(1) INTRODUCTION

Once, the acronym SAS actually did stand for *Statistical Analysis System*. Now, when you use the term SAS, you are referring to a collection of integrated software products that you can use for data entry/management/analysis, report writing, statistical analysis, forecasting, graphics, mapping, etc. Some of these products are...

BASE	"basic product" - needed to run all other products - data analysis, report writing, some statistical analysis, some graphics, programming
STAT	advanced statistical analysis (regression, analysis of variance)
GRAPH	presentation quality graphics for just about any type of output device or graphics file type (laser printer, PDF, GIF, JPEG, PNG...)
ETS	time series
ETC.	"...more than you'll ever need or use..."

You are all probably familiar with word processing programs such as *Word*. If so, you know that there are many features within such programs that you rarely if ever use. The main thing you do with a word processing package is create documents (letters, term papers). You do 95% of your work with about 5% of the features that are available, but it is nice to know that the other 95% of the software allows you to perform tasks such as generating a table of contents, automatically inserting footnotes, or creating mailing lists.

Unless working as a SAS consultant or teaching SAS becomes your career, you will probably use a very small portion of all the available features in SAS. Just as with a word processing package, you'll most likely use 5% of the features to do a significant percentage of your data management, analysis, and reporting. However, given the breadth of tasks that can be accomplished with SAS and the size of the total package, mastering only that 5% might still seem daunting. If you encounter a bookcase full of SAS documentation, it is even hard to know where to start to find that important 5%. Hopefully, these notes will point you in the correct direction.

So, what is SAS? The jargon used at the start of these notes about "...a collection of integrated software products..." does not tell you too much. A simpler definition is that SAS is a programming language that also has a collection of pre-written programs that help you to work with data. The programming is usually accomplished with something called a data step, while the pre-written programs are referred to as procedures (or PROCs). When you hear someone say that they have written a SAS job, it is most likely a combination of one or more data steps and/or one or more PROCs (that's it...data steps and procedures). That SAS job may also contain some more "stuff" such as a few "global statements", but most likely data steps and PROCs comprise most if not all of the SAS job.

There is just one last introductory item to mention. All those pre-written programs, the PROCs, can only be used with data that is stored in something called a SAS data set. If you have some experience working with data in a program such as Excel or Access, you know that they expect your data to be in a specific type of file, usually an XLS file for Excel and an MDB file for Access. If you consider your thoughts and ideas as "data", you know that a program such as Word expects your writing to be in a specific type of file, usually a DOC file. The same is true for SAS in that PROCs expect your data are in a SAS data set.

What you should remember...

Remember the terms data step, PROCs (or procedures), SAS data sets. You really do not know what these things are just yet, but those three concepts are very important, part of mastering that essential 5%. And one more thing to remember...there are most likely exceptions to everything you have read thus far.

...SIMPLE SAS JOBS - STARTING WITH A SAS DATA SET

Despite what you might have heard, getting work done with SAS is not that hard. Let's look at some simple SAS jobs that contain all the things you are supposed to remember from the introduction: procedures; a data step; a SAS data set. There are even a few global statements. The first SAS job prints a report, computes some simple statistics, and shows you the attributes of a SAS data set. If your entire experience with a computer is 'point-and-click', the following look at SAS might not be enough to dissuade you of the notion that 'SAS is hard'. Hopefully that is not the case. Even if you are the type that gets discouraged easily, please keep reading. That moment of clarity is within reach if you just keep at it. By the way, there is a point-and-click interface to SAS called Enterprise Guide. There are also other point-and-click applications within SAS, for example SAS/Insight. These notes are not meant to hide these interfaces and applications from you. These notes present a programming approach to learning SAS that is applicable in more situations you will encounter in working with data than any of the other ways of using SAS.

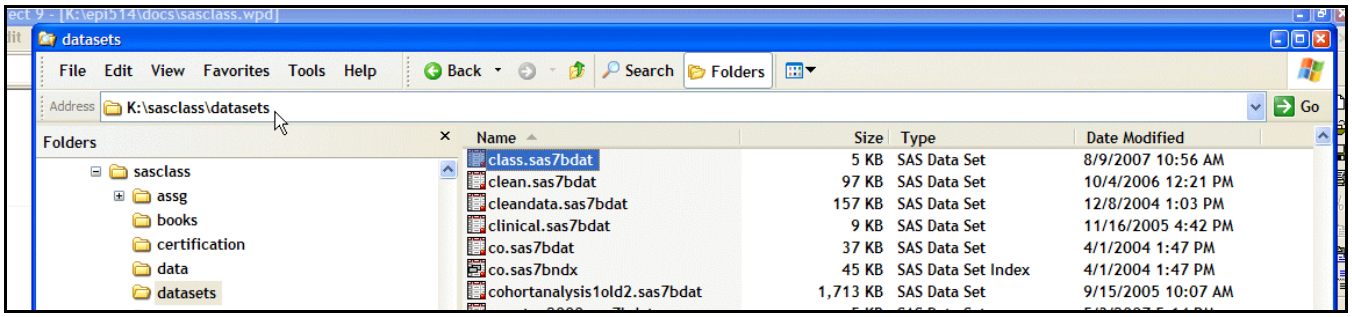
...Example 1.1

```
* LIBNAME is a global statement - tells SAS where your data are located (a folder);  ①
libname mydata "k:\sasclass\datasets"; ②

* PROC PRINT is a SAS procedure that displays VALUES of your data;
proc print data=mydata.class; ③
run;

* TITLE is a global statement - adds a title to PROC output;
title "SAS-SUPPLIED DATA SET CLASS"; ④

* PROC MEANS is a SAS procedure that computes basic statistics;
proc means data=mydata.class; ⑤
run;

* PROC CONTENTS is a SAS procedure that displays ATTRIBUTES of your data;
proc contents data=mydata.class; ⑥
run;
```

The first thing you might have noticed is that there are a lot of semi-colons (;). Here are a few more things to remember. You just learned that a SAS job is most likely a combination of one or more data steps and/or one or more PROCs and maybe some global statements. Now you should remember that data steps and PROCs are made up of statements and that each statement ends with a semi-colon. Most statements start with a keyword (in this example the keywords are: LIBNAME, PROC, RUN, TITLE). How many statements in example 1.1 ... just count the semi-colons ... how about 13. The first statement is a comment, one of the global statements in SAS ①. There are two types of comment statements in SAS, but for now concentrate on learning about one of them. The one shown above starts with an asterisk (*) and ends with a semi-colon (;) and all the text in between is treated as text you have written for reasons such as explaining your SAS job to others or so you can remember what each section of your SAS job does. How many comments are there in example 1.1 ... how about 5.

LIBNAME is another global statement ②. This statement is used to tell SAS the location of one or more SAS data sets. The word LIBNAME is followed by a libref. A libref serves as a shortcut for the name of a folder (directory). The final entry in the LIBNAME statement is the name of a folder (directory). In this example, the LIBNAME statement creates the libref "mydata" that is a shortcut for the folder "k:\sasclass\datasets". That might seem like a lot of SAS jargon, but understanding the role of a LIBNAME statement is very important and there is more to learn about this statement. For now, just try to remember ... LIBNAME followed by libref followed by a folder name. If you still have room to remember a bit more ... (from SAS on-line documentation) ... ***A libref can be one to eight characters in length. It must start with a letter (A through Z) or an underscore (_), but it can otherwise contain any combination of letters, numbers, and underscores.***

Next is a SAS procedure, PROC PRINT ③. The PRINT procedure lists the values of the variables in a SAS data set. In this example, the data set is MYDATA.CLASS. This is your introduction to how SAS data sets are named. All SAS data sets have a two part name, a libref followed by the name of a file. That is another important concept...SAS data set names contain information on both the location of the data set (remember that a libref is a shortcut for the name of a folder) and the name of a file within that location (a file within a folder). If you listed the files in the folder "k:\sasclass\datasets" you would see a file named CLASS.



As was mentioned with the LIBNAME statement, there is a more to learn about using PROC PRINT. For now, it is enough to remember that...PROC PRINT will list the values of the variables in a SAS data set and that you use PROC PRINT by specifying a SAS data set name. PROC PRINT produces the output shown below, on the left .

The SAS System

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

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

The default behavior of PROC PRINT is to display all the observations and variables in a data set. The columns are labeled with the variable names while the rows are numbered with the observation numbers. Those terms, variables and observations, should be added to what you remember from this example. If you had entered these data into an Excel spreadsheet, they might look something like the figure above, on the right. In Excel, you refer to your data as being in rows and columns. When your data are in a SAS data set, they are in observations and variables. How many observations and variables are there in the data set MYDATA.CLASS ... how about 19 observations and 5 variables.

Another global statement is next, TITLE ④. When you use a SAS procedure that produces printed output, there is a default SAS-supplied title that appears on all your output...*The SAS System*. Look at the PROC PRINT output on the previous page. If you like that title, you do not need any title statement(s). However, you can provide your own title with a TITLE statement, the word TITLE followed by text within either single or double quotes. If you want no title, use the word TITLE followed immediately by a semi-colon (no text in quotes). You can provide up to ten titles using TITLE1, TITLE2, ..., TITLE10 in place of the word TITLE.

PROC MEANS is another SAS procedure ⑤. It computes basic statistics on all the numeric variables in a SAS data set. The term 'numeric variable' is new and yet to be described, just be patient. You can see that you use this SAS procedure in the same way that you used PROC PRINT... the word PROC followed by the procedure name followed by DATA= and the name of a SAS data set. Yes, there is a lot more to learn about PROC MEANS, but it does work with just the minimal amount of information you see in example 1.1. PROC MEANS produces the output shown below.

SAS-SUPPLIED DATA SET CLASS

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	19	13.3157895	1.4926722	11.0000000	16.0000000
Height	19	62.3368421	5.1270752	51.3000000	72.0000000
Weight	19	100.0263158	22.7739335	50.5000000	150.0000000

The default behavior of PROC MEANS is to calculate the mean and standard deviation and to find the minimum, and maximum values for each numeric variable. The names of the variables and the number of observations used for the calculation of statistics is also displayed. Notice the title.

The last step in this example is another procedure, PROC CONTENTS ⑥. PROC PRINT showed you the values of the variables in your data. PROC MEANS computes statistics based on the values of numeric variables in your data. PROC CONTENTS does not use or tell you anything about variable values. What it does give you is information about the attributes of your data in the output shown below.

SAS-SUPPLIED DATA SET CLASS

Data Set Name	MYDATA.CLASS	Observations	19
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Thursday, August 09, 2007 10:56:28 AM	Observation Length	40
Last Modified	Thursday, August 09, 2007 10:56:28 AM	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	101
Obs in First Data Page	19
Number of Data Set Repairs	0
File Name	k:\sasclass\datasets\class.sas7bdat
Release Created	9.0101M3
Host Created	XP_PRO

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

There is a lot of information in the PROC CONTENTS output. What you should notice (and remember) is that it includes: the name of the data set; the number of observations and variables in the data set; a list of the variables in the data set and their attributes. All variables in a SAS data set have the four attributes shown at the bottom of the output. Ignoring the first column for now, the first attribute is the variable name. Just as there is a rule for naming a libref (bottom of page 2), there are rules for naming variables in SAS data sets and note that the same rule also applies to the naming of SAS data sets. ... (from SAS on-line documentation) ... ***A SAS name can be up to 32 characters long. The first character must be a letter (A, B, C, . . . , Z) or underscore (_). Subsequent characters can be letters, digits (0 to 9), or underscores. Note that blanks are not allowed.***

The next attribute is the variable type ... either Num (for numeric) or Char (for character). If you are given a SAS data set, the attributes of your variables have already been determined. They have been given names. As you saw in the PROC PRINT output and in the Excel spreadsheet on page 3, the variable names in a SAS data set correspond to column names in a spreadsheet. How is a variable assigned a type? That is a bit harder to explain. A variable whose values are not numbers (variables NAME and SEX) must be a character variable. A variable whose values are all numbers (variables AGE, HEIGHT, and WEIGHT) can be either a numeric or character variable. That decision is usually based on whether the variable can be meaningfully analyzed as numeric data ... if you calculate descriptive statistics such as the mean or median, do those values have any real meaning. It would not make sense to calculate the mean or median of some variables whose values are always numbers such as a zip code or social security number. Another way to choose a variable type is to base your decision on the type of data it represents ... nominal, ordinal, interval, ratio ... with character variables for nominal and ordinal data and numeric variables for interval or ratio data For now, just remember that there are two types of variables in SAS data sets, character and numeric and the only rule you must follow is that non-numeric data must be a character variable.

The last two attributes shown in the PROC CONTENTS output are length (abbreviated Len) and position in the data set (the column labeled #). Just as the columns in a spreadsheet have a position (on page 3, column 1 is NAME, column 2 is SEX, ... , column 5 is WEIGHT), the variables in a SAS data set have a position. If you look at the output from PROC PRINT on page 3, the order of the variables is NAME-SEX-AGE-HEIGHT-WEIGHT. The corresponds to the numbers in the '#' column in PROC CONTENTS (#1 NAME ... #5 WEIGHT). Length is more difficult to explain. It is the amount of room needed to store the value of the variable in the data set. That does not explain much since you may have noticed that all the numeric variables have a length of 8, while the lengths of the two character variables are 8 for NAME (though the longest name is 7 letters) and 1 for SEX (finally, a length that makes sense since the values are only 1 letter, either M or F). For now, just remember that position and length are attributes of variables in a SAS data set. If you have a bit more memory space left, you could also remember that default length of all variables (numeric and character) in a SAS data set is 8 ... more explanation will follow when we actually create a SAS data set from some of our own data.

Before moving on to another example, in addition to all the semi-colons in example 1.1, you might have noticed that there are a lot of RUN statements. Though not all of them are necessary, it is good practice to end every use of a procedure with a RUN statement. Once again, there will be more about this later, in chapter 2.

The next example shows how to add another variable to a SAS data set. It also shows how to add two more attributes to a variable, a format and a label. Given that the data set contains the height and weight, we can compute the body mass index (BMI) for each person. With height in inches and weight in pound, BMI is computed as shown on the right.

$$BMI = 730 \times \left[\frac{weight(pounds)}{height(inches)^2} \right]$$

...Example 1.2

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

data mydata.bmi; ①
set mydata.class; ②
* formula for BMI with weight in pounds and height in inches;
bmi = 703 * weight / height**2; ③
label bmi = 'Body Mass Index'; ④
format bmi 4.1; ⑤
run;

title "NEW DATA SET BMI";
proc contents data=mydata.bmi varnum; ⑥
run;

proc print data=mydata.bmi label; ⑦
run;

*
modify default behavior of PROC MEANS
compute statistics for one variable (BMI)
compute statistics for data grouped by another variable (SEX)
;

proc means data=mydata.bmi mean min q1 median q3 max; ⑧
class sex; ⑨
var bmi; ⑩
run;
```

There is something new in this example, a data step, one of the two fundamental parts of SAS (the other being procedures) ①. The data step is used for many purposes, including creating new data sets and modifying already existing data sets ... (from SAS on-line documentation) ... ***The DATA step is one of the basic building blocks of SAS programming. It creates the data sets that are used in a SAS program's analysis and reporting procedures. Understanding the basic structure, functioning, and components of the DATA step is fundamental to learning how to create your own SAS data sets.*** The data step gives SAS the features of a programming language. For example, a common task with public health and epidemiologic data is age adjusting a rate. There is no procedure in SAS that will compute an age adjusted rate. However, you can write a short program in a data step that will compute either a direct or indirect age adjusted rate.

The reason we are using a data step in this example is to create a new data set name MYDATA.BMI by modifying an already existing data set named MYDATA.CLASS. The new data set will contain all the observations and variables in the old data set, plus a new variable named BMI. The first statement in a data step is always a data statement, starting with the word DATA followed by the name of one or more data sets ... yes, there are data steps, data statements, and data sets and it is important to understand all of them. Within a data step, you read an observation in an already existing SAS data set with a SET statement, SET followed by the name of a data set ②. The next statement creates the new variable BMI using the formula shown on the previous page ③. When writing equations in SAS, the new variable is placed to the left of the = sign. This is called an assignment statement, the results of the equation on the right are assigned as the value of the variable on the left. The equation includes multiplication (*), division (/), and raising a variable to a power (**).

Two new attributes are added to the variable BMI. A LABEL statement is used to assign the label 'Body Mass Index' ④ and a FORMAT statement is used to assign a rule for the display of the value of BMI ⑤. Variable labels can be up to 256 characters in length and labels allow you to supplement information that is conveyed by the name of a variable. Not everyone will know what BMI represents, so we add a label. The label will be used in place of or in addition to the variable name in the output of SAS procedures. A format is a rule for displaying the value of a variable. When SAS calculates the value of BMI, it does so

with many decimal places. When you assign the format 4.1 to the variable BMI, you are asking SAS to display the value of BMI in up to 4 characters, including a decimal point and rounded to one decimal place (leaving room for two numbers to the left of the decimal point). Just as all PROCS were ended with a RUN statement in example 1.1, a RUN statement ends the data step.

PROC CONTENTS is used with the VARNUM option ⑥ A portion of the PROC CONTENTS output shows the variables in the new data set in variable number (VARNUM) order, including the new variable BMI with two new attributes, a format and a label.

Variables in Creation Order

#	Variable	Type	Len	Format	Label
1	Name	Char	8		
2	Sex	Char	1		
3	Age	Num	8		
4	Height	Num	8		
5	Weight	Num	8		
6	bmi	Num	8	4.1	Body Mass Index

PROC PRINT is used the display the values of all the variables in the new data set ⑦. Notice that a LABEL option is added to the procedure. This asks PROC PRINT to display variable labels in place of variable names as the column headers. The output looks as follows.

NEW DATA SET BMI

Obs	Name	Sex	Age	Height	Weight	Body Mass Index
1	Alfred	M	14	69.0	112.5	16.6
2	Alice	F	13	56.5	84.0	18.5
3	Barbara	F	13	65.3	98.0	16.2
4	Carol	F	14	62.8	102.5	18.3
5	Henry	M	14	63.5	102.5	17.9
6	James	M	12	57.3	83.0	17.8
7	Jane	F	12	59.8	84.5	16.6
8	Janet	F	15	62.5	112.5	20.2
9	Jeffrey	M	13	62.5	84.0	15.1
10	John	M	12	59.0	99.5	20.1
11	Joyce	F	11	51.3	50.5	13.5
12	Judy	F	14	64.3	90.0	15.3
13	Louise	F	12	56.3	77.0	17.1
14	Mary	F	15	66.5	112.0	17.8
15	Philip	M	16	72.0	150.0	20.3
16	Robert	M	12	64.8	128.0	21.4
17	Ronald	M	15	67.0	133.0	20.8
18	Thomas	M	11	57.5	85.0	18.1
19	William	M	15	66.5	112.0	17.8

Only one variable, BMI, has a label and that label replaces the variable name BMI in the output. The values of BMI are rounded to one decimal place since a format of 4.1 was added to the variable BMI in the data step.

The final step in example 1.2 is PROC MEANS ⑧. In example 1.1, PROC MEANS was used without any options or additional statements. In this example, options and two new statements are added. In place of the default statistics (remember what they are?), specific statistics are requested. Most should be obvious except for Q1 and Q3 which request the 25th and 75th percentile values respectivley (quartile 1 and quartile 3).

The CLASS statement specifies that statistics are to be computed for each value of variable SEX ⑨. Two sets of statistics will be calculated, one for males (SEX=M) and the other for females (SEX=F). A VAR statement limits the computation of statistics to one variable, BMI, rather than all numeric variables in the data set ⑩. The output from PROC MEANS looks as follows.

NEW DATA SET BMI

Analysis Variable : bmi Body Mass Index

Sex	N Obs	Mean	Minimum	Lower Quartile	Median	Upper Quartile	Maximum
F	9	17.0510391	13.4900007	16.1567884	17.0776953	18.2708984	20.2464000
M	10	18.5942434	15.1173120	17.7715036	17.9718208	20.3414352	21.4296601

Notice that the label (Body Mass Index) appears in addition to the variable name (bmi). Even though BMI was assigned the format 4.1, that format is not used in the output from PROC MEANS. You will have to wait to lean how to get rid of all those decimal places.

After the first example, it was mentioned that most SAS statement start with a keyword and end with a semi-colon. Example 1.2 includes some new keywords: DATA, SET, CLASS, VAR. One exception to the rule of 'start with a keyword' is the assignment statement that creates the new variable BMI. That statement starts with the name of a variable, not a keyword. How many statements are there in example 1.2 ... how about 18 ... remember, just count the semi-colons, not the number of lines.

What you should remember...

Everything (just joking) ... SAS jobs are combinations of data steps and/or procedures (PROCs). The building blocks of data steps and PROCs are SAS statements. All SAS statements end with a semi-colon and most start with a keyword. You can (and should) add comments to your SAS jobs.

A LIBNAME statement creates a shortcut called a libref that can be used to tell SAS the location (folder) of one or more SAS data sets.

PROCs work with SAS data sets. PROC PRINT displays the values of variables. PROC CONTENTS displays the attributes of variables. PROC MEANS computes basic statistics for numeric variables. For now, you do not have to remember the options or extra statements that were used in the various procedures.

You can create or modify a data set with a data step. Within a data step, a SET statement is used to read observations from a data set. Data sets comprise observations and variables. All variables have the attributes: name, type, length, position. They might also have additional attributes; format, label.

What are the rules for the names of variables, data sets, and librefs? The rules cover both content and length of the names. What is the maximum length of a variable label?

And yes, anything you see in **BOLD** type is important.

...SIMPLE SAS JOBS - STARTING WITH A RAW DATA

The first two examples used data that was already in a SAS data set, MYDATA.CLASS. What if you have data in some other format, how can you create a SAS data set from that data? From now on, any data that is not in a SAS data set is referred to as raw data, so the question can be rephrased to how do you convert raw data into a SAS data set. Given that raw data can come in a wide variety of forms, that question has a variety of answers. We can start with some data in a common format, a text file (sometimes referred to as plain text or ASCII text or text data or raw data or data).

Assume that someone has given you a text file name HEALTH.TXT. You copy it onto your PC and put it into a folder named C:\EPI514\DATA. It contains data on 80 subjects. The first 10 records look as follows (on the left) and you are also given a description of the contents of the file (on the right).

```
423237875F1263.3156.3 64104 41
740761016F1657.0100.7 64106 64
112538488 1763.0156.3 96109 65
422202451M1771.0237.1 64125 76
486084198F  64.3114.8 76104 61
296857158M1862.9151.8 68 95 58
770488386M1865.6164.7 60118 68
795140256F1861.8123.9 88 92 46
838563140F1964.8105.4 76112 62
077047601F1963.1136.1 68    48
```

HEALTH.TXT

columns (start-end)	data
1-9	social security number
10	gender
11-12	age (years)
13-16	height (inches)
17-21	weight (pounds)
22-24	pulse
25-27	systolic blood pressure
28-30	diastolic blood pressure

...Example 1.3

```
* a data step that uses formatted input to create a SAS data set;
data patients; ①
infile 'k:\epi514\data\health.txt'; ②
input ③
@01 ssn       $9. ④
@10 gender    $1.
@11 age        2. ⑤
@13 height     4.
@17 weight     5.
@22 pulse      3.
@25 sbp        3.
@28 dbp        3.
;
bmi = 703 * weight / height**2; ⑥
label
ssn = 'SOCIAL SECURITY NUMBER'
sbp = 'SYSTOLIC BLOOD PRESSURE'
dbp = 'DIASTOLIC BLOOD PRESSURE'
bmi = 'BODY MASS INDEX' ;
format bmi 4.1
;
run;

* an option that causes all variable names to be displayed in uppercase;
options validvarname=upcase; ⑦

* BAD PRACTICE ... use a PROC without specifying a data set ... it works;
proc means maxdec=1 n mean std var clm; ⑧
run;

proc print;
where ranuni(0) lt 0.2; ⑨
run;
```

A data step is used to read the raw data and convert it to a SAS data set ①. You might have noticed that the name of the data set only has one part, PATIENTS, and there is no libref. There is no LIBNAME statement in the SAS job. If you do not use a libref when you create a data set, SAS provides a libref for you, WORK. Remember that a libref is a short cut that refers to a location of one or more SAS data sets. What folder is associated with the libref WORK? For now (and maybe forever), it is enough to know that the WORK library is a folder on your PC. One very important feature of using the WORK library is that all data sets in that library are erased when you exit SAS. For this reason, when you create a data set without specifying a libref other than WORK, you are creating a temporary SAS data set. Librefs are used for the data sets in example 1.1 and 1.2 (MYDATA.CLASS, MYDATA.BMI) and they are referred to as permanent SAS data sets since they are not erased when you exit SAS.

The INFILE statement tells SAS the location of your raw data ②. It is important to remember the distinction between a LIBNAME and INFILE statements. A LIBNAME statement refers to a folder that contains one or more data sets. An INFILE statement refers to a file, not a folder. The keyword INFILE is followed by the name of the file where the data are located. You need to specify the full file name, the folder plus the name of the data file.

Once you have told SAS the location of your data, an INPUT statement gives instructions for reading the values of the data ③. In examples 1.1 and 1.2, most of the variables already had names and types. Only one new variable was created, BMI. In this example, you are creating a new data set and all the decisions about variable names and types are up to you. There are many ways to write an INPUT statement. When data are nicely organized as in the raw data file HEALTH.TXT, one way to write the INPUT statement is: choose names for the variables in your raw data and write those names in a column; to the left of the each variable name, write an @ followed by the first column in the raw data file where data for that variable is located; to the right of the name of each character variable, write the number of columns occupied by the data for that variable, add a period after the number, add a $ prior to the number of columns ④; to the right of each numeric variable, do the same thing you would do for a character variable, but leave out the $ ⑤. The description you write is preceded by the keyword INPUT and ends with a semi-colon. This type of INPUT statement is referred to as formatted input. Notice that the INPUT statement is written on 10 lines, but it is still only one statement.

As in the last example, the variables weight and height are used to compute the body mass index ⑥. The new variable is included in the LABEL statement and a FORMAT statement is added to control the appearance of the variable BMI.

The OPTIONS statement is another global statement ⑦. When you use SAS, there are a lot of options that are preset. One of them, VALIDVARNAME, controls how variable names appear when they are used in the output of PROCs. The default is to show them exactly how they first appeared when a data set was created. In this example, all the variable names first appeared in lower case. If you would like all variable names to appear in upper case regardless of how they first appeared, the OPTIONS statement in this example should be used, VALIDVARNAME=UPCASE. To reset, use VALIDVARNAME=V7.

Two procedures are used without specifying a data set. The comment says this is bad practice, even though it works in this situation. If you use a PROC without specifying a data set name, the PROC uses the last data set created in the SAS job, in this case it is PATIENTS. Do not do this ... always specify a data set. An option is used with PROC MEANS, MAXDEC=1 ⑧. In the previous examples, the output from PROC MEANS displayed many decimal places. The number of decimal places displayed can be limited using the MAXDEC option. Just as was done in example 1.2, some statistics keywords are used to produce the output shown at the top of the next page. Notice the column labeled N. PROC MEANS uses all the non-missing data within each numeric variable to compute statistics. Though there are 80 observations in the data set, some numeric variables have missing values.

Variable	Label	N	Mean	Std Dev	Variance	Lower 95% CL for Mean	Upper 95% CL for Mean
AGE		78	34.6	13.2	174.1	31.6	37.6
HEIGHT		79	65.8	3.9	15.1	64.9	66.6
WEIGHT		80	159.4	34.9	1216.4	151.6	167.1
PULSE		79	72.9	12.4	154.1	70.1	75.6
SBP	SYSTOLIC BLOOD PRESSURE	79	114.9	14.7	217.2	111.6	118.3
DBP	DIASTOLIC BLOOD PRESSURE	80	70.3	10.8	116.4	67.9	72.7
BMI	BODY MASS INDEX	79	25.8	5.0	24.7	24.7	27.0

An extra statement is used in PROC PRINT ⑨. Normally, PROC PRINT display all the observations is an data set. The WHERE statement shown in this example is an easy way to display a random sample of observations. The number of observations to be displayed is controlled by the number at the end of the statement, in this case the 0.2 requests that approximately 20% of the observations be printed. The output is shown below. For now, it is not necessary to understand why the WHERE statement produces a random sample of observations.

Obs	SSN	GENDER	AGE	HEIGHT	WEIGHT	PULSE	SBP	DBP	BMI
3	112538488		17	63.0	156.3	96	109	65	27.7
4	422202451	M	17	71.0	237.1	64	125	76	33.1
5	486084198	F	.	64.3	114.8	76	104	61	19.5
9	838563140	F	19	64.8	105.4	76	112	62	17.6
12	424244337	M	20	69.7	137.4	88	112	44	19.9
18	755760959	F	23	64.7	108.8	72	98	61	18.3
28	908880187	F	27	65.1	119.0	68	100	53	19.7
33	863622766	M	29	70.0	162.4	56	116	64	23.3
50	324323549	M	37	66.1	169.8	84	112	79	27.3
65	650051148	F	47	58.2	195.6	88	114	79	40.6
76	887337740	F	56	63.4	181.2	80	116	71	31.7
78	311294413	M	58	70.8	169.1	68	125	78	23.7

Look at the column labeled Obs. The number in that column is the position of the observation in the data set, equivalent to what row you would be viewing if the data were in a spreadsheet instead of a data set. Also notice that missing character data (Obs=3, GENDER) is displayed as spaces, while missing numeric data (Obs=5, AGE) is displayed as a period.

Another common format for data is delimited. This means that the values of the variables in each record are separated by a delimiter, some common delimiters being: space, comma, slash, tab. Below are the data used in example 1.3. The box on the left shows the first 10 records in the file HEALTH.TXT. The box on the right shows the first 10 records in a new data file, HEALTH.CSV, in which the values of the variables in each record are separated by commas. The two files contain the same data, just stored in different formats. In addition to the commas in the file on the right, notice one other major difference between the files. On the left, the values of all the variables occur in the same columns across all the records in the file (the column locations of variables is shown on page 9). On the right, look at the last entry on each line. The column location of that number varies across records. Also look at how missing data are represented in the two files. In the TXT file on the left, there are spaces in records 3, 5, and 10 (no gender in 3, no age in 5, no systolic blood pressure in 10). In the CSV file on the right, missing data are represented by two consecutive commas. By the way, what does "CSV" represent?

```
423237875F1263.3156.3 64104 41
740761016F1657.0100.7 64106 64
112538488 1763.0156.3 96109 65
422202451M1771.0237.1 64125 76
486084198F  64.3114.8 76104 61
296857158M1862.9151.8 68 95 58
770488386M1865.6164.7 60118 68
795140256F1861.8123.9 88 92 46
838563140F1964.8105.4 76112 62
077047601F1963.1136.1 68    48
```

HEALTH.TXT

```
423237875,F,12,63.3,156.3,64,104,41
740761016,F,16,57,100.7,64,106,64
112538488,,17,63,156.3,96,109,65
422202451,M,17,71,237.1,64,125,76
486084198,F,,64.3,114.8,76,104,61
296857158,M,18,62.9,151.8,68,95,58
770488386,M,18,65.6,164.7,60,118,68
795140256,F,18,61.8,123.9,88,92,46
838563140,F,19,64.8,105.4,76,112,62
077047601,F,19,63.1,136.1,68,,48
```

HEALTH.CSV

...Example 1.4

```
* a data step that uses list input to create a SAS data set;
data patients;
infile 'k:\epi514\data\health.csv' dsd; ①
input
ssn     :  $9.  ②
gender  :  $1.
age
height  ③
weight
pulse
sbp
dbp
;

bmi = 703 * weight / height**2;

label
ssn = 'SOCIAL SECURITY NUMBER'
sbp = 'SYSTOLIC BLOOD PRESSURE'
dbp = 'DIASTOLIC BLOOD PRESSURE'
bmi = 'BODY MASS INDEX'
;
format bmi 4.1;
run;

options validvarname=upcase;

title;
proc ttest data=patients; ④
class gender;
var bmi;
run;

ods listing close; ⑤
ods html style=barrettsblue;

title 'OBSERVATIONS WITH MISSING DATA';
proc print data=patients;
where ssn is missing or gender is missing or nmiss(age, pulse, sbp, dbp, bmi) ne 0; ⑥
run;
title;

ods html close; ⑦
ods listing;
```

The INFILE statement tells SAS the location of your raw data ①. The DSD option is added to the INFILE statement to let SAS know three things: the data are delimited; the delimiter is a comma; two consecutive delimiters indicate missing data. You can use the DSD option together with the DLM option if a delimiter other than a comma is used. More about that later.

The INPUT statement is similar to that used in example 1.3. However, notice that there are no @'s to tell SAS the column locations of the data for the variables. Since the values occur in different columns, you cannot specify column locations. In this example, you are again creating a new data set from scratch and all the decisions about variable names and types are up to you. When data are delimited as in the raw data file HEALTH.CSV, one way to write the INPUT statement is: choose names for the variables in your raw data and write those names in a column; to the right of each character variable, write a colon followed by the maximum number of spaces needed to accommodate the value of the variable, add a period after the number, add a $ prior to the number of columns ②. Notice that nothing is written to the right of the numeric variables ③. The description you write is preceded by the keyword INPUT and ends with a semi-

colon. This type of INPUT statement is referred to as list input. Notice that as with example 1.3, the INPUT statement is written on 10 lines, but it is still only one statement. The remaining statements in the data step are the same as those used in example 1.3. The only differences thus far are the extra option on the INFILE statement and the different type of INPUT statement.

PROC TTEST is used to compare the BMI between males and females ④. The procedure is preceded by a TITLE statement with no title specified. Do you remember what that does? The SAS code used with this procedure looks similar to that used with PROC MEANS in example 1.2. However, rather than just computing descriptive statistics, PROC TTEST performs a hypothesis test, comparing the mean BMI among males to that of females. The output looks as follows.

Statistics

Variable	GENDER	N	Lower CL Mean	Mean	Upper CL Mean	Lower CL Std Dev	Std Dev	Upper CL Std Dev	Std Err	Minimum
BMI	F	39	23.718	25.735	27.751	5.0832	6.22	8.0162	0.996	17.646
BMI	M	39	24.789	25.907	27.026	2.82	3.4506	4.4471	0.5525	19.502
BMI	Diff (1-2)		-2.441	-0.172	2.0961	4.3416	5.0297	5.979	1.139	

T-Tests

Variable	Method	Variances	DF	t Value	Pr > |t|
BMI	Pooled	Equal	76	-0.15	0.8801
BMI	Satterthwaite	Unequal	59.4	-0.15	0.8802

Equality of Variances

Variable	Method	Num DF	Den DF	F Value	Pr > F
BMI	Folded F	38	38	3.25	0.0004

You may or may not recognize a lot of the terms in the above output. The upper section contains many of the same descriptive statistics you have already seen in PROC MEANS output. In addition to those, the results of a t-test are presented. For those of you with some statistics background: is there a difference in mean BMI between males and females; did you use the pooled or Satterthwaite method, why; what else would you do with these data in addition to presenting the t-test results.

The last step in the SAS job is PROC PRINT. Just to show you that SAS can produce output that is a bit (actually a lot) nicer looking than what you have seen thus far, the output delivery system (ODS) is used. ODS is relatively new to SAS. Since these notes cover only a small portion of SAS (remember the introduction, you are seeing 5%), they will include only a limited amount of the things you can do with the output delivery system. You should remember that ODS is now part of SAS and it gives you the capability of producing professional looking output (both text and graphics, including statistical graphics) and that it also allows you to select among portions of the output from SAS PROCs in place of producing all of the default output.

The first step in using ODS to create output is to select a destination. The destination specifies a type of output and among those you can choose are HTML (web page), RTF (document), and PDF (for use with Adobe Acrobat). One ODS destination that is always chosen by default is the output window in the display manager, referred to as the listing destination. So, we close the listing destination (no output will appear in the output window) and we open the HTML destination and specify one option ⑤. The STYLE option allows you to use either SAS-supplied 'looks' for the output (colors, fonts) or a style that you have defined. In this example, a SAS-supplied style is used.

PROC PRINT is used with a WHERE statement ⑥. In example 1.3, a WHERE statement was used to print a random sample of observations. In this example, the WHERE statement requests that we only print observations that have a missing value for one or more of the variables. The details are not important. Just remember that this type of selection among observations is possible.

The last step is to close the HTML destination and reopen the listing destination ⑦. Until you close a destination you have opened in ODS, all procedure output will be written to the destination and you cannot use that output until the close. The output looks as follows. Only the observations with a missing value for at least one variable are printed. The 'look' is much different from that produced in the listing destination.

OBSERVATIONS WITH MISSING DATA

Obs	SSN	GENDER	AGE	HEIGHT	WEIGHT	PULSE	SBP	DBP	BMI
3	112538488		17	63.0	156.3	96	109	65	27.7
5	486084198	F	.	64.3	114.8	76	104	61	19.5
10	077047601	F	19	63.1	136.1	68	.	48	24.0
44	103141200	M	.	68.3	204.6	60	110	66	30.8
52	432053180	F	37	.	141.4	72	124	85	.
80	863181626	M	73	68.3	186.6	.	153	87	28.1

What you should remember...
A data step is used to convert raw data into a SAS data set. Within a data step, an INFILE statement tells SAS the location of your raw data. An INPUT statement describes the organization of the your data with the raw data file. SAS can convert just about any type of raw data into a data set. The two types of data used in this section are 'nicely organized' and comma-delimited. Each type of data requires different types of INFILE and INPUT statements.

In addition to permanent data sets, there are also temporary data sets. The temporary data sets are stored in a library named WORK and they are deleted upon completion of a SAS session. Data sets in the WORK library can be used in a SAS job with only the second portion of the full data set name. For example, a temporary data set named PATIENTS need not be referred to as WORK.PATIENTS. If only PATIENTS is used, SAS will look in the WORK library.

There are options and statements within SAS procedures that allow you to alter their default behavior. ODS (the output delivery system) can be used to produce nice looking procedure output.

It is not important to remember all the little details shown in the SAS jobs thus far. Up to now, you are just expected to be developing a general knowledge of what SAS jobs look like and some idea as to what you can do with SAS. Remembering the details starts in the next chapter, but anything you remember from this chapter will help.

SAS LOG...

Thus far, all the emphasis has been on the looking at SAS jobs and the output produced by SAS procedures. We have yet to look at another very important part of using SAS, the log file. Each time you run a SAS job, a log file is produced. One thing you will quickly discover if you are new to SAS is that when you write a SAS job, you are very likely to make mistakes. You will leave out a semi-colon at the end of a SAS statement. You will use a SAS statement in an incorrect location in the SAS job. Unfortunately the number of different types of errors you can make is quite large. The SAS log will attempt to show you where your error(s) occurred and give you information that will help you to fix your mistakes.

We will use the data step in example 1.4 to show what you will see in the SAS log when you have mistakes in your SAS code. First, we will use the data step with no errors in the SAS code. The SAS log looks as follows.

```
1     * a data step that uses list input to create a SAS data set;
2     data patients;
3     infile 'k:\epi514\data\health.csv' dsd ;
4     input
5     ssn      :  $9.
6     gender   :  $1.
7     age
8     height
9     weight
10    pulse
11    sbp
12    dbp
13    ;
14
15    bmi = 703 * weight / height**2;
16
17    label
18    ssn = 'SOCIAL SECURITY NUMBER'
19    sbp = 'SYSTOLIC BLOOD PRESSURE'
20    dbp = 'DIASTOLIC BLOOD PRESSURE'
21    bmi = 'BODY MASS INDEX'
22    ;
23    format bmi 4.1;
24    run;

NOTE: The infile 'k:\epi514\data\health.csv' is:
      File Name=k:\epi514\data\health.csv,
      RECFM=V,LRECL=256

NOTE: 80 records were read from the infile 'k:\epi514\data\health.csv'.
      The minimum record length was 31.
      The maximum record length was 36.
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 15:20   1 at 15:28
NOTE: The data set WORK.PATIENTS has 80 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

The first text you see is your SAS code with line numbers added on the left. Then there are a series of NOTES. Usually, any blue text you see in the log is good, not an indication that there were errors that prevented your SAS code from working. That is not always true, but "blue is good" is a good rule to remember. Most of the notes shown above fit the rule. You are told what raw data file was used, how many records were read from that file, and the number of observations and variables that are in your newly created data set. The third note has some extra information that could be considered as violating the rule. You are told that line 15 of the SAS code (the numbers on the left) produced a variable with a missing value one time. On line 15, you calculate the BMI. You know from looking at the output produced by example 1.4 (top of page 14) that one observation has a missing value for HEIGHT. Since you have no value for HEIGHT, SAS cannot calculate a BMI using the equation in line 15 so a missing value is created (again, look at the output on the top of page 14). Here is another rule that you will have to remember at some point, so why not start now. Any equation that you write on your own in a data step that has missing data on the right side of the ='s will result in a missing value being assigned to the value on the left of the ='s. That is not really an error in the log, just some information about your data. The SAS code worked and created a data set.

We will start with a mistake that is easy to correct ... provide the incorrect location of the raw data in the INFILE statement ...

```
infile 'k:\health.csv' dsd;
```

The log now shows ...

```
ERROR: Physical file does not exist, k:\health.csv.
NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PATIENTS may be incomplete.  When this step was stopped there were 0 observations and 9 variables.
WARNING: Data set WORK.PATIENTS was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

SAS cannot find a file name HEALTH.CSV in the folder K:\. The says that the file does not exist. This error is easy to see and easy to correct.

Now, we will provide the correct location of the file, but leave off the DSD option that informs SAS that the file contains comma-delimited data ...

```
infile 'k:\epi514\data\health.csv';
```

The first portion and last portion of the log are shown below.

```
NOTE: The infile 'k:\epi514\data\health.csv' is:
      File Name=k:\epi514\data\health.csv,
      RECFM=V,LRECL=256

NOTE: Invalid data for age in line 3 1-32.
NOTE: Invalid data for height in line 4 1-33.
NOTE: Invalid data for weight in line 5 1-33.
NOTE: Invalid data for pulse in line 6 1-34.
NOTE: Invalid data for sbp in line 7 1-35.
NOTE: Invalid data for dbp in line 8 1-34.
RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0
8         795140256,F,18,61.8,123.9,88,92,46 34
NOTE: Invalid data errors for file ''k:\epi514\data\health.csv'' occurred outside the printed range.
NOTE: Increase available buffer lines with the INFILE n= option.
ssn=423237875 gender=7 age=. height=. weight=. pulse=. sbp=. dbp=. bmi=. _ERROR_=1 _N_=1
NOTE: Invalid data for age in line 11 1-35.
NOTE: Invalid data for height in line 12 1-35.
NOTE: Invalid data for weight in line 13 1-35.
NOTE: Invalid data for pulse in line 14 1-33.
NOTE: Invalid data for sbp in line 15 1-35.
NOTE: Invalid data for dbp in line 16 1-34.
16        715834528,F,22,60.7,124.3,64,97,64 34
NOTE: Invalid data errors for file ''k:\epi514\data\health.csv'' occurred outside the printed range.
NOTE: Increase available buffer lines with the INFILE n= option.
```

```
79        769395646,F,59,63.5,163.1,76,105,69 35
NOTE: Invalid data errors for file ''k:\epi514\data\health.csv'' occurred outside the printed range.
NOTE: Increase available buffer lines with the INFILE n= option.
ssn=590088609 gender=6 age=. height=. weight=. pulse=. sbp=. dbp=. bmi=. _ERROR_=1 _N_=10
NOTE: LOST CARD.
ssn=863181626 gender=  age=. height=. weight=. pulse=. sbp=. dbp=. bmi=. _ERROR_=1 _N_=11
NOTE: 80 records were read from the infile 'k:\epi514\data\health.csv'.
      The minimum record length was 31.
      The maximum record length was 36.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      10 at 107:11
NOTE: The data set WORK.PATIENTS has 10 observations and 9 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

In between those two sections of the log, there a many other lines that keep repeating the same text each time SAS tries to read data from the file. In the lower portion there is an additional message about a LOST CARD. But you should also notice another message ... a data set named PATIENTS was created and it has 10 observations and 9 variables. All the messages are in that friendly blue color, but something is wrong since the log also says that we read 80 records to produce those 10 observations and it is not clear from the log what should be done to correct the mistake.

Now we fix the INFILE statment (put back the DSD option) and add a different mistake leaving out the multiplication sign in the equation that calculates BMI ...

```
bmi = 703 weight / height**2;
```

The log directs you to the line of SAS code that contains the error and underlines the variable name

```
131  bmi = 703 weight / height**2;
                ------
                22
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, *, **, +, -, /, <, <=, <>, =, >, ><, >=, AND, EQ, GE, GT, LE, LT, MAX, MIN, NE, NG
              OR, ^=, |, ||, ~=.

132
133  label
134  ssn = 'SOCIAL SECURITY NUMBER'
135  sbp = 'SYSTOLIC BLOOD PRESSURE'
136  dbp = 'DIASTOLIC BLOOD PRESSURE'
137  bmi = 'BODY MASS INDEX'
138  ;
139  format bmi 4.1;
140  run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PATIENTS may be incomplete.  When this step was stopped there were 0 observations and 9 variables.
WARNING: Data set WORK.PATIENTS was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

WEIGHT. You then see and error statement that informs you of a syntax error. A syntax error is caused by a statement that SAS just cannot understand. Think of it as trying to speak to someone in a foreign language and doing things such as leaving out words, using extra words, or using the wrong words. Any one of those mistakes might cause an error in how the other person understands what you are trying to say. Notice you are not told that you should provide a multiplication sign. What you are told is that in place of the word 'weight' SAS expects to find one of those many symbols or text you see in the log (there is a multiplication sign in the list). The actual message extends even further to the right with more potential text to correct your error.

Next we will make one of the most common errors. Let us remove the semi-colon from the end of the INPUT statement ...

```
input
ssn     :  $9.
gender  :  $1.
age
height
weight
pulse
sbp
dbp

bmi = 703 * weight / height**2;
```

```
217  input
218  ssn     :  $9.
219  gender  :  $1.
220  age
221  height
222  weight
223  pulse
224  sbp
225  dbp
226
227  bmi = 703 * weight / height**2;
                                  --
                                  22
                                   -
                                   200
ERROR 22-322: Syntax error, expecting one of the following: a name, an integer constant, arrayname, *, $, &, (, +, -, /, //, :, ;, =, ?, @, @@, [, {, ~.

ERROR 200-322: The symbol is not recognized and will be ignored.

228
229  label
230  ssn = 'SOCIAL SECURITY NUMBER'
231  sbp = 'SYSTOLIC BLOOD PRESSURE'
232  dbp = 'DIASTOLIC BLOOD PRESSURE'
233  bmi = 'BODY MASS INDEX'
234  ;
235  format bmi 4.1;
236  run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PATIENTS may be incomplete.  When this step was stopped there were 0 observations and 9 variables.
WARNING: Data set WORK.PATIENTS was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

An error occurs, but the log does not say something as simple as that you left out a semi-colon. It does not even point to the correct location of the error. Instead, it looks as if your error is in the equation for calculating BMI. The moral to this example ... it is not always easy to locate your mistakes.

The final error we will introduce is once again leaving out a semi-colon, but this time we remove it from the end of the equation used to calculate BMI ...

```
bmi = 703 * weight / height**2
```

```
249
250  bmi = 703 * weight / height**2
251
252  label
     -----
     79
ERROR 79-322: Expecting a ;.

253  ssn = 'SOCIAL SECURITY NUMBER'
254  sbp = 'SYSTOLIC BLOOD PRESSURE'
255  dbp = 'DIASTOLIC BLOOD PRESSURE'
256  bmi = 'BODY MASS INDEX'
257  ;
258  format bmi 4.1;
259  run;

NOTE: The SAS System stopped processing this step because of errors.
WARNING: The data set WORK.PATIENTS may be incomplete.  When this step was stopped there were 0 observations and 9 variables.
WARNING: Data set WORK.PATIENTS was not replaced because this step was stopped.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

This is somewhat reassuring in that you left out a semi-colon and the error message says that SAS is expecting a semi-colon. The log does not say that you left the semi-colon off the end of the line with the equation. It does underline the word 'label' and that word is the first portion of SAS code after the missing semi-colon ... maybe you could figure out where that missing semi-colon belongs.

What you should remember...
ALWAYS look at the log file after you have run a SAS job.

Some mistakes cause a SAS job to fail. These are mistakes that you will most likely notice. Other mistakes do not cause a SAS job to fail and a data set is created. That data set may or may not contain what you think it contains and what you want it to contain. This type of mistake you might not notice. Therefore,

ALWAYS look at the log file after you have run a SAS job.