Introduction to Fundamentals of SAS

Getting Started in the SAS System

The SAS system is a collection of products, available from the SAS Institute. The SAS runs on a wide variety of computers and operating systems. In this course we will be using *PC version of SAS* under the Microsoft Windows Operating System.

SAS Display Manager

The *SAS Display Manager* (DM) is the standard interactive environment for the SAS system. In here, you will see various "*windows*".

- **EDITOR** window***:*** You can enter a SAS program or bring in a SAS program from a file. This window has an editor that allows you to write and edit your program. When completed, you can submit your program.
- **LOG** window: This window displays the program lines as the program executes, prints out messages about your data sets as they are created, and gives error messages as errors are created. Note that the error messages are shown in **RED** color.
- **OUTPUT** window: This window displays results from your SAS program. You can switch among these windows.
- **RESULTS Viewer – SAS Output**: This window will be displayed **after** the program is run.
- **EXPLORER** window gives you quick access to SAS files.

Various Modules of SAS

Base SAS is the main module which provides data manipulation and programming capability and some elementary descriptive statistics.

SAS/STAT is the module that includes all the statistical programs except the elementary ones supplied with the base package.

SAS/GRAPH is a module that provides high quality graphs. The base SAS and SAS/STAT do provide graphics but SAS/GRAPH provides high quality graphs, maps, and charts.

SAS/FSP is a module that allows you to search, modify, or delete records directly from SAS data file. It provides for data entry with sophisticated data checking capabilities. FSP stands for **F**ull **S**creen **P**roduct. Procedures available with FSP are FSBROWSE, FSEDSIT, FSPRINT, FSLIST, and FSLETTER.

SAS/IML IML stands for "Interactive Matrix Language" which is used for matrix manipulations.

A few other modules are **SAS/AF** (SAS **A**pplications **F**acility), **SAS/ETS (E**conometric and **T**ime **S**eries package), **SAS/OR** (**O**perations **R**esearch programs), **SAS/QC** (programs for **Q**uality **C**ontrol), **SAS/ACCESS, SAS/CONNECT**, and **SAS/ASSIST**.

Introduction to Fundamentals of SAS

Basic Structure of the SAS System

Every SAS program is organized into 2 steps:

DATA step: This is the heart of the SAS's programming. It puts data in a form that the SAS program can use. ***DATA*** step reads SAS datasets or raw data, performs transformations, creates new variables, and recodes existing variables.

PROC step: This step uses procedures to do many things to the data, such as sorting, analyzing, or printing them.

Both ***DATA*** and ***PROC*** steps consist of SAS statements.

- ✓ **Each SAS statement begins with a keyword** that implies its function
    - *For example*:
        - *INPUT* statement reads inputs
        - *FREQ* procedure produces frequency distribution
        - *RUN* statement tells SAS to run the DATA or PROC just completed
- ✓ A **semicolon (;)** is required to denote the end of each SAS statement.
- ✓ The spacing of the statements or lower (or upper) case is ***not*** important in SAS.

SAS has many different ***PROC***s that can be organized into functional categories:

- ✓ PROCs that manage datasets
- ✓ PROCs that analyze the contents of a data set
- ✓ PROCs that graphically display the contents of a data set. …..etc.

- ✓ SAS not only provides how to create a data set within the SAS environment but also provides many different ways to create a SAS data set by letting you import data from many different formats such as, to name a few, Excel, ASCII, Access, SPSS formats.
- ✓ Data sets are organized in the form of "***rows***" and "***columns***". A "*row*" refers to an observation, and a "*column*" refers to a variable name.

Note: Please keep in mind that the data come in many different formats, and we will learn later on how to convert them into SAS data sets that can be used in SAS programs.

Introduction to Fundamentals of SAS

Creating a SAS data set as part of the DATA step programming

Here is an example of a **7-line SAS program and 3 lines of comments**, followed by a **detailed explanation** (in **RED**) of each SAS statement.

```
Data sample;
Input id gender $ lastname $ sys_bp dia_bp examdate mmddyy6.;
Format examdate mmddyy10.;
Datalines;
1 MALE . 124 89 011915
2 FEMALE JOHN 130 92 022216
3 MALE WALLIS 112 80 123102
4 MALE BROWN 120 80 .
5 MALE WU 122 95 042117
6 MALE MASON 120 79 031012
7 MALE PETERSON 118 . 122511
;
proc print data = sample;
run;
```

* **Comment line 1**: The above SAS program is a simple SAS program but has many basic features Of SAS in it;

* **Comment line 2**: The comment statements **begin with an asterisk (*) and ends with a SEMICOLON (;);**

/* **Comment line 3**: This is another way of writing a comment statement. That is, **starting with /* and ending with */.** This form of comment can be embedded within a SAS statement and can include semicolon within the comment itself */

SAS Statement 1:

data sample;

This statement creates a *temporary* SAS data set called SAMPLE

DATA is a SAS key word.

The name SAMPLE can be any name that meets SAS's naming convention as described below:

- **Names in SAS can contain letters, numbers, and underline characters but cannot begin with a number.**
- **Names can be in upper case, lower case, or a mixture.**
- **Names must be 32 characters or fewer in length.**

The semicolon indicates the end of a SAS statement. Each SAS statement must end with a semicolon.

Introduction to Fundamentals of SAS

SAS Statement 2:

input id gender $ lastname $ sys_bp dia_bp examdate mmddyy6.;

INPUT is a SAS key word that tells SAS that what follows the INPUT is a list of VARIABLES that will be stored in the data set. Again these variable names should meet the SAS's naming convention as described on the previous page.

$ indicates that the variable is a character type variable. In this example, GENDER and LASTNAME are character type variable. Both letters and numbers can be entered as data to character variables.

The numeric type variables are ID, SYS_BP and DIA_BP. Only numbers can be entered as data to numeric variables.

EXAMDATE is a date type variable. Note that in this example, dates will be entered as a six-digit number (as an example 011999 for January 19, 1999).

In order to tell SAS to read this date, we use what is called INFORMAT (or INPUT FORMAT)

The syntax of the INFORMAT is mmddyy6.

Note that the period at the end is part of the syntax. The first two digits are for the MONTH, the next two digits are for the DAY and the last 2 digits are for the YEAR, entered as a two-digit year.

SAS Statement 3:

format examdate mmddyy10.;

By default, SAS stores any date as the *number of days* from January 1, 1960.

Dates after January 1, 1960 are stored as positive numbers, and the dates before January 1, 1960 are stored as negative numbers.

For example:

01/01/1959 is stored as -365
01/01/2001 is stored as 14976

In order to print a date, entered as 011999, in the form 01/19/1999, SAS expects you to have a format statement as shown in this SAS statement.

MMDDYY10. allows SAS to print a 6-digit number 011999 using 10 spaces as 01/19/1999.

SAS Statement 4:

datalines;

This statement says the DATA statements are done and the next thing the program should look for are the data themselves that will go into the SAS data set (SAMPLE) being created above.

We do not need this statement if the data comes from external sources.

Introduction to Fundamentals of SAS

SAS Statement 5:

```
1 MALE . 124 89 011915
2 FEMALE JOHN 130 92 022216
3 MALE WALLIS 112 80 123102
4 MALE BROWN 120 80 .
5 MALE WU 122 95 042117
6 MALE MASON 120 79 031012
7 MALE PETERSON 118 . 122511
;
```

These 6 lines represent the data that will go into the SAS data set, SAMPLE. Note that the SEMICOLON is placed all by itself after the last data line.

This data set has 1 missing value for the CHARACTER variable, LASTNAME; 1 missing value for the NUMERIC variable, DIA_BP; 1 missing value for the DATE type variable, EXAMDATE.

The missing value is entered as a period (.) for all types of variables, be it CHARACTER, NUMERIC or DATE TYPE.

SAS Statement 6 and 7:

```
proc print data = sample;
run;
```

These 2 lines invoke SAS PRINT procedure once you submit (run) the program and prints the TEMPORARY SAS data set, SAMPLE.

How to include comments within a SAS program?

After Statements 6 & 7, you see 3 lines of texts in the program. These are the examples of comments that one can put anywhere in the SAS program.

Note that each comment statement begins with an "asterisk (*) and ends with a "semicolon (;)

Another form of including comments in any SAS program is to begin with /* and end with */.

Introduction to Fundamentals of SAS

After running the above SAS program, following is what you see in the **LOG WINDOW**. **There are no errors in the program**.

This is what you see in the LOG WINDOW

```
1    Data sample;
2    Input id gender $ lastname $ sys_bp dia_bp examdate mmddyy6.;
3    Format examdate mmddyy10.;
4    Datalines;

NOTE: The data set WORK.SAMPLE has 7 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds


12   ;
13   proc print data = sample;
NOTE: Writing HTML Body file: sashtml.htm
14   run;

NOTE: There were 7 observations read from the data set WORK.SAMPLE.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          0.26 seconds
      cpu time           0.14 seconds
```

****

Following is what you see in the **RESULTS WINDOW**.

Obs	id	gender	lastname	sys_bp	dia_bp	examdate
1	1	MALE		124	89	1/19/2015
2	2	FEMALE	JOHN	130	92	2/22/2016
3	3	MALE	WALLIS	112	80	12/31/2002
4	4	MALE	BROWN	120	80	.
5	5	MALE	WU	122	95	4/21/2017
6	6	MALE	MASON	120	79	3/10/2012
7	7	MALE	PETERSON	118	.	12/25/2011

REMARKS:

Note that the missing values were entered as a period (.) for all types of variables.

However, after the data set is created by SAS, they are stored internally as follows:

- ✓ **Missing value for the character type variable is internally stored as a "blank"**
- ✓ **Missing value for the numeric type (and DATE type) variable is internally stored as a period (.)**

Introduction to Fundamentals of SAS

Use of PROC CONTENTS

```
proc contents data=sample;
run;
```

PROC CONTENTS procedure is illustrated;

PROC CONTENTS is a procedure that describes the contents of the SAS data file (such as names and types of variables, formats, number of observations in the data set, and labels) and lists the variables alphabetically.

If you use: proc contents data=sample position, then

It produces the output same as described above and in addition, it lists the variable names in the same order as they are created in the data set.

Every procedure ends with the RUN statement.

If data=sample is not used, then SAS simply gives the contents of the data set that was created last.

Use of PROC MEANS

```
proc means data=sample;
var sys_bp dia_bp;
run;
```

PROC MEANS procedure is illustrated.

PROC MEANS produces the descriptive statistics (n, mean, standard deviation, minimum and maximum) for all the continuous variables listed after the key word VAR.

In this example, PROC MEANS works on the variables, SYS_BP and DIA_BP.

Use of PROC FREQ

```
proc freq data=sample;
tables gender;
run;
```

PROC FREQ procedure is illustrated.

PROC FREQ produces the frequency (such as how many of each category) for all the categorical variables listed after the key word TABLES.

In this example, PROC FREQ works on the variable, GENDER.

TEMPORARY versus PERMANENT SAS Data set.

Note that in the example just discussed, the SAS data set SAMPLE is a temporary SAS data set and is not saved anywhere.

When once you exit SAS, the SAS data set, SAMPLE, will no longer exist;

Let us suppose that we want to create a **PERMANENT** SAS data set by name **SAMPLE** in the directory **C:\CAPHR\DAT_FILE**.

Add the following SAS statements:

```
libname dat_file 'c:\caphr\dat_file\';
data data_file.sample;
      set sample;
run;
```

LIBNAME is introduced. More on this later.

This sets SAMPLE equal to another data set with the same name, SAMPLE, but then it is stored permanently in the folder, DAT_FILE.

We will discuss more about this when we talk about the LIBNAME statements later on.

Remark:

On the next page, I have briefly summarized all of the key words used in the above program. Please read this carefully.

A typical question can be, for example,

"what is the significance of a $ sign?" or

"what symbol do you use to end a SAS statement?"

In the above program, we have used many **SAS keywords** such as:

libname points to the subdirectory in which the data file needs to be stored permanently

$ indicates that the type of the variable is *character*.

input is the key word which followed by the list of variables in the data set.

datalines statement signals the beginning of the lines of data.

mmddyy6. is called an "*input format*" (informat). This indicates to SAS to expect the dates to be entered as a six-digit number. As an example: **122902**

mmddyy10. is called a "*format*". This indicates to SAS to print date as **12/29/2002**

set tells SAS to use this data file **SAMPLE** to create the new data file.

data dat_file.sample;
creates a *permanent* data set called ***SAMPLE*** and stores it in the subdirectory, **dat_file,** which is a subdirectory in **c:/caphr/**

Proc stands for PROCEDURE

Proc contents;
displays the contents of the data file

Proc print; is a SAS procedure that prints the dataset for all the variables in the order in which they were read. This prints the data from the **latest SAS data** set created in the program.

Proc means; computes the mean, standard deviation, minimum, maximum for the variables listed following the key word **VAR**. This is only for the **continuous** variables.

Proc freq; computes frequency tabulation for the variables listed following the key word **TABLES**. This is only for the **categorical** variables.

Note: **A single SAS data set can have up to 32,767 variables**.

In the next few pages, we will discuss various ways in which the data can be read into SAS

Please note that much of the explanation I have given above in "red" keeps coming back again and again all through this course and in all of the SAS programs that are in your lecture notes.

In the programs that follow, I will explain anything that is new.

Please run the following programs as many times as is required in order to understand them completely.

Data Step Programming Examples

Example 1: Reading Data values separated by spaces (no missing values)

```
options nodate nonumber;
```

OPTIONS is a SAS key word. NODATE and NONUMBER in OPTIONS statement are to avoid printing dates and page numbers on the output;

```
data example1;
```

EXAMPLE1.SAS (example of LIST input)

```
input name $ id $ gender $ height weight center drug $ changescore;
```

INPUT statement can be used in three different ways:

- ✓ LIST input
- ✓ COLUMN input
- ✓ FORMATTED input

The above INPUT statement is referred to as LIST INPUT. Data in this format are entered without regard to column location.

Two restrictions are placed:

First, variable values must be separated by one or more blanks.

Second, there must be as many values on each line of data as there are variable names in the INPUT statement.

As an example: `John 123 male 59 155 1 placebo 17`

Note that each data value is separated by at least one blank space and there are altogether 8 pieces of data on each line, matching 8 variables listed in the INPUT statement.

```
label
      height = "Height of the participant"
      weight = "Weight of the participant"
      center = "Study center number"
      drug = "Treatment: Drug or Placebo"
      changescore = "change score";
```

- LABEL statement: Each label can have up to 256 characters. It goes right after the INPUT statement.
- Note that LABEL statement begins with the key word LABEL, followed by the variable name and its description enclosed in single or double quotes.
- Also note that the semicolon goes at the end of the last variable with its description.

```
datalines;
John 123 male 59 155 1 placebo 17
Wallis 221 female 52 101 1 drug 23
Peterson 323 male 65 210 2 placebo 14
Wu 105 female 55 120 2 placebo 6
Sanders 143 male 60 175 2 drug 9
;
proc print data=example1;
      title 'Tabulation of data from EXAMPLE1 data set';
```

- TITLE statement begins with the key word TITLE followed by at least one blank space and the text for the title enclosed again in either single or double quotes.
- There can be as many TITLE statements in any SAS program as one needs.
- It is a good practice to have TITLE s for each SAS PROCEDURE.
- For the second title, the syntax is, for example:
  title2 'Whatever the text you want to put here';
- For the third title, the syntax is, for example:
  Title3 'Whatever the text you want to put here';
  and so on;

```
run;

proc means data=example1;
      var height weight changescore;
      title 'Output from the Procedure MEANS';
run;

proc freq data=example1;
      title 'Frequency distribution of the categorical variables';
      tables drug center;
run;

* The following statements read only observations 2 through 4;
data temp;
      set example1 (firstobs=2 obs=4);
Run;
```

SET is another SAS key word. The purpose of this is to create a temporary SAS data set using an existing SAS data set. That is, basically making a copy of an existing SAS data set and calling it by another name.

In here, we are using an existing SAS data set called EXAMPLE1 & making a copy of it and calling it TEMP. This way I am working with the copy of the original file and keeping the original data set intact.

Suppose we want to read the observations 2 through 4, then we use
FIRSTOBS=2 OBS=4;
as part of the SET statement as shown above.

Note that the SAS data set, TEMP, has only 3 observations

```
proc print data=temp;
      title 'Tabulation of data from observation 2 through 4';
run;
```

Notes:

√ LABEL statement: Each label can have up to 256 characters.

√ NODATE and NONUMBER in OPTIONS statement are to avoid printing date and page number on the output.

A few examples of OPTIONS Statement:

Here are a few examples of the use of OPTIONS statement

Note that there will be questions on these concepts in the midterm exam.

Any number of **OPTIONS** statements may appear anywhere in the SAS program.

√ `OPTIONS pageno=n;` (starts numbering output pages with ***n***)

```
options pageno = 10;
```

Means the page numbers for the output begins with 10.

√ `OPTIONS ps = n;` (controls the maximum number of lines per page of output)

```
options ps = 45;
```

Means 45 lines are printed on each page of the output.

√ `OPTIONS ls = n;`

(Controls the maximum length of output lines. The default is 78 in *interactive* environments, and 132 in *batch*. ***n*** may range from 64 to 256)

√ `OPTIONS firstobs = n;` SAS begins processing with observation number **n**.

```
options firstobs = 25;
```

Means SAS begins processing from the observation 25

√ `OPTIONS obs = n;` SAS uses *up to and including* observation number **n**.

```
options obs = 10;
```

Means SAS uses the first 10 observations of the data set.

Introduction to Fundamentals of SAS

Example 2: Reading Data values separated by spaces (with Missing values being present)

- **The data set for this program has a lot of missing values.**
- **While entering the data into a SAS data set, missing value for both character type and numeric type variables will be entered as a period (.)**
- **However, after the SAS data set is created, the missing value for the character type variable is stored internally as a "blank" whereas for numeric type data, the missing value is stored internally as a period (.).**

EXAMPLE2.SAS

```
data example2;
      input age race $ parity meno bmi bmi2 stage type;
datalines;
69    W     0      1      23.43 22.41 1     .
.     W     2      1      31.31 29.23 1     1
61    AA    2      1      22.85 24.15 1     .
82    W     2      1      34.27 32.78 0     .
37    AA    2      0      21.68 20.89 0     1
71    .     2      1      41.84 40.76 1     .
57    AA    2      1      29.93 28.34 0     1
75    AA    .      1      .     22.32 0     0
65    W     .      .      .     .     .     .
55    W     0      1      29.59 27.67 1     .
;
*************************************************************;
proc print data=example2;
      title 'Tabulation of data from EXAMPLE2 data set';
run;
```

Note Below: Missing character data will be stored as **"blanks"**
Missing numeric data will be stored as "**period" (.)**

SAS OUTPUT

Obs	age	race	parity	meno	bmi	bmi2	stage	type
1	69	W	0	1	23.43	22.41	1	.
2	.	W	2	1	31.31	29.23	1	1
3	61	AA	2	1	22.85	24.15	1	.
4	82	W	2	1	34.27	32.78	0	.
5	37	AA	2	0	21.68	20.89	0	1
6	71		2	1	41.84	40.76	1	.
7	57	AA	2	1	29.93	28.34	0	1
8	75	AA	.	1	.	22.32	0	0
9	65	W	.	.	.	.	.	.
10	55	W	0	1	29.59	27.67	1	.

Introduction to Fundamentals of SAS

Example 3: Reading Data values separated by spaces with one data piece for a character variable being greater than 8 characters

By default, a character type variable can accept a maximum of 8 characters as the data.

Suppose a particular character variable (such as RACE in this example) has more than 8 characters as its data. We need to make SAS accept more than 8 characters as the data to a character variable.

This can be accomplished in the following 3 ways using:

- ✓ **INFORMAT statement**
- ✓ **LENGTH statement**
- ✓ **Modifying the INPUT statement.**

```
**********************************************************; EXAMPLE3.SAS
data example3;
informat race $16.;          * Note the syntax $16.;
input age race $ parity meno bmi bmi2 stage type;
datalines;
69    WHITE 0     1     23.43 22.41 1     .
.     WHITE 2     1     31.31 29.23 1     1
61    AFRICAN-AMERICAN  2     1     22.85 24.15 1     .
82    WHITE 2     1     34.27 32.78 0     .
37    AFRICAN-AMERICAN  2     0     21.68 20.89 0     1
71    .     2     1     41.84 40.76 1     .
57    AFRICAN-AMERICAN  2     1     29.93 28.34 0     1
75    AFRICAN-AMERICAN  .     1     .     22.32 0     0
65    WHITE .     .     .     .     .     .
55    WHITE 0     1     29.59 27.67 1     .
;
proc print data=example3;
title 'Tabulation of data from EXAMPLE3 data set';
      var age race parity meno bmi bmi2 stage type;
run;
**********************************************************;
data example3;
length race $ 16;            * Note the syntax $ 16;
input age race $ parity meno bmi bmi2 stage type;
datalines;
69    WHITE 0     1     23.43 22.41 1     .
.     WHITE 2     1     31.31 29.23 1     1
61    AFRICAN-AMERICAN  2     1     22.85 24.15 1     .
82    WHITE 2     1     34.27 32.78 0     .
37    AFRICAN-AMERICAN  2     0     21.68 20.89 0     1
71    .     2     1     41.84 40.76 1     .
57    AFRICAN-AMERICAN  2     1     29.93 28.34 0     1
75    AFRICAN-AMERICAN  .     1     .     22.32 0     0
65    WHITE .     .     .     .     .     .
55    WHITE 0     1     29.59 27.67 1     .
;
proc print data=example3;
title 'Tabulation of data from EXAMPLE3 data set';
      var age race parity meno bmi bmi2 stage type;
run;
```

Introduction to Fundamentals of SAS

```
*****************************************************;
data example3;
input age race : $16. parity meno bmi bmi2 stage type;
                      * Note the syntax :  $16.;
datalines;
69    WHITE 0     1     23.43 22.41 1     .
.     WHITE 2     1     31.31 29.23 1     1
61    AFRICAN-AMERICAN  2     1     22.85 24.15 1     .
82    WHITE 2     1     34.27 32.78 0     .
37    AFRICAN-AMERICAN  2     0     21.68 20.89 0     1
71    .     2     1     41.84 40.76 1     .
57    AFRICAN-AMERICAN  2     1     29.93 28.34 0     1
75    AFRICAN-AMERICAN  .     1     .     22.32 0     0
65    WHITE .     .     .     .     .
55    WHITE 0     1     29.59 27.67 1     .
;
proc print data=example3;
title 'Tabulation of data from EXAMPLE3 data set';
      var age race parity meno bmi bmi2 stage type;
run;
*************************************************;
```

Note:

- ✓ Use of INFORMAT statement and the LENGTH statement.
- ✓ Alternative methods are also shown
- ✓ Comment lines start with an "*asterisk*" and ends with a semi-colon (;)

Example 4: Reading data values where missing values are represented by Periods except at the end of short data lines

In the data set used here note that the missing values are represented by periods as before but except at the end of the short data line.

As an example, look at the 4th observation:

82 W 2

The data for rest of the variables is missing but has not been represented by periods. This is an example of a short data line.

To make SAS read such data lines without any error, use MISSOVER (or TRUNCOVER) option as follows:

Infile datalines missover;

This statement tells SAS that if you reach the end of a data line and have not yet read values for all variables, set all the remaining values to missing.

EXAMPLE4.SAS

```
* An example that uses MISSOVER option;
data example4;
      infile datalines missover;
      input age race $ parity meno bmi bmi2 stage type;
datalines;
69 W 0 1 23.43 22.41 1 .
. W 2 1 31.31 29.23 1 1
61 AA 2 1 22.85 24.15 1 0
82 W 2
37 AA 2 0 21.68 20.89
57 AA 2 1 29.93 28.34 0 1
75 AA . 1 . 22.32 0 0
65 W . . . . . .
;
proc print data=example4;
      title 'Tabulation of data from EXAMPLE4 data set';
run;
```

Note: "*Missover*" option says that if you reach the end of a data line and have not yet read values for all variables, set all the remaining values to missing.

```
*****************************************;
* The same example as above but using TRUNCOVER option;
data example4;
      infile datalines truncover;
      input age race $ parity meno bmi bmi2 stage type;
datalines;
69 W 0 1 23.43 22.41 1 .
. W 2 1 31.31 29.23 1 1
61 AA 2 1 22.85 24.15 1 0
82 W 2
37 AA 2 0 21.68 20.89
57 AA 2 1 29.93 28.34 0 1
75 AA . 1 . 22.32 0 0
65 W . . . . . .
;
proc print data=example4;
      title 'Tabulation of data from EXAMPLE4 data set';
run;
```

Introduction to Fundamentals of SAS

Example 5:

Reading data values separated by commas, missing values are represented by two adjacent commas except at the beginning of a record, and the data for character variables are placed in double quotes

Sometimes the data set comes in the following format as shown below. That is,

- **The data for character type variable is enclosed in " "**
- **Missing values are represented by two adjacent commas except at the beginning of a data line as in 2nd record.**

In order to make SAS read such data sets correctly use the statement,

infile datalines dsd; `* DSD = Delimiter Sensitive Data;`

EXAMPLE5.SAS

```
options nodate nonumber;
data example5;
      infile datalines dsd;
input lastname $ id $ gender $ gpa height weight;
datalines;
"smith","123","male",3.5,,155
,"221","female",3.7,52,101
"hopkins","323",,,,,
"reed","105","female",3.0,55,,
"dobbs","143","male",3.5,60,175
;
proc print data=example5;
      title 'Tabulation of data from EXAMPLE5 data set';
run;
```

Note:

"dsd" option allows you to read comma-delimited data, to treat two consecutive commas as a missing value, and to remove the double quotes from quoted strings.

Example 6: Reading data values using starting and ending column numbers

This is an example of the second type of input: COLUMN input

Note that the data for the variable NAME starts at column 1 and can go up to column 14.

Data for ID starts at column 16 and end at column 18. Similarly for other variables.

Most external files have a regular pattern for data location. Column input is one of the two methods that can be used to read such data.

EXAMPLE6.SAS

```
options nodate nonumber;
data example6;
      input name $ 1-14 id $ 16-18 gender $ 23-28 gpa 31-33 height 35-36
            weight 39-41;
datalines;
victor          123    male    3.5 59  155
joan            221    female  3.7 52  101
roger           323    male    2.4 65  210
amy             105    female  3.0 55  120
steve           143    male    3.5 60  175
;
proc print data=example6;
      title 'Tabulation of data from EXAMPLE6 data set';
run;
```

Note:

Data for the variable *NAME* should start at column1 and can go up to column 14. Data for *ID* should start at column 16 and end at column 18. Similarly for other variables.

Example 7: Reading data values with missing data for character and numeric Variables Using starting and ending column numbers

* **This example is same as Example 6;**

EXAMPLE7.SAS (example of COLUMN input)

```
options nodate nonumber;
data example7;
            input name $ 1-14 id $ 16-18 gender $ 23-28 gpa 31-33
                  height 35-36 weight 39-41;
datalines;
victor          123    male    3.5     155
                221    female  3.7 52  101
roger           323
amy             105    female  3.0 55
steve           143    male    3.5 60  175
;
proc print data=example7;
      title 'Tabulation of data from EXAMPLE7 data set';
run;
```

Introduction to Fundamentals of SAS

Example 8: Reading data values using pointers (@) and informats

This is an example for the third type of input: FORMATTED input:

Like the column style, FORMATTED input requires that you know the location of the beginning of the variable in the line of data and how many columns it uses.

Instead of using starting and ending column numbers, we have use column pointer symbol (@) and informats to read the data values to create a SAS data set.

In this example, the data for the variable NAME begins in column 1 and you can enter up to 14 characters ($14 indicates character type variable of length 14). The 15th column is a blank space. The data for GPA starts in column 31 and can use 3 digits with no decimal part (that is why the format is 3.)

EXAMPLE8.SAS (example of FORMATTED input)

```
options nodate nonumber;
data example8;
      input @1 name $14.
            @16 id $3.
            @23 gender $6.
            @31 gpa 3.
            @35 height 2.
            @39 weight 3.;
datalines;
victor          123    male    3.5 59  155
joan            221    female  3.7 52  101
roger           323    male    2.4 65  210
amy             105    female  3.0 55  120
steve           143    male    3.5 60  175
;
proc print data=example8;
      title 'Tabulation of data from EXAMPLE8 data set';
run;
```

Note:

Instead of using starting and ending column numbers, we have used ***column pointers*** (@) and ***informats*** to read the data values to create a SAS data set called EXAMPLE8.

Introduction to Fundamentals of SAS

Example 9: Use of @@ to read multiple observations that are on the same line.

Sometimes multiple observations are placed on the same data line.

For example, in this data set there are only 5 variables but on each line you see more than 5 pieces of data.

In order to make SAS understand that there are multiple observations on the same line and to make it allocate first 5 pieces of data as one record, the next 5 pieces of data as the second record and so on, we use two @ symbols as @@ at the end of the INPUT statement.

EXAMPLE9.SAS

```
*****************************************************************;
* In the following data set, multiple observations are placed on
  the same line. In order to make SAS understand how to read
  this data, one needs to use @@;

* There are five variable in the INPUT statement. The symbol @@
  allows SAS to read the following data; Note that there are 4 records per
  child, and there are 10 children. There should be a total of 40 records.;
*****************************************************************;
data example9;
input number child age distance gender @@;
datalines;
1 1 8 21 0 2 1 10 20 0 3 1 12 21.5 0 4 1 14 23 0 5 2 8 21 0 6 2 10 21.5 0 7 2
12 24 0 8 2 14 25.5 0 9 3 8 20.5 0 10 3 10 24 0 11 3 12 24.5 0 12 3 14 26 0
13 4 8 23.5 0 14 4 10 24.5 0 15 4 12 25 0 16 4 14 26.5 0 17 5 8 21.5 0 18 5
10 23 0 19 5 12 22.5 0 20 5 14 23.5 0 21 6 8 20 0 22 6 10 21 0 23 6 12 21 0
24 6 14 22.5 0 25 7 8 21.5 0 26 7 10 22.5 0 27 7 12 23 0 28 7 14 25 0 29 8 8
23 0 30 8 10 23 0 31 8 12 23.5 0 32 8 14 24 0 33 9 8 20 0 34 9 10 21 0 35 9
12 22 0 36 9 14 21.5 0 37 10 8 16.5 0 38 10 10 19 0 39 10 12 19 0 40 10 14
19.5 0
;
run;
*************************************************************;
proc print data=example9;
      title "Use of @@ to read the data set that has multiple";
      title2 "observations placed on the same line";
run;
*************************************************************;
```

Note:

- If you run the program without @@ symbol, then, the program reads the first 5 pieces of data as one record, and ignores the rest of the data.

- Please note that the program does not give any error statements in the Log Window.

Introduction to Fundamentals of SAS

Summary of LIST, COLUMN, and FORMATTED

LIST input:

Data in this format are entered without regard to column location.

Two restrictions are placed:

First, variable values must be separated by one or more blanks.
Second, there must be as many values in each line of data as there are variable names in the INPUT statement.

```
input name $ id $ gender $ height weight center drug $ changescore;
```
(EXAMPLE1)

- *$* sign identifies the variable as a *character* variable.
- Missing value should be represented by a period (**.**)
- Default length for character data value is 8
- Character data values cannot contain spaces

COLUMN input

Most external files have a regular pattern for data location. Column input is one of the two methods that can be used to read such data.

```
Input   name $ 1-14 id $ 16-18 gender $ 23-28 gpa 31-33 height 35-36 weight 39-41;
```
(EXAMPLE6)

FORMATTED input:

Like the column style, FORMATTED input requires that you know the location of the beginning of the variable in the line of data and how many columns it uses.

It uses what is called *input formats* ***or*** *informats* for short. These *informats* are instructions that tell the SAS how to read raw data.

```
input   @1 name $14.
        @16 id $3.
        @23 gender $6.
        @31 gpa 3.
        @35 height 2.
        @39 weight 3.;
```
(EXAMPLE8)

Which Input Format to use?

- The INPUT format one selects is based partly on the characteristics of the data and partly on preferences one develops over time.
- List INPUT is useful if you have few observations and/or few variables
- Column and formatted INPUT give more flexibility if the data have to be read from external sources such as universities, hospitals, and government agencies.