A Brief Introduction to SAS

March 24th 2004.

Table of Contents

I. Introduction

I should start by acknowledging that I started with the UMass Statistical Computing Center's SAS/WIN Workshop sas8.pdf from December 6, 1999 and used it as the basis for this document, but they are not the same document and the original http://www.math.umass.edu/%7Escc/software/handouts/sas8.pdf would be worth reading to fill in any gaps I have created in this version.

SAS Availability and Cost

Any student, staff or faculty at UMASS may purchase a license for SAS/WIN[1] from OIT, Lederle Bldg. (A118) (8:30 a.m. to 5:00 p.m. weekdays). The cost of the license is an initial fee of $100 for student or $200 for faculty/staff (plus $35 for the CD) and then an annual fee[2] of $40.

Check out http://www.oit.umass.edu/hss/software/site.html for details.[3]

SAS/WIN is available in all OIT/PCCO classrooms and labs on campus. You need an OIT account to use these labs.

http://www-pcco.oit.umass.edu/utilities/software.asp

Check out the Web page on http://www-pcco.oit.umass.edu/ for details of the labs' locations, hours and availability.

(http://www-pcco.oit.umass.edu/areas/areahours.asp).

Hardware and Software Requirements

SAS Version 8 requires an Intel or Intel-compatible Pentium class processor machine running Windows 95 or 98 (Windows 95 must be updated with Year 2000 updates provided by Microsoft. Windows 98 must be updated[4] with the planned Service Pack 1 or to Windows 98, Second Edition.), or Windows NT

---

[1] The manufacturers of SAS and SYSTAT no longer support the Macintosh versions of their software. OIT can distribute the last available version of each (6.12 and 5.2.1, respectively), but there are no upgrades or bug-fixes planned.

[2] SAS Licenses from UMASS run from Jul 1st to June 30th.

[3] PC Classroom Operations (PCCO) provides computer classrooms for faculty using computer technology in their coursework. Students, faculty and staff may use the computer facilities for homework (or other University-related work) when a class is not scheduled.

[4] http://v4.windowsupdate.microsoft.com/en/default.asp or similar will get you to the right place at Microsoft for these updates. The Web page link may be towards the top of the list when you select the Start menu.

(Version 4.0 updated with Service Pack 4 or higher) or Windows 2000. The memory requirements are: 32 MB minimum for Windows and Windows NT. A mouse and math coprocessor are assumed to be present.

Distribution is by CD. There are three CDs. CD1, CD2 and an OnlineDoc* CD.

On installation, when it asks you for the GIS CD it means CD2. Make sure that you wait for your system to recognize the CD before clicking on the OK Button.

You will get a print out of what your SETINIT.SAS file should look like with your CDs. This file contains your licensing information without which SAS will not run. This file will be in the following location if you accepted the defaults during installation.

C:\Program Files\SAS Institute\SAS\V8\\CORE\SASINST\SETINIT.SAS

When editing the file, make sure you go character by character through the text, because it won't just be dates and passwords that will change.

Documentation

The CD OnlineDoc* contains complete reference documentation for SAS. You may access this by opening the following file using a Web browser.

CDROM:/sasdoc/sashtml/main.htm

The SAS Institute has a publication Web page with stuff that may be useful to you at:

http://www.sas.com/

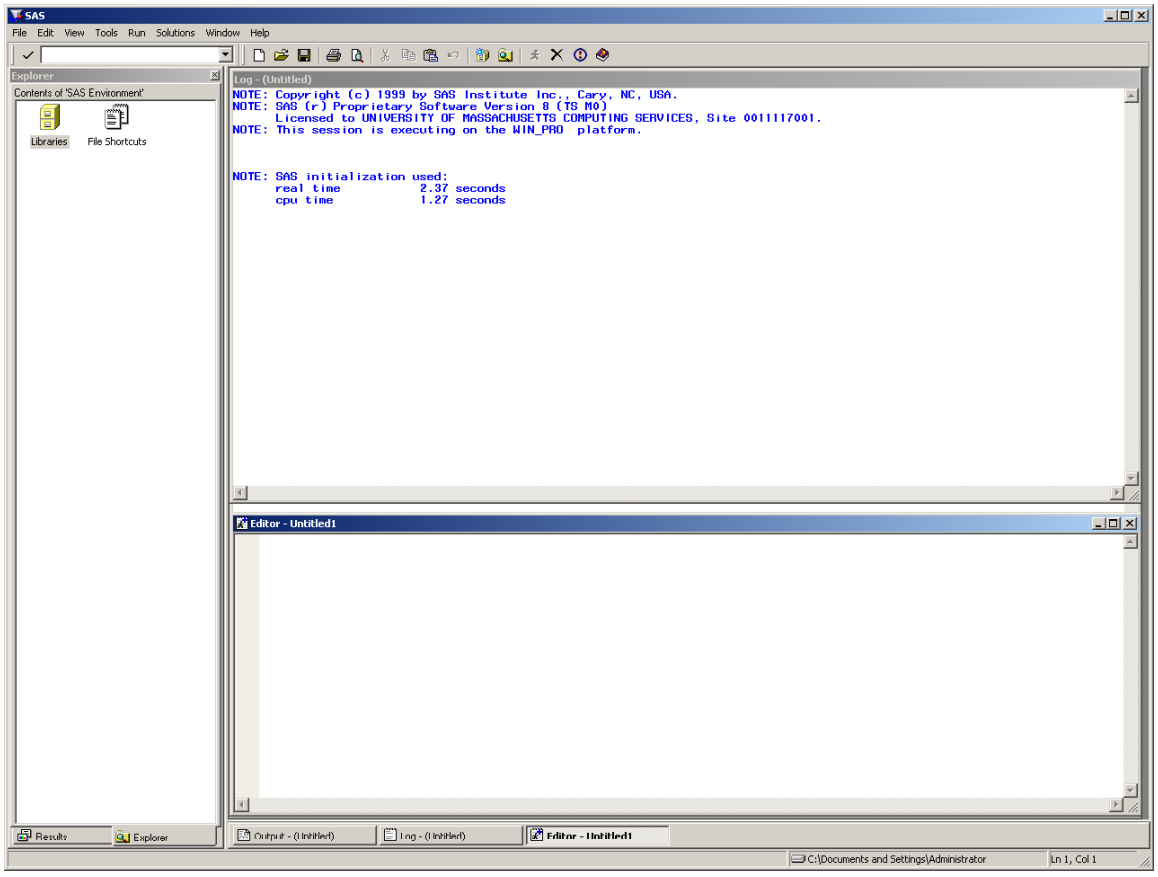
II. SAS Windowing Environment

SAS provides a number of windows for accomplishing all the basic SAS tasks. Note that SAS is very context sensitive and you need to be aware of which of the SAS windows is the active one when you issue commands from the pull down menus, icons etc…

When you first start SAS software, five main SAS windows are open, with three of them tiled and ***visible*:**

- ***Explorer****
- Results
- ***Editor****
- ***Log*,*** and
- Output.

Your screen will look as follows:



In the ***Explorer*** window, you can view and manage your SAS files and create shortcuts to non-SAS files. Use this window to create new libraries and SAS files; open any SAS file; and perform most file management tasks such as moving, copying, and deleting files.

In the ***Program Editor*** window, you enter, edit, and submit SAS programs. To open your SAS programs, click in the Program Editor window then from the File menu select the open option to select your SAS program. The Log window displays messages about your SAS session and any SAS programs you submit.

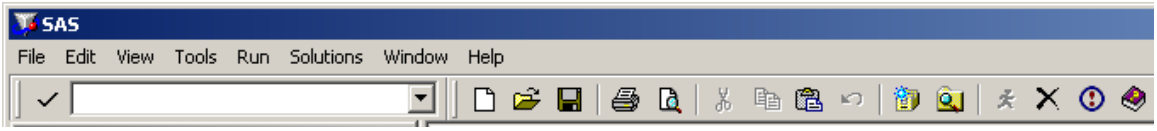
In the ***Output*** window, you can browse output from SAS programs you submit. By default, the Output window is positioned behind the Program Editor and Log windows. When you create output, the Output window automatically moves to the front of your display.

The ***Results*** window helps you navigate and manage output from SAS programs you submit. You can view, save, and print individual items of output. By default, the Results window is positioned behind the Explorer window and is empty until you submit a SAS program that creates output. Then it moves to the front of your display.

The ***Log*** window allows you to see what SAS thought was going on when it ran your programs and contains useful error messages that will help you to know where you went wrong!

SAS Commands

There are SAS commands for performing a variety of tasks. You have up to three ways to issue commands: menus, the tool bar, or the SAS command bar (or command line). The following figure shows the location of these three methods of issuing SAS commands in the Windows operating environment default view:



Menus

Pull-down menus are located at the top of your screen. The choices in the menu are: File, Edit, View, Tools, etc. The menus will change according to which window is active.

SAS command bar

The command bar, directly below the menu, is a place that you can type in SAS commands. Most of the commands that you can type in the command bar are also accessible through the pull-down menus or the tool bar.

Tool Bar

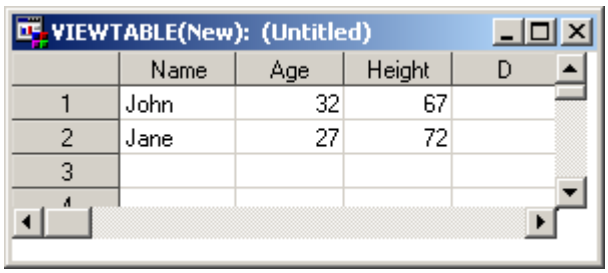
The tool bar is to the right of the command bar and gives you quick access to commands that are already accessible through the pull-down menus.

The Viewtable Window for entering data

This has an unpleasant interface, which requires too much fiddling about, as far as I am concerned, and so I recommend that if you want to type in data sets you do so in Excel and then load them into SAS. However…

Besides the 5 main windows mentioned above, there are many other SAS windows available for various tasks. The Viewtable window is a way to create new data sets or **browse** and edit existing data sets. The Viewtable window displays tables (another name for data sets) in a tabular format. To open the Viewtable window, select **Table Editor** from the **TOOLS** menu. An empty Viewtable window will appear.

This table contains no data. Instead you see rows (or observations) labeled with numbers and columns (or variables) labeled with letters. We will now enter the small dataset given in the following Viewtable window. First we will give the columns more meaningful names by clicking on the column heading for A and typing the word name, then tab to column heading B and type height, and so forth. Once we have typed in the appropriate column headings (variable names), we will type in the data as in the following Viewtable window. SAS will automatically figure out if your columns are numeric or character based on the first row of data.



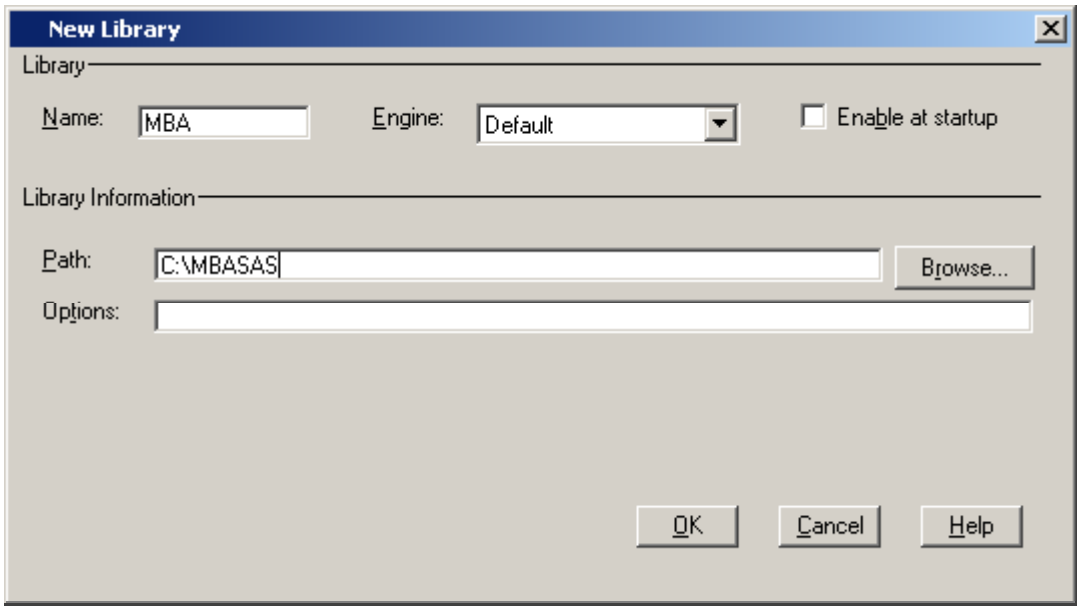
Saving your data

Once you have typed in your data and are satisfied with the accuracy you should save your data. From the **File** menu, select **Save As**. From the Save As dialog box, select a library and then specify the member name of your data (or table). **Libraries** are directories (folders). **Members** are files[5]. Initially, you are presented with 4 libraries that have been created by SAS and are controlled by SAS: **Maps**, **Sashelp**, **Sasuser** and **Work**.

Note that **Work** is a temporary library or workspace that exists for the duration of your SAS session but is erased at the end of each session.

If you wish to create a permanent SAS data set, then you need to first create a new library. Click on the **New Library** icon.  When the New Library dialog box appears, type in a name for the new library and select its location (path).

For this class I will use library name: MBA and path C:\MBASAS (N.B. You have to create the folder in Windows Explorer ***before*** you can declare its existence to SAS. SAS will not create the folder for you.



Then click **OK** and when you return to Save As dialog box, select **MBA** under libraries and then type in **Member Name**: "SASData1" and click **Save**.

If you look at this file from Windows Explorer it will be called SASData1.sas7bdat in C:\MBASAS .

---

[5] Whilst all this new jargon seems unnecessary, it is the result of SAS having been developed on Mainframe computers and being common to mainframes, Unix Servers and PCs.

The windows filename is SASData1.sas7bdat, but within the SAS program you may refer to this dataset by giving either the path or library name and only the first part of the name, i.e.

C:\MBASAS\ SASData1 **or** *MBA.SASData1* .

If your data needs were such that you were going to type them in yourself, you would probably use Excel or Minitab. Where SAS is very useful is in dealing with large datasets.

III. SAS language

There are menu-driven front ends to SAS; SAS/ASSIST, the Analyst, for example, which make SAS appear like a point-and-click program. However, you will have much more flexibility using SAS if you learn to write your own program using the SAS language.

A SAS program is a sequence of SAS statements executed in order that give instructions to SAS. There are very few rules for the SAS language. The most important rule is:

Every SAS statement ends with a semicolon. That does not mean the end of every line, since a statement can run over many lines.

SAS programs consist of two basic building blocks: **DATA** steps and **PROC** steps.

A typical SAS program starts with a **DATA** step to create, reference or manipulate a SAS data set.

SAS Statement	Effect
DATA MBAClass ;	Declares a *reference* to a dataset
SET **'C:\MBASAS**SASData1**'** ;	Relates the physical file to the *reference*.
Htcm=Height*2.54 ;	Declares a variable, htcm, which will be calculated on the fly.
PROC print data=MBAClass;	Uses the SAS procedure "print" on the data set referenced by MBAClass.
RUN;	Declares the end of the program.

We can refer to the permanent SAS dataset that we are using to create the new one in two ways: **specify the complete location** where the dataset is stored as above or use the **shortcut name** that we created earlier to refer to this location as below

SET **MBA.**SASData1 ;	Relates the physical file to the *reference*.

If we fail to give the location or use the shortcut name then by default SAS will look for this dataset in the temporary library called WORK[6].

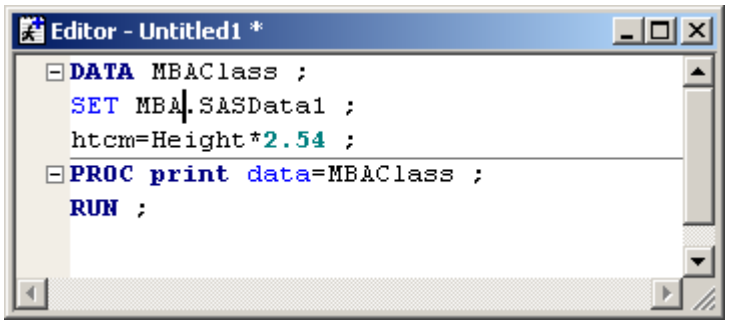
Your final statement in a SAS program must be a **RUN** statement. Note that you can write several "***programs***" and execute them one after another in a single SAS edit. Each separate "***program***" needs its own **RUN** statement or odd things will happen.

[6] The WORK library's contents are deleted when you exit SAS.

IV. Submitting a Program in SAS Windowing Environment

Getting your program in the Program Editor

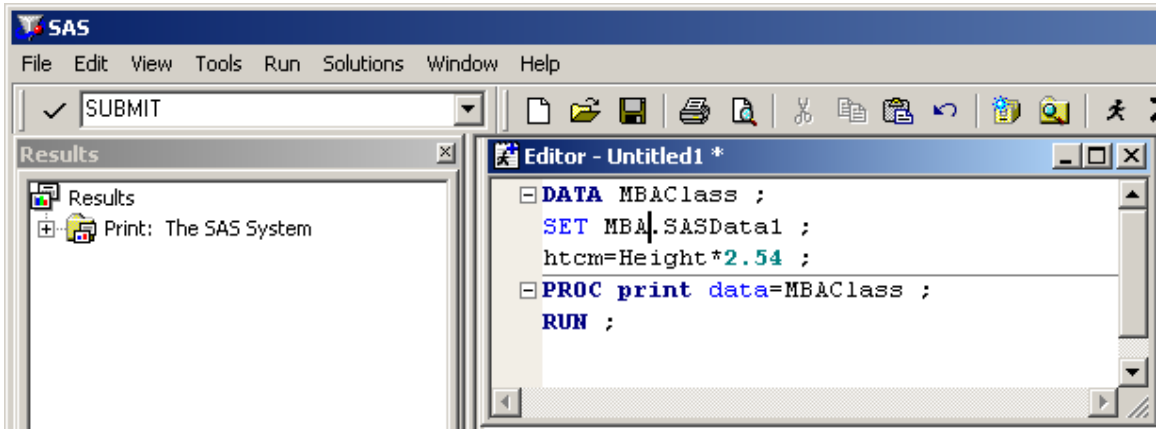
The first thing you need to do is get your program into the Program Editor window. You can either type your program into the Program Editor window, or you can bring the program into the Program Editor window from a file. Click on the Program Editor window and then type the above program into the window. Your Program Editor window should look as follows:



Submitting your program

You have 3 ways to execute the **SUBMIT** command:

1. Use the **Submit** icon on the tool bar (the runner figure)  .
2. Make the Program Editor window active and enter **SUBMIT** in the command line



3. or make the Program Editor window active and select **Submit** from the **RUN** pull-down menu.

Getting your program back

Unfortunately for most of us, our programs do not run perfectly every time. If you have an error in your program, you will most likely want to edit the program and run it again. To get your program back in the `Program Editor` window, use the **`RECALL`** command. You can: either make the `Program Editor` the active window, then enter **`Recall`** in the command line; or select **`Recall Last Submit`** from the **`RUN`** pull-down menu.

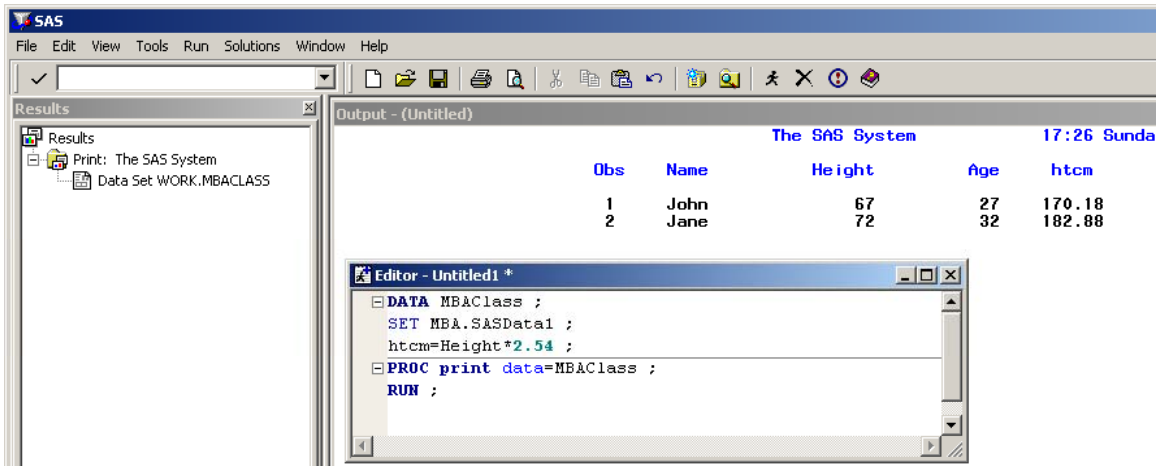
The **`RECALL`** command will bring back the last block of statements you submitted. If you use the **`RECALL`** command again, it will insert the block of statement submitted before the last one, and so on, until it retrieves all the statements you submitted.

I think that you are better served by explicitly saving your program, **BEFORE** you "***submit***" it for two reasons.

1. It is easier to simply reopen the program in the Editor than to recall it, and
2. Sometimes SAS will recall the program it created from the program you submitted, rather than the program that you actually submitted. Sounds complicated, but we will see examples later on.

Viewing the SAS Log, Results and Output

After you submit your program, the results of your program go into the Log and Output windows. If your program produced any output, then you will also get new entries in your Results window. The Results window is like a table of contents for your SAS output. The following figure is an example of what your screen might look like after you submit a program.



Printing or saving the contents of the Output window

If you want to print or save the entire contents of the Output window, first activate the Output window by clicking in it, then select either **Print** or **Save As** from the **FILE** pull-down menu.

The Results window

When you have a lot of output, the Results window can be very helpful. The Results window is like a table of contents for your output. It lists each procedure that produces output, and if you open, or expand, the procedure in the Results tree, you can see each part of the procedure output. You may also print or save each part.

V. Getting data into SAS

Before you can analyze your data with SAS, your data must be in a form that SAS can read.

SAS can read from ASCII files, delimited or fixed length records, Excel and Access Database tables amongst others. (In a work setting you might see Oracle or other DBMS and SAS can connect to those too!!).

From Excel

If you want a specific excel range of data name the range in excel and then write and submit a SAS program like the one that follows:

SAS Statement	Effect
`PROC IMPORT`	`Invokes the SAS Procedure to Import data from external file`
`OUT=MBA.class`	`Specifies the Name of the SAS Dataset name by which the data will be referenced.`
`DATAFILE="F:\ExcelToSAS.xls"`	`Specifies the name and location of the file to be imported.`
`DBMS=EXCEL2000 REPLACE ;`	`Specifies the type of data file from which the data will be imported.`
`SHEET="Sheet1" ;`	`Specifies the sheet within the excel file containing the data.`
`GETNAMES=YES ;`	`Specifies that the first row contains column names.`
`RUN;`	
`PROC PRINT DATA=MBA.class ;`	`Prints data to the Output window so we can see something happened.`
`RUN;`	

Notes:

- `PROC IMPORT OUT=MBA.class DATAFILE= "F:\ExcelToSAS.xls" DBMS=EXCEL2000 REPLACE ;` is **ONE** statement, hence the single semi-colon at the end of the word REPLACE.
- To import a named range from excel use the command: `RANGE="NameOfNamedRange";` in place of `SHEET="NameOfSheet";`
- REPLACE makes SAS overwrite an existing SAS data set and is optional. If you do not specify REPLACE, PROC IMPORT does not overwrite an existing data set.
- GETNAMES. If your excel file has text in one of the columns, this feature does not work. If you have no column names, then use "`GETNAMES=NO;`"

From Access

Write your query in Access to isolate the data you are interested in or prepare your table. Make notes of the names of the query or the table and the physical location of the Access database, then write and submit a SAS program like the one that follows:

SAS Statement	Effect
`PROC IMPORT`	Invokes the SAS Procedure to Import data from external file
`Table="SF10037"`	Specifies the name of the table or query in the Access Database to be imported.
`OUT=MBA.Table37`	Specifies the Name of the SAS Dataset name by which the data will be referenced.
`DBMS=ACCESS REPLACE ;`	Specifies the type of data file from which the data will be imported.
`DATABASE="F:\MA2000SF1Table37.mdb";`	Specifies the name and location of the Access Database from which the Query or Table is to be imported.
`GETNAMES=YES ;`	Specifies that the first row contains column names.
`RUN;`	
`PROC PRINT DATA=MBA.Table37 (obs=20);`	Prints 20 records from the data to the Output window so we can see something happened. (Note with large files, printing all observations can create problems.
`RUN;`	

Notes:

- The example uses table 37 or the 2000 Census SF1 file for Massachusetts, which contains 130,064 records.
- Your database may be have security and passwords associated with it and there are statements by which these can be incorporated into your program. Refer to the SAS Online documentation.
- `PROC IMPORT TABLE="SF10037" OUT=MBA.Table37 DBMS=ACCESS REPLACE ;` Is **ONE** statement, hence the single semi-colon at the end of the word REPLACE.
- GETNAMES. If your excel file has text in one of the columns, this feature does not work. If you have no column names, then use "GETNAMES=NO;"

From Delimited ASCII files

The Census for 2000 is downloadable in comma delimited ASCII files, one for each table for each state, so you could write and submit a SAS program like the one that follows:

SAS Statement	Effect
`PROC IMPORT`	`Invokes the SAS Procedure to Import data from external file`
`DATAFILE="F:\SF10037.csv"`	`Specifies the name of the table or query in the Access Database to be imported.`
`OUT=MBA.SF10037.csv`	`Specifies the Name of the SAS Dataset name by which the data will be referenced.`
`DBMS=dlm`	`Specifies the type of data file from which the data will be imported.`
`REPLACE ;`	
`DELIMITER="," ;`	`Specifies the name and location of the Access Database from which the Query or Table is to be imported.`
`GETNAMES=YES ;`	`Specifies that the first row contains column names.`
`RUN;`	
`PROC PRINT DATA=MBA. SF10037.csv (obs=20);`	`Prints 20 records from the data to the Output window so we can see something happened. (Note with large files, printing all observations can create problems.`
`RUN;`	

Notes:

```
PROC IMPORT DATFILE=F:\”SF10037.csv” OUT=MBA.SF10037 DBMS=DLM REPLACE ;
```
is **ONE** statement, hence the single semi-colon at the end of the word REPLACE.

DELIMITERS can be textual characters like commas or ampersands. For tab delimited files specify DBMS=tab as in the following example:

```
PROC IMPORT DATAFILE="F:\TabToSAS.txt"
            OUT=MBA.TabToSAS
            DBMS=tab
            REPLACE;
   GETNAMES=yes;
RUN;

PROC PRINT DATA=MBA.TabToSAS (obs=20);;
RUN;
```

Saving your program

Now that we have written a SAS program with several lines, we will save the program for future use. From the Program Editor window, **RECALL** your program and then from the **File** menu, choose **Save As** and type the name.

Exiting the SAS system

Since we have saved the program, we can now exit the SAS program and then come back later and start up where we left off. To exit from SAS, from the **File** menu click on **Exit**.

Including a SAS program

We now wish to continue where we left off before exiting the SAS system. We can do this: start SAS again and bring the program that we just saved into the Program Editor window. Then submit it to the system for processing: from the **File** pull-down menu, select Open. When the Open dialog box appears, click on file you want and then click on **Open**. When the program appears in the Program Editor window, then you can **Submit** to get back to where you were before you exited SAS.

VI. More Procedures

Now that we have successfully imported data from a variety of file types and have created the temporary SAS datafiles, we can run some statistical procedures on the data.

Cumulative Frequencies

SAS Statement	Effect
PROC IMPORT DATAFILE="F:\TabToSAS.txt" OUT=MBA.TabToSAS DBMS=tab REPLACE; GETNAMES=yes; RUN;	Reads the data from TabToSAS.txt into the SAS dataset MBA.TabToSAS.
PROC FREQ DATA=MBA.TabToSAS;	Specifies that the FREQ procedure is to be invoked using data from the SAS dataset MBA.TabToSAS.
TABLES H001001 H002001;	Specifies the names of the variables in the SAS dataset for which cumulative frequency tables are wanted.
RUN;	

Descriptive Statistics

SAS Statement	Effect
PROC IMPORT DATAFILE="F:\SF10037.csv" OUT=MBA.SF10037CSV DBMS=dlm REPLACE; DELIMITER=','; GETNAMES=yes; RUN;	**Reads the data from SF10037.csv into the SAS dataset MBA.Sfoo37CSV.**
PROC MEANS DATA=MBA.SF10037CSV;	Specifies that the MEANS procedure is to be invoked using data from the SAS dataset MBA.SF10037CSV.
VAR H001001 H002001 H003001 H004001 H005001 H006001 H007001 H008001;	Specifies the names of the variables in the SAS dataset for which descriptive statistics are wanted.
RUN;	

Notes:

- If you omit the VAR statement SAS will calculate the statistics for all of the numeric variables in the dataset.

Covariance and Correlation

SAS Statement	Effect

`PROC IMPORT DATAFILE="F:\SF10037.csv"`
`OUT=MBA.SF10037CSV`
`DBMS=dlm`
`REPLACE;`
`DELIMITER=',';`
`GETNAMES=yes;`
`RUN;`
	Reads the data from SF10037.csv into the SAS dataset MBA.Sfoo37CSV.

`PROC CORR`	Specifies that the CORR procedure is to be invoked.
`COV`	Specifies the Covariance Optional calculation
`DATA=MBA.SF10037CSV;`	Specifies the SAS dataset to be used. (MBA.SF10037CSV).
`RUN;`	

Notes:

- The statement `PROC CORR COV DATA=MBA.SF10037CSV;` is one statement.

The DO Statement

SAS has a do statement as you can see from the following example:

SAS Statement	Effect
`data andrew;`	**Declares the SAS dataset andrew which by default will be in the WORK library.**
`m = 1000;` `n = 5;`	Declares 2 variables and assigns values to them.

`do i to m;`
`sum = 0;`
`do j = 1 to n;`
`x= sqrt(20)* normal(0) + 1;`
`sum = sum + x;`
`end;`
`end`
	Includes two nested do statements.
	Sqrt(argument) returns the square root of the argument.
	Normal(0)returns a pseudo-random number having a normal distribution with a mean of 0 and a standard deviation of 1

`RUN;`	

More on the Results window

Now that we have run more procedures, we will revisit the Results windows to explore its uses. The following output shows what your Results window might look like after running the above procedures plus the previous print:

There is one entry in the window for each procedure. But note the plus sign (**+**) to the left of each procedure. You can expand the results of each procedure by clicking on its plus sign.

Using the Results window it is now possible to print or save just the parts of the output you want. First highlight the item you want in the Results window, then click on the right mouse button. Then select either **Print** or **Save As** from the pop-up menu. You may also print from the **File** pull-down menu once you highlight the output you want.

VII. Manipulating your Data

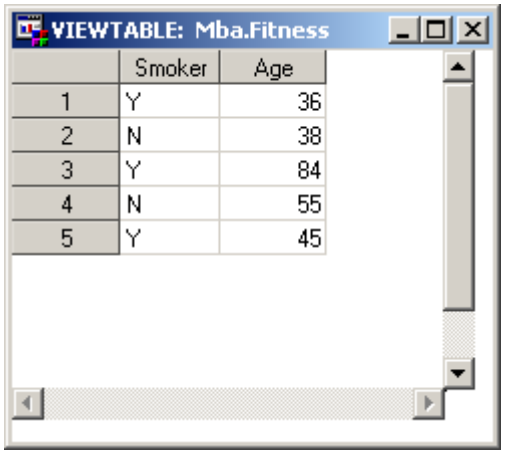
Calculations

We have seen that we can create variables and perform calculations such as the conversion of inches into centimeters.

We have seen that you can generate normally distributed random values.

Creating a permanent SAS dataset from a temporary dataset

Assume we have a SAS dataset called Fitness, which looks as follows:



	Smoker	Age
1	Y	36
2	N	38
3	Y	84
4	N	55
5	Y	45

I can create two SAS Datasets from "*Fitness"* and I can export the two SAS datasets as delimited ASCII files or to Excel or to Access or to other destinations.

SAS Statement	Effect
data MBA.Smoker MBA.NonSmoker;	**This declares two new SAS Datasets.**
set MBA.Fitness;	This declares the source SAS dataset.
if Smoker='Y' then output MBA.Smoker; else output MBA.NonSmoker;	This IF THEN; ELSE; statement directs records from MBA.Fitness to MBA.Smoker or MBA.NonSmoker according to the value in the Smoker field of MBA.Fitness.
run;	
proc print data=MBA.Smoker; **run;** **Proc print** data=MBA.NonSmoker; **run;**	**Print out the two new SAS Datasets to be sure that we achieved our objective.**
proc export data=MBA.Smoker outfile="F:/Smoker.csv" dbms=dlm; delimiter=','; **run;**	**Use the procedure EXPORT taking the SAS data set MBA.Smoker and sending the contents to the file F:\Smoker.csv which will be a delimited ASCII file delimited with commas.**
proc export data=MBA.NonSmoker outfile="F:/NonSmoker.csv" dbms=dlm; delimiter=','; **run;**	**Use the procedure EXPORT taking the SAS data set MBA.NonSmoker and sending the contents to the file F:\NonSmoker.csv which will be a delimited ASCII file delimited with commas.**

LIBNAME statement

One method for creating a new library is to use the **LIBNAME** statement in a SAS program. If you type in the following SAS statements into the Program Editor window and submit, you are creating a library called **IN** that points to the directory c:\windows\desktop\saswork.

libname in 'c:\windows\desktop\saswork'; run;

VIII. Working with SAS Dates

Dates can be tricky to work with. Some months have 30 days, some 31, some 28 and don't forget leap year. SAS dates simplify all this. A SAS date is the number of days since January 1, 1960. The table below lists four dates and their values as SAS dates:

Date SAS date value

January 1, 1959 -365

January 1, 1960 0

January 1, 1961 366

January 1, 2001 14976

SAS has special tools for working with dates: **informats** for reading dates, **functions** for manipulating dates, and **formats** for printing dates. You may refer to the SAS Language manual or SAS online help for a list of date **informats**, **formats** and **functions**.

Informats

To read variables that are dates, you use formatted style input with a date **informat**.

The **INPUT** statement below tells SAS to read a variable named *BDAY* using the MMDDYY10. informat:

INPUT bday mmddyy10.;

SAS has a variety of date informats for reading dates in many different forms. All of these informats convert your data to a number equal to the number of days since January 1, 1960. Refer to the documentation for a complete list of all possible informats.

Dates in SAS expressions

Once a variable has been read with a SAS date informat, it can be used in arithmetic expressions like other numeric variables. For example, if a library book is due in three weeks, you could find the due date by adding 21 days to the date it was checked out:

datedue = datechek + 21;

Functions

SAS date functions perform a number of handy operations. For example, the **TODAY** function returns a SAS date value equal to today's date. The statement below uses the TODAY function to compute a person's current age in years:

Age=(TODAY() – bday)/365.25;

Formats

If you print a SAS date value, SAS will by default print the actual value (the number of days since January 1, 1960). Since this is not very meaningful to most people, SAS has a variety of formats for printing dates in different forms. Refer to SAS documentation for a complete list of possible formats. The **FORMAT** statement below tells SAS to print the variable *BDAY* using the WEEKDATE17.Format:

FORMAT bday weekdate17.;

An example using SAS Dates

We have a small data file that contains a person's last name and their date of birth. We would like to compute their age today and print out the results. Below is the program we would use:

We use the informat mmddyy10. to read the persons date of birth. This will read data information in the form of **mmddyy** form, where mm, dd, and yy are integers representing the month, day, and year.

The month, day, and year fields can be separated by blanks or special characters. The 10. informs SAS of the total width of the date value. If you use the value of 10 you will have room to enter year as a 4 digit value. If you use the value of 8, you will have to enter year as a 2 digit value which we wouldn't recommend.

We then calculate age by using the **TODAY()** function to calculate today's date and then subtract from that the person's date of birth. Finally, we divide by 365.25 to get their age in years.

We then print the data. In order that *BDAY* will be printed in a form that people recognize , we need to use the format statement with the **PROC PRINT**. We use the same format as we did informat so we can see the date of birth printed as mm/dd/yyyy. Since we are printing *BDAY* with a format, in order to show you what its value would be as a julian date, we create a another variable (julbday=bday) so we can print this without a format.

Appendix A

Example code for reading fixed length records files into a SAS dataset.

```
data a; infile 'c:\pums\pumsasc3.dat' lrecl=147;
input serno 1-7 state 8-9 faminc 10-16 child 17-18 mrace 19 mage 20-21
mweeks 22-23 mhours 24-25 mearn 26-31 wrace 32 wage 33-34
wweeks 35-36 whours 37-38 wearn 39-44 magesq 45-48 wagesq 49-52
group1 53 group2 54 group3 55 mstud 56 wstud 57 mhs 58 mcol 59
whs 60 wcol 61 mvetdum 62 wvetdum 63 mdis1 64 mdis2 65
wdis1 66 wdis2 67 woccserv 68 woccfrm 69
mwork 70 wwork 71 extra 72-77 mindag 78 mindcon 79 mindmfg 80
mindtcu 81 mindrtrd 82 mindfire 83 mindbsrv 84  mindprof 85
mindpub 86 mindmil 87
windag 88 windcon 89 windmfg 90 windtcu 91 windrtrd 92 windfire 93
windbsrv 94 windprof 95 windpub 96 windmil 97 south 98
own1 99 own2 100 group4 101
lmearn 102-108 .4 lwearn 109-115 .4 weduc 116-117 meduc 118-119
woccprod 120 woccoper 121
mindwtrd 122 mindpsrv 123 mindent 124 mindmin 125
windwtrd 126 windpsrv 127 windent 128 windmin 129
moccmgr 130 mocctech 131 moccserv 132 moccfrm 133 moccprod 134
moccoper 135 woccmgr 136 wocctech 137 mocc 138-140 wocc 141-143
meduc2 144-145 weduc2 146-147;
mtothrs=mweeks*mhours;  mtothrs=mtothrs+1; mleisure=1/mtothrs;
wtothrs=wweeks*whours;  wtothrs=wtothrs+1; wleisure=1/wtothrs;
run;
```

Notes:

pumsasc3.dat file has fixed length records of length (lrecl=147)and contains the following variables in the following positions in the record:

serno 1-7 state 8-9 faminc 10-16 child 17-18 mrace 19 mage 20-21 mweeks 22-23 mhours 24-25 mearn 26-31 wrace 32 wage 33-34 wweeks 35-36 whours 37-38 wearn 39-44 magesq 45-48 wagesq 49-52 group1 53 group2 54 group3 55 mstud 56 wstud 57 mhs 58 mcol 59 whs 60 wcol 61 mvetdum 62 wvetdum 63 mdis1 64 mdis2 65 wdis1 66 wdis2 67 woccserv 68 woccfrm 69 mwork 70 wwork 71 extra 72-77 mindag 78 mindcon 79 mindmfg 80 mindtcu 81 mindrtrd 82 mindfire 83 mindbsrv 84 mindprof 85 mindpub 86 mindmil 87 windag 88 windcon 89 windmfg 90 windtcu 91 windrtrd 92 windfire 93 windbsrv 94 windprof 95 windpub 96 windmil 97 south 98 own1 99 own2 100 group4 101 lmearn 102-108 .4 lwearn 109-115 .4 weduc 116-117 meduc 118-119 woccprod 120 woccoper 121 mindwtrd 122 mindpsrv 123 mindent 124 mindmin 125 windwtrd 126 windpsrv 127 windent 128 windmin 129 moccmgr 130 mocctech 131 moccserv 132 moccfrm 133 moccprod 134 moccoper 135 woccmgr 136 wocctech 137 mocc 138-140 wocc 141-143 meduc2 144-145 weduc2 146-147;

And 6 new variables were generated:

Mtothrs, mtothrs, mleisure, wtothrs, wtothrs, wleisure

Appendix B.

Example code for a Normal Probability Plot.

```
options pagesize=60 linesize=80;
data a;
infile 'a:t1-3.dat' ;
input x1 SolarRad x3 x4 x5 x6 x7;
proc univariate data=a NORMAL PLOT;
   TITLE 'Normal Probability Plot for Solar Radiation';
   VAR SolarRad ;
run;
```