Learn SAS ® in 50 minutes

Subhashree Singh, The Hartford, Hartford, CT

ABSTRACT

SAS ® is the leading business analytics software used in a variety of business domains such as Insurance, HealthCare, Pharmacy, Telecom etc. Very often, business analysts and other professionals with little or no programming experience are required to learn SAS. This paper will explain the key concepts and the minimum knowledge required to get started with SAS. The information presented in this paper will enable new SAS users to get up and running in a very short amount of time.

INTRODUCTION

The SAS software includes a number of different components such as Base SAS, SAS/STAT, SAS/GRAPH, SAS/AF etc. This paper will explain the key concepts of Base SAS to enable a novice to seamlessly get up to speed on SAS and perform most of the tasks that are expected of a beginner.

1. SAS AND ITS RULES

1.1 WHAT IS SAS?

SAS stands for Statistical Analysis System.

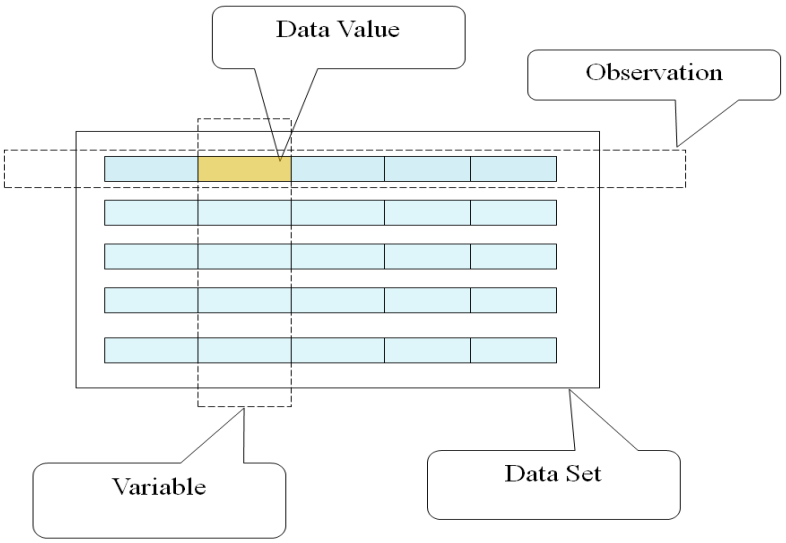
Useful for the following types of task:

- Data entry, retrieval, and management
- Report writing and graphics
- Statistical and mathematical analysis

1.2 UNDERSTANDING TERMS IN A SAS DATA STRUCTURE

- DATA VALUE: The basic unit of information.
- VARIABLE: A set of data values that describes a given attribute. 2 main types of variable types: numeric and character
- OBSERVATION: All the data values associated with a particular record.
- DATA SET: It is a collection of observation.

Figure:



1.3 RULES FOR SAS NAMES

- SAS variable/variable names must be between 1 and 32 characters long.
- The first character must be a letter or an underscore.
- Characters after the first may be letters, digits or underscores.

Example: `NAME, _NAME_, FILE1, _NULL_` etc.

1.4 RULES FOR SAS STATEMENTS

- SAS statements may begin in any column of the line.
- SAS statements end with a semicolon (;).
- Some SAS statements consist of more than one line of commands.
- A SAS statement may continue over more than one line.

One or more blanks should be placed between items in SAS statements. If the items are special characters such as '=', '+', '$', the blanks are not necessary.

There are two major building blocks in SAS.

- DATA step
- SAS built in procedures

2. DATA STEP

Used for: Names the SAS data set and creates the dataset.

In a Data step the following can be performed.

- defining the variables
- read input files
- assign values to the variables,
- creating new variables,
- merging two data sets
- formatting and labeling variables
- assignment of missing values.

If a variable is used in a SAS program but not initialized then SAS automatically assign a missing value to it.

Numeric missing values are represented by a single period (.). Character missing values are represented by a single blank enclosed in quotes (' ').

Syntax:

```
DATA <SOMENAME>;
```

The name should be 1-32 characters and must begin with a letter or underscore.

Example: `DATA EMPDAT;`

The following SAS statements can be used in a DATA step.

2.1 INFILE AND INPUT STATEMENT

Used for: INFILE is used to read external files (such as mainframe file, text files, comma delimited files etc). INPUT is used to define names and order of variables for the SAS dataset.

Syntax:

```
INFILE file-specification <options>;
INPUT variable variable_type column(s);
```

Example1:

```
DATA EMPFL;
      INFILE 'c:\emp\external\emp1.dat';
      INPUT
         @001  EMPNO $CHAR6.
         @007  NAME  $CHAR15.
         @022  AGE   3.;
RUN;
```

In this example external file “emp1.data” stored in the location “c:\emp\external” will be read and copied into SAS file EMPFL. Three variables EMPNO, NAME and AGE will be read.

2.2 SET STATEMENT

Used for: Reads an existing SAS dataset or concatenating SAS datasets
Syntax:

```
SET <SAS dataset name> <OBS=n>;
```

Where “n” is the number of observations you want to read from the file.

Example1:
Copies first 100 records of OLDFILE1 to NEWFILE.

```
DATA NEWFILE;
    SET OLDFILE1 (OBS=100);
RUN;
```

Example2:

Concatenate two datasets.

Input SAS dataset:

SAS data set NWEST

OBS	STATE	POP
1	WA	3.4
2	OR	2.1

SAS data set SWEST

OBS	STATE	POP
1	NM	1.1
2	AZ	1.8

```
DATA COMMON;
     SET NWEST SWEST;
RUN;
```

Output SAS data set

SAS data set COMMON

OBS	STATE	POP
1	WA	3.4
2	OR	2.1
3	NM	1.1
4	AZ	1.8

2.3 IF/THEN; ELSE; STATEMENT

Used for: Used for conditional checking. Used in a data step.

Syntax:

```
IF expression THEN statement;
<ELSE statement;>
```

Example:
```
IF LANG='Spanish' or LANG='French' THEN
      NEWLANG='NotEngl';
ELSE
      NEWLANG='English';
```

Example 2:
```
IF status='M' AND type=1 THEN
    count=count+1;
```

2.4 SUBSETTING "IF" STATEMENT

Used for: To subset, or take a portion of the data set

Syntax:
```
      IF var_name = somevalue1;
      IF var_name = somevalue2 THEN DELETE;
```

Example:
```
DATA FORGNER;
     IF LANG= 'ENGLISH';
     IF TAX < 20000 THEN DELETE;
RUN;
```

In the above example it will select all observations where LANG equals 'ENGLISH' and where TAX greater than or equal to 20000.
Notice that values for character variables must be enclosed in quotes and values must match exactly including case.

2.5 LIBNAME STATEMENT

Used for: Associates a libref with a SAS library. It's kind of location pointer.
This is generally used we want to save the SAS dataset in a permanent location. When a libref is not associated with SAS dataset, SAS assumes it is created or read from SAS work area which is temporary in nature.

Syntax:
```
      LIBNAME libref <engine>'('SAS-data-library-1' <,...'SAS-data-library-n'> ) ';
```

Example:

In windows –
```
      LIBNAME EMPLIB1 'C:\data\SAS\EMPDATA';
```

The path 'C:\data\SAS\EMPDATA' will be assigned to libref EMPLIB1. Say there is a SAS dataset EMPDAT in that library, it can be access as,
```
DATA EMPFILE1;
    SET EMPLIB1.EMPDAT;
RUN;
```

In Unix –
```
LIBNAME CLAIM '\data\CL\CLAIM\INPUT';
DATA CLAIM.LOSS_HISTORY;
     SET LOSS_TEMP;
RUN;
```
In this example the temporary file LOSS_TEMP will be stored in a permanent location '\data\CL\CLAIM\INPUT' with a new name LOSS_HISTORY.

2.6 MERGE STATEMENT

Used for: Joins corresponding observations from two or more SAS data sets. Two types one to one, match merging.

Syntax:

```
DATA new-sasdataset;
      MERGE sasdsname1 sasdsname2 …. sasdsnamen;
      BY var1 var2 varm;
RUN;
```

Input data sets must be sorted by the same BY variables before you can merge them. Usually merge statement followed by sort procedure.

Example:
Merge the two files SURVY with NAMES by the variable NAME.

```
DATA NEW;
   MERGE NAMES
         SURVY;
   BY NAME;
RUN;
```

SAS data set NAMES

OBS	NAME	GENDER
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	27	72	PAUL
4	35	54	RENEE
5	37	56	RENEE
6	32	68	TONY

SAS data set NEW

OBS	AGE	HEIGHT	NAME	GENDER
1	28	64	JENNIFER	F
2	35	60	MANUEL	M
3	27	72	PAUL	M
4	35	54	RENEE	F
5	37	56	RENEE	F
6	32	68	TONY	M

3. SAS BUILT IN PROCEDURES

SAS provides large number of procedures which is invoked in a "PROC step" which starts with the keyword PROC. Some important SAS built in procedures are described in this section.

3.1 PROC CONTENTS

Used for: Lists structure of a SAS dataset. It is very useful when you receive an existing SAS file from someone and you need to start working on it. Before you start you may want to see the structure of this file.

Syntax:

```
PROC CONTENTS DATA=<SAS dataset name> <options> ;
RUN;
```

Example:
To list the structure of the SAS dataset EMPFL.

```
PROC CONTENTS DATA=EMPFL;
RUN;
```

The following is the sample result. SAS also prints other metadata information such as FORMAT, INFORMAT, LABEL, no. of observations, SORT info, INDEX info etc which were not shown here. Please refer to online documentation for details.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	AGE	Num	8
1	EMPNO	Char	6
2	NAME	Char	15

3.2 PROC PRINT

Used for: Lists data as a table of observations.
Large datasets are very hard to visualize, so it is useful when you want to print couple of observations for some variables.

Syntax:

```
PROC PRINT <options>;
      VAR variable(s) <options>;
RUN;
```

Example:
To print variables NAME, AGE and GENDER of SAS dataset FILE1.

```
PROC PRINT DATA=FILE1;
     VAR NAME AGE GENDER;
RUN;
```

Output will look like:

OBS	NAME	AGE	GENDER
1	KEVIN	15	M
2	JANE	12	F
3	MARY	17	F

3.3 PROC DELETE

Used for: Delete one or more SAS dataset. It is good programming practice to delete unwanted SAS datasets to increase SAS work space.

Syntax:

```
PROC DELETE DATA=<Data set name(s)>;
RUN;
```

Example:

```
PROC DELETE DATA=OLDDATA1 CLAIM.OLDDATE2;
RUN;
```

The above statement temporary SAS datasets OLDDATE1 and permanent SAS dataset CLAIM.OLDDATA2 will be deleted.

3.4 PROC SORT

Used for: To rearrange the observations in SAS datasets according to one or more variables.

Syntax:

```
PROC SORT <NODUP/NODUPKEY> DATA=  SAS-file name <OUT=SAS-filename>;
    BY <descending> variable-1 … <descending> variable-n;
RUN;
```

You can sort in ascending or descending order. Default is ascending. BY variables can be numeric or character.

Example:

```
PROC SORT DATA=ORIG OUT=SORTORIG;
   BY GENDER DESCENDING AGE;
RUN;
```

Sorts file first in ascending order of GENDER and within those categories, 1 (Male) and 2 (Female), the cases are further sorted in descending order of AGE.

PROC SORT is used most often for sorting a data set so that other SAS procedures can process that data set in subsets using BY statements. Data sets must also be sorted before they can be merged or updated.

3.5 PROC FREQ

Used for: The FREQ procedure produces one way to n-way frequency tables and creates distribution reports of the variable values. This procedure used for categorical data.

Syntax:

```
PROC FREQ <option>;
      BY variables;
      OUTPUT <OUT=SAS-data-set> <options>;
      TABLES requests </ options>;
RUN;
```

Example:

Find out unique values of the variable department in SAS dataset ‘EmpFile’.

```
PROC FREQ DATA= EmpFile;
    TABLES Department /OUT=Freq_Department;
RUN;
```

The FREQ Procedure

Department

Department	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Audit	6	37.50	6	37.50
Finance	6	37.50	12	75.00
Research	4	25.00	16	100.00

3.6 PROC MEANS

Used for: The MEANS procedure summarizes data. It computes descriptive statistics for variables within groups of observations or across all observations. This procedure generally used for continuous data.

Syntax:

```
PROC MEANS <option(s)> <statistic-keyword(s)>;
      BY variable-1 ... variable-n>;
      CLASS variable(s) </ option(s)>;
      OUTPUT <OUT=SAS-data-set> <output-statistic>;
RUN;
```

Example:

Find out no. of employees, average, maximum, minimum, sum of salary for 'EmpFile' by department.

```
PROC MEANS DATA= EmpFile N MEAN MAX MIN SUM;
     VAR salary;
     BY department;
     OUTPUT OUT=SumEmpFile;
RUN;
```

Remember to sort the file by variable 'department ' before using the variable in 'BY' statement of proc mean as given below.

```
PROC SORT DATA= EmpFile;
     BY department;
RUN;
```

Input Filea

EMP_ID	Department	Salary
1	Finance	23000
2	Audit	53000
3	Audit	42000
4	Finance	80000
5	Research	45220
6	Audit	21444
7	Finance	45200
8	Research	12000
9	Audit	45000
10	Finance	12400
11	Research	53000
12	Audit	42000
13	Finance	80000
14	Research	45220
15	Audit	21444
16	Finance	45200

Output File

Department	_STAT_	Salary
Audit	N	6
Audit	MIN	21444
Audit	MAX	53000
Audit	MEAN	37481.33
Audit	STD	13058.24
Finance	N	6
Finance	MIN	12400
Finance	MAX	80000
Finance	MEAN	47633.33
Finance	STD	28125.55
Research	N	4
Research	MIN	12000
Research	MAX	53000
Research	MEAN	38860
Research	STD	18278.39

4. SOME IMPORTANT SAS FUNCTIONS

This section is for advance reading.
For user's benefit some important SAS functions are mentioned below which can be used in a data step.

4.1. DATE (): produces the current date as SAS date, value representing the number of days between January 1, 1960 and the current date.
E.g., `Today_Date = DATE();`
The value of Today_Date will be 18499 which means August 25, 2010.
Then a SAS data format can be used to read the date. Refer to PUT function for details.

4.2. DAY (DATE): It returns an Integer representing the day of the month from SAS DATE value.
E.g. `Day1=DAY(Today_Date);`
The value of Day1 will be 25.

4.3. MONTH (DATE): It returns the numeric value representing the month from SAS DATE value.
E.g. `Month1=MONTH(Today_Date);`
The value of Month1 will be 8.

4.4. YEAR (DATE): It returns the 4 digits numeric value representing the year from SAS DATE value.
E.g. `Year1=YEAR(Today_Date);`
The value of Year1 will be 2010.

4.5. LENGTH(argument): Calculate length of a variable or a constant. Default length of numeric variable is 8.
E.g.
```
String2='Rocky Hill';
   Len1=LENGTH(String2);
```
The value of Len1 will be 10 which is the length of the string 'Rocky Hill'.

4.6. SUM(variable-1, variable-2, ….,variable-n): Calculates sum of non-missing arguments.
E.g.
```
Num1=5;
   Num2=2.5;
   Sum1=SUM(Num1,Num2);
```
The value of Sum1 will be 7.5

4.7. MIN(variable-1, variable-2, ….,variable-n): Returns the value of the lowest of its non-missing arguments.
E.g. `Min1=MIN(9,0,-2,45);`
The value of Min1 will be -2

4.8. MAX(variable-1, variable-2, ….,variable-n): Returns the value of the lowest of its non-missing arguments.
E.g. `Max1=MAX(9,0,-2,45);`
The value of Max1 will be 45

4.9. INT(argument): This function taken one numeric argument and returns integer portion of the argument.
E.g. `Int1= INT(332.502);`
The value of Int1 will be 332

4.10. CEIL(argument): The CEIL function returns the smallest integer that is greater than or equal to the argument.
E.g.
```
Ceil1=CEIL(9.231);
   Ceil2=CEIL(-9.231);
```
The value of Ceil1 will be 10 and Ceil2 will be -9.

4.11. FLOOR(argument): The FLOOR function takes one numeric argument and returns the largest integer that is less than or equal to the argument.
E.g.
```
Floor1=FLOOR(9.231);
   Floor2=FLOOR(-9.231);
```
The value of Floor1 will be 9 and Floor2 will be -10.

4.12. PUT (source,format) : Write the value of source with a specified format. PUT function is used to convert numeric value to a character value and also to format numeric and date variables into various formats.
E.g.
```
Date_Formatted = PUT(Today_Date,DDMMYY10.);
```
Here DDMMYY10. is the DATE format. Please refer to SAS online documentation to get a list of various formats.
The value of Date_Formatted will be 25/08/2010.

4.13. UPCASE(argument): Converts all letters in an argument to uppercase.
E.g.
```
String2='Rocky Hill';
   Upcase1=UPCASE(String2);
```
The value of Upcase1 will be 'ROCKY HILL'.

4.14. LOWCASE(argument): Converts all letters in an argument to uppercase.
E.g.
```
String2='Rocky Hill';
   Lowcase1=LOWCASE(String2);
```
The value of Lowcase1 will be 'rocky hill'.

4.15. CATS(variable-1, variable-2, ….,variable-n) and ||: It removes leading and trailing blanks, and concatenates the arguments.
|| (two pipe signs) also can be use to concatenate multiple strings. Unlike CATS function || does not remove leading and trailing blanks.
E.g.
```
Var1="United  ";
   Var2=" States";
   Con1=CATS(Var1,Var2,"of America");
   Con2='Var1 || Var2 || "of America"';
```
The value of Con1 will be 'UnitedStatesof America' where as value of Con2 will be 'United Statesof America'.

4.16. TRANWRD(source, target, replacement): TRANWRD replaces or removes all occurrences of a word in a string.
E.g.
```
OldText='This is OLD';
   OldWord='OLD';
   NewWord='NEW';
   NewText=TRANWRD(OldText,OldWord,NewWord);
```
The value of NewText will be 'This is NEW'.

4.17. SUBSTR (argument, position<, n>) : It returns the portion of the string specified in the argument from the position up to the number of characters specified by n.
E.g.
```
String1='THE HARTFORD';
   SubString1=SUBSTR(String1,5,4);
```
The value of SubString1 will be 'HART'.

4.18. TRIM (argument): It copies a character argument removing any trailing elements.
E.g.
```
String3='SAS Institute   ';
   Trim1=trim(String3);
```
The value of Trim1 will be 'SAS Institute'.

CONCLUSIONS

This paper is submitted to help out new SAS users to accustom them to SAS in less time.

REFERENCES:

http://support.sas.com/documentation/onlinedoc/91pdf/index.html

ACKNOWLEDGMENTS

Thanks to the SAS online support for prompt responses to my questions.

Many thanks to Adil Khan for reviewing my paper and encourage me to write this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Subhashree Singh
The Hartford
One Hartford Plaza
Hartford, CT 06155
Work Phone: 860-547-2398
Email: Subhashree.Singh@thehartford.com, Subhashree_S123@yahoo.com

****