SAS® Macro Magic For Beginners

Kirk Paul Lafler

Data & Systems Architect, Analytics Consultant, Data Engineer, Data Scientist, Educator, & Author

Empowering learners with data science, analytics, and practical insight

KirkLafler@cs.com

https://www.linkedin.com/in/kirkpaullafler/

Bio for Kirk Paul Lafler

Kirk Paul Lafler is an internationally recognized Enterprise Data & Systems Architect, Analytics Consultant, Data Engineer, Data Scientist, Educator, & Author who brings deep, cross-disciplinary expertise spanning analytics, data science, SQL, SAS, database systems, Python, application development, and systems thinking. His work is grounded in the intersection of rigorous technical execution and measurable business outcomes. Recognized for distilling complexity into clear, actionable insight, Kirk equips organizations and professionals to build capability and confidence through hands-on, applied instruction and leadership.

Widely respected for his ability to translate complex technical concepts into clear, structured, and approachable learning experiences, Kirk's training courses and presentations consistently resonate with audiences ranging from hands-on practitioners to technical leaders. His sessions are known for combining deep technical insight with real-world examples, best practices, and immediately actionable techniques where his technical content doesn't just explain what to do - it shows how and why, empowering participants with greater confidence, capability, and impact.

As a consultant, Kirk has partnered with organizations across industries to improve analytics capabilities, optimize systems, and enable teams to work more efficiently and confidently. As an educator and author, he is passionate about helping professionals grow their skills, strengthen their problem-solving abilities, and achieve better outcomes through smarter use of data and software.

As the author of several books, including PROC SQL: Beyond the Basics Using SAS, Third Edition (SAS Press, 2019), Kirk is a frequent invited speaker, educator, and keynote presenter at conferences, symposiums, and professional summits worldwide, and a recipient of 29 "Best" awards for contributed papers, hands-on workshops (HOWs), and posters.

Agenda

Session 1: SAS Macro Basics

(30 minutes)

Session 2: Parameterized Macros

(40 minutes)

Session 3: Loops & Conditional Logic

(40 minutes)

Session 4: SAS Macro Debugging

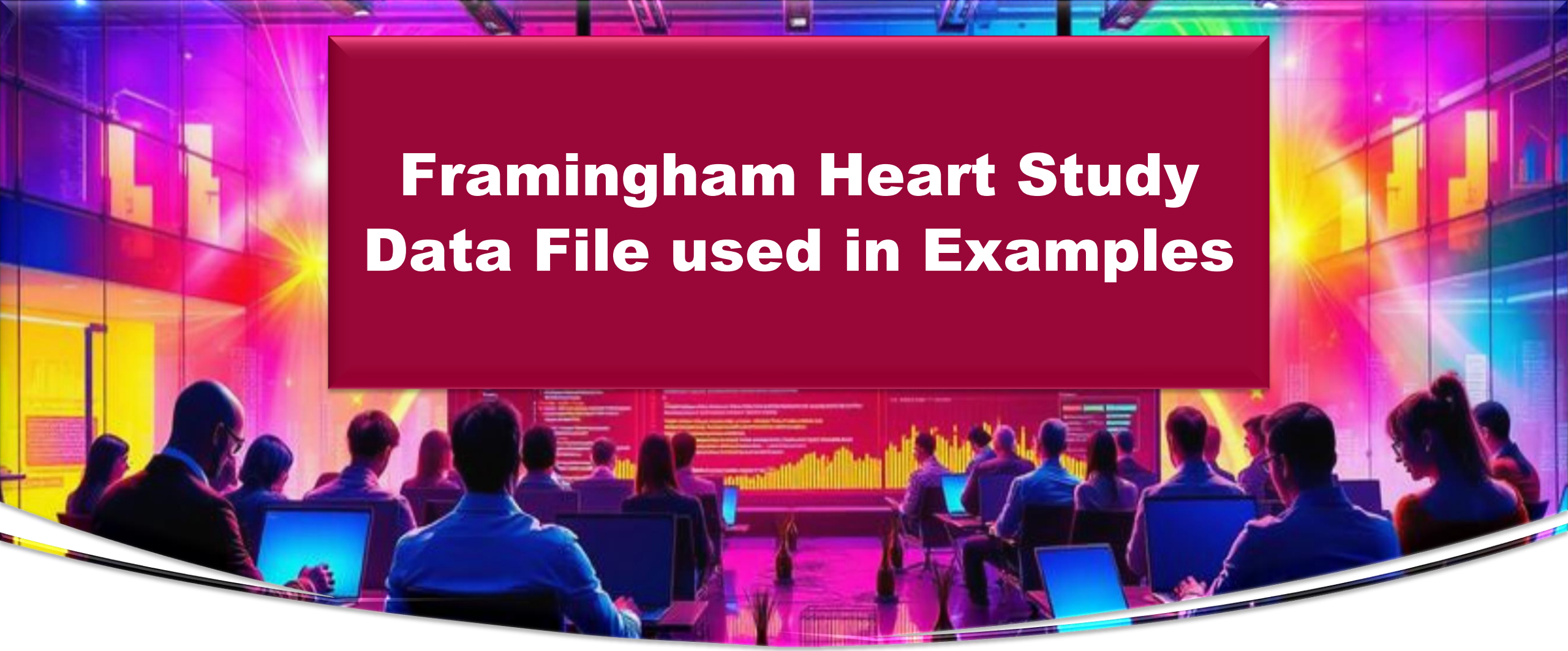
(40 minutes)

Session 5: Automation & Reporting

(40 minutes)

Session 6: Advanced SAS Macro Techniques

(30 minutes)



Framingham Heart Study Data File used in Examples

Data File Used in Examples

- **The synthesized Framingham Heart Study data file contains demographic, lifestyle, clinical, and laboratory variables commonly used in cardiovascular disease (CVD) risk modeling.**
- **The details of the data file consist of:**
  - ✓ **Number of Observations: 500**
  - ✓ **Number of Variables: 22**
  - ✓ **Unit of Analysis: Individual participant.**

Framingham Heart Study Data

age	sex	education	smoking	alcohol	physical_activi	sbp	dbp	height	weight	bmi	cholestero	hdl	ldl	triglyceride	glucose	diabetes	heart_rate	cvd_event	hypertensi	medications	death
68	female	high school	never	none	moderate	127.4	84.4	167.2	77.9	27.9	184	49.7	97.5	140.5	104.5	0	70.3	0	0	none	alive
58	male	college graduate	former	moderate	moderate	131.5	87.8	178.5	61.4	19.3	210.8	48.4	120.2	139	76	0	72.7	0	0	none	alive
44	male	college graduate	never	moderate	moderate	125.5	84.6	169.5	51.8	18	198.2	68.3	90.3	99.3	121.7	0	58.9	0	0	none	alive
72	female	college graduate	current	moderate	moderate	117	96.7	155.9	78.9	32.5	207.3	33.8	132	149.2	104.2	0	82	0	1	none	alive
37	male	college graduate	never	moderate	high	133.4	79.9	169.5	91.6	31.9	173.2	38.6	100	146.2	104	0	64	0	0	antihypertensive	alive
50	female	college graduate	never	moderate	low	138.8	86.7	187	82.1	23.5	246.9	65.5	132	164.5	99.8	0	53.3	0	0	none	alive
68	female	college graduate	never	none	low	108.2	69.1	182.5	75	22.5	153	41.1	81.3	155.1	89.6	0	68.9	0	0	none	alive
48	male	less than high sch	current	none	low	158.2	76.1	169.4	66.2	23.1	219.5	63.5	112.1	197.9	104.4	0	70.3	0	1	cholesterol	alive
52	male	some college	former	heavy	high	128	87	179.5	58.6	18.2	247.3	75.5	122.3	205.4	109.5	0	71.6	0	0	antihypertensive	non-cvd
40	female	college graduate	former	none	moderate	130.4	88.5	166.3	87.5	31.6	219.2	44	131.4	87	65.1	0	83.6	0	0	antihypertensive	alive
40	female	some college	never	moderate	low	144.2	77.1	153	88.7	37.9	191.9	60.2	93.3	202.1	89.2	0	67.6	0	1	both	alive
53	female	some college	former	moderate	high	134.7	79.3	160.8	51.8	20	178.9	32.7	110.4	78.8	94.9	0	68.8	0	0	none	alive
65	male	less than high sch	never	moderate	high	149.1	64.8	185.6	98.8	28.7	241.4	61.3	131.8	185.3	105.1	0	59	0	1	antihypertensive	alive
69	female	college graduate	current	moderate	high	135.7	76.4	167.3	83.6	29.9	143.1	53	61.5	187.2	123.9	0	78.2	0	0	cholesterol	alive
53	female	high school	never	moderate	high	117.8	88.9	167.6	96	34.2	161.2	57.1	71.9	35.3	84.6	0	71.8	0	0	antihypertensive	alive
32	female	less than high sch	never	moderate	low	137.2	85.8	167	54.9	19.7	184.1	40.3	107	155.6	90	0	65.8	0	0	none	alive
51	male	high school	never	heavy	moderate	107.1	85	189	54.5	15.3	249.5	25.1	174.5	121.9	89.7	0	51	0	0	none	alive
31	female	some college	never	moderate	moderate	132.2	80.5	186.3	72.8	21	215.2	57.5	114.7	104.5	84	0	84.4	1	0	cholesterol	alive
53	female	some college	former	none	high	129.3	80.1	192.2	82.5	22.3	161.3	19	110	231.1	93.1	0	57.7	0	0	none	alive
73	male	less than high sch	never	moderate	moderate	125.8	73.4	168.4	101.9	35.9	187.6	45.2	104.9	110.8	115.4	0	64.1	1	0	cholesterol	alive
59	female	less than high sch	former	heavy	moderate	127.4	87	173	85.6	28.6	175.4	64.1	76.2	101.6	78.9	0	73.6	1	0	antihypertensive	alive
67	female	high school	never	none	moderate	92.4	84.2	154.8	71.4	29.8	171.6	52.9	84.4	154.5	108.6	0	58	0	0	antihypertensive	alive
31	male	college graduate	former	moderate	low	119	84.9	184.8	59.6	17.5	239.2	79.2	112.2	191.1	66	0	55.1	0	0	cholesterol	non-cvd
50	female	high school	never	moderate	moderate	131.9	74.7	158.3	93.5	37.3	145.1	65.4	50.7	141.5	88.8	0	48.8	0	0	antihypertensive	alive
62	male	less than high sch	never	moderate	high	133.2	58.5	172.2	60.5	20.4	264.4	40.2	171.3	220	93.4	0	77.2	0	0	none	alive

Framingham Heart Study Data Dictionary

The synthesized Framingham Heart Study data file's variables and associated attributes are displayed, below.

Variable Name	Type	Description	Units / Categories
age	Numeric	Age of participant	Years
sex	Categorical	Biological sex	male, female
education	Categorical	Highest education level attained	high school, college graduate
smoking	Categorical	Smoking status	never, former, current
alcohol	Categorical	Alcohol consumption level	none, moderate
physical_activity	Categorical	Physical activity level	low, moderate, high
sbp	Numeric	Systolic blood pressure	mmHg
dbp	Numeric	Diastolic blood pressure	mmHg
height	Numeric	Height	centimeters
weight	Numeric	Weight	kilograms
bmi	Numeric	Body Mass Index	kg/m^2

Variable Name	Type	Description	Units / Categories
cholesterol	**Numeric**	**Total serum cholesterol**	**mg/dL**
hdl	Numeric	High-density lipoprotein cholesterol	mg/dL
ldl	Numeric	Low-density lipoprotein cholesterol	mg/dL
triglycerides	Numeric	Serum triglycerides	mg/dL
glucose	Numeric	Fasting blood glucose	mg/dL
diabetes	Binary	Diabetes diagnosis indicator	0 = No, 1 = Yes
heart_rate	Numeric	Resting heart rate	beats per minute
cvd_event	Binary	Cardiovascular disease event occurrence	0 = No event, 1 = Event
hypertension	Binary	Hypertension diagnosis indicator	0 = No, 1 = Yes
medications	Categorical	Current medication status	none, antihypertensive
death	Categorical	Vital status	alive, deceased

Import the Framingham Heart Study to SAS7BDAT

Code

```
FILENAME REFFILE 'Framingham Heart Study.csv' ;
LIBNAME MYDATA '/home/Data Sources' ;

PROC IMPORT
     DATAFILE=REFFILE
     DBMS=CSV
     OUT=MYDATA.Framingham_Heart_Study ;
  GETNAMES=YES ;
RUN ;

PROC CONTENTS
  DATA=MYDATA.Framingham_Heart_Study ;
RUN ;
```

Results

The CONTENTS Procedure

Data Set Name	MYDATA.FRAMINGHAM_HEART_STUDY	Observations	500
Member Type	DATA	Variables	22
Engine	V9	Indexes	0
Created	04/08/2026 16:08:17	Observation Length	200
Last Modified	04/08/2026 16:08:17	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat
1	age	Num	8	BEST12.	BEST32.
5	alcohol	Char	8	$8.	$8.
11	bmi	Num	8	BEST12.	BEST32.
12	cholesterol	Num	8	BEST12.	BEST32.
19	cvd_event	Num	8	BEST12.	BEST32.
8	dbp	Num	8	BEST12.	BEST32.
22	death	Char	7	$7.	$7.
17	diabetes	Num	8	BEST12.	BEST32.
3	education	Char	21	$21.	$21.
16	glucose	Num	8	BEST12.	BEST32.
13	hdl	Num	8	BEST12.	BEST32.
18	heart_rate	Num	8	BEST12.	BEST32.
9	height	Num	8	BEST12.	BEST32.
20	hypertension	Num	8	BEST12.	BEST32.
14	ldl	Num	8	BEST12.	BEST32.
21	medications	Char	16	$16.	$16.
6	physical_activity	Char	8	$8.	$8.
7	sbp	Num	8	BEST12.	BEST32.
2	sex	Char	6	$6.	$6.
4	smoking	Char	7	$7.	$7.
15	triglycerides	Num	8	BEST12.	BEST32.
10	weight	Num	8	BEST12.	BEST32.

Session 1:
SAS Macro Basics to Unlock the Power Behind SAS Automation

Session 1: Introduction

Every SAS programmer eventually reaches a point where repetitive coding such as copying and pasting code and making small edits becomes a nuisance. In this session, we introduce the foundational concepts that transform SAS from a procedural tool into a dynamic, flexible programming environment.

You'll explore what macros are, how they work, why they're essential for efficient and scalable programming, what macro variables are, understand the difference between DATA / PROC steps versus macro processing, how macros reduce redundancy, and understand the difference between compile-time and execution-time behavior.

This session lays the groundwork for everything that follows, so you'll not only understand macro syntax, but recognize opportunities to apply macros in your everyday SAS work.

Highlights: SAS Macro Basics

- A SAS macro is a code generation facility used to create dynamic, reusable SAS programs.
- Operate through the macro processor, which runs before SAS compiles and executes code.
- Triggered by special symbols:
  - ✓ % for macro statements.
  - ✓ & for macro variables.
- Macros are defined using %MACRO / %MEND and invoked by name.
- Macro variables store text values and are resolved prior to execution.
- The output of macro processing is standard SAS code (DATA steps and PROC steps).
- Support automation of repetitive tasks, improving efficiency and consistency.

What is the SAS Macro Language?

Part of the base product – no additional licenses required

Tool for extending and customizing the behavior of SAS software

A text substitution and code generation tool

Designed to automate repetitive tasks, reducing manual coding and improving efficiency

What is the SAS Macro Language, continued

Supports macro variables (&variable) that store text values and can be resolved during code execution.

Uses macros (%MACRO / %MEND) to define reusable blocks of code that can be invoked as needed.

Enables parameterized programming, allowing users to pass values into macros for flexible, reusable solutions.

Commonly used for batch processing, report generation, simulation studies, and large-scale data workflows.

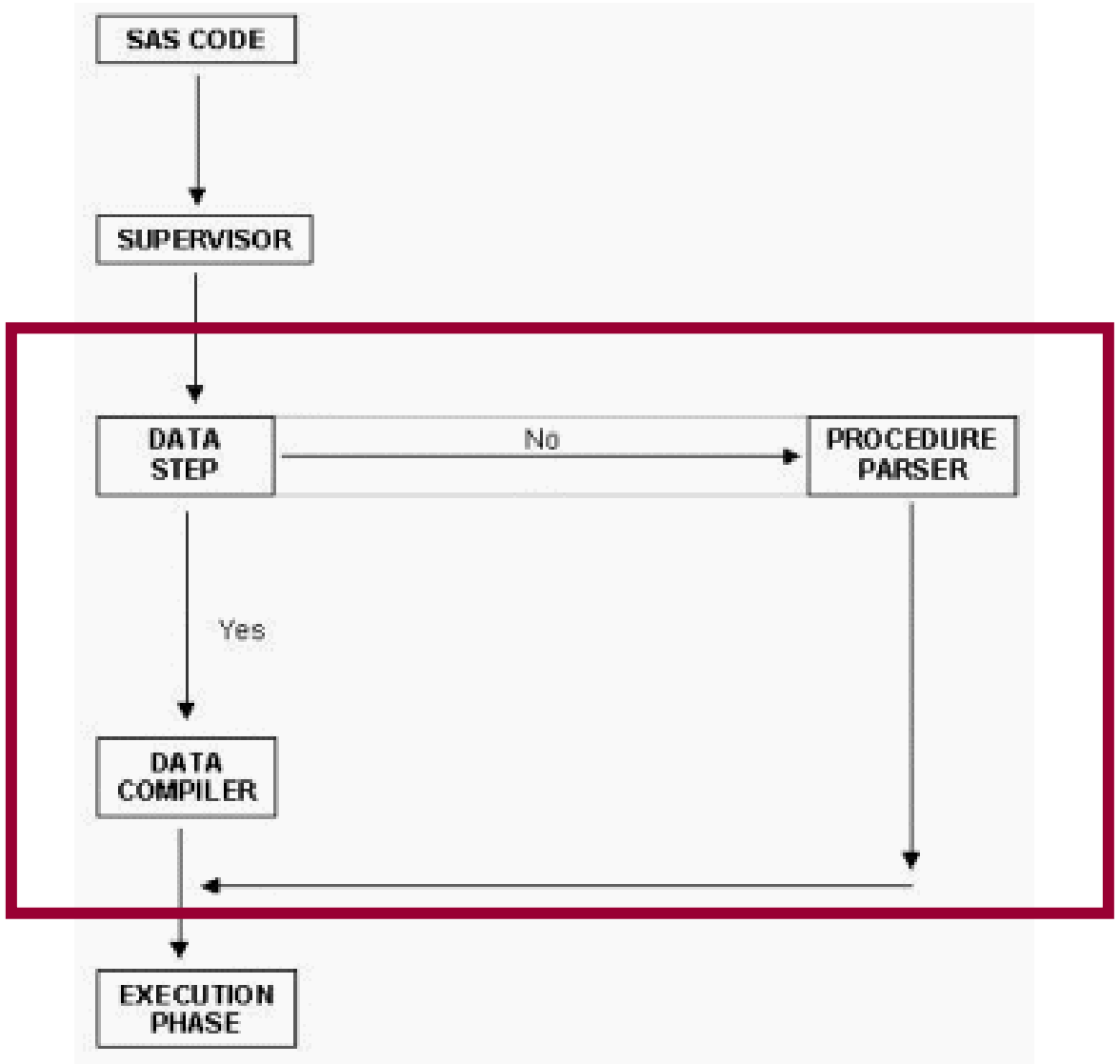
What is a SAS Macro?

- **SAS macros are used to generate and control SAS code dynamically.**
- **In simple terms, a SAS macro:**
  - ✓ **Is a code-writing tool.**
  - ✓ **Allows you to write flexible, reusable, and automated SAS programs by substituting text and creating SAS statements before the code runs.**
  - ✓ **Is a named block of code defined using %MACRO and %MEND.**
  - ✓ **Variable (e.g., &name) stores a value that can be reused throughout your program.**
  - ✓ **Must have its macros and macro variables resolved before SAS compiles and executes any code.**

SAS Program Control Flow without Macros

- **When SAS code is submitted for execution without a macro; the SAS Supervisor performs the following steps:**
  - ✓ **The Supervisor determines whether the code is a DATA step or a PROC step.**
  - ✓ **DATA step code is compiled first; if successful, it proceeds to execution.**
  - ✓ **PROC step code is already compiled and is sent to the procedure parser to interpret parameters, then executed.**

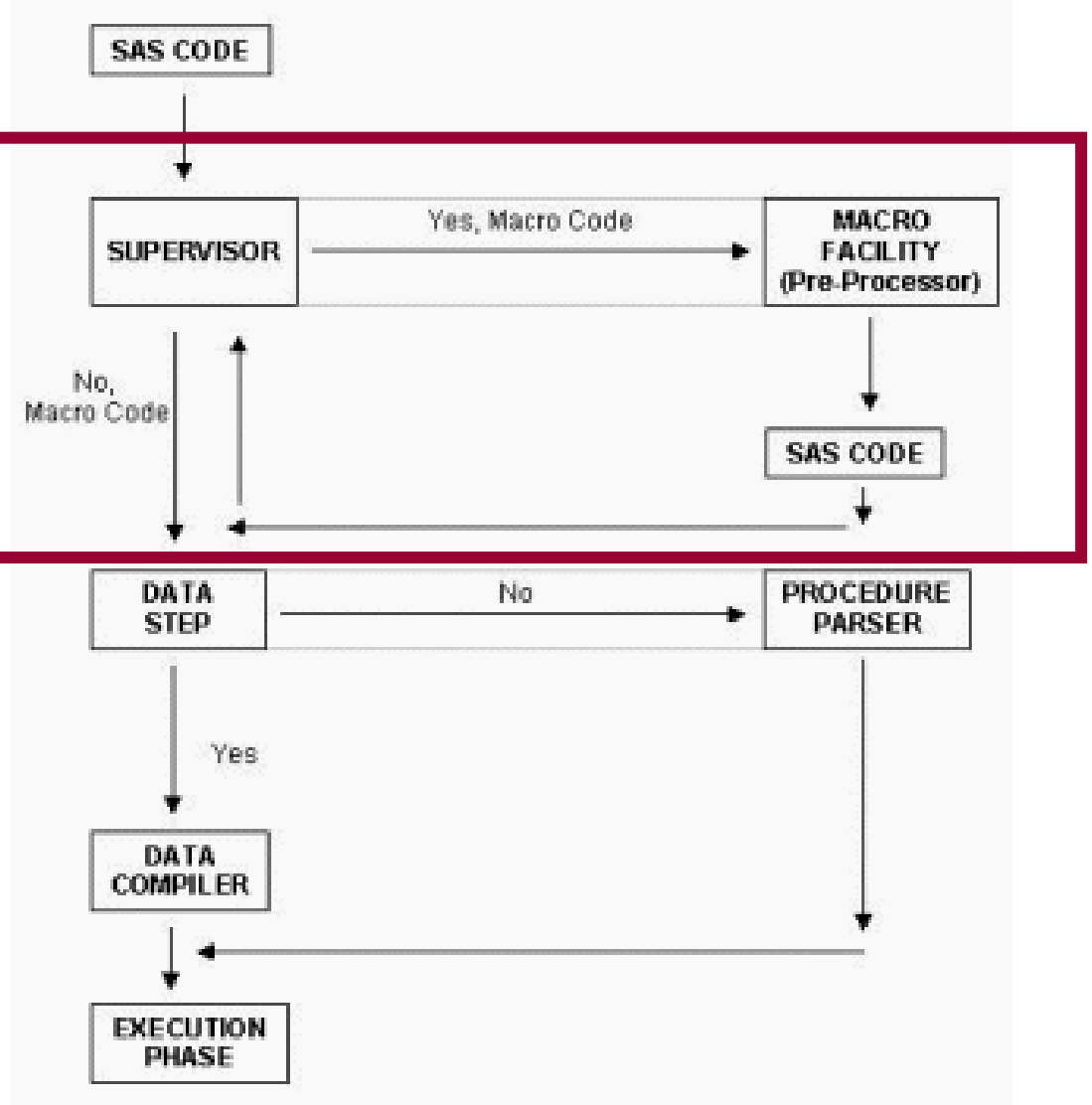
SAS Program Control Flow without Macros



SAS Program Control Flow with Macros

- **When SAS code containing macros is submitted for execution, the SAS Supervisor performs the following steps:**
  - ✓ **Scans the code for (& or %) characters. If found, the macro processor is invoked.**
  - ✓ **The macro processor resolves all macro variables and macro logic, generating standard SAS code.**
  - ✓ **Once macro resolution is complete, the Supervisor determines whether the code is a DATA step or a PROC step.**
  - ✓ **DATA step code is compiled first; if successful, it proceeds to execution.**
  - ✓ **PROC step code is already compiled and is sent to the procedure parser to interpret parameters, then executed.**

SAS Program Control Flow with Macros



Exploring SAS Layers

- SAS runs in two layers:
  - ✓ 1st Layer: Macro Layer (text generation).
  - ✓ 2nd Layer: SAS Execution Layer (DATA/PROC steps).
- Macros live entirely in the first layer.
- Simple Analogy of a SAS Macro:
  - ✓ Write a macro.
  - ✓ Provide inputs (parameters).
  - ✓ It generates a finished SAS program.
  - ✓ SAS then runs the program.

Create a Simple Macro to Print a Message

Code #1

```
%macro hello;
  %put Hello, SAS Macro World!;
%mend hello;
%hello;
```

Results

The SAS Log shows:

Hello, SAS Macro World!

Defining a Macro Variable

- Defined by the user
- Specify a macro variable with a %LET statement:
  - ✓ %LET DSN = WORK.Framingham ;
- Values can range from 0 bytes to 32K
- Valid characters include:
  - ✓ Letters
  - ✓ Keyboard characters
  - ✓ Numbers
  - ✓ Blanks between characters
- Values are always stored as characters
- Case of letters are preserved
- Leading and trailing blanks are not stored

Define a Macro Variable with %LET

Code #2

```
%let study = Framingham;
%put &study Dataset Loaded;
```

Results

The SAS Log shows:

Framingham Dataset Loaded

Reference a Macro Variable

Code #3

```
%let nobs = 10;
proc print data=framingham (obs=&nobs);
run;
```

Results

Prints the first 10 observations.

Macro Variable from Dataset with CALL SYMPUT

Code #4

```
data _null_;
 set framingham;
 if _n_=1 then call symput('first_age',
    age);
run;
%put First participant age = &first_age;
```

Results

SAS Log displays first participant's age.

Define Macro Variables with Multiple %LETs

Code #5

```
%LET DSN = Framingham ;
%LET VAR = Sex ;
%LET CLASS_VAR = Height Weight ;
PROC MEANS DATA=&DSN MAXDEC=0 ;
  VAR &VAR ;
  CLASS &CLASS_VAR ;
RUN ;
```

Results

Displays patients' height and weight grouped by sex.

Macro Referencing Environments

- **Global Macro environment**
  - ✓ macro variable created outside of a macro
  - ✓ macro variable created in %GLOBAL statement
  - ✓ display with %PUT _GLOBAL_;
- **Local Macro environment**
  - ✓ available only during macro execution
  - ✓ macro variable created inside a macro
  - ✓ macro variable created in %LOCAL statement
  - ✓ macro with one or more parameters
  - ✓ display with %PUT _LOCAL_;

Key Takeaways: SAS Macro Basics

- SAS macros are a code generation tool, not a data analysis tool.
- Macros eliminate repetitive coding across programs and projects.
- Macro variables store text, not data values.
- Macros operate before DATA and PROC steps.
- SAS first scans for macro triggers (% and &).
- SAS runs in two layers:
  - ✓ 1st Layer: Macro Layer (text generation).
  - ✓ 2nd Layer: SAS Execution Layer (DATA/PROC steps).
  - ✓ Macros live entirely in the first layer.
- The macro processor resolves macro variables and executes macro logic.
- The result is standard SAS code, which is then compiled and executed.

Session 2: Parameterized Macros to Write Flexible and Reusable Code

Session 2: Introduction

Once you understand the basics of macros, the next step is to make them truly powerful: parameterization. Hard-coded values limit flexibility, but parameterized macros allow you to write code once and reuse it across multiple scenarios with ease.

In this session, you'll learn how to design macros that accept inputs, making them adaptable, modular, and reusable. We'll cover positional and keyword parameters, default values, and best practices for structuring macro interfaces. You'll also see how parameterized macros improve consistency and reduce errors in real-world workflows.

Through hands-on examples, you'll move from static programs to dynamic tools that can handle varying datasets, variables, and analytical needs. This session is where your macros begin to feel like true "functions" in a programming sense – powerful, reusable, and efficient.

Highlights: Parameterized Macros

- Parameterized macros allow you to pass values into a macro, making code flexible and reusable.
- Defined with parameters inside the %MACRO – %MEND statements.
- Support both positional parameters (ordered) and keyword parameters (named).
- Can include default values, making macros easier to use and safer.
- Enable dynamic code generation across datasets, variables, and time periods.
- Help standardize logic across projects while reducing duplication.

Application Details for Parameterized Macros

Parameterized macros are widely used in:

- **Reporting automation**
  Generate the same report for different regions, time periods, or products.
- **Data pipeline workflows**
  Apply the same transformation logic to multiple datasets.
- **Clinical / life sciences programming**
  Reuse validated logic across studies while changing inputs (e.g., population, endpoints).
- **Batch processing**
  Run the same job across multiple scenarios using different parameter values.

Types of Parameterized Macros

- Keyword Parameters (Recommended – More readable & Flexible).
- Positional Parameters (Defined & Called in Order, Less Flexible).
- Hybrid Parameters (Defined & Called with Positional, then Keyword).

Keyword Macro Routine to Print Dataset

Code #6

```
%macro summary(data=);
  proc means data=&data;
  run;
%mend summary;
%summary(data=Framingham);
```

Results

Displays Summary statistics for numeric variables.

Keyword Macro with Multiple Parameters

Code #7

```
%macro print_data(ds=, obs=5);
  proc print data=&ds(obs=&obs);
  run;
%mend print_data;
%print_data(ds=Framingham, obs=8);
```

Results

Prints first 8 observations.

Keyword Macro to Calculate Mean of a Variable

Code #8

```
%macro mean_var(var=);
  proc means data=Framingham mean;
    var &var;
  run;
%mend mean_var;
%mean_var(var=BMI);
```

Results

Prints the Mean (or arithmetic average) of BMI.

Keyword Macro with Multiple Parameters

Code #9

```
%macro stats(ds=, var=, n=);
  proc means data=&ds n mean std;
    var &var;
    where age <= &n;
  run;
%mend stats;
%stats(ds=Framingham, var=SYSBP, n=50);
```

Results

Summary stats for SYSBP age ≤50.

Keyword Macro with Assigned Default Values

Code #10

```
%macro stats(ds=Framingham,
             var=SYSBP,
             n=50);
  proc means data=&ds n mean std;
    var &var;
    where age <= &n;
  run;
%mend stats;
%stats(n=250);
```

Results

Summary stats for SYSBP age ≤250.

Keyword Macro to Filter by Gender

Code #11

```
%macro gender_stats(gender=);
  proc means data=framingham mean;
    var BMI SYSBP;
    where sex="&gender";
  run;
%mend gender_stats;
%gender_stats(gender=F);
```

Results

Mean BMI and SYSBP for females.

Best Practices: Parameterized Macros

- Keyword parameters are preferred for clarity and maintainability.
- Always include default values where appropriate.
- Add basic validation for required parameters.
- Use meaningful parameter names (avoid cryptic abbreviations).
- Keep macros focused and modular.
- Hybrid parameters are defined & called with positional, then keyword.
- Use debugging options (MPRINT, SYMBOLGEN) when developing.

Key Takeaways: Parameterized Macros

- Parameterized macros turn SAS programs into flexible tools.
- They are essential for scalable, production-level SAS programming.
- The real power comes from combining parameters + macro logic + dynamic code generation.
- Well-designed parameterized macros reduce errors, improve consistency, and save significant time.
- Mastering these techniques is a major step from basic SAS coding to advanced automation.

Session 3:

Loops & Conditional Logic to Add Intelligence to Your Macros

Session 3: Introduction

With parameterized macros in your toolkit, the next leap is adding logic, enabling your programs to make decisions and repeat tasks automatically. This session introduces the control structures that bring intelligence and adaptability to your SAS macros.

You'll learn how to use macro loops to iterate over lists, datasets, or time periods, eliminating repetitive coding patterns. Conditional logic using %IF-%THEN-%ELSE will allow your macros to respond dynamically to different inputs and scenarios. Together, these tools enable you to build programs that are not only reusable but also responsive and scalable.

We'll emphasize practical use cases, such as generating multiple reports, processing batches of data, and customizing outputs based on conditions. By the end of this session, your macros will evolve from simple templates into smart, decision-driven programs.

Highlights: Loops & Conditional Logic

- Enable dynamic code generation by repeating and conditionally creating SAS statements before execution.
- Operate entirely in the macro processing phase, not during DATA or PROC step execution.
- %DO loops allow you to iterate over ranges, lists, or conditions, generating multiple blocks of SAS code.
- %IF-%THEN-%ELSE provides decision-making capability to control which code is produced.
- Work with macro variables (text values) rather than dataset values.
- Help reduce manual coding, duplication, and maintenance effort.
- Require careful attention to syntax, resolution timing, and proper use of %DO/%END blocks.

Simple %DO Loop

Code #12

```
%macro print_vars;
  %do i=1 %to 4;
      %let var=%scan(&vars,&i);
      proc print data=Framingham (obs=5);
        var &var;
      run;
  %end;
%mend print_vars;
%print_vars;
```

Results

Prints first 5 observations of each variable.

Macro with %IF-%THEN

Code #13

```
%macro check_age(age=);
  %if &age < 50 %then %put Young
 participant;
  %else %put Older participant;
%mend check_age;
%check_age(age=45);
```

Results

Log shows Young participant.

Macro Looping Over Dataset

Code #14

```
%macro loop_datasets;
  %let dslist = framingham temp;
  %do i=1 %to 2;
      %let ds=%scan(&dslist, &i);
      proc print data=&ds(obs=3);
      run;
  %end;
%mend loop_datasets;
%loop_datasets;
```

Results

Prints first 3 observations of each dataset.

Conditional PROC Execution

Code #15

```
%macro cond_run(var=);
  %if &var=AGE %then %do;
      proc freq data=framingham;
        tables age;
      run;
  %end;
%mend cond_run;
%cond_run(var=AGE);
```

Results

Frequency of age printed.

Loop through Age Groups

Code #16

```
%macro age_groups;
  %do i=30 %to 70 %by 10;
      proc means data=framingham;
        var BMI;
        where age between &i and
              eval(&i+9);
      run;
  %end;
%mend age_groups;
%age_groups;
```

Results

BMI means for 30–39, 40–49, etc.

Best Practices: Loops & Conditional Logic

- Keep loop logic simple and readable.
- Use meaningful loop variable names.
- Add debugging options (MPRINT, MLOGIC) when troubleshooting.
- Combine with parameterized macros for maximum flexibility.

Key Takeaways: Loops & Conditional Logic

- Parameterized macros turn SAS programs into flexible tools.
- They are essential for scalable, production-level SAS programming.
- The real power comes from combining parameters + macro logic + dynamic code generation.
- Well-designed parameterized macros reduce errors, improve consistency, and save significant time.
- Mastering these techniques is a major step from basic SAS coding to advanced automation.

Session 4:

SAS Macro Debugging to Identify, Interpret, and Fix Problems with Confidence

Session 4: Introduction

Even the most well-written macros can produce unexpected results, and debugging macro code can feel like navigating a maze without a map. This session equips you with the essential tools and techniques to diagnose and resolve macro-related issues effectively.

You'll learn how to interpret SAS log messages, identify common macro pitfalls, and use debugging options such as MPRINT, MLOGIC, and SYMBOLGEN to trace execution. We'll also explore strategies for isolating problems, validating macro variables, and understanding how macro resolution impacts program behavior.

Rather than fearing errors, you'll begin to see them as clues. This session builds your confidence and equips you with a systematic approach to troubleshooting, ensuring that you can develop and maintain macros with clarity and precision.

Highlights: SAS Macro Debugging

- Focus on how SAS generates code, not just how it executes it.
- Most common issue stems from incorrect macro variable resolution or unexpected text substitution.
- Key Debugging Techniques:
  - ✓ Turn on debugging options early when developing macros:

    ```
    OPTIONS MPRINT MLOGIC SYMBOLGEN;
    ```
  - ✓ Use %PUT statements to print macro variable values and trace execution.
- Common Problem Areas:
  - ✓ Missing or incorrect & or % triggers.
  - ✓ Unresolved macro variables (e.g., typos or scope issues).
  - ✓ Misuse of %IF vs. DATA step IF statements.
  - ✓ Missing %DO / %END blocks.

Enabling Debugging Options

Code #17

```
options mprint mlogic symbolgen;
%macro test;
 %let var=age;
 proc means data=framingham;
   var &var;
 run;
%mend test;
%test;
```

Results

Identify how &var resolves in the log.

Detect a Misspelled Macro Variable

Code #18

```
%macro summary;
  %let variable=totchol;
  proc means data=framingham;
    var &varible;
  run;
%mend summary;
%summary;
```

Results

Use SYMBOLGEN to diagnose the issue.

Macro Variable Scope Issue

Code #19

```
%macro outer;
  %let var=age;
  %macro inner;
    proc means data=framingham;
      var &var;
    run;
  %mend inner;
  %inner;
%mend outer;
%outer;
```

Results

Modify to ensure proper scope using %GLOBAL or %LOCAL.

Debugging Conditional Logic

Code #20

```
%macro check_gender(gender=);
  %if &gender = M %then %do;
      proc freq data=framingham;
        tables sex;
      run;
  %end;
%mend check_gender;
%check_gender(gender=F);
```

Results

Add debugging output to trace logic flow.

Loop Debugging

Code #21

```
%macro loop_test;
  %do i=1 %to 3;
      %put Iteration: &i;
  %end;
%mend loop_test;
%loop_test;
```

Results

Use MLOGIC to trace loop execution.

Debugging Generated Code with MPRINT

Code #22

```
%macro print_data(ds=);
  proc print data=&ds;
  run;
%mend print_data;
%print_data(ds=framingham);
```

Results

Examine generated SAS code in the log.

Using %PUT for Debugging

Code #23

```
%macro calc;
  %let x=10;
  %let y=20;
  %let z=%eval(&x + &y);
  %put Result: &z;
%mend calc;
%calc;
```

Results

Enhance logging to show intermediate values.

Best Practices: SAS Macro Debugging

- Keep macros simple and modular.
- To reveal generated code, macro logic flow, and variable resolution turn on the following debugging options:

```
OPTIONS MPRINT MLOGIC SYMBOLGEN;
```

- Always inspect the generated SAS code:
  - ✓What SAS executes is the resolved code, not the macro itself.
  - ✓Use MPRINT output to verify correctness.
- Use %PUT statements liberally:
  - ✓Print macro variable values and checkpoints.
- Build and test incrementally:
  - ✓Develop macros in small pieces.
  - ✓Test each component before combining into larger logic.

Key Takeaways: SAS Macro Debugging

- Macros generate code that SAS executes.
- The SAS log is your primary debugging tool.
- Focus on the resolved code, the log, and the flow of text substitution.
- Effective macro debugging is about seeing through the macro layer.
- Use debugging options to expose macro behavior:
  - ✓ MPRINT → shows generated code.
  - ✓ MLOGIC → shows macro logic flow.
  - ✓ SYMBOLGEN → shows macro variable resolution.
- Print and trace values during execution.
- Macro variables are text, not data values.
- Macro logic runs before DATA/PROC execution.

Session 5:
Automation & Reporting to Turn Macros into Productivity Engines

Session 5: Introduction

At this stage, you've built a strong foundation in macro programming. Now it's time to put the skills you've learned to work in one of the most impactful areas of SAS: automation and reporting.

In this session, you'll learn how to leverage macros to streamline repetitive workflows, generate consistent reports, and manage complex analytical processes with minimal manual intervention. We'll explore techniques for automating report generation across multiple datasets, variables, or time periods, and demonstrate how macros can integrate with PROC steps to produce polished outputs.

You'll also see how macros support scalability, allowing you to handle larger workloads with less effort and fewer errors. This session highlights the real-world value of macros, showing how they can save time, improve accuracy, and enhance productivity in professional environments.

Highlights: SAS Macro Automation & Reporting

- Enables end-to-end automation of repetitive reporting and data workflows.
- Uses parameterized macros to drive flexible, reusable reporting solutions.
- Generates dynamic SAS code for different datasets, time periods, and business rules.
- Leverages macro loops (%DO) to iterate across reporting scenarios.
- Applies conditional logic (%IF-%THEN) to customize outputs.
- Integrates with Output Delivery System to produce reports in PDF, Excel, HTML, and more.
- Produces standardized, consistent, and presentation-ready outputs.

Macro for Multiple PROC MEANS

Code #24

```
%macro multi_means(vars=);
  %do i=1 %to %sysfunc(countw(&vars));
      %let v=%scan(&vars, &i);
      proc means data=framingham mean;
        var &v;
      run;
  %end;
%mend multi_means;
%multi_means(vars=BMI SYSBP DIASTBP);
```

Results

Produce the results by running the macro code.

Generate a PDF Report Dynamically

Code #25

```
%macro pdf_report(var=);
  ods pdf file="report_&var..pdf";
  proc means data=framingham;
    var &var;
  run;
  ods pdf close;
%mend pdf_report;
%pdf_report(var=CHOL);
```

Results

Produce the results by running the macro code.

Macro to Export Dataset Subset

Code #26

```
%macro export_subset(age_limit=);
  data subset;
    set framingham;
    where age < &age_limit;
  run;
  proc export data=subset
       outfile="subset_&age_limit..csv"
       dbms=csv replace;
  run;
%mend export_subset;
%export_subset(age_limit=50);
```

Results

Produce the results by running the macro code.

Macro Chaining Multiple Steps

Code #27

```
%macro analyze(var=);
  %mean_var(var=&var);
  %pdf_report(var=&var);
%mend analyze;
%analyze(var=SYSBP);
```

Results

Produce the results by running the macro code.

Macro for Correlation Matrix

Code #28

```
%macro corr_matrix(vars=);
  proc corr data=framingham;
    var &vars;
  run;
%mend corr_matrix;
%corr_matrix(vars=BMI SYSBP DIASTBP
                  CHOL);
```

Results

Produce the results by running the macro code.

Best Practices: Automation & Reporting

- Use parameterized macros to make reports flexible (e.g., by date, gender, education, smoking, etc.).
- Prefer keyword parameters with defaults for clarity and usability.
- Keep macros modular and single-purpose (one macro = one task).
- Establish consistent naming conventions for macros, parameters, and outputs.
- Avoid hardcoding by making inputs data-driven whenever possible.
- Automate BY-group or iterative reporting using %DO loops.
- Use conditional logic (%IF-%THEN) to customize report content.
- Standardize output formats (PDF, Excel, HTML) using ODS (Output Delivery System).
- Optionally include a debug mode parameter to toggle verbose logging.

Key Takeaways: Automation & Reporting

- Macros enable repeatable, parameter-driven workflows for reports and data processes.
- Parameterization is the foundation of scalability.
- Once built, macros can run consistently with minimal intervention.
- ODS can produce standardized reports in PDF, Excel, HTML, and more.
- Loops & conditional logic allow selective and iterative report generation.
- Use macros for code generation and DATA/PROC steps for data processing.
- Well-structured macros can be reused across projects and teams.
- Automated reporting ensures uniform formats, naming, and presentation.
- Macros deliver consistent, efficient, and high-quality results at scale.

Session 6:

Advanced Techniques for Beginners to Bridge the Gap to Mastery

Session 6: Introduction

In the final session, we bring everything together and introduce a set of advanced yet accessible techniques that elevate your macro programming skills to the next level. While still beginner-friendly, this session challenges you to think more strategically about how macros can be designed and applied.

You'll explore topics such as dynamic code generation, indirect referencing, macro variable scoping, and working with lists and metadata. These techniques allow you to write more sophisticated, flexible, and maintainable macros that can adapt to complex scenarios.

This session is not about overwhelming you with complexity, it's about expanding your perspective. By the end, you'll have a clearer vision of what's possible with SAS macros and a solid foundation to continue your journey toward advanced macro programming.

Highlights: Advanced Macro Techniques

- Keyword parameters with defaults enable flexible, user-friendly macros.
- Macros can encapsulate entire analytical procedures by including PROC MEANS, PROC FREQ, PROC COMPARE, and PROC SGPLOT.
- Parameter-driven PROC COMPARE macros support quick, repeatable dataset checks.
- Efficiently iterate over datasets or variables without manual repetition.
- Using &prefix._: allows flexible selection of related variables without hardcoding names.
- Parameterized PROC SGPLOT chart generation enables fast, repeatable visual reporting.
- Parameterizing CLASS and VAR enables flexible subgroup analysis.
- Macros support both simple and multi-step workflows.

Keyword Macro with Optional Parameter

Code #29

```
%macro optional_param(var=SYSBP,
                      ds=framingham);
  proc means data=&ds;
     var &var;
   run;
%mend optional_param;
%optional_param( );
%optional_param(var=BMI);
```

Results

Produce the results by running the macro code.

Macro for Dataset Comparison

Code #30

```
%macro compare_datasets(ds1=, ds2=);
  data Framingham_2;
    set Framingham(drop=death);
  run;
  proc compare base=&ds1 compare=&ds2;
  run;
%mend compare_datasets;
%compare_datasets(ds1=Framingham
                  ds2=Framingham_2);
```

Results

Produce the results by running the macro code.

Dynamic Variable Selection

Code #31

```
%macro select_vars(prefix=);
  data selected;
    set framingham(keep=AGE SEX
                   &prefix._:);
  run;
%mend select_vars;
%select_vars(prefix=SYSBP);
```

Results

Produce the results by running the macro code.

Iterate through Multiple Datasets

Code #32

```
%macro process_datasets(list=);
 %do i=1 %to %sysfunc(countw(&list));
     %let ds=%scan(&list, &i);
     %if %sysfunc(exist(&ds)) %then %do;
         proc means data=&ds;
         run;
     %end;
     %else %put
    WARNING: Dataset &ds does not exist.;
  %end;
%mend process_datasets;
%process_datasets(list=framingham temp);
```

Results

Produce the results by running the macro code.

Macro to Handle Missing Values

Code #33

```
%macro miss_check(var=);
  proc freq data=framingham;
    tables &var / missing;
  run;
%mend miss_check;
%miss_check(var=CHOL);
```

Results

Produce the results by running the macro code.

Macro to Generate Bar Charts

Code #34

```
%macro bar_chart(var=);
  proc sgplot data=framingham;
    vbar &var;
  run;
%mend bar_chart;
%bar_chart(var=SEX);
```

Results

Produce the results by running the macro code.

Macro to Automate PROC FREQ

Code #35

```
%macro freq_loop(vars=);
  %do i=1 %to %sysfunc(countw(&vars));
      %let v=%scan(&vars, &i);
      proc freq data=framingham;
        tables &v;
      run;
  %end;
%mend freq_loop;
%freq_loop(vars=SEX SMOKE DIABETES);
```

Results

Produce the results by running the macro code.

Macro to Create Summary by Group

Code #36

```
%macro group_summary(group=, var=);
  proc means data=framingham mean std;
    class &group;
    var &var;
  run;
%mend group_summary;
%group_summary(group=SEX, var=BMI);
```

Results

Produce the results by running the macro code.

Macro to Generate Dynamic Reports for Many Variables

Code #37

```
%macro multi_report(vars=);
  %do i=1 %to %sysfunc(countw(&vars));
      %let v=%scan(&vars,&i);
      %pdf_report(var=&v);
  %end;
%mend multi_report;
%multi_report(vars=BMI SYSBP DIASTBP
                   CHOL);
```

Results

Produce the results by running the macro code.

Best Practices: Advanced Macro Techniques

- Use keyword parameters with sensible defaults.
- Encapsulate complete procedures within macros to promote reuse and standardization.
- Design macros to be parameter-driven rather than hardcoding.
- Automate repetition with macro loops.
- Build reusable macros to standardize validation and comparison workflows.
- Support flexible subgroup analysis with CLASS and VAR statements.
- Ensure macros can handle single tasks as well as be combined into larger, multi-step processes.
- Well-designed macros enable you to scale from simple analyses to fully automated reporting systems with minimal code duplication.

Key Takeaways: Advanced Macro Techniques

- Keyword parameters with defaults make macros adaptable, user-friendly, and reusable.
- Macros transform procedures into reusable tools.
- Looping through datasets and variables eliminates redundancy and supports large-scale workflows.
- Techniques like &prefix._: remove the need for hardcoded variable lists.
- Dynamically controlling CLASS & VAR supports diverse subgroup analyses.
- Macros scale from simple tasks to full workflows.

Conclusion

Session 1: SAS Macro Basics

(30 minutes)

Session 2: Parameterized Macros

(40 minutes)

Session 3: Loops & Conditional Logic

(40 minutes)

Session 4: SAS Macro Debugging

(40 minutes)

Session 5: Automation & Reporting

(40 minutes)

Session 6: Advanced SAS Macro Techniques

(30 minutes)

SAS Macro Magic for Beginners

Kirk Paul Lafler
Data & Systems Architect, Analytics Consultant, Data Engineer, Data Scientist, Educator, & Author
Empowering learners with data science, analytics, and practical insight
KirkLafler@cs.com
https://www.linkedin.com/in/kirkpaullafler/