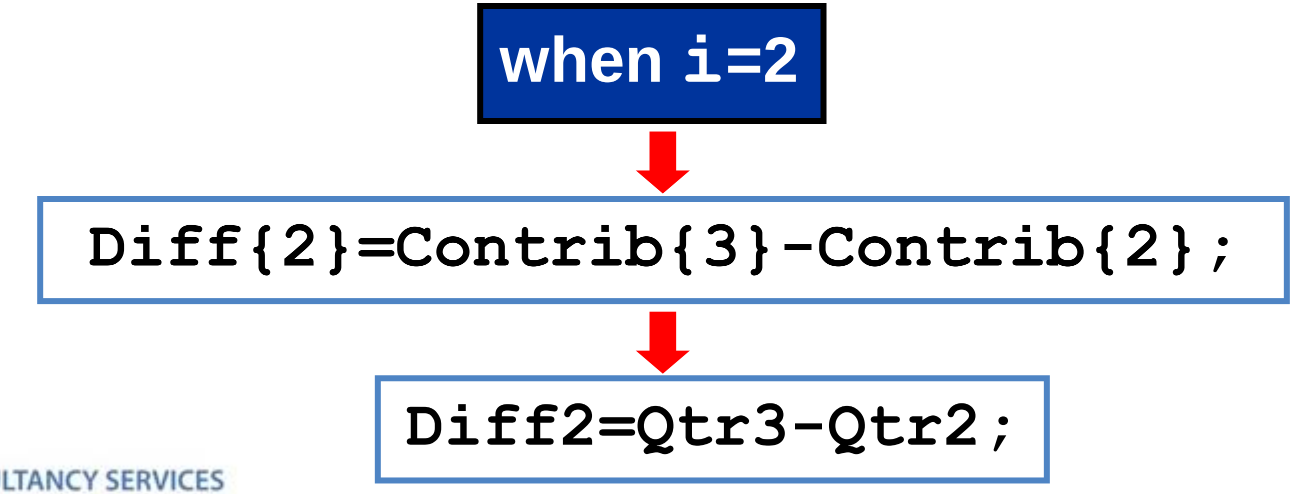
when i=1

```
Diff{1}=Contrib{2}-Contrib{1};
```

```
Diff1=Qtr2-Qtr1;
```

Creating Variables with Arrays

```
data change;
   set Study.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```



when i=2

```
Diff{2}=Contrib{3}-Contrib{2};
```

```
Diff2=Qtr3-Qtr2;
```

Creating Variables with Arrays

```
data change;
   set Study.employee_donations;
   drop i;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{3};
   do i=1 to 3;
      Diff{i}=Contrib{i+1}-Contrib{i};
   end;
run;
```

when i=3

```
Diff{3}=Contrib{4}-Contrib{3};
```

```
Diff3=Qtr4-Qtr3;
```

Creating Variables with Arrays

```
proc print data=change noobs;
   var Employee_ID Diff1-Diff3;
run;
```

Partial PROC PRINT Output

```
Employee_ID     Diff1     Diff2     Diff3

     120265         .         .         .
     120267         0         0         0
     120269         0         0         0
     120270       -10        -5         .
     120271         0         0         0
     120272         0         0         0
     120275         0         0         0
     120660         0         0         0
     120662         .         .         0
```

Question
&
Answer

Assigning Initial Values to an ARRAY

The ARRAY statement has an option to assign initial values to the array elements.

General form of an ARRAY statement:

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

Example:

```
array Target{5} (50,100,125,150,200);
```

Use commas or spaces to separate values in the list.

Assigning Initial Values to an ARRAY

When an *initial-value-list* is specified, all elements behave as if they were named in a RETAIN statement. This is often used to create a *lookup table*, that is, a list of values to refer to during DATA step processing.

```
array Target{5} (50,100,125,150,200);
```

PDV

R Target1 N 8	R Target2 N 8	R Target3 N 8	R Target4 N 8	R Target5 N 8
50	100	125	150	200

Example

Read **`Study.employee_donations`** to determine the difference between employee contributions and the quarterly goals of $10, $20, $20, and $15. Use a lookup table to store the quarterly goals.

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Compilation: What Variables Are Created?

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4

Compilation: What Variables Are Created?

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

No variables created

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4

Compilation: What Variables Are Created?

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1

Diff2	Diff3	Diff4

Compilation: What Variables Are Created?

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1

Diff2	Diff3	Diff4	Goal1	Goal2	Goal3	Goal4

Compilation: What Variables Are Created?

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1

Diff2	Diff3	Diff4	Goal1	Goal2	Goal3	Goal4	i

Compilation: Drop Flags Are Set

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1

Diff2	Diff3	Diff4	D ▶ Goal1	D ▶ Goal2	D ▶ Goal3	D ▶ Goal4	D ▶ i

Compilation: Retain Flags Are Set

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1

Diff2	Diff3	Diff4	R D Goal1	R D Goal2	R D Goal3	R D Goal4	D i

PDV Is Initialized

```
data compare(drop=i Goal1-Goal4);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

Initialize PDV

Partial PDV

Employee_ ID	Qtr1	Qtr2	Qtr3	Qtr4	Diff1
.	.	.	.	.	.

Diff2	Diff3	Diff4	D R Goal1	D R Goal2	D R Goal3	D R Goal4	D i
.	.	.	10	20	20	15	.

TATA CONSULTANCY SERVICES
Experience certainty.

Creating a Temporary Lookup Table

You can use the keyword _TEMPORARY_ in an ARRAY statement to indicate that the elements are not needed in the output data set.

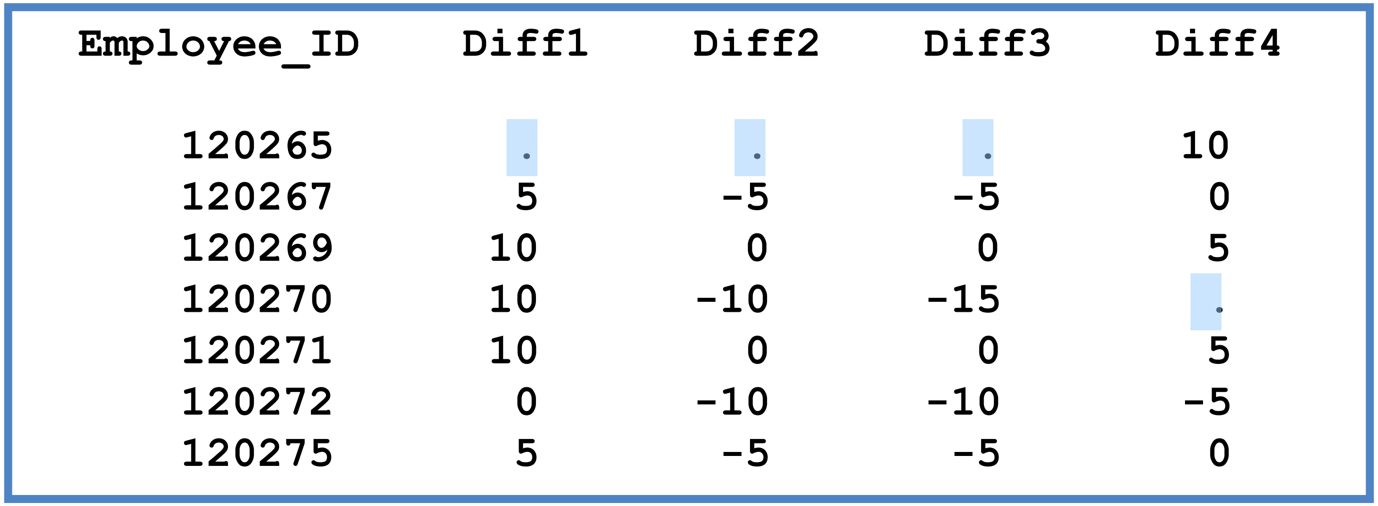
```
data compare(drop=i);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} _temporary_ (10,20,20,15);
   do i=1 to 4;
      Diff{i}=Contrib{i}-Goal{i};
   end;
run;
```

p207d17

Output: Creating a Temporary Lookup Table

```
proc print data=compare noobs;
   var employee_id diff1-diff4;
run;
```

Partial PROC PRINT Output



```
  Employee_ID     Diff1     Diff2     Diff3     Diff4

     120265         .         .         .        10
     120267         5        -5        -5         0
     120269        10         0         0         5
     120270        10       -10       -15         .
     120271        10         0         0         5
     120272         0       -10       -10        -5
     120275         5        -5        -5         0
```

What can be done to ignore missing values?

The SUM Function Ignores Missing Values

The SUM function ignores missing values. It can be used to calculate the difference between the quarterly contribution and the corresponding goal.

```
data compare(drop=i);
   set Study.employee_donations;
   array Contrib{4} Qtr1-Qtr4;
   array Diff{4};
   array Goal{4} _temporary_ (10,20,20,15);
   do i=1 to 4;
      Diff{i}=sum(Contrib{i},-Goal{i});
   end;
run;
```

Output: Lookup Table Application

```
proc print data=compare noobs;
   var employee_id diff1-diff4;
run;
```

Partial PROC PRINT Output

```
Employee_ID     Diff1     Diff2     Diff3     Diff4

     120265       -10       -20       -20        10
     120267         5        -5        -5         0
     120269        10         0         0         5
     120270        10       -10       -15       -15
     120271        10         0         0         5
     120272         0       -10       -10        -5
     120275         5        -5        -5         0
```

The missing values were handled as if no contribution were made for that quarter.

Poll

Quiz

Quiz

Using pencil and paper, write an ARRAY statement to define a temporary lookup table named `Country` with three elements, each two characters long. Initialize the elements to AU, NZ, and US. Refer to the syntax below.

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

Quiz – Correct Answer

Using pencil and paper, write an ARRAY statement to define a temporary lookup table named **`Country`** with three elements, each two characters long. Initialize the elements to AU, NZ, and US. Refer to the syntax below.

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

```
array Country{3} $ 2 _temporary_ ('AU','NZ','US');
```

Question
&
Answer

Chapter Review

1. An iterative DO loop must have a stop value? True or False?

2. A DO WHILE statement tests the condition at the ____________________and a DO UNTIL statement tests the condition at the ______.

3. A _______ will always execute at least once, but a __________might never execute.

4. What is the out of range value for this DO loop?

```
do year=2000 to year(today());
```

continued...

Chapter Review Answers

1. An iterative DO loop must have a stop value. True or False
   False. It might have a list of values.

2. A DO WHILE statement tests the condition at the **top of the loop**, and a DO UNTIL statement tests the condition at the **bottom**.

3. A **DO UNTIL** statement will always execute at least once, but a **DO WHILE** statement might never execute.

4. What is the out of range value for this DO loop?

```
do year=2000 to year(today());
```

The upcoming year, so in 2009 the final value of year will be 2010.

continued...

Chapter Review

5. A single array can contain both numeric and character elements. True or False?

6. What is wrong with the following array definition?

```
array value{5} v1-v6;
```

7. Write a DO statement to process every element in the following array: `array num{*} n:;`

8. What keyword causes a lookup table to be stored in memory instead of in the PDV ?

Chapter Review Answers

5. A single array can contain both numeric and character elements. True or False? **False**

6. What is wrong with the following array definition?

```
array value{5} v1-v6;
```

The subscript and the number of items in the element list does not agree.

7. Write a DO statement to process every element in the following array: **`array num{*} n:;`**

```
do i=1 to dim(num);
```

8. What keyword causes a lookup table to be stored in memory instead of in the PDV ? **_TEMPORARY_**

THANK YOU

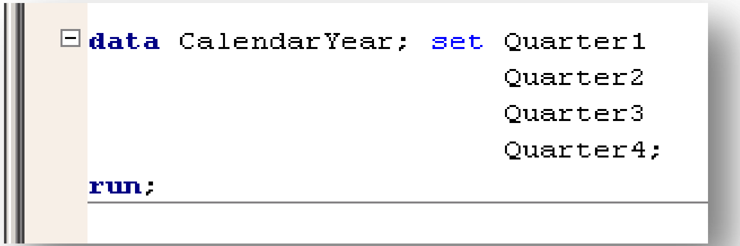
Achieving business excellence through efficient and effective business processing and winning insights.

That's certainty

Appending Datasets

SET statement

- The simplest method for adding observations to a SAS dataset is through the SET statement

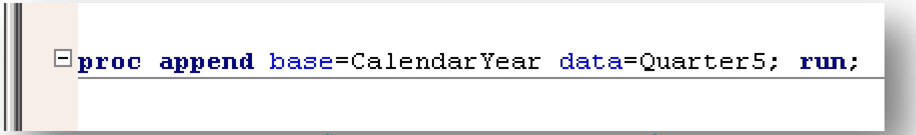


```
data CalendarYear; set Quarter1
                       Quarter2
                       Quarter3
                       Quarter4;
run;
```

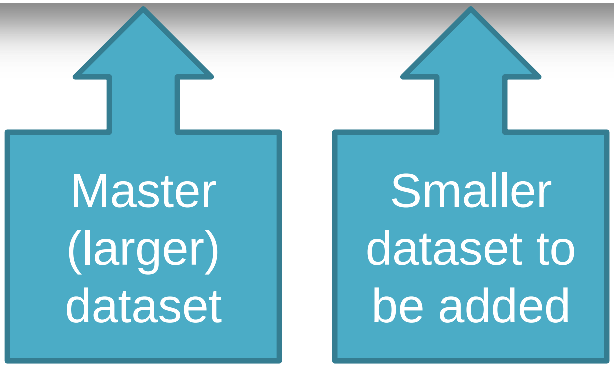
- Datasets are simply stacked on top of each other (concatenation)
- Can use a BY statement to interleave datasets (if input datasets are sorted)
- Duplicate observations are not overwritten

PROC Append

- SET statement method works best if datasets are small and manageable
- For better processing efficiency, use PROC Append



```
proc append base=CalendarYear data=Quarter5; run;
```



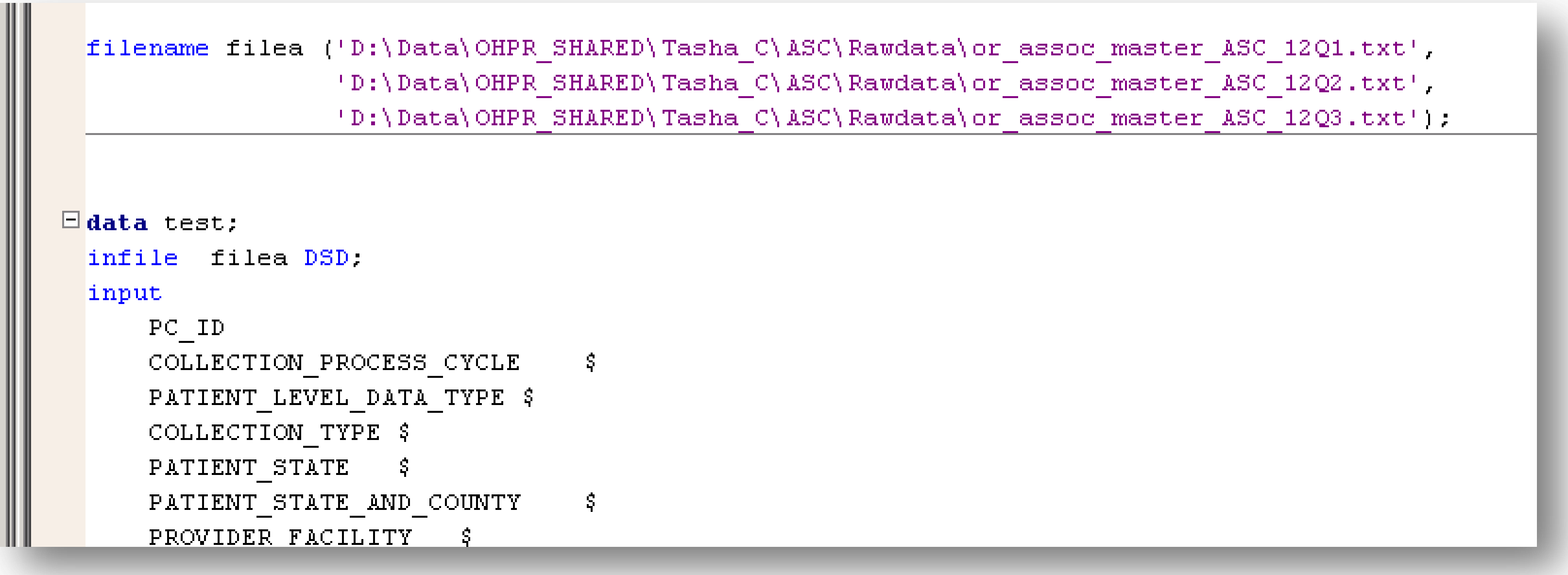
Master (larger) dataset

Smaller dataset to be added

- Can only append one dataset at a time
- Input datasets must have same variables and attributes (otherwise use FORCE option)

Appending raw data

- Use FILENAME statement to append raw data during input



```
filename filea ('D:\Data\OHPR_SHARED\Tasha_C\ASC\Rawdata\or_assoc_master_ASC_12Q1.txt',
                'D:\Data\OHPR_SHARED\Tasha_C\ASC\Rawdata\or_assoc_master_ASC_12Q2.txt',
                'D:\Data\OHPR_SHARED\Tasha_C\ASC\Rawdata\or_assoc_master_ASC_12Q3.txt');


data test;
infile  filea DSD;
input
    PC_ID
    COLLECTION_PROCESS_CYCLE    $
    PATIENT_LEVEL_DATA_TYPE $
    COLLECTION_TYPE $
    PATIENT_STATE   $
    PATIENT_STATE_AND_COUNTY    $
    PROVIDER_FACILITY   $
```

Joins and Merge

Joins and Merge

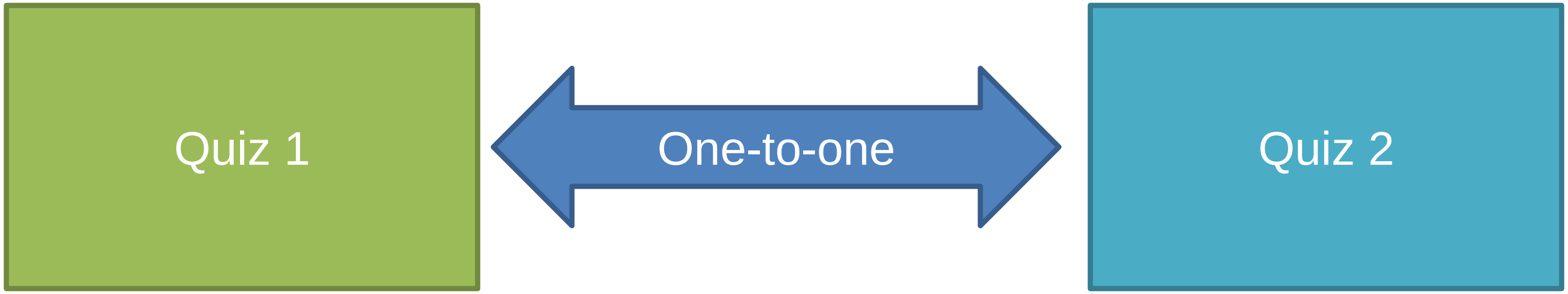
Joins and merges

- Merging data – combining columns from two or more datasets
- Can use DATA step MERGE statement or PROC SQL
- Can produce inner joins and outer joins
- Can merge entire datasets or subsets of datasets
- Can produce one-to-one and one-to-many, but not many-to-many joins
  - Can produce many-to-many joins in PROC SQL

MERGE statement

- Input datasets must have a common identifying variable (primary key)
- Input datasets must be sorted by this key variable
- Key variable must have same name and attributes
- All other variables must have a unique name or they will be overwritten by last merged dataset

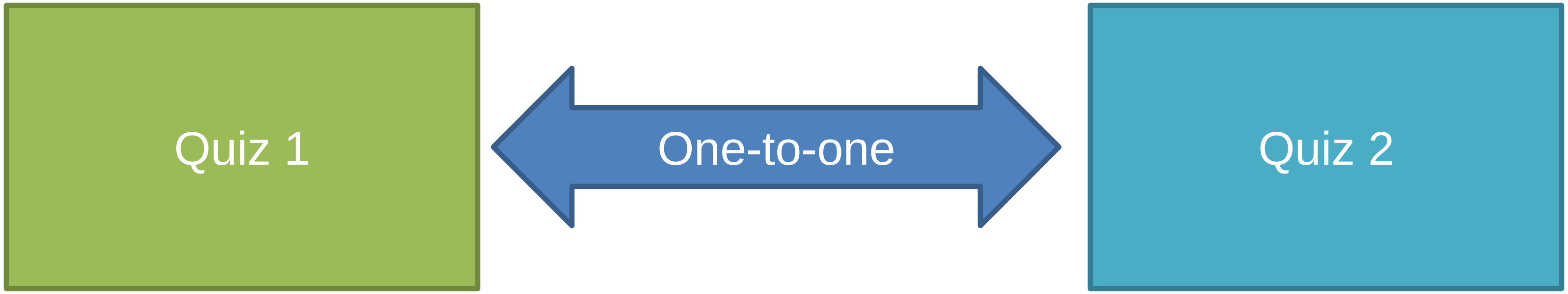
One-to-one joins



Student_ID	Quiz1
001	85
002	86
003	95
004	97

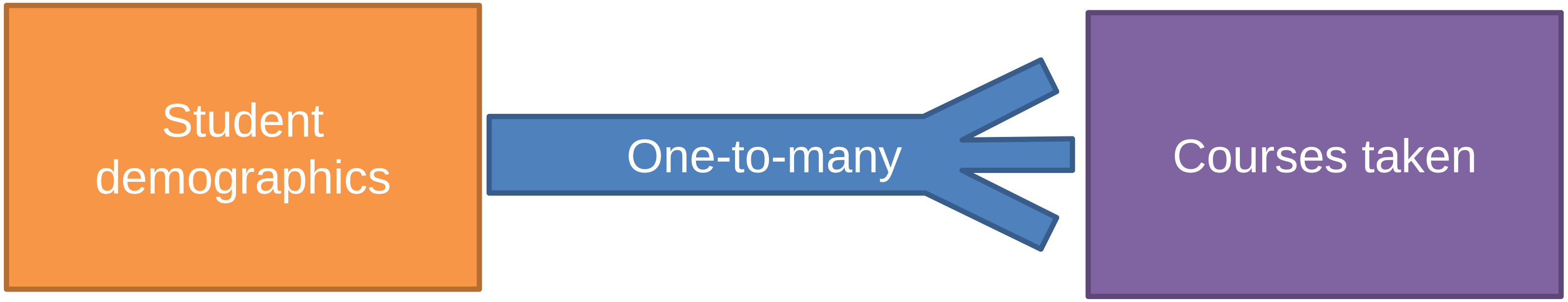
Student_ID	Quiz2
001	96
002	88
003	85
004	94

One-to-one joins



Student_ID	Quiz1	Quiz2
001	85	96
002	86	88
003	95	85
004	97	94

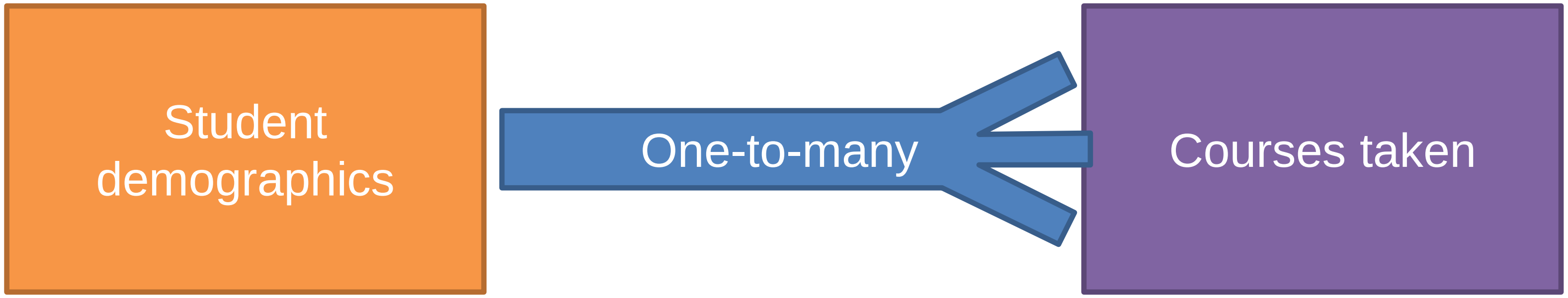
One-to-many joins



Student_ID	Gender
001	F
002	M
003	M
004	F

Student_ID	Course_Title
001	Psychology 101
001	Philosophy 105
001	Math 212
002	Writing 222
002	Psychology 101
002	Spanish 301
…	…

One-to-many joins

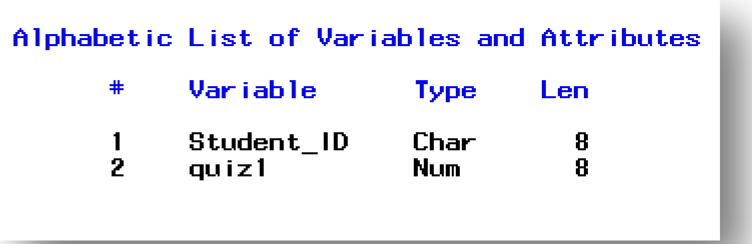


Student_ID	Gender	Course_Title
001	F	Psychology 101
001	F	Philosophy 105
001	F	Math 212
002	M	Writing 222
002	M	Psychology 101
002	M	Spanish 301
…	…	…

MERGE statement

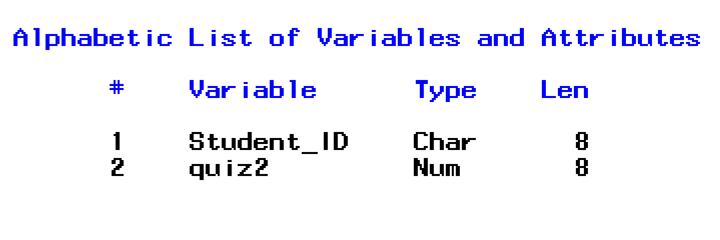
- Example – 3 datasets

PersQuiz



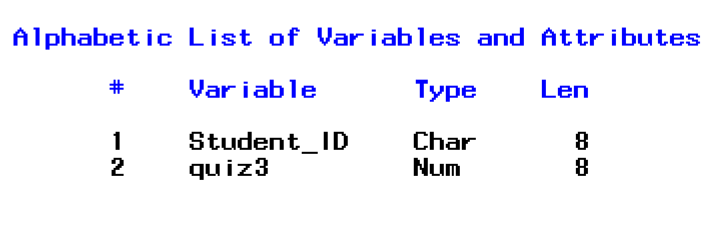
```
Alphabetic List of Variables and Attributes

     #    Variable      Type    Len

     1    Student_ID    Char      8
     2    quiz1         Num       8
```

SocQuiz



```
Alphabetic List of Variables and Attributes

     #    Variable      Type    Len

     1    Student_ID    Char      8
     2    quiz2         Num       8
```

CogQuiz

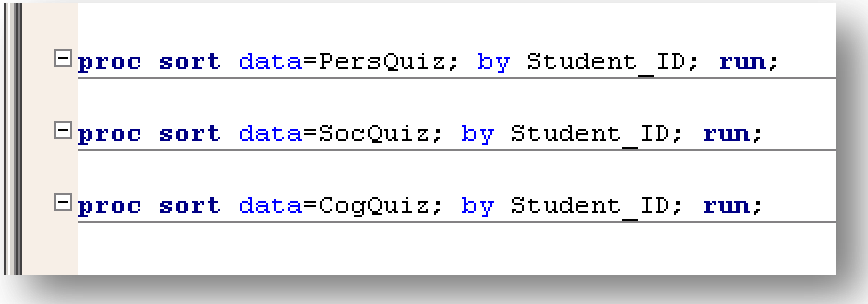


```
Alphabetic List of Variables and Attributes

     #    Variable      Type    Len

     1    Student_ID    Char      8
     2    quiz3         Num       8
```

Want one dataset
with all three quiz
grades

MERGE statement

- First sort the datasets using the PROC Sort

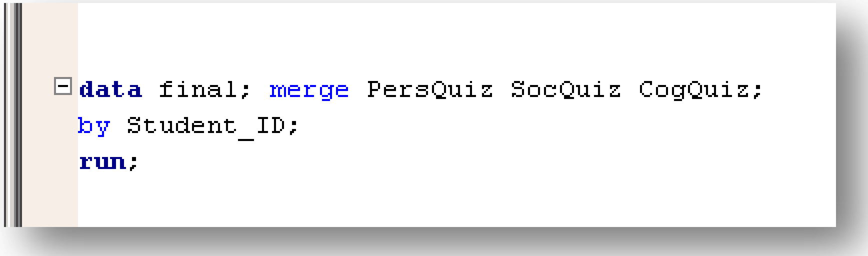


```
proc sort data=PersQuiz; by Student_ID; run;
proc sort data=SocQuiz; by Student_ID; run;
proc sort data=CogQuiz; by Student_ID; run;
```

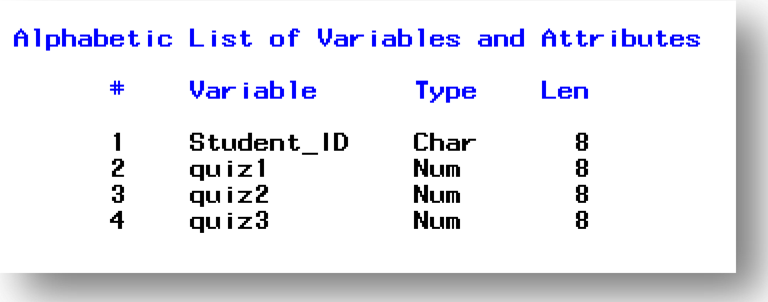
Datasets **must** be sorted or the merge will not work properly

MERGE statement

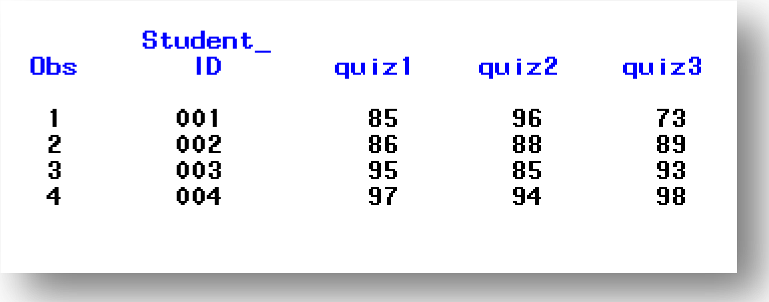
- Then merge in the DATA step



```
data final; merge PersQuiz SocQuiz CogQuiz;
by Student_ID;
run;
```



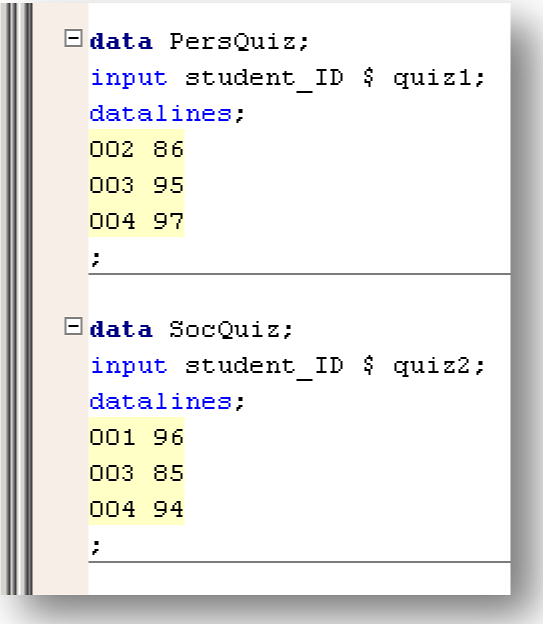
```
Alphabetic List of Variables and Attributes

      #    Variable      Type    Len

      1    Student_ID    Char      8
      2    quiz1         Num       8
      3    quiz2         Num       8
      4    quiz3         Num       8
```



```
         Student_
  Obs       ID       quiz1    quiz2    quiz3

   1       001         85       96       73
   2       002         86       88       89
   3       003         95       85       93
   4       004         97       94       98
```

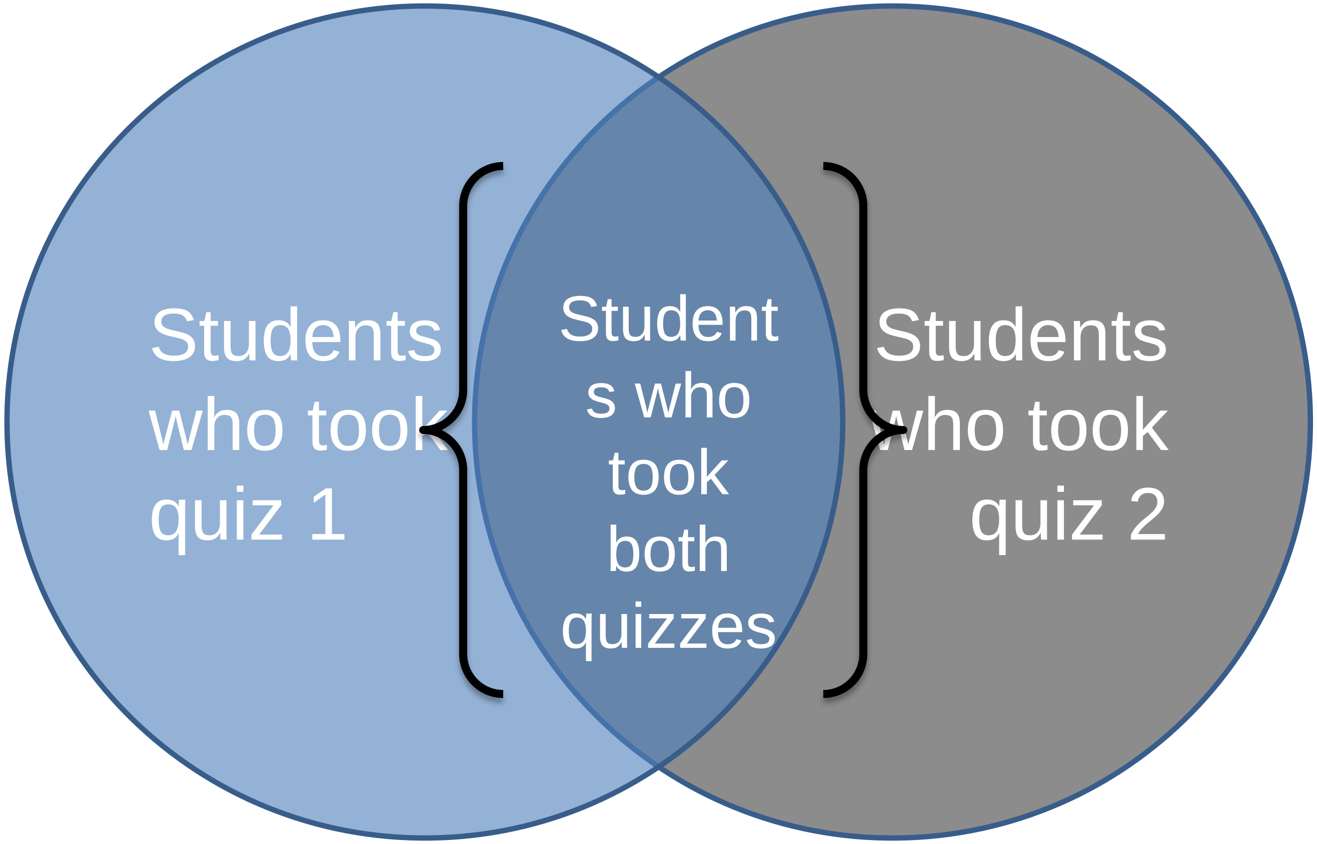
Inner Joins

- Example – 2 datasets
  - Some students took quiz 1, but not quiz 2
  - Some students took quiz 2, but not quiz 1
  - Some students took both quizzes



```
data PersQuiz;
input student_ID $ quiz1;
datalines;
002 86
003 95
004 97
;

data SocQuiz;
input student_ID $ quiz2;
datalines;
001 96
003 85
004 94
;
```

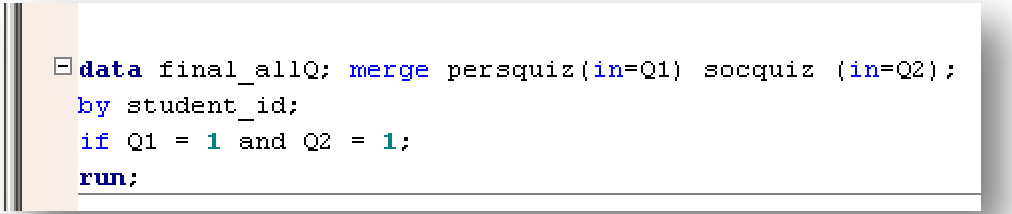
Inner Joins


Students who took quiz 1
Students who took both quizzes
Students who took quiz 2

Inner Joins

- Use the IN= option

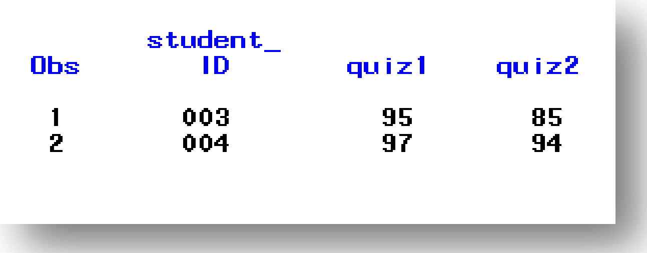
Use IN= to create temporary aliases for the input datasets



```
data final_allQ; merge persquiz(in=Q1) socquiz (in=Q2);
by student_id;
if Q1 = 1 and Q2 = 1;
run;
```

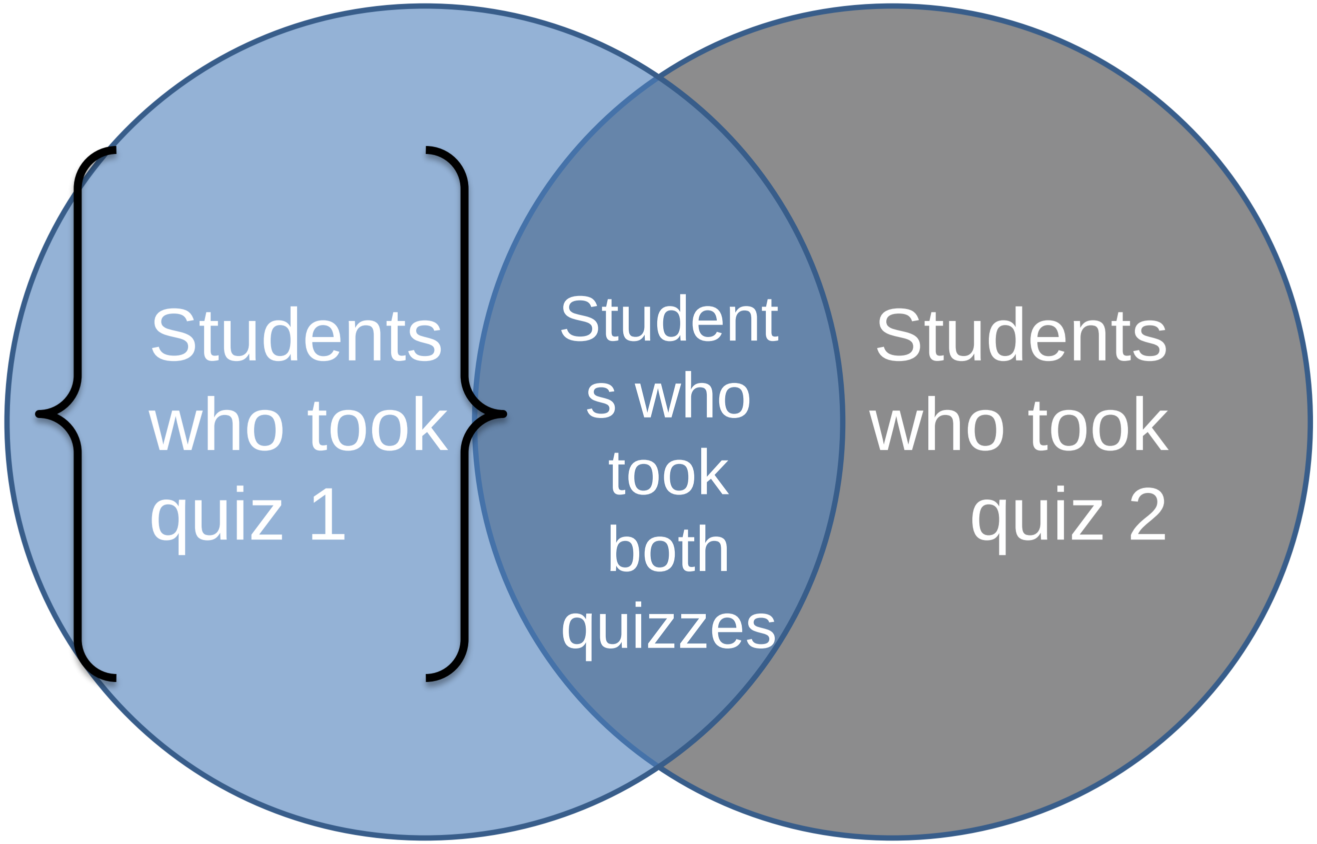
Use subsetting IF to only include observations in both datasets

`Q1 = 1` means the observation is in the Q1 (PersQuiz) dataset



```
            student_
   Obs         ID         quiz1     quiz2

    1         003           95        85
    2         004           97        94
```

Excluding Joins



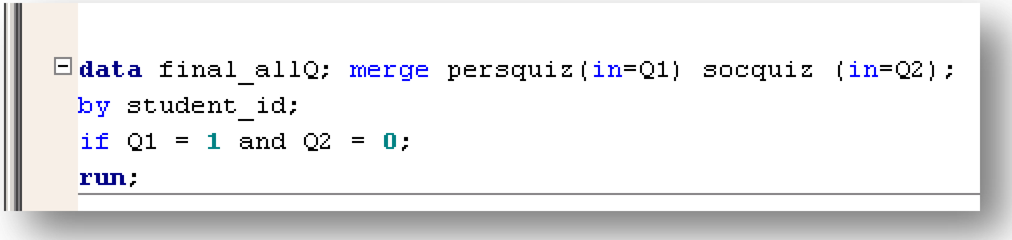
Students who took quiz 1

Students who took both quizzes

Students who took quiz 2

Excluding Joins

- Who took Quiz 1, but not Quiz 2?

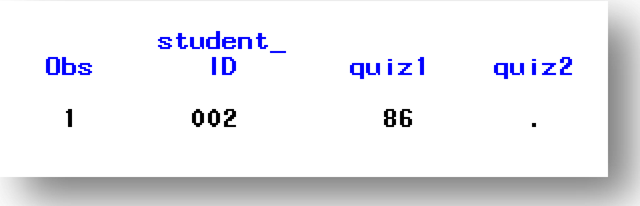


```
data final_allQ; merge persquiz(in=Q1) socquiz (in=Q2);
by student_id;
if Q1 = 1 and Q2 = 0;
run;
```

We'll discuss other inner and outer joins with PROC SQL

Use subsetting IF to only include observations in one dataset that are not in the other

Q2 = 0 means the observation is in **not** Q2 (SocQuiz) dataset



```
           student_
 Obs          ID        quiz1     quiz2

  1          002          86        .
```