

# Overview of SAS System

**Neelesh Singh**

# Module Content

- SAS Introduction
- SAS Installation
- SAS Components and Products
- Components of a SAS Program
- Data and Proc Steps
- Observation and Variables
- About SAS datasets
- Compiling and executing SAS programs

# What is SAS

- Its stands for “*Statistical Analysis Software*”
- It was very popular language before Open Sources tools like Python and R.
- It was created in 1960 and is owned by The SAS Institute
- Its platform independent and you can run it on Windows, Mac, Unix and Linux.
- Integrated system of software products.
- Most of large and older organizations have SAS implemented for one of more areas.
- SAS is used for:
  - *Data Entry, Retrieval, and Management.*
  - *Report Writing and Creating Graphics.*
  - *Statistical and Mathematical Analysis.*
  - *Business Forecasting and Decision Support.*
  - *Operations Research and Project Management.*
  - *Web Applications.*

# SAS Installation

- *SAS is distributed in digital (downloadable) as well as in DVDs*
- *Windows users use setup.exe to start the deployment wizard and Unix users use Setup.sh.*
- *University edition for students*
- *Organisations use PC SAS or unix SAS or Both*
- *SAS Licenses*
- *If you want to know which products are available in your system with the validate date:*

**PROC SETINIT;**

**Run;**

Each statement ends at Semicolon

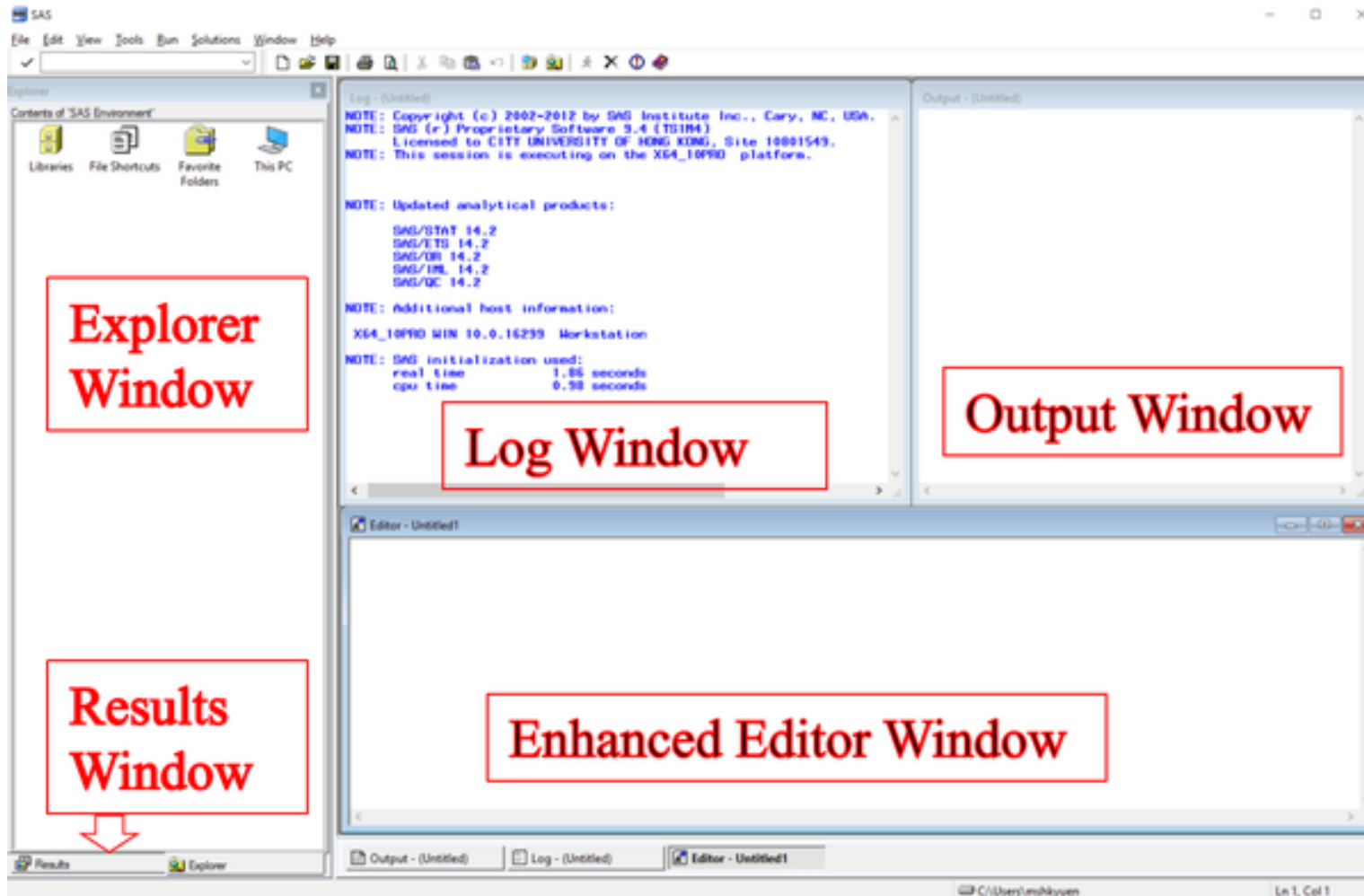
RUN statement executes previous DATA or PROC Step.

# Components of SAS


- Base SAS.
- STAT / GRAPH / OR / FSP / AF / IML / ASSIST / QC /  
CONNECT / INSIGHT / EIS / ETS / MDDDB Server / ACC-PC
- Product for industries  
([https://www.sas.com/en\\_in/software/all-products.html#a-z](https://www.sas.com/en_in/software/all-products.html#a-z))
- Enterprise Miner
- SAS Viya (Dashboarding Solution)

This training program will focus mainly on Base SAS.

# Running the SAS



To Run in PC SAS,

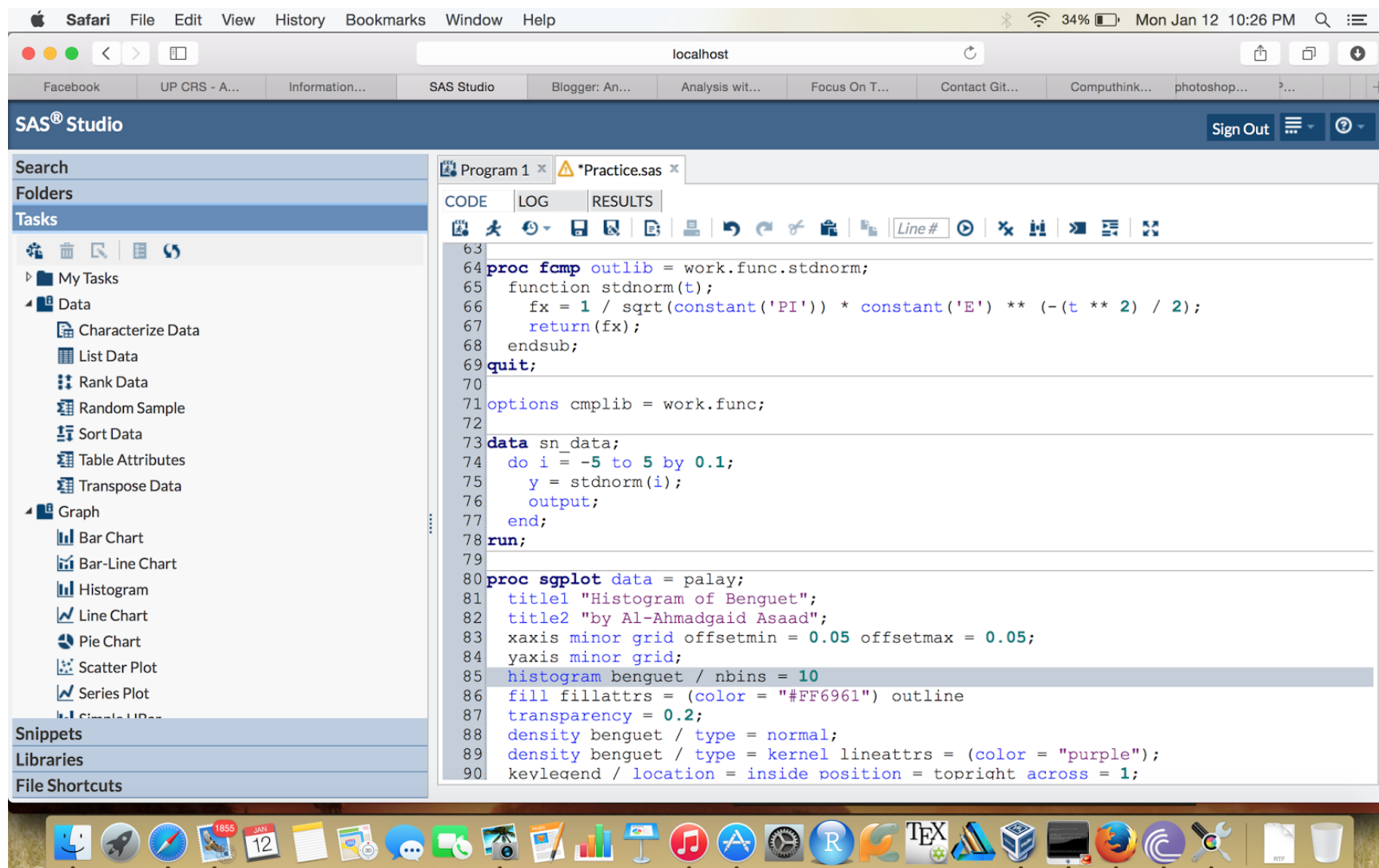
- Press 
- Use Run > Submit
- F3 or F8

- DATA and PROC steps we write in program are in **black**
- notes that SAS System report to you are in **blue**
- SAS gives you warning in **green text**
- Errors that cause SAS to Stop are in **red**

To Run in Unix

```
sas /sas/code/test1.sas -log  
/sas/code/log/test1.log
```

# SAS Studio – University Edition



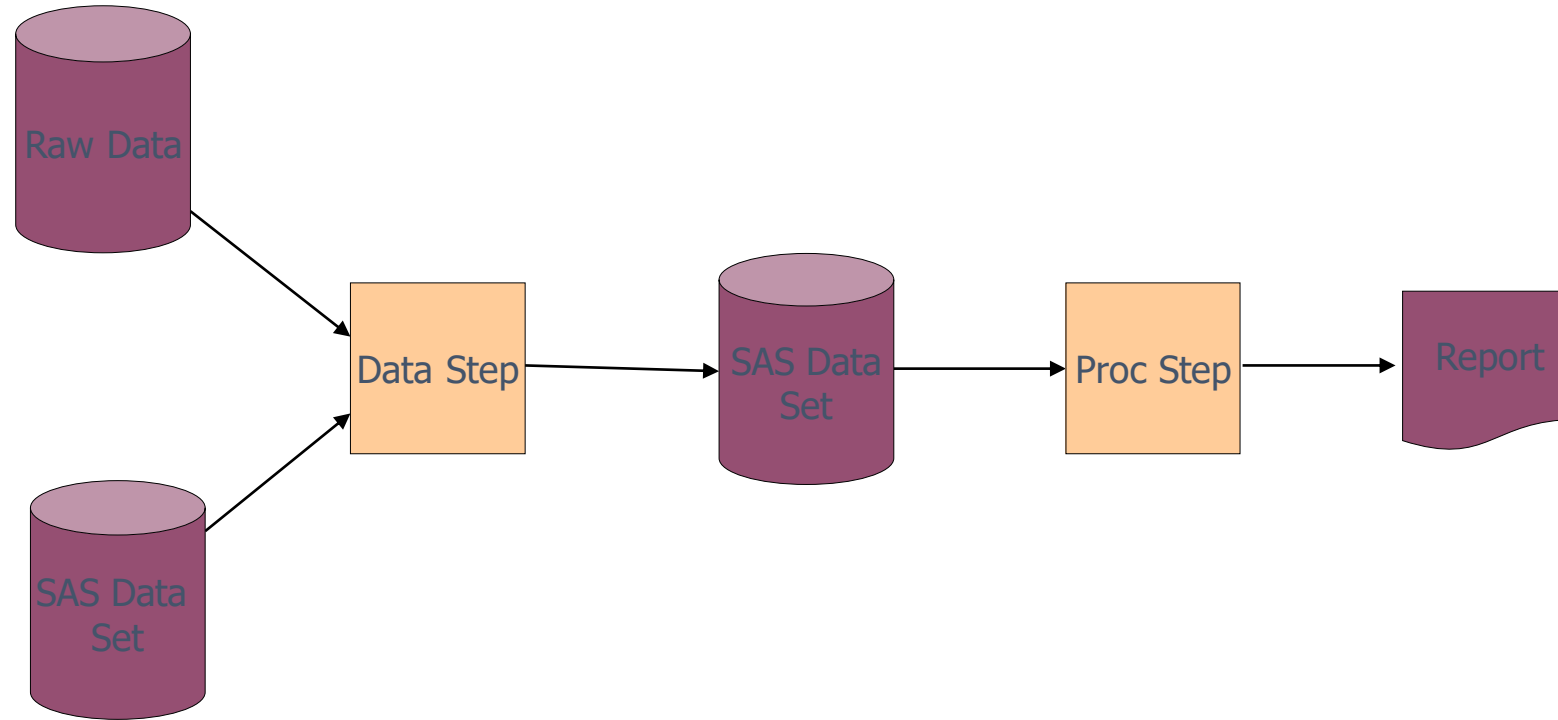
# The Base SAS Software

The Base SAS Software consists of

- SAS Language
- SAS Procedures
- Macro Facility
- Data Step Debugger
- Output Delivery System (ODS)



# Components of a SAS Program



# Components of a SAS Program

- Data Step

This step creates a SAS data set by processing input data. The input data can be in the form of raw data, another SAS data set or assignment statements.

- Proc Step

This step will execute a SAS procedure, with a SAS data set as input. It will help us to analyze the data in a data set, produce reports and other results.

# SAS Data Step example

- Extension of SAS data: .sas7bdat
- Extension of SAS Code: .sas
- DATA Step in SAS is used to create a data from another data, datasets or raw file.

DATA Cars;  
SET SASHELP.cars;  
Run;

SAS Help Library

DATA Cars;  
SET SAS1.cars;  
Run;

Library created by User

Libname SAS1 '/folders/myshortcuts/SAS'

**SET Statement reads from another dataset**

# SAS PROC Example

- Procedures in SAS are very simple to use, these 'canned' ready to use procedures makes process simple and differentiate SAS from other software products.
- SAS supports four categories of procedures: 1) reporting, 2) statistical, 3) scoring, and 4) utility

```
PROC PRINT Data = Cars;
```

```
RUN;
```

# Special Procedure – PROC SQL

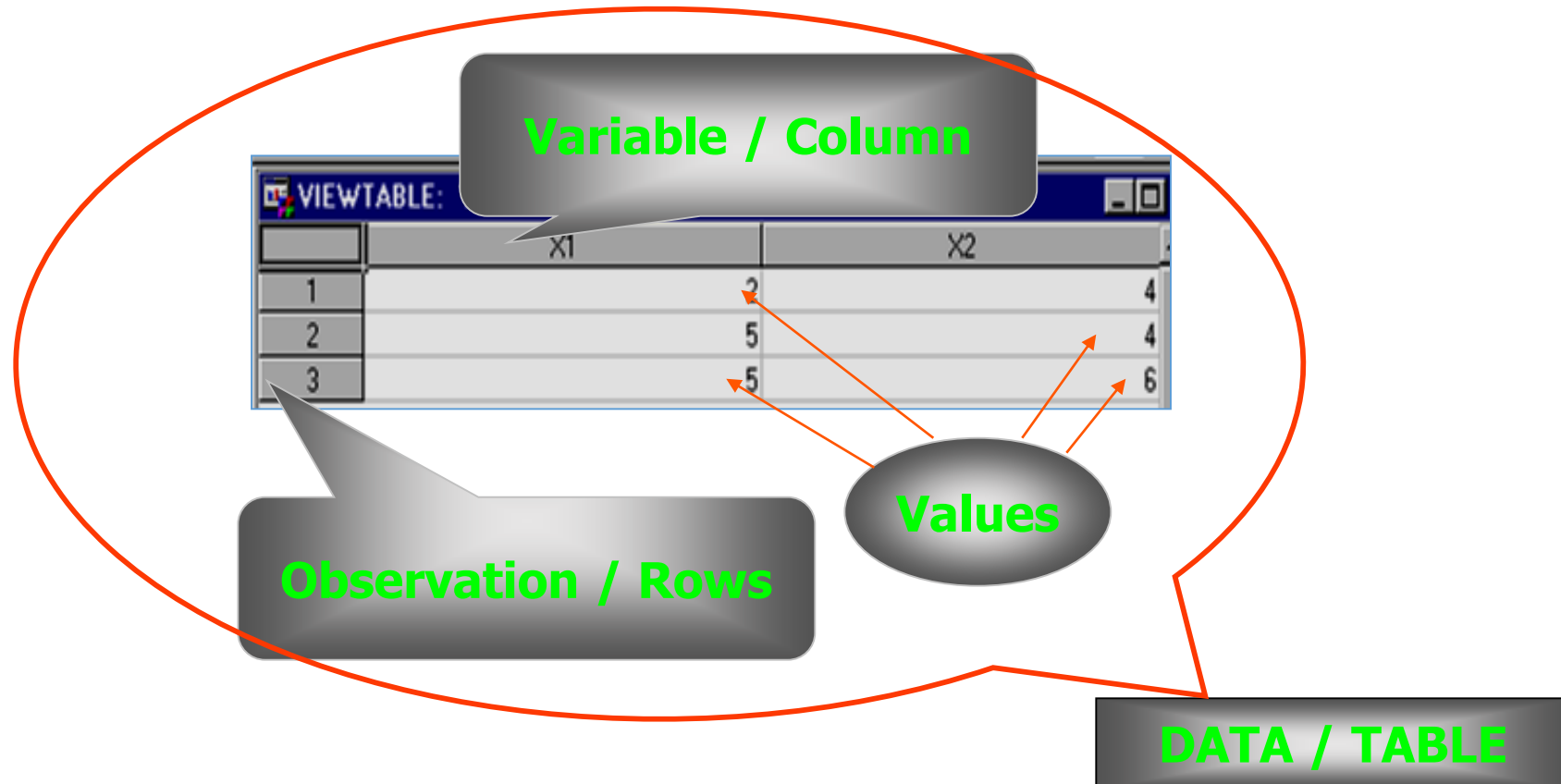
**PROC SQL** uses **SQL** to create, modify, and retrieve data from tables and views. In a way it can serve the purpose of both simple Data as well as Proc Sets

```
PROC SQL ;  
    SELECT SUM(Amt) AS Spend  
    FROM MyTable  
    WHERE Month = 'Dec' ;  
QUIT ;
```

**Data Set in SAS is TABLE in SQL...**

PROC SQL end with the  
QUIT Syntax instead of  
RUN.

# Observation and Variables



**Only 2 types of variables - Numeric and Character**

# SAS Variables

There are two types of variables

## Character

- Contains any value, letters, numbers, special characters and blanks
- Maximum length of 32,767 bytes
- One byte stores one character

## Numeric

- Stored as floating point numbers of 8 bytes by default
- 16 or 17 significant digits can be stored in these 8 bytes
- Length not restricted to 8 bytes

# Changing the variable type

- Put () and Input() functions can be used to convert from numeric to character or character to numeric
- Put () function always return character however input will return variable type numeric or character based on parameters.

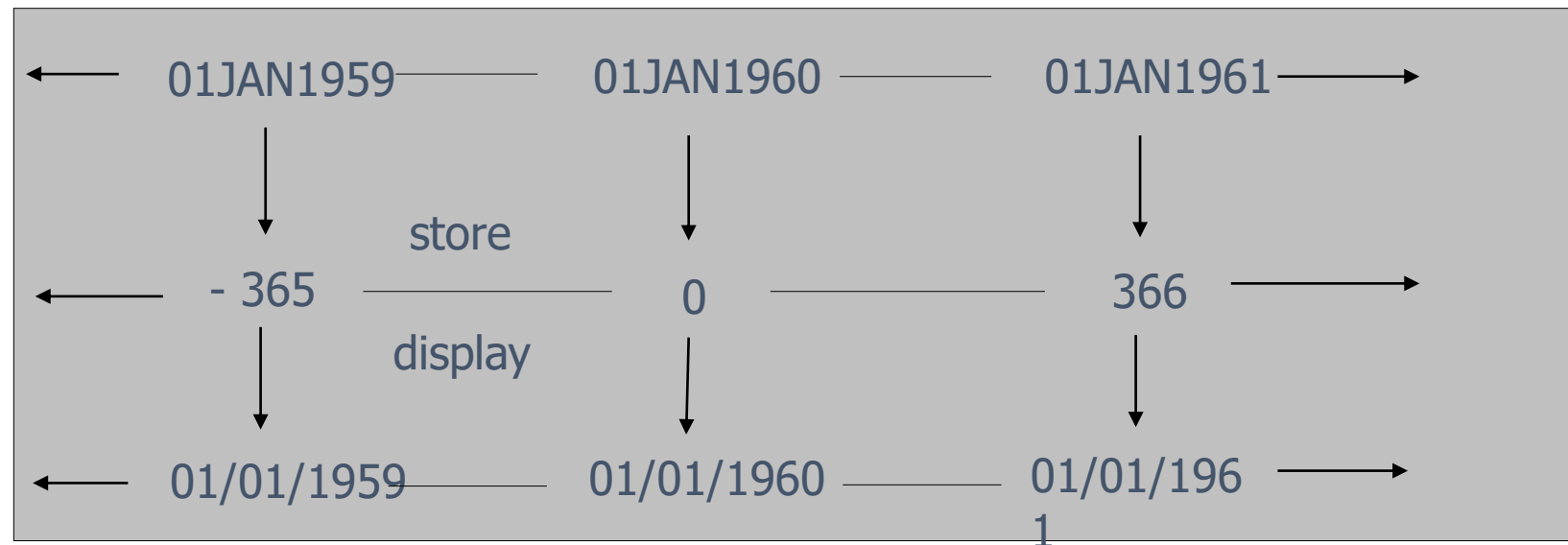
	Input Type	Input Value	Output Type	Output Value
PUT(name, \$12.);	Character	'Sumeet'	Character	'Sumeet '
PUT(age, 4.);	Number	20	Character	' 20'
INPUT(agechar, 4.);	Character	'20'	Numeric	20
INPUT(agechar, \$4.);	Character	'20'	Numeric	' 20'



# SAS Date Values

SAS stores date values as numeric values.

A SAS date value is stored as the number of days between January 1, 1960 and a specific date.



# Missing Data Values

A value must exist for every variable for each observation.

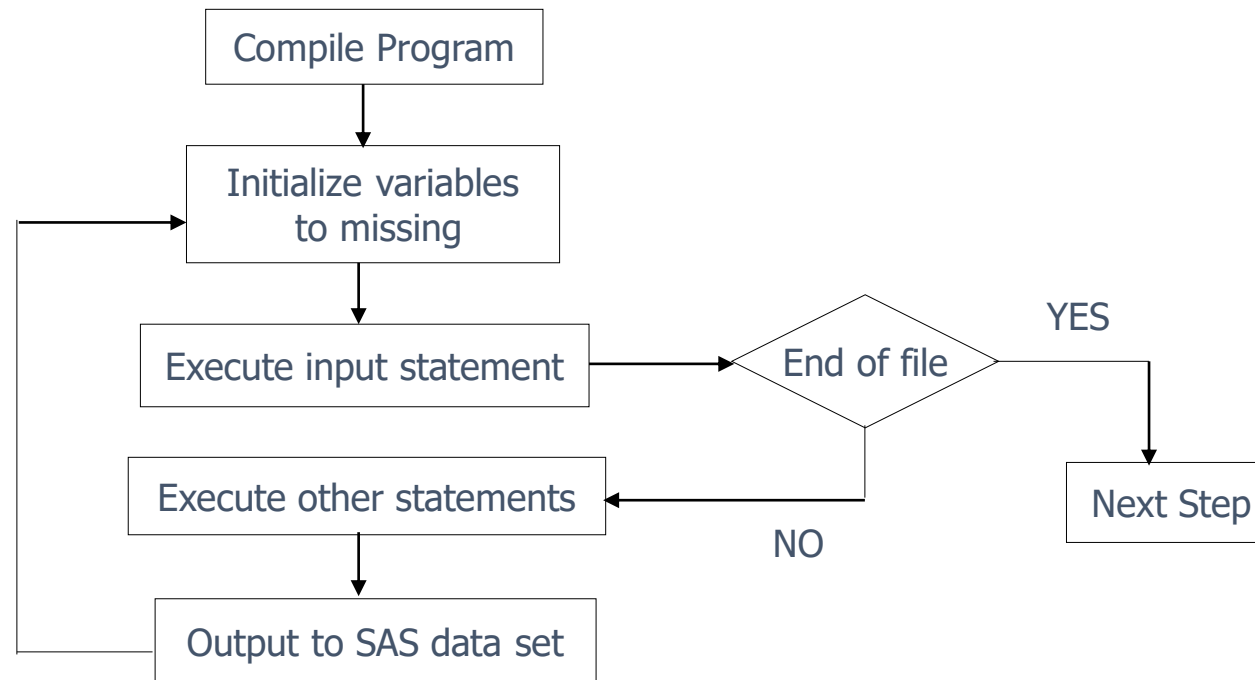
Missing values are valid values.

- A character missing value is displayed as a blank ( )
- A numeric missing value is displayed as a period (.)

# Compilation and Execution

The SAS data step is processed in two phases

- Compilation
- Execution



# Compiling the Data Step

At compile time SAS creates

- an input buffer to hold the current flat file record

a program data vector (PDV) to hold the current observation

the descriptor portion of the output data set

--	--	--	--	--	--	--	--	--	--

Lastname \$12.	Firstname \$12.	Open date mmddyy8.	Credit Limit 4.	Card Type \$10.

Lastname \$12.	Firstname \$12.	Open date mmddyy8.	Credit Limit 4.	Card Type \$10.
-------------------	--------------------	-----------------------	--------------------	--------------------

# Executing the Data Step

- The execution starts with the data statement. Every time this statement executes, the automatic variable `_n_` is incremented by 1.
- The variables in PDV are set to missing.
- The record from the flat file is read into the input buffer.
- Other statements in the step get executed for the current record.
- At the end of the step, the observation is written to the output data set, the pointer returns to the top of the data step and the variables in the PDV are reset to missing.
- The data step terminates when SAS encounters the end of the flat file.

# SAS Libraries

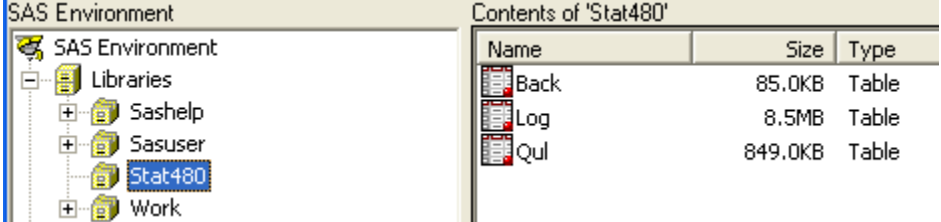
- SAS libraries are a folder located on a user's disk or shared drive which allow users to safely store things like data sets and user-defined formats.

To create a library

```
LIBNAME Stat480 'C:\Users\Bank\Data';
```

To refer a dataset:

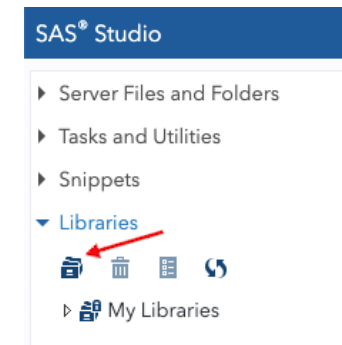
```
BANKDATA.Log
```



Contents of 'Stat480'		
Name	Size	Type
Back	85.0KB	Table
Log	8.5MB	Table
Qul	849.0KB	Table

SAS has a built-in temporary library called *Work* which stores data you are working on in your current session.

In university Edition which runs on virtual environment, we would need to create the folder using “Shared Folders” option in settings



# SAS Variables Naming Conventions

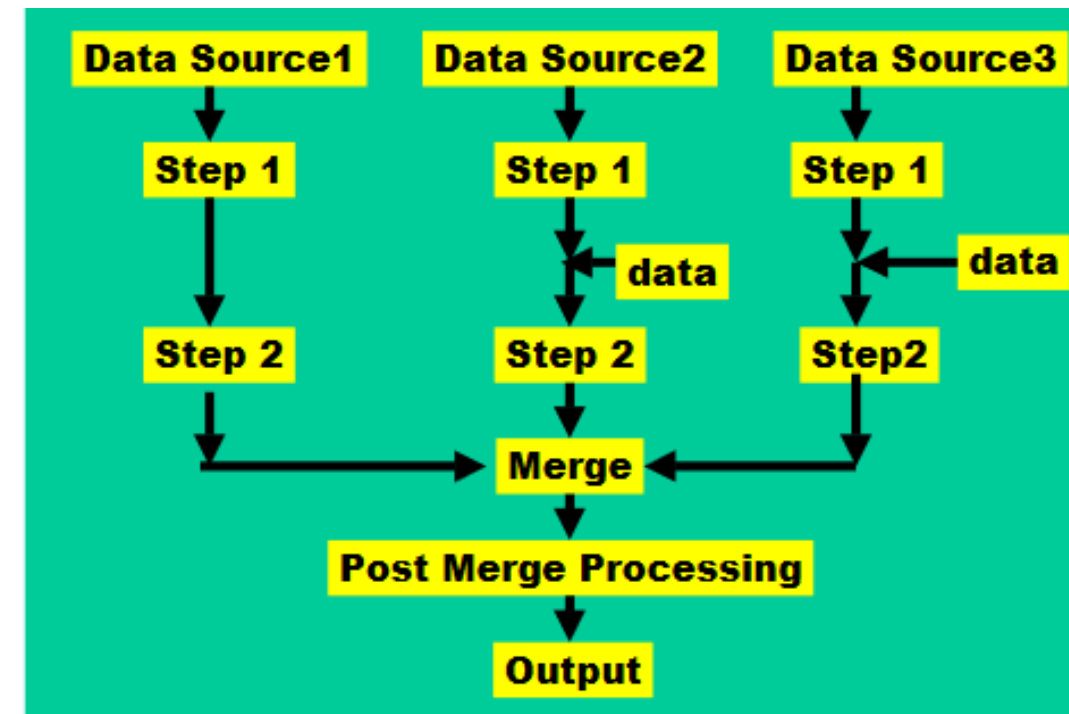
- Names can have up to 32 characters
- Names can start with letters or under score(\_)
- We can use letters and numbers
- We can't use any special characters or space except under score. In filerefs only, you can use the dollar sign (\$), pound sign (#), and at sign (@)
- SAS reserves a few names for automatic variables and variable lists, SAS data sets, and librefs.
- When creating variables, do not use the names of special SAS automatic variables (for example, \_N\_ and \_ERROR\_) or special variable list names (for example, \_CHARACTER\_, \_NUMERIC\_, and \_ALL\_).
- When associating a libref with a SAS data library, do not use these: SASHELP, SASMSG SASUSER, WORK
- When you create SAS data sets, do not use these names: \_NULL\_, \_DATA\_, \_LAST\_

# Coding Best Practices

Divide the program into logical steps and use indentation and comments. This makes program much easier to read, maintain and change in future

POOR CODE – NO SPACES & SPAGHETTI CODE	Good – has structure- macros at top- uses indenting- has comments of business rules & program logic – collect logic (Where and transforms in one data step).
<pre>data w; set rsw. RTLWP503;  data m; set rsm.FF00028QW;  data W1; set W; where wk_end_dt GT '2007-08-17';  data M1; set M; where sales_lvl_ind ="D";  data W2; set W1; Age=age*12;  data M2; set M1; Age=age*12; Run</pre>	<pre> /***** Section: Set Macros *****/ %let Maxdate=2008-08-17; /***** Section: Weeks *****/ data Weekly; /*We Need weekly data for some odd reason*/   set rsw. RTLWP503;   /*Program started Aug 17*/   where wk_end_dt GT '2007-08-17'd;   AgeMo=age*12; run; /***** Section: Months -We need monthly district level data 4 QC *****/ data montlyChk;   set rsm. RTLWZ_FF00028QW;   where sales_lvl_ind ="D";   AgeMo=age*12; run;</pre>

Its best to perform operations in one dataset, followed by other and then merge, rather than making merges at various stages to avoid loss of data and keep the code clean





# Thank You