

Fundamental Concepts of the SAS® Programming Language

Tom Winn
Office of the State Auditor
State of Texas

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.
® indicates USA registration.

Goals & Objectives for this Tutorial

- My goal is that you will be able to use Base/SAS software to access, analyze, manage, and present data in various simple formats.
 - You will learn how to utilize data which either are internal to your program (instream data), or which are stored in another computer file (existing SAS data files or external files).
 - You will learn how to prepare various types of simple reports.
- What are your expectations and interests?

Outline of Topics to be Covered

- An overview of the SAS System,
- Introduction to SAS programming,
- Creating SAS data files,
- Managing SAS data files,
- Modifying SAS data files,
- Processing data with a few fundamental SAS procedures,
- Writing customized reports,
- Other topics (if there is enough time).

The SAS System

- The *SAS System* is an integrated suite of information delivery software products. It is a library of modular components which are tied together by a central supervisory program.
- Applications of the SAS System include executive information systems; data entry, retrieval, and management; report writing and graphics; statistical and mathematical analysis; business planning; forecasting, and decision support; operations research and project management; statistical quality improvement; computer performance evaluation; and applications development.

The SAS System, continued

- Originally (1976), "SAS" stood for "Statistical Analysis System", however, the applicability of SAS has grown far beyond that single purpose.
- Besides statistics, elements of the SAS System also include web publishing, data warehousing, data mining, and business intelligence solutions, a variety of industry-specific (e.g., IT management, pharmaceutical, oil & gas, and human resources) and cross-industry products, as well as state-of-the-art applications development tools.
- Today, the SAS System is a comprehensive system for data management and analysis.

The SAS System, continued

- SAS software collects data from almost any platform and data format; it cleans and transforms the data into information which decision makers will understand; and it stores the information in an open and efficient data storage structure.
- To explore information, SAS software provides multidimensional data analysis, query and reporting, executive information systems, data mining, data visualization, and applications development capabilities. SAS solutions are client/server- and Web-enabled.
- SAS says about itself, "*The Power to Know.™*"

The SAS System, continued

- Components of the SAS System include (partial listing):
 - Base SAS
 - SAS/ACCESS
 - SAS/AF
 - SAS/ASSIST
 - SAS/CALC
 - SAS/CONNECT
 - SAS/EIS
 - SAS/Enterprise Miner
 - SAS/ETS
 - SAS/FSP
 - SAS/GIS
 - SAS/GRAPH
 - SAS/IML
 - SAS/INSIGHT
 - SAS/InteNet
 - SAS/LAB
 - SAS/MDDDB Server
 - SAS/OR
 - SAS/QC
 - SAS/SHARE
 - SAS/SPECTRAVIEW
 - SAS/STAT
 - SAS/TOOLKIT
 - SAS/Warehouse Administrator

The SAS System, continued

- This tutorial is about fundamental elements of the SAS programming language. It covers elementary DATA step, and Base/SAS PROCedure programming.

Suggested References:

- ★ SAS Institute Inc., SAS OnlineDoc, Version 8
- ★ SAS Institute Inc., Getting Started With the SAS System, Version 8
- ★ SAS Institute Inc., SAS Language: Reference, Version 6, First Edition (1990)
- ★ SAS Institute Inc., SAS Procedures Guide, Version 6, First Edition (1990)
- SAS Institute Inc., SAS Language and Procedures: Usage, Version 6, First Edition (1989)
- SAS Institute Inc., SAS Language and Procedures: Syntax, Version 6, First Edition (1990)

Additional Reading Suggestions:

- SAS Institute Inc., SAS Language and Procedures: Introduction, Version 6, First Edition (1990)
- SAS Institute Inc., SAS Applications Guide, 1987 Edition
- ★ Ronald P. Cody & Raymond Pass, SAS Programming By Example (1995)
- ★ Lora D. Delwiche & Susan J. Slaughter, The Little SAS Book: A Primer, Second Edition (1998)
- Frank DiIorio, SAS Applications Programming: A Gentle Introduction
- SAS Institute Inc., SAS Companion for the MVS Environment, Version 6, First Edition (1990)
- SAS Institute Inc., SAS Companion for the Microsoft Windows Environment, Version 6, First Edition (1993)

Introduction to SAS Programming

- All SAS jobs are a sequence of *SAS steps*, which are made up of instructions, which are called *SAS statements*.
- There are only two kinds of SAS steps:
 - ✓ *DATA* steps are used to read, edit, and transform data (raw data or SAS data files), to prepare SAS data sets.
 - ✓ *PROC* steps are ready-to-use procedures which analyze or process SAS data sets.
- In general, data must be in a SAS data file before they can be processed by SAS procedures.

Introduction to SAS Programming, continued

```
DATA STUDENTS;
  INPUT NAME $ 1-14 SEX $ 15
        SECTION $ 17-19 GRADE;
  DATALINES;
... data lines ...
;

PROC SORT DATA=STUDENTS;
  BY SECTION DESCENDING GRADE;

PROC PRINT DATA=STUDENTS;
  BY SECTION;
RUN;
```

Introduction to SAS Programming, continued

- There are two kinds of SAS data sets:
 - ✓ SAS data files (or tables), and
 - ✓ SAS data views.
- A SAS data file contains:
 - ✓ the *descriptor portion*, which provides SAS procedures and some DATA step statements with descriptive information (data set attributes and variable attributes) about the data, and
 - ✓ the *data portion*, a rectangular structure containing the data values, with *rows* (customarily called *observations*), and *columns* (customarily called *variables*); and which is passed to most procedures, observation by observation.
- A *SAS catalog* is a type of SAS file which stores many different types of information used by the SAS System.
- All SAS files reside in a *SAS data library*.

Introduction to SAS Programming, continued

- The SAS System processes the program in two steps.
 - ① it compiles the program,
 - ② it executes the program.
- When the program is compiled, a *program data vector* (PDV) is constructed for each DATA step. It is an area of memory which includes all variables which are referenced either explicitly or implicitly in the DATA step.

Introduction to SAS Programming, continued

- At execution time, the PDV is the location where the current working values are stored as they are processed by the DATA step.
- Variables are added to the PDV sequentially as they are encountered during parsing and interpretation of SAS source statements.
- Each step (DATA or PROC) is compiled and executed separately, in sequence. And at execution time within each DATA step, each observation is processed iteratively through all of the SAS programming statements of the DATA step.

Introduction to SAS Programming, continued

- SAS procedures (PROCs) are programs which are designed to perform specific data processing and analysis tasks on SAS data sets.
- Base/SAS procedures fall into the following categories:
 - ✓ SAS Utilities -- APPEND, CATALOG, CIMPORT, COMPARE, CONTENTS, COPY, CPORT, DATASETS, DBCSTAB, DISPLAY, EXPLODE, EXPORT, FORMAT, FSLIST, IMPORT, OPTIONS, PMENU, PRINTTO, RANK, REGISTRY, SORT, SPELL, SQL, STANDARD, TRANSPOSE, TRANTAB;
 - ✓ Descriptive Statistics -- CORR, FREQ, MEANS, SQL, SUMMARY, TABULATE, UNIVARIATE;
 - ✓ Reporting -- CALENDAR, CHART, FORMS, PLOT, PRINT, REPORT, SQL, TABULATE, TIMEPLOT.

Creating SAS Data Files

- Since SAS procedures can operate only on SAS data sets, then the first step in processing any raw data using SAS will be to transform them into a SAS data set.
- Whenever the SAS System creates a SAS DATA file, it does the following:
 - it reads the DATA statement, creates the structure of a SAS data set, and marks the statement as the place to begin the processing of each line of data;
 - it uses the description of the data in the INPUT statement to read the data line, and to produce an observation;
 - it uses the observation to execute any other SAS statements that are in the DATA step;
 - it adds the observation to the data set being created; and
 - it returns to the beginning of the DATA step for the processing of the next observation.
- All SAS DATA step statements are executed once for each observation.

Creating SAS Data Files, continued

- SAS statements begin with an identifying keyword, and end with a semicolon.


```
DATA STUDENTS;
    INPUT NAME $ 1-14 SEX $ 15
          SECTION $ 17-19 GRADE;
DATALINES;
... data lines ...
PROC SORT DATA=STUDENTS;
    BY SECTION DESCENDING GRADE;
PROC PRINT DATA=STUDENTS;
    BY SECTION;
RUN;
```
- SAS statements are free-format.
 - ✓ They can begin anywhere, and end anywhere.
 - ✓ A single statement may continue over several lines.
 - ✓ Several statements may be on a single line.
 - ✓ Blanks (as many as desired) are used to separate fields. Other special characters also may be used to separate fields.

Creating SAS Data Files, continued

- The *data portion* of a SAS data file is a collection of data values arranged in a rectangular table.

	Variables			
	NAME	SEX	SECTION	GRADE
Observation 1	Abdallah	F	Mon	46
Observation 2	Anderson	M	Wed	76
Observation 3	Aziz	F	Wed	87
Observation 4	Byar	M	Wed	77
Observation 5	Black	M	Fri	70
...				
Observation 50	Zabinski	M	Fri	89

- The *rows* in the table are called *observations*.
- The *columns* in the table are called *variables*.
 - There are two kinds of variables: *character* variables, and *numeric* variables.
 - Each variable has a name.

Creating SAS Data Files, continued

- There are rules for naming SAS data sets and variables:
 - ✓ 1 to 32 characters in length (8 character maximum in Version 6 and earlier versions)
 - ✓ start with A-Z or _ (underscore)
 - ✓ continue with letters, numbers, or underscores.
- It is recommended that you choose meaningful names.

Creating SAS Data Files, continued

- Character data can consist of up to 32,767 characters (max. of 200 characters in Version 6 and earlier versions). Character values may include letters, numbers, blanks and special characters, although generally you should not include any semicolons within the data.
- Numeric data values must be numbers, and they may be preceded by a + or -. Unless the data are being read using a special SAS informat, do not include commas or dollar signs in numeric data values.
- SAS assigns the value of a decimal point (".") to *missing numeric values*, and a blank (" ") to *missing character values*. You can enter these values into your data to indicate missing values.

Creating SAS Data Files, continued

- Every SAS data set has a name and is physically stored on some type of media (disk, tape, etc.). In simple jobs, the SAS data sets are stored on temporary space, but they can be stored "permanently".
- A *temporary SAS data set* exists only for the duration of the current SAS job, or interactive SAS session. A *permanent SAS data set* exists after the end of the current SAS job or interactive SAS session.
- Both types of SAS data sets have two-level names, of the form *libref.data-set-name*, where *libref* is a reference to the name of a SAS data library (a collection of SAS files).

Creating SAS Data Files, continued

- With temporary SAS data sets, the SAS System automatically assigns the libref *WORK* and you specify the data set name.
- When you create a permanent SAS data set, you must specify both the libref and the data set name. The SAS System does not assign the libref for you.
- Although there are other methods for certain operating environments, the LIBNAME statement is the most universal method of assigning a libref. The general form is
LIBNAME libref 'SAS-data-library';

Creating SAS Data Files, continued

- Here is an example of a LIBNAME statement in the Windows environment:
LIBNAME mylib1 'C:\mySASlib';
- Here is an example of a LIBNAME statement in certain mainframe environments:
LIBNAME mylib2 'data.set.name';
- Here is an example of reading data from a "permanent" SAS data set (in the SAS data library whose previously defined libref is 'MYLIB'):
DATA EXAMPLE;
SET MYLIB.STUFF;
- Here is an example of creating a "permanent" SAS data set (also in the SAS data library whose previously defined libref is 'MYLIB'):
DATA MYLIB.TESTDATA;
SET SAMPLE1;

Creating SAS Data Files, continued

- A SAS DATA statement instructs the SAS System to create and name a SAS data set. It has the general syntax:

```
DATA data-set-name-1 (options-1)
    data-set-name-2 (options-2)
    ...
    data-set-name-k (options-k);
```
- Many SAS data sets can be created in a single DATA step.
- DATA step options include such things as:
 DROP=, IN=, FIRSTOBS=, KEEP=,
 OBS=, RENAME=, WHERE=, and others.

Creating SAS Data Files, continued

- The two major functions of the DATA statement are:
 - ✓ to signal the beginning of the DATA step, and
 - ✓ to name the data set(s) being created.
- When creating temporary SAS data sets, the data set name can be supplied by the programmer:

```
DATA STUDENTS;
    INPUT NAME $1-14 SEX $15
          SECTION $17-19 GRADE;
    DATALINES;
    ... data lines ...
```
- or, if the name is omitted in the DATA statement, the SAS System will provide a name (DATA1, DATA2, etc.):

```
DATA;
    INPUT NAME $1-14 SEX $15
          SECTION $17-19 GRADE;
    DATALINES;
    ... data lines ...
```

Creating SAS Data Files, continued

- The **INPUT Statement** provides descriptive information about the raw data that is needed by the SAS System in order to organize the data into a SAS data set.
 - ✓ The INPUT statement assigns names to the SAS variables, it identifies each variable's type and, if necessary, it specifies the column locations from which they are to be read.
 - ✓ There are three modes for the INPUT statement: list, column, and formatted. (It is possible to mix these input styles, if it is done without ambiguity.)

Creating SAS Data Files, continued

- **List input** is when we list the variables in the order in which they appear in the input record.

```
DATA STUDENTS;
    INPUT NAME $ SEX $ SECTION $ GRADE;
    DATALINES;
    ... data lines ...
```
- Whenever the list input style is used, the INPUT statement scans over blanks in the data until it finds a character, then it reads in the field until the next blank is encountered.
- A \$ is used to indicate character variables.
- In list input:
 - ✓ each field is specified in order,
 - ✓ fields must be separated with blanks,
 - ✓ data values for character data may not contain any embedded blanks, and may not exceed 8 characters in length (unless a LENGTH statement is used to increase the default length).

Creating SAS Data Files, continued

- **Column input** is when we specify *where* the variables may be found on the input record, according to their column position.

```
DATA STUDENTS;
    INPUT NAME $1-14 SEX $15
          SECTION $17-19 GRADE;
    DATALINES;
    ... data lines ...
```
- Whenever column input is used, the field positions for the variables are fixed.
- A \$ is used to indicate character variables.
- In column input,
 - ✓ data fields may be read in any order,
 - ✓ blank fields are read as missing,
 - ✓ embedded blanks are allowed in character values,
 - ✓ the length of character data values can range from 1 to 200,
 - ✓ fields or parts of fields can be re-read.

Creating SAS Data Files, continued

- **Formatted input** is when we specify *where* the variables may be found on the input record through the use of pointer controls, and we specify *how* the data values are to be read by specifying an informat for each variable.

```
DATA STUDENTS;
    INPUT NAME $14. @15 SEX $1
          +1 SECTION $3. +1 GRADE 3.;
    DATALINES;
    ... data lines ...
```
- Here are some pointer controls:
 - @n means "go to column n",
 - +n means "move the pointer n positions".
- Here are a few commonly-used informats:
 - w. means "standard numeric",
 - w.d means "standard numeric with decimal point inserted",
 - \$w. means "standard character".

Creating SAS Data Files, continued

- The SAS System can handle data in virtually any form, from almost any input file. We point at the data file in two ways, depending upon the circumstances.
 - ✓ Data which are specified explicitly in the job stream are preceded by a **DATALINES** (or a **CARDS**) statement.
 - ✓ Data which are stored in some external (*non-SAS*) file are referred to by an **INFILE** statement.
- Regardless of where the data are stored, an **INPUT** statement is used to describe the data for the SAS System to read.

Creating SAS Data Files, continued

- Recall that data which are stored in an external (*non-SAS*) file are referred to by an **INFILE** statement.
- Here are two examples of reading data from external files:

Example 1:

```
//FILE1 DD DSN=USER.B30.TWIN504.TRNGCLS.DATA,DISP=OLD
```

DATA STUDENTS;

INFILE FILE1;

INPUT NAME \$ 1-14 SEX \$ 15 SECTION \$ 17-19 GRADE;

Example 2:

DATA STUDENTS;

INFILE 'C:\MyData\FILE2.dat';

INPUT NAME \$ 1-14 SEX \$ 15 SECTION \$ 17-19 GRADE;

Creating SAS Data Files, continued

- The **LABEL Statement** is used to assign descriptive labels to variables.
LABEL variable = 'label' ...;
- Labels may be up to 256 characters in length (maximum of 40 characters in Version 6 and earlier versions).
- Many SAS procedures use the variable labels for printing.
- Labels defined in a SAS DATA step are stored in the descriptor portion of the SAS data set, along with such other variable attributes as type, length, informat, and format.

A Few Words About Working With Dates and Times Using SAS

- Whenever SAS reads date value inputs, it converts them into integers. **SAS dates** are positive or negative integers representing the number of elapsed days between January 1, 1960 and the specified date.
- Similarly, SAS converts time values into integers, representing the number of seconds since midnight of the current day.

A Few Words About Working With Dates and Times Using SAS, continued

- You may use **date constants** or **time constants** in a SAS expression by writing the date or time enclosed in quotes, and followed by a D (date), a T (time), or DT (date:time).
- To read data that are date or time values, SAS has a variety of **informats**.
- To write date or time values in reports, SAS has numerous **formats**.
- Consult the suggested references for complete descriptions and examples.

Managing SAS Data Files

- Three DATA step statements that read SAS data sets: **SET**, **MERGE**, and **UPDATE**.

Managing SAS Data Files, continued

- We use a **SET** statement to access one or more SAS data files which were defined previously.

```
DATA NEW1;  
  SET STUDENTS; /* from libref WORK */  
  
DATA NEW2;  
  SET MYLIB.CLASS; /* from libref MYLIB */
```

Managing SAS Data Files, continued

- We also use a **SET** statement to *concatenate* previously-defined SAS data files. In concatenation, the data files are *stacked vertically*, one on top of the other.

```
DATA NEW;  
  SET SAMPLE1 SAMPLE2;
```

- We use a **SET** statement with a **BY** statement to *interleave* observations from two previously-defined SAS data files, both of which are already sorted according to values of the specified BY-variable.

```
DATA NEW;  
  SET SAMPLE1 SAMPLE2;  
  BY GRPNUM;
```

Managing SAS Data Files, continued

- We use a **MERGE** statement to join corresponding observations from two or more SAS data files into single observations in a new SAS data file.
- Whenever a **MERGE** statement is used with a **BY** statement, the observations are *matched* according to the values of the variable specified in the **BY** statement. The SAS data files must be sorted according to the variable specified in the **BY** statement

```
DATA NEW;  
  MERGE NAMES ADDRESSES;  
  BY IDNUM;
```

Managing SAS Data Files, continued

- Whenever a **MERGE** statement is not accompanied by a **BY** statement, the observations are combined in a *one-to-one* manner; that is, the first observation in the first SAS data set is combined with the first observations in the other SAS data sets, the second observation in the first SAS data set is combined with the second observations in the other SAS data sets, and so on, until it reaches the last observation in one of the SAS data files being combined.

```
DATA NEW;  
  MERGE NAMES ADDRESSES;
```

Managing SAS Data Files, continued

- We use an **UPDATE** statement for the special case of combining observations with matched key values from two SAS data files, a master data file, and a transactions data file. **UPDATE** uses observations from the transactions data file to change the matching observations in the master data file. Both of the input SAS data files must be sorted according to the variable specified in the **BY** statement

```
DATA NEWMSTR;  
  UPDATE MASTER TRNSACTN;  
  BY IDNUM;
```

Modifying SAS Data Files

- Once you have gotten your data into a SAS data set, you may want to change or manipulate them in some ways.
- SAS has a full range of statements which can be used to modify data. For example, some statements delete observations, select only certain variables, create new variables, or perform specific tasks based upon certain conditions.

Modifying SAS Data Files, continued

- SAS data files can be changed by:
 - ✓ selecting a subset of the observations,
 - ✓ selecting variables,
 - ✓ adding new variables,
 - ✓ changing data values,
 - ✓ adding observations.

Modifying SAS Data Files, continued

- Deleting observations from a SAS data set can be accomplished in either of these (logically-equivalent) ways:
 - ✓ the DELETE statement, or
 - ✓ the subsetting-IF statement.

```
DATA FEMALES;  
SET STUDENTS;  
IF SEX = 'M' THEN DELETE;
```

```
DATA FEMALES;  
SET STUDENTS;  
IF SEX = 'F';
```

Modifying SAS Data Files, continued

- Deleting variables from a SAS data set can be accomplished in either of these ways:
 - ✓ the DROP statement, or
 - ✓ the KEEP statement.

```
DATA TRAINING;  
SET STUDENTS;  
DROP SECTION;
```

```
DATA TRAINING;  
SET STUDENTS;  
KEEP NAME SEX GRADE;
```

Modifying SAS Data Files, continued

- Creating and redefining variables from existing variables is accomplished with an *assignment statement*. An assignment statement is in the form of an equation, the left side of which is a variable name, either new or existing, and the right side of which may be a constant, another variable, or an expression.

```
DATA INCOME;  
SET INCOME;  
NETINC = SALES - COSTSOLD - EXPENSE  
+ OTHERINC;  
OPERCOST = .15 * SALES;  
DIVISION = REGION || DEPARTMENT;
```

Modifying SAS Data Files, continued

- Conditional logic is used to perform specific operations based upon certain conditions. This is accomplished through the use of IF-THEN/ELSE statements, which are of the form:

```
IF condition(s) THEN action1;  
ELSE action2;
```

- DATA STUDENTS;
SET STUDENTS;
LENGTH GROUP \$ 11;
IF GRADE >= 90 THEN GROUP = 'HIGH-PASS';
ELSE IF 75 <= GRADE < 90
THEN GROUP = 'MEDIUM-PASS';
ELSE IF 60 <= GRADE < 75
THEN GROUP = 'LOW-PASS';
ELSE GROUP = 'FAIL';

Modifying SAS Data Files, continued

- SAS includes numerous functions which can simplify programming tasks by transforming input-arguments into desired output-results. For example, SAS functions may perform mathematical & statistical computations, character-string manipulation, or date & time calculations. SAS functions may be nested.
- Consult the suggested references for complete descriptions and examples.

Processing Data with a Few Fundamental SAS Procedures

- SAS procedures are pre-written computer programs that process SAS data sets. SAS procedures produce reports, manage files, or analyze data.

- A **PROC Statement** is used to invoke a SAS procedure.

`PROC xxxxxx options ;`

Processing Data with a Few Fundamental SAS Procedures, continued

- The following SAS statements may be used in any SAS PROC step:

- ✓ a **BY** statement may be used to identify groupings for sorting the procedure output,
- ✓ a **WHERE** statement may be used to subset the input data set before it is processed by the procedure,
- ✓ a **LABEL** statement may be used to identify descriptive headings to be used in place of the variable names in reports generated by the procedure,
- ✓ a **FORMAT** statement may be used to specify how data values should be written in procedure output,
- ✓ a **TITLE** statement may be used to specify a title line to be printed at the top of procedure reports,
- ✓ a **FOOTNOTE** statement may be used to specify a footnote to be printed at the bottom of procedure reports.

Processing Data with a Few Fundamental SAS Procedures, continued

- We will discuss a few procedures which commonly are used for sorting, summarizing, and printing data.

Processing Data with a Few Fundamental SAS Procedures, continued

- **PROC SORT** is used for sorting the observations in a SAS data file. The general syntax is:

```
PROC SORT DATA=data-set-name1
      OUT=data-set-name2 option(s);
      BY variable1 variable2 ... variablem
      DESCENDING variablen;
```

Processing Data with a Few Fundamental SAS Procedures, continued

- **PROC MEANS** is used for summarizing and generating descriptive statistics for the data in a SAS data file. The general syntax is:

```
PROC MEANS DATA=data-set-name1 option(s);
      VAR variable-list1;
      CLASS variable-list2;
      ID variable-list3;
      BY variable-list4;
      OUTPUT OUT=data-set-name2
      statistic(variable-list)= name-list;
```

- Some of the output statistics which may be requested with the MEANS procedure are the following: N, NMISS, MIN, MAX, RANGE, SUM, MEAN, MEDIAN, STDDEV, VAR.

Processing Data with a Few Fundamental SAS Procedures, continued

- **PROC FREQ** may be used for producing simple frequency counts and cross-tabulation tables, for analyzing distributions of categorical data values. The general syntax is:

```
PROC FREQ DATA=data-set-name;
      BY variable-list;
      TABLES requested-variable-combinations
      / option(s);
```

- Frequency counts for all values of a single variable would be specified by the following:
`TABLES variable-name / LIST MISSING;`
- A 2-way frequency table for all combinations of the values of two variables would be specified by:
`TABLES variable1*variable2 / LIST MISSING;`

Processing Data with a Few Fundamental SAS Procedures, continued

- PROC PRINT is used for producing listings of the data portion of a SAS data file. The general syntax is:

```
PROC PRINT DATA=data-set-name option(s);
  VAR variable-list1;
  ID variable-list2;
  BY variable-list3;
  PAGEBY variable;
  SUMBY variable;
  SUM variable-list4;
```

Processing Data with a Few Fundamental SAS Procedures, continued

- Let's examine some simple examples of using SAS procedures.

Here is a DATA step:

```
DATA SATURN;
  LENGTH COLOR $10 TRANS $9;
  INPUT YEAR MODEL $ PRICE MILEAGE
         COLOR $ TRANS $ AC;
  DATALINES;
96 SC2 12475 23705 white automatic 1
96 SL2 8489 55480 black manual 1
99 SL1 14007 3015 dark_green automatic 0
96 SL2 11048 44480 purple automatic 0
96 SC2 11224 14433 red manual 0
94 SC2 7884 85201 med_blue automatic 0
97 SW2 12668 31912 white automatic 1
95 SL1 9464 29734 blue_green automatic 1
96 SL2 11264 24424 copper automatic 1
93 SL2 8142 49770 blue_green automatic 0
97 SL2 12996 19830 white automatic 1
98 SC2 13848 20004 white automatic 0
98 SL1 12998 13454 silver automatic 1
96 SL2 10994 67663 light_plum manual 1
97 SW2 13628 20789 gold automatic 1
96 SL1 9996 44084 dark_green automatic 1
95 SL1 8588 47716 med_blue manual 1
95 SC2 8968 80931 red manual 0
93 SL2 7965 77004 med_blue automatic 1
97 SC2 13988 29778 silver automatic 1
96 SL2 12248 14543 gold automatic 1
94 SC1 8996 57688 plum manual 1
95 SC1 9428 49882 red manual 1
96 SC2 11468 64401 purple automatic 1
97 SW2 12994 17422 white manual 0
;;;
RUN;
```

The procedure output of all SAS jobs is accompanied by the *SAS Log*, which documents the step-by-step processing of the job. Take time to examine the SAS Log, even when you think that your output looks okay. This is the SAS Log for the preceding DATA step example:

```
1 DATA SATURN;
2   LENGTH COLOR $10 TRANS $9;
3   INPUT YEAR MODEL $ PRICE MILEAGE
4         COLOR $ TRANS $ AC;
5   DATALINES;
```

NOTE: The data set WORK.SATURN has 25 observations and 7 variables.

NOTE: DATA statement used:
real time 0.43 seconds

```
31 ;;;
32 RUN;
```

Continuing our Example:

```
33 PROC PRINT DATA=SATURN;
34 TITLE 'Original SATURN Data Set';
```

NOTE: There were 25 observations read from the dataset WORK.SATURN.

NOTE: PROCEDURE PRINT used:
real time 0.49 seconds

Obs	COLOR	TRANS	YEAR	MODEL	PRICE	MILEAGE	AC
1	white	automatic	96	SC2	12475	23705	1
2	black	manual	96	SL2	8489	55480	1
3	dark_green	automatic	99	SL1	14007	3015	0
4	purple	automatic	96	SL2	11048	44480	0
5	red	manual	96	SC2	11224	14433	0
6	med_blue	automatic	94	SC2	7884	85201	0
7	white	automatic	97	SW2	12668	31912	1
8	blue_green	automatic	95	SL1	9464	29734	1
9	copper	automatic	96	SL2	11264	24424	1
10	blue_green	automatic	93	SL2	8142	49770	0
11	white	automatic	97	SL2	12996	19830	1
12	white	automatic	98	SC2	13848	20004	0
13	silver	automatic	98	SL1	12998	13454	1
14	light_plum	manual	96	SL2	10994	67663	1
15	gold	automatic	97	SW2	13628	20789	1
16	dark_green	automatic	96	SL1	9996	44084	1
17	med_blue	manual	95	SL1	8588	47716	1
18	red	automatic	95	SC2	8968	80931	0
19	med_blue	automatic	93	SL2	7965	77004	1
20	silver	automatic	97	SC2	13988	29778	1
21	gold	automatic	96	SL2	12248	14543	1
22	plum	manual	94	SC1	8996	57688	1
23	red	manual	95	SC1	9428	49882	1
24	purple	automatic	96	SC2	11468	64401	1
25	white	manual	97	SW2	12994	17422	0

```
35 PROC SORT DATA=SATURN;
36 BY YEAR MODEL;
```

NOTE: There were 25 observations read from the dataset WORK.SATURN.

NOTE: The data set WORK.SATURN has 25 observations and 7 variables.

NOTE: PROCEDURE SORT used:
real time 0.00 seconds

```
37 PROC PRINT DATA=SATURN;
38 TITLE 'SATURN Data Set After Sorting';
39 RUN;
```

NOTE: There were 25 observations read from the dataset WORK.SATURN.

NOTE: PROCEDURE PRINT used:
real time 0.04 seconds

SATURN Data Set After Sorting

OBS	COLOR	TRANSM	YEAR	MODEL	PRICE	MILEAGE	AC
1	blue-green	automatic	83	SL2	8442	140770	0
2	red-blue	automatic	83	SL2	7985	77804	1
3	plum	manual	84	SC1	8000	57888	1
4	red-blue	automatic	84	SC2	7984	88201	0
5	red	manual	85	SC1	8420	48862	1
6	red	manual	85	SC2	8008	60821	0
7	blue-green	automatic	85	SL1	8484	20734	1
8	red-blue	manual	85	SL1	8380	87716	1
9	white	automatic	86	SC2	13475	23708	1
10	red	manual	86	SC2	11224	14423	0
11	purple	automatic	86	SC2	11688	84601	1
12	dark-green	automatic	86	SL1	8980	44004	1
13	black	manual	86	SL2	8488	58480	1
14	purple	automatic	86	SL2	11048	44480	0
15	copper	automatic	86	SL2	11284	24424	1
16	light-plum	manual	86	SL2	10884	67843	1
17	gold	automatic	86	SL2	12248	14843	1
18	silver	automatic	87	SC2	14880	20778	1
19	white	automatic	87	SL2	12880	18820	1
20	white	automatic	87	SC2	12888	31812	1
21	gold	automatic	87	SC2	13828	20788	1
22	white	manual	87	SC2	12884	17822	0
23	white	automatic	88	SC2	13848	20804	0
24	silver	automatic	88	SL1	12888	13484	1
25	dark-green	automatic	89	SL1	14887	3015	0

```

40  PROC MEANS DATA=SATURN;
41      VAR PRICE MILEAGE;
42      CLASS YEAR MODEL;
43      OUTPUT OUT=SUMRY MEAN=AVGPRICE AVGMILGE;
44      TITLE 'Using PROC MEANS with the SATURN
Data Set';

NOTE: There were 25 observations read from the
dataset WORK.SATURN.
NOTE: The data set WORK.SUMRY has 28 observations
and 6 variables.
NOTE: PROCEDURE MEANS used:
      real time          0.16 seconds

```

Using PROC MEANS with the SATURN Data Set
The MEANS Procedure

YEAR	MODEL	MEAN	STD DEV	MINIMUM	MAXIMUM
83	SL2	8227.20	277.2877	7985.00	8442.00
84	SC1	8200.00	178.8854	8000.00	8400.00
	SC2	8192.00	178.8854	7984.00	8400.00
85	SC1	8327.20	178.8854	8008.00	8420.00
	SC2	8327.20	178.8854	8008.00	8420.00
86	SL1	8980.00	277.2877	8488.00	8980.00
	SL2	8980.00	277.2877	8488.00	8980.00
86	SC1	11776.00	178.8854	11224.00	12248.00
	SC2	11776.00	178.8854	11224.00	12248.00
86	SL1	8980.00	277.2877	8488.00	8980.00
	SL2	8980.00	277.2877	8488.00	8980.00
87	SC1	13880.00	178.8854	12888.00	14880.00
	SC2	13880.00	178.8854	12888.00	14880.00
88	SC1	13880.00	178.8854	12888.00	14880.00
	SC2	13880.00	178.8854	12888.00	14880.00
88	SL1	12888.00	178.8854	12000.00	13888.00
	SL2	12888.00	178.8854	12000.00	13888.00
89	SL1	14887.00	178.8854	14000.00	15000.00
	SL2	14887.00	178.8854	14000.00	15000.00

```

45  PROC PRINT DATA=SUMRY;

NOTE: There were 28 observations read
from the dataset WORK.SUMRY.
NOTE: PROCEDURE PRINT used:
      real time          0.05 seconds

```

Using PROC MEANS with the SATURN Data Set

OBS	YEAR	MODEL	_TYPE_	PRICE	AVGPRICE	AVGMILE
1	83	SL2	0	8442.00	140770.00	0.00
2	83	SL2	1	7985.00	77804.00	1.00
3	84	SC1	0	8200.00	57888.00	0.00
4	84	SC1	1	8400.00	84000.00	1.00
5	84	SC2	0	7984.00	88201.00	0.00
6	84	SC2	1	8400.00	60821.00	1.00
7	85	SC1	0	8008.00	20734.00	0.00
8	85	SC1	1	8420.00	48862.00	1.00
9	85	SC2	0	8008.00	20734.00	0.00
10	85	SC2	1	8420.00	48862.00	1.00
11	86	SL1	0	8488.00	20778.00	0.00
12	86	SL1	1	8980.00	44004.00	1.00
13	86	SL2	0	8488.00	58480.00	0.00
14	86	SL2	1	8980.00	44004.00	1.00
15	86	SC1	0	11224.00	14423.00	0.00
16	86	SC1	1	12248.00	14843.00	1.00
17	86	SC2	0	11688.00	84601.00	0.00
18	86	SC2	1	11048.00	44480.00	1.00
19	86	SL1	0	8980.00	44004.00	0.00
20	86	SL1	1	8980.00	44004.00	1.00
21	86	SL2	0	8980.00	44004.00	0.00
22	86	SL2	1	8980.00	44004.00	1.00
23	87	SC1	0	12888.00	18820.00	0.00
24	87	SC1	1	14880.00	20778.00	1.00
25	87	SC2	0	12888.00	18820.00	0.00
26	87	SC2	1	14880.00	20778.00	1.00
27	88	SL1	0	12888.00	18820.00	0.00
28	88	SL1	1	14887.00	3015.00	0.00

```

46  PROC FREQ DATA=SATURN;
47      TABLES YEAR*MODEL;
48      TITLE 'Using PROC FREQ with the
SATURN Data Set';
49  RUN;

NOTE: There were 25 observations read
from the dataset WORK.SATURN.
NOTE: PROCEDURE FREQ used:
      real time          0.33 seconds

```

