(2) PROGRAMS, DATA SETS, RULES

...SAS PROGRAMS

For many of you, the only experience you have with working with data involves using some "point-and-click" interface. You work with data by selecting analysis tools via drop-down menus and the click of a mouse button. SAS is different in that you write statements that instruct SAS how to read, analyze, and report data. This is not quite 100% true in that new features have been added to SAS that allow more of an interactive style of working with data. However, the statement is true enough in that it is how SAS will be explained in the remainder of these notes.

If you have some programming experience using other languages, you might have noticed in the first example that you do not have to write much code to get results (especially if your experience has been with a COBOL or a similar programming language). If you do not have any experience writing programs, just think of learning how to write SAS statements as being similar to learning how to express your ideas by writing in a foreign language, one where you have to be precise with the syntax and where the only really important punctuation mark is a semicolon. The other analogy to learning a foreign language is that it is quite easy to learn the nouns, verbs, and other parts of speech. The difficult part is putting them all together so what you say or write makes sense. This is also true of SAS. Learning about all the various parts such as data steps, PROCs, and global statements is not as difficult as learning how to put the various parts together properly to do something with or answer a question about your data.

A SAS program (or SAS job) can be made up of one or more data steps, one or more PROCs (procedures), or a combination of data steps and PROCs. There might also be global statements such as the LIBNAME, TITLE, and OPTIONS statements shown in chapter 1. The data step has a number of different functions. One of them is to convert raw data into a SAS data set. Data steps range from the very simple or to the very complex. The data step gives you the functionality of a programming language such as C, COBOL, or BASIC. SAS PROCs (short for procedures) are pre-written programs that work with data that are stored as a SAS data set.

The data steps and PROCs in a SAS program are all discrete units of SAS code made up of SAS statements. You can think of each of these discrete units as being a mini-SAS program since when you run a SAS job, each of the units (the data steps and/or PROCs) each run independently and run in the order in which they appear in the SAS job. Imagine you just start the SAS software on your PC and you enter the following statements in the enhanced editor window.

...Example 2.1

```
proc print data=mydata.class; ①
run;

libname mydata "k:\sasclass\datasets"; ②

title "SAS-SUPPLIED DATA SET CLASS";
```

You would see the SAS log shown on the right. The first thing that happens when you run the SAS job is PROC PRINT ①. That procedure tries to use a SAS data set name MYDATA.CLASS. However, you have yet to create the libref MYDATA in a LIBNAME statement ②. In the above SAS code, the LIBNAME statement occurs after PROC PRINT and since all the steps occur in sequence, the PROC cannot find your data set.

```
1    proc print data=mydata.class;
ERROR: Libname MYDATA is not assigned.
2    run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

3
4    libname mydata "k:\sasclass\datasets";
NOTE: Libref MYDATA was successfully assigned as follows:
      Engine:        V9
      Physical Name: k:\sasclass\datasets
5
6    title "SAS-SUPPLIED DATA SET CLASS";
```

...Example 2.2

```
libname mydata "k:\sasclass\datasets";

proc print data=mydata.class;
run;

title "DATA SET CLASS"; ①
```

SAS-SUPPLIED DATA SET CLASS

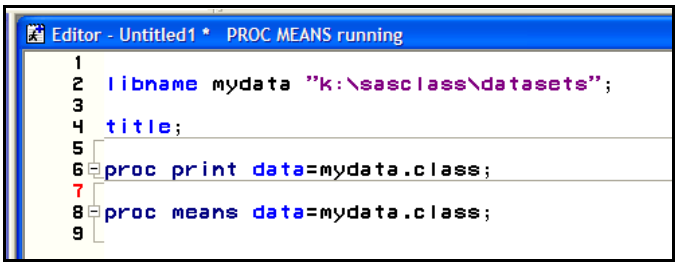
Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84.0
10	John	M	12	59.0	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90.0
13	Louise	F	12	56.3	77.0
14	Mary	F	15	66.5	112.0
15	Philip	M	16	72.0	150.0
16	Robert	M	12	64.8	128.0
17	Ronald	M	15	67.0	133.0
18	Thomas	M	11	57.5	85.0
19	William	M	15	66.5	112.0

You correct your mistake and move the LIBNAME statement above the procedure. You also decide to change the title ①. The output is shown on the right. PROC PRINT was able to use the data set since you moved the LIBNAME statement. However, look at the title. Since your new title is in a TITLE statement that occurs after PROC PRINT, the old title that is still in place from example 2.1 is placed on the output.

You may have noticed in all the examples thus far that all data steps and PROCs have ended with a RUN statement. The RUN statement is not always necessary, but it is good practice to end every data step and every PROC with a RUN statement. For example, in the following SAS job, PROC PRINT would run correctly, but PROC MEANS would never finish.

...Example 2.3

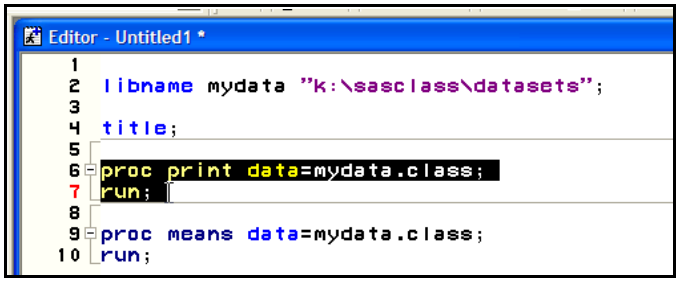
```
libname mydata "k:\sasclass\datasets";

proc print data=mydata.class;

proc means data=mydata.class;
```



If you looked at the output window, you would see the results from PROC PRINT. You would see no output from PROC MEANS. If you looked at the enhanced editor window, you would see the message on the right in the blue banner bar above the SAS code. Why is the PROC still running?

Each data step and PROC in a SAS job is a unit of work to be done by SAS. SAS knows that it should complete a unit of work when it encounters another unit of work or a RUN statement. In example 2.3, PROC PRINT completed since SAS encountered another unit of work, PROC MEANS. PROC MEANS never completed since there are no more units of work (data steps or PROCs) and no RUN statement.

There are other reasons to use RUN statements to complete data steps and PROCs. One is that most SAS statements are either part of a data step or part of a PROC. It is easier to keep track of where the various statements belong if you place them between DATA and RUN or PROC and RUN. A common error is to have a SAS statement out of place in a SAS job. This is less likely to occur if you use RUN statements. Another reason to use RUN statements is that you can highlight only a portion of the SAS code in a SAS job and have SAS run only that portion if it ends with a RUN statement (as shown on the right).



Thus far, the only recommendation about how you structure your SAS programs is that you use RUN statements to end data steps and PROCs. You have a lot of freedom in how you write SAS jobs. You have already been told that most SAS statements begin with a keyword and end with a semi-colon. The number of lines you use for a statement and the spacing within and between statements is up to you. The examples used thus far all have had a certain style. Most statements are on one line. However INPUT statements have been written on multiple lines. There are blank lines between sections of the programs (data steps, PROCs, global statements) and sometimes there are blank lines within sections. This style of writing SAS code is intended to make it easy to read and easier to find errors. It is not always easy to find errors, but it can be made easier by not cramming all your SAS code into as little space as possible.

What you should remember...
Learning SAS is analogous to learning a foreign language whose building blocks are data steps and PROCs. These building blocks together with global statements comprise most SAS programs. The order of data steps, PROCs, and global statements is important since they are executed in the sequence in which they appear in a SAS job. Data steps and PROCs are run as soon as SAS sees the next data step, the next PROC, or a RUN statement. It is good practice to end data steps and PROCs with RUN statements for a number of reasons: easier to keep track of SAS statements and where they belong in a SAS job; all units of work will complete; makes it possible to highlight and run portions of a SAS job. Develop a style of writing SAS code that makes it easy for you (and others) to both read your SAS jobs and determine what you are trying to accomplish with various data steps and PROCs.

...SAS DATA SETS

A SAS data set is data in a proprietary format (similar to storing data in a spreadsheet used in Excel, or in a database that is used in Access, or even in a document that is used by Word) that can be "understood" by SAS. When you save data in a spreadsheet or your thoughts and ideas in a document, there are "extras" that are also saved. Some of these "extras" are items that control the appearance such as the font and font size. In a spreadsheet, you can assign a format to various cells and that information is also saved with your data.

When you save your data in a SAS data set, the attributes you assign to your data are also saved together with additional information about the data set. The output from PROC CONTENTS shown on pages 4 and 7 show that the "extras" in a SAS data set include information on both the attributes of all the variables in the data set (position, name, type, length, format, label) and attributes of the data set itself such as: the name of the data set; creation date; number of observations; number of variables.

If you are familiar with spreadsheets, you know that you refer to data as being stored in ROWS and COLUMNS. Each row usually represents some person, place, or thing and each column usually represents some characteristic of the data in each row. If you are familiar with databases, you know that you refer to data as being stored in RECORDS and FIELDS. Each record represents some person, place, or thing and each field represents some characteristic of the data in each record.

The analogous terms used for SAS data sets are OBSERVATIONS (instead of rows or records) and VARIABLES (instead of columns or fields). SAS data sets are rectangular in that each observation has the same number of variables. There may be observations where you do not have any data for a given variable but that observation still contains all the variables in the data set.

SAS data sets can be thought of as having two parts, a descriptor portion (sometimes referred to as the header) and the data portion. When you look at the output from PROC CONTENTS, that information resides in the descriptor portion of the data set. PROC DATASETS can be used to alter some of the information stored in the descriptor portion of a data set.

...Example 2.4

```
* same as example 1.4 but no LABEL or FORMAT statements;
data patients;
infile 'k:\epi514\data\health.csv' dsd;
input
ssn     :  $9.
gender  :  $1.
age
height
weight
pulse
sbp
dbp
;
bmi = 703 * weight / height**2;
run;

proc contents data=patients;
run;

proc datasets lib=work nolist; ①
modify patients; ②
label ③
ssn = 'SOCIAL SECURITY NUMBER'
sbp = 'SYSTOLIC BLOOD PRESSURE'
dbp = 'DIASTOLIC BLOOD PRESSURE'
bmi = 'BODY MASS INDEX'
;
format bmi 4.1; ④
quit; ⑤

proc contents data=patients;
run;
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	age	Num	8
9	bmi	Num	8
8	dbp	Num	8
2	gender	Char	1
4	height	Num	8
6	pulse	Num	8
7	sbp	Num	8
1	ssn	Char	9
5	weight	Num	8

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
3	age	Num	8		
9	bmi	Num	8	F4.1	BODY MASS INDEX
8	dbp	Num	8		DIASTOLIC BLOOD PRESSURE
2	gender	Char	1		
4	height	Num	8		
6	pulse	Num	8		
7	sbp	Num	8		SYSTOLIC BLOOD PRESSURE
1	ssn	Char	9		SOCIAL SECURITY NUMBER
5	weight	Num	8		

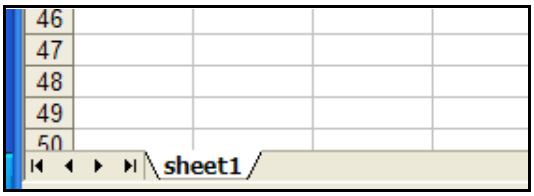
The data step from example 1.4 is used, but the LABEL and FORMAT statements are left out. The results of the first PROC CONTENTS are shown in the top box. There are no labels or formats. PROC DATASETS is used to modify the data set ①. Rather than specifying a data set with DATA= as has been done with all the PROCs thus far, a library is specified with LIB=. Since the data set PATIENTS is a temporary data set, we use LIB=WORK. The option NOLIST suppresses a listing in the LOG of the names of all the data sets in the WORK library. A MODIFY statement specifies the name of the data set to be used in the PROC ②. The LABEL ③ and FORMAT ④ statements add new attributes to the variables in the data set. Notice that a QUIT statement (not a RUN statement) ends the PROC ⑤. The results of the second PROC CONTENTS are shown in the bottom box. The new attributes are now part of the data set. This is the most efficient method for adding/removing attributes such as formats and labels to/from an already existing data set.

At one time, SAS PROCs only would work with data in the form of a SAS data set. This is no longer true since SAS procedures can now use data stored in a number of different formats. If you are a bit of a computer geek, you may be familiar with terms such as DDE (dynamic data exchange) or ODBC (open data base connectivity) that refer to data in a proprietary format for one piece of software being read by another. Both DDE and ODBC can be used within SAS to access data that is not in a SAS data set. SAS allows access on non-SAS data using data base engines. You can use data stored in ACCESS, EXCEL, ORACLE, SYBASE without having to first convert your data to a SAS data set. On page 5, there is figure on the right showing data in an Excel spreadsheet. The following will use the data directly from the spreadsheet. This is 'extra' and not something you are expected to be able to do at this point. However, you should know that SAS can use data that resides in non-SAS files.

...Example 2.5

```
* the name of an Excel spreadsheet;
libname x 'k:\epi514\xls\class.xls'; ①

* use the spreadsheet in a proceedure;
proc means data=x.'sheet1$'n; ②
run;

* add a new variable --- read the old workbook, write a new workbook;
data x.sheet2; ③
set x.'sheet1$'n; ④
bmi = 703 * weight / height**2;
run;

libname x clear; ⑤
```

This LIBNAME statement is different from those used previously in that it specifies the name of a file (a spreadsheet), not the name of a folder ①. The data in the Excel file are used in PROC MEANS without first converting them to a SAS data set ②. After DATA= you see the libref x followed by some other text. That other text is the name of the workbook within the spreadsheet you are using. If you are familiar with Excel, you will already know that on the bottom of the display of a spreadsheet, you will see the name of one or more workbooks. The only workbook within the file CLASS is SHEET1 (look a the box on the right). Just accept the fact that you have to: add a '$' to the end of the workbook name; place the name in quotes; add an 'n' after the name of the workbook.



If you look at the LOG (top box on the right) after using PROC MEANS you will see that SAS treated the data in the Excel file just like a SAS data set. It is also possible to write to an Excel file directly. A data step is used to create a new workbook, SHEET2, in the Excel file ③. Notice that when you create the new workbook, you use a standard SAS data set name. But the libref is the Excel file. A set statement reads the workbook data just as if it were in a data set ④. A new variable is created, BMI. The workbook SHEET2 will have all the data from SHEET1 plus the new variable BMI. As you can see in the LOG (lower box above), SAS treated the existing workbook SHEET1 as a SAS data set and also created a new workbook named SHEEt2 in the Excel file. Both PROC MEANS and the data step look as if you are working with SAS data sets, not workbooks in an Excel file. SAS 'owns' the Excel file until you submit the last LIBNAME statement ⑤. Until you do that, you will not be able to use the spreadsheet in Excel.

```
554
555  * use the spreadsheet in a proceedure;
556  proc means data=x.'sheet1$'n;
557  run;

NOTE: There were 19 observations read from the data set X.'sheet1$'n.
NOTE: PROCEDURE MEANS used (Total process time):
      real time               0.21 seconds
      cpu time                0.07 seconds
```

```
NOTE: SAS variable labels, formats, and lengths are not written to DBMS tables.
NOTE: There were 19 observations read from the data set X.'sheet1$'n.
NOTE: The data set X.sheet2 has 19 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time               0.09 seconds
      cpu time                0.03 seconds
```

For now, forget about using non-SAS data sets within SAS data steps and PROCs. Work under the assumption that if you want to use data in any SAS procedures, your data must be in the form of a SAS data set. That makes understanding the data step very important since it is the only way for you to take raw data and convert it into a SAS data set.

All SAS data set names have two parts, a libref and the name of a data set within the folder specified by the libref. That folder is a SAS data set library. It could be on a flash drive, on the PC hard disk, on a network drive, wherever. If you create a SAS data set in the course of a SAS session and only use the second portion of the data set name (leave out the libref), SAS creates the data set in the WORK library. The WORK library is in a location that was specified when SAS was installed. If you are curious as to the name of the folder that is being used as the WORK library, you can submit the following single line of SAS code and the folder name will be listed in the LOG...

```
libname work list;
```

As soon as you exit SAS, all the files in the WORK library created during your SAS session are erased. Data sets written to the WORK library are referred to as temporary. If you want to keep any or all of the data sets created within a SAS session, they must be created using a two level name using both a libref and a data set name. Data sets written to a library other than WORK are referred to as permanent. A permanent data set named BMI is created in example 1.2. A temporary data set names PATIENTS is created in both example 1.4 and example 2.4 . Creating a permanent SAS data set allows use to use it again in another SAS session, as is shown in example 1.1. In that example, the previously created and saved data set CLASS is used.

What you should remember...
A SAS data set is data in a proprietary format that can be used in SAS PROCs and data steps. Storing your data in data set is similar keeping it in an Excel spreadsheet. Data sets comprise observations (equivalent to spreadsheet rows) and variables (equivalent to spreadsheet columns). PROC CONTENTS will give you information about a data set, including but not limited to: number of observations; number of variables; a list of all the variables and their attributes. PROC DATASETS allows you to change some of the attributes of a data set and it is the most efficient method for doing so. All data sets have a two part name made up of a libref plus a data set name. The libref tells SAS a location of the data set and that location is a folder, also called a data set library. The data set name is the name of a file within the folder. Data sets can be thought of as being either temporary or permanent. Temporary data sets are created in library with the name WORK. Permanent data sets are created in a library other than WORK and if you are creating or using a permanent data set, your SAS job should contain a LIBNAME statement.

...RULES

There are a number of rules that you will have to remember if you want to run SAS programs. There are also a lot of rules that you do not have to remember (that is why we have SAS manuals and on-line help). The following is a set of rules (borrowed from a SAS consultant) that you should try to remember since most are common to all the SAS code you will ever write. Yes, there are manuals and on-line help, but you do not want to look up everything.

- SAS statements begin with a KEYWORD and end with a semicolon. In the example SAS jobs seen thus far, some of the keywords are DATA, INPUT, LABEL, INFILE, LIBNAME.
- Both data set and variable names are limited to 32 characters.
- Librefs created in a LIBNAME statement are limited to 8 characters.
- Data set name, variable names, and libref names must start with a letter or an underscore. Subsequent characters may be letters, numbers, or underscores.
- There are only two types of SAS variables, CHARACTER and NUMERIC. Unless you explicitly state otherwise, variables are assumed to be NUMERIC.
- The default length of the content of all SAS variables is eight.

- The maximum length of the content of a SAS character variable is 32,000 (actually ... 32,767)

- SAS programs are made up of one or more units of work. Units of work are either data steps or PROCs, and a single SAS program usually contains more than one unit of work.

- A SAS System unit of work is terminated by a RUN statement or by another unit of work.

- SAS operates on units of work in the exact order that they appear in the program.

- Units of work are made up of SAS statements and many statements are specific to either a data step or a PROC. However, some statements can be used in either data steps or PROCs including FORMAT, LABEL, and WHERE.

- A data step operates on one observation at a time.

- A PROC operates on all the observations in a data set.

- There are at least three ways to do any one thing with SAS - one wrong, and at least two right ways.

- ***VERY IMPORTANT RULE...*** For each of the above rules there exists at least one exception, knowledge of which is not required to learn SAS System fundamentals.

In addition to the rules, there are also some suggestions.

- SAS statements can be written in 'free format' since SAS recognizes the beginning of a statement via a keyword and keeps reading a statement until it encounters a semicolon. You can break SAS statements across lines, start in any column, use upper and/or lower case, embed blanks, and include blank lines. More than one SAS statement can be put on one line. This gives you all the flexibility to become a real sloppy programmer. You should use the flexibility to develop a style, not to be sloppy.

- Avoid using SAS keywords (DATA, INPUT, etc.) as variable names.

- Just because you can use up to 32 characters to name variables does not mean that you should get into the habit of using long variable names. Use short variable names that make sense to you, that they remind you of the content of the variable. Then, use labels to show more about what the variable represents. Remember, if you use SBP to name a variable that represents systolic blood pressure, you can use SBP each time to refer to that variable. If you use SYSTOLIC_BLOOD_PRESSURE as the variable name, you will be doing a lot more typing each time you want to use that variable in a data step or procedure.

Try to keep the rules in mind as you look at SAS code since this will help you to commit them to memory.

What you should remember...
Everything, yes everything.