

---

**BIOM601 (Biostatistics I)**  
**Laboratory Session #1 (Fall 2008)**  
**SAS PROGRAMMING AND FILE MANAGEMENT**

**Objective:** Learn basic steps of SAS programming. The DATA step will be explored in detail. Handling data with PROC SORT and PROC PRINT will be covered. Principles and praxis of debugging SAS programs will be covered.

### 1. Overview of laboratory activities

BIOM601 is designed to provide you with knowledge to summarize and analyze data from scientific research and to understand and evaluate results of statistical analyses in the literature. The laboratory portion of this course will introduce the SAS system. You will learn its environment and the essentials of its programming language. At the end of the semester you will be able to write your own SAS programs to carry out most of the statistical analyses discussed in the lectures. Laboratory notes and supporting files are found at the BIOM601 website: <http://ansc.umd.edu/siewerdt/BIOM601Master.html>.

### 2. File management

In order to standardize and organize your work, it is suggested that you follow file naming rules. Avoid using spaces in file names. Use the same name in all associated files, changing only the extension. The following file extensions should be used to identify the type and source of your files:

- .DAT** files that contain data for analysis.
- .SAS** SAS program files that were provided or that you have written.
- .LOG** files created by SAS that tell how the program ran and associated notes and error messages.
- .LST** files created by SAS that contain the results from the requested analysis.

It is not generally necessary to keep the last two types of files since you can always run the SAS program at a later time.

### 3. General Structure of a SAS Program

SAS programs are organized in two “sections”. These are like paragraphs that start with either the word DATA or the word PROC. A paragraph is made up of SAS statements that can be thought of as sentences. Statements may start with a SAS key word such as VAR for variable list or INPUT to enter the data format. A **DATA** step is how you tell SAS where the data is stored, inform the format of the data set, create new variables, delete variables, or merge data files, among other functions. The **PROC** step tells SAS what analysis you want applied to the data. For example, PROC PRINT will print a data set, PROC MEANS provides descriptive statistics, and PROC TTEST performs a hypothesis test using Student’s *t* test. SAS can hold in its memory several data sets simultaneously. When multiple data sets are read into SAS at the same time, it is interesting to specify which set is to be used for a given PROC. This is done by adding the statement DATA=NAME, where NAME is the name of the data set to be used. If you omit this information, SAS uses as the

default the last data set that was created. In programming terms, DATA steps are line processors and perform operations on each line of data in the whole data set, while PROC steps are generally variable processors and will perform operations on variables rather than data lines.

It is important to remember that **every SAS statement must end with a semicolon**. With the exception of a particular data input procedure, a program line in SAS does not end until a semicolon is found. If you forget this detail, you will be frustrated by error messages. On some occasions SAS provides error messages that are generated when you run the program. These usually point out to syntax mistakes and allow for easy correction.

A short introduction to the process of writing SAS programs is provided below. Specific details on each part of the SAS program will be covered as progress is made in laboratory sessions. A useful resource is the SAS manual, available online. Offline help offers answers to specific questions. You are encouraged to use both resources.

- **Program Title** Because we are sharing printers, it is important that the first line of every program is a title that includes your name, so that you will be able to identify your printed output. The first line in every SAS program should be: TITLE '*Your name*'. The text inside the single quotes is treated by SAS as a label and is not processed as a command.

**TITLE 'SIEWERDT';**

- **Clearing Previous Output** SAS does not automatically clear your log and output windows from previous runs when you submit a program. You can clear them with the mouse through a number of point and clicks. However, you will often forget to do so and, after running several programs you may find it difficult to tell which portion of the log and output came from a particular run. Therefore the second line of every program that you write this semester should be:

**DM 'LOG; CLEAR; OUT; CLEAR;';**

- **Options** Following these two program lines it is helpful to specify options, so that printouts conform to printer fonts being used or to the screen size on your monitor. This is where you specify how wide and long to make each page of output. A line size (**ls**) of 75 or 90 characters and a page size (**ps**) of 60 lines are good choices. The statement **pageno=1**, will start page numbering with 1 each time the program is submitted, when this option is included.

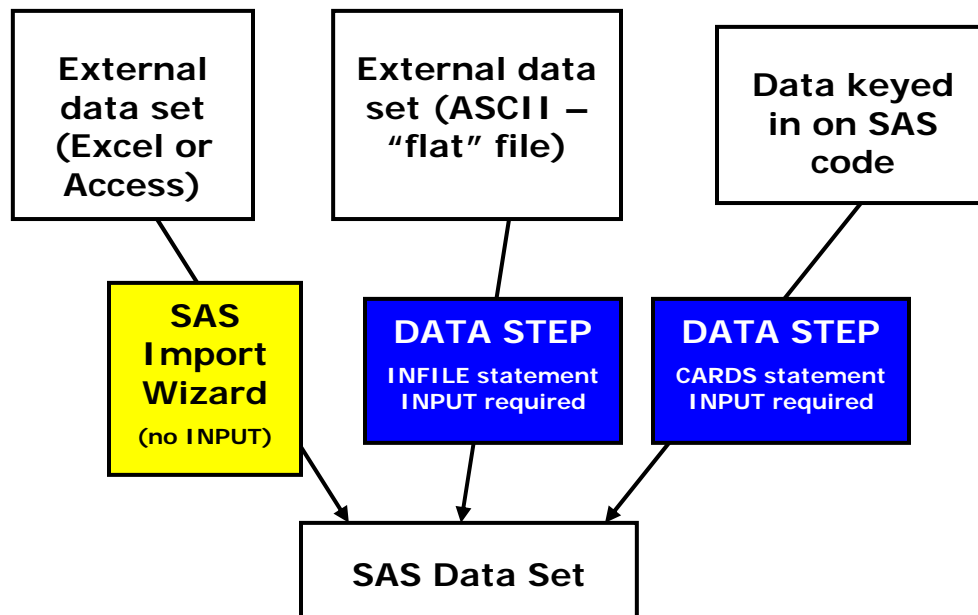
**OPTIONS ls=75 ps=60 pageno=1;**

**4. Data Step** There are many forms to read data into a SAS file.

- a) If you have a data set stored already in a "flat" (ASCII) file, then the keyword **INFILE** tells SAS where the external data is stored (e.g. on a flash disk, on the server, on the hard drive). Then, the **INPUT** statement describes the order of the variables in the data set by giving each variable a name and telling SAS whether the variable is numeric or character. Many different data formats can be read using features of the INPUT statement. In this course, to keep it simple, variables will be separated with one or more

blank spaces and missing values will be represented by a period (.).

- b) You may place the data as additional lines inside the SAS program. This is done using the CARDS statement (or DATAFILES, a single word), instead of INFILE. An INPUT statement is still required by SAS to identify the variables. For small data sets and where no other program is likely to need access to the same data, this is a satisfactory approach.
- c) You may import data from some spreadsheets, such as Microsoft Excel, or general database programs, such as Microsoft Access, by using the SAS import procedures (PROC IMPORT or select "Import Data" from the "File" drop-down menu, then follow the prompts) to bring data into a SAS data step. Either method replaces the entire initial DATA step using INFILE or INPUT (See Figure 1).



**Figure 1.** Alternatives for input of data into SAS

Note that there is no need for the DATA step when the Import Wizard is used. If you decide to use the Import Wizard for reading the data, the set will already be in SAS, ready to be analyzed.

The data step can be also used to create new variables. For example, you may have two variables in the data set: weight of 10 corn grains (variable GRAIN) and oil content of the 10 corn grains (variable OIL). You can create a new variable to express the oil content in percentage (call this new variable OILPC) by using the instruction:

**OILPC=100\*OIL / GRAIN;**

When you need to apply a transformation to your original, this is where it is done. Several functions can be used, like taking square roots, taking logarithms, or employing a trigonometric function. Become thoroughly familiar with the DATA step because no analysis

can be done with SAS unless you first load your data set into SAS.

**5. Procedures (PROCs)** Once a data set is read into SAS, there are many things that you can do on the variables in that data set. For example you can sort the data, print the data, compute means and standard errors of variables, run t-tests and regressions. There are often many procedures used in a single SAS program. Many PROCs also have output capabilities that can be used to create new data sets for use by other PROCs.

- **QUIT Statement** It is suggested that the last line in every SAS program be the statement shown below. Using the QUIT statement will clear SAS of any pending commands and leave it ready for new analyses without any residual commands from the previous programs.

**QUIT;**

**6. Annotating SAS programs** It is helpful to you and to the teaching assistants if you put comments and titles in your program to remind yourself what you are doing and why. Comment lines can be put anywhere in a program. Comment lines start with an asterisk (\*) and end with a semi-colon. For longer comments, spanning many lines, use the format: /\* *your comments* . . . \*/. SAS will not process anything after a "/\*", until it finds the corresponding "\*\*/" that ends the comment.

**7. Saving and retrieving files** Once you have created a data set or program with SAS you will want to save that file for future use. Use the tool button bar or the pull-down menu to save your files. To retrieve a previously saved file into the program editor you should also use the tool button bar or the pull down menu. Note that you can only retrieve files into an editor window (not the LOG or OUTPUT windows). The files are saved in ASCII format, which is convenient if you wish to work on your SAS code in a word processor.

**8. An Exercise** Double-click on the SAS icon to begin. You will notice that there are several ways in SAS to do the same thing. You only need to know one way to get the task done. Take notes and keep them handy until you become comfortable working in the SAS environment.

- a) Create a data set. Go to the SAS editor (or PGM) window and type in the following lines of data:

```
1975 Estimate 2620 0
1979 Census 1637 1
1981 Census 1490 2
1984 Census 1103 2
1987 Estimate 1125 2
1995 Estimate 1250 6
2004 Census 1590 5
```

In each line the values are, respectively: year; methodology (census or estimate); size of the giant panda (*Ailuropoda melanoleuca*) population in the wild; number of births in captive panda populations.

- b) Select **FILE - Save as** on the drop-down menu to save the data on your flash disk with

the name PANDAS.DAT

- c) Clear the editor window. (Type in the word “clear” on the command line or choose the Edit drop-down menu and select “Clear All”.)
- d) Open the file PANDAS.SAS and save it on your flash disk. Open the file by clicking on it or use the pull-down menu features (“File” then “Open Program”). Look through this program. The program was written to make a very simple analysis of the data set that you just saved (make sure that the INFILE statement reflects the correct name and location of the data file that you just created). Use the editor to change the name in the title where ‘*Your name*’ appears. Remember, the first line of all your programs this semester should be the TITLE statement with your name, so that you can identify your print-outs as they come off the printer. The program was written with the expectation that the data file PANDAS.DAT is saved on the A: drive. Change the code if you saved your copy of the file elsewhere.
- e) **SUBMIT** the program. You may use the F8 key, the pull-down menu, or click on the icon in the tool bar (the little “running man”). The program will run and should produce both a log and an output. Go to each of these windows and look through the results. Always start by the LOG screen because it will alert you if any errors occurred.
- f) You may want to save your corrected program to a flash disk. Using the same file name is fine, since SAS will simply replace the older version of the program.

**9. A second exercise** Clear your previous work. Download the Excel file *frogs.xls* (found on the lecturer’s website (<http://ansc.umd.edu/siewerdt/BIOM601Master.html>)). Open the file and examine its structure. Pay especial attention on how the data was stored. Use the simplified guidelines on page 3, item (c) to transfer this data set from an Excel format into a SAS format. The first line in the Excel file identifies the variables that are found in each column: the first one has the species of frogs and the second one contains the measurements of length taken on femur bones from seven male frogs of each species. Use SAS (not Excel) to print the data set, to ensure that you read it into SAS correctly. Annotate your program to remind you of what you did. The coded names used to identify the species were: ESCULENT for *Rana esculenta* (the edible frog), RIDIBUND for *Rana ridibunda* (European marsh frog), WHITES *Litoria caerulea* (White’s tree frog), and COLORADO *Bufo alvarius* (Colorado river toad).

**10. SAS Tutorial** SAS has excellent tutorial capabilities built in the software. You should not restrict yourself to study what will be presented in the laboratory sessions. There is only a limited portion of material that will be covered, mostly directed at uses of SAS for some statistical analyses. Since there is not enough available time to explore many of the resources available in SAS, you should try to learn as much as possible by means of independent study. You are encouraged to explore the *SAS tutorial for beginning users* to familiarize yourself with additional features of SAS. To access the SAS tutorial, follow this sequence:

- Help (drop-down menu)
- Getting Started with SAS Software
- New SAS Programmer (quick-start guide)

## 11. The DATA step

Remember to start every SAS program with a title statement, and some “housekeeping” lines. Before you can perform data analysis, you need to create one or more data sets in SAS. This is accomplished by the DATA step. In the previous laboratory session, you were briefly shown three different ways of importing data into SAS.

Every DATA step starts with a line where the name of the data set is identified. Try to keep data set names to 8 characters and always start with a letter. Numbers may be used in the name, but SAS will not accept symbols like \*, \$, &, or #. Most typographic symbols have special meaning in SAS programming language and are reserved for other uses. The line

### **DATA REACTION;**

starts the DATA step by instructing SAS to create a set named “REACTION”. Also, all DATA steps will end with the instruction

### **RUN;**

indicating that the data should be read into SAS and that all other instructions given in the DATA step should be processed at this time. There are several ways to specify where the data is physically located, what variables are to be read, and other instructions to be followed. The series of examples below will present many of these options. Unless you are importing data directly from a spreadsheet or a database file, the statement

### **INPUT <variables>;**

will always be present in your DATA step. INPUT specifies the names of the variables being read. Variable names may include numbers and underscore signs, but no symbols are allowed. When a variable is not numeric, the use of the \$ sign after the name of the variable is mandatory. The variable is then identified by SAS as an alphanumeric (or character) variable. SAS will not perform any mathematical calculations (e.g., obtain means and variances) on alphanumeric variables.

## 12. Example 1: *Reading data with the CARDS statement (file: swim1.sas)*

```
TITLE 'YOUR NAME';
TITLE2 'IMPROVEMENT OF STARTING REACTION TIME IN SWIMMERS';
DM 'LOG; CLEAR; OUT; CLEAR;';
OPTIONS LS=75 PS=60 PAGENO=1;
DATA REACTION;
    /* DESCRIPTION OF VARIABLES (UNIT: MILLISECONDS)
    INITIAL: REACTION TIME (RT) BEFORE TRAINING PROGRAM
    FINAL: RT AFTER TRAINING PROGRAM
    TOTALIMP: TOTAL IMPROVEMENT IN RT
    WEEKIMP: AVERAGE WEEKLY IMPROVEMENT IN RT */
INPUT GENDER $ WEEKS INITIAL FINAL;
TOTALIMP=INITIAL-FINAL;
WEEKIMP=TOTALIMP/WEEKS;
CARDS;
```

```
F 6 361 364
F 6 318 316
F 12 347 325
F 12 338 317
F 18 302 266
F 18 356 317
M 6 313 312
M 6 328 330
M 12 299 280
M 12 366 342
M 18 320 283
M 18 339 297

RUN;
PROC PRINT;
RUN;
PROC PRINT NOOBS;
RUN;
QUIT;
```

The statement CARDS in the SAS program allows you to include the data directly into the program. If you don't have an extensive data set, this is probably the best strategy to read data, since all information is kept in the same place. Take a careful look at the four-line comment was placed at the beginning of the DATA step, to describe the variables used. SAS treated everything that was between the starting */\** and the ending *\*/* as a comment, and has not processed any of that information. After the variables were read with the INPUT line, two new variables were created (TOTALIMP AND WEEKIMP) based on previous variables in the data set. Also notice that the program asks for the data set to be printed twice. Look at the first printout of the data and check to see if the original structure of the data set has been respected. Then compare the second printout with the first one and spot the difference.

### 13. Example 2: *Reading data with the INFILE statement (files: swim2.sas, swim2.dat)*

The main difference from the previous example will be that the data is now physically placed in another data set, which is saved elsewhere (the program assumes that it is in a floppy disk; change if appropriate). This may be convenient when the data set is large and placing all data lines in your SAS program would make it too long. The presence of the INFILE statement and the absence of the CARDS statement are essentially the only differences in the program. The statement CARDS is no longer necessary because SAS has already been told (with the INFILE statement) that the data is stored elsewhere.

```
TITLE 'YOUR NAME';
TITLE2 'IMPROVEMENT OF STARTING REACTION TIME IN SWIMMERS';
DM 'LOG; CLEAR; OUT; CLEAR;';
OPTIONS LS=75 PS=60 PAGENO=1;

DATA REACTION;
```

```
/* DESCRIPTION OF VARIABLES (UNIT: MILLISECONDS)  
INITIAL: REACTION TIME (RT) BEFORE TRAINING PROGRAM  
FINAL: RT AFTER TRAINING PROGRAM  
TOTALIMP: TOTAL IMPROVEMENT IN RT  
WEEKIMP: AVERAGE WEEKLY IMPROVEMENT IN RT */  
INFILE 'A:SWIM2.DAT';  
INPUT GENDER $ WEEKS INITIAL FINAL;  
TOTALIMP=INITIAL-FINAL;  
WEEKIMP=TOTALIMP/WEEKS;  
RUN;  
PROC PRINT;  
RUN;  
PROC PRINT NOOBS;  
RUN;  
QUIT;
```

After running this SAS program, you should obtain the same output produced by the previous example. You may use either approach to store and read your data. □