Best Practice

Programming Techniques Using SAS® Software

a presentation by

Kirk Paul Lafler

Senior SAS® Consultant, Application Developer, Data Scientist, Educator and Author

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

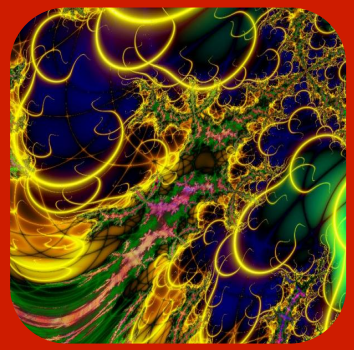
@sasNerd

Topics

Good Programming Concepts

Naming Conventions

Comments and Documentation



Configuration Management

Coding Conventions

SET and EMPLOYEES Data Sets

	Variable1	Variable2	Variable3	Variable4	Variable5
1	Jones	Sam	M	San Diego	CA
2	Harrison	Mary	F	Los Angeles	CA
3	Flamingo	Tom	M	San Francisco	CA
4	Abbott	Jane	F	Orlando	FL
5	Mantel	Margaret	F	Portland	OR
6	Bennett	Arnold	M	Atlanta	GA

Set

	LastName	FirstName	Gender	City	State
1	Jones	Sam	M	San Diego	CA
2	Harrison	Mary	F	Los Angeles	CA
3	Flamingo	Tom	M	San Francisco	CA
4	Abbott	Jane	F	Orlando	FL
5	Mantel	Margaret	F	Portland	OR
6	Bennett	Arnold	M	Atlanta	GA

Employees

MOVIES and ACTORS Data Sets

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

Movies

Actors

	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	Christmas Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	Ghost	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegge	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet

CARS Data Set

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3	6	220	20	28	3462	104	179
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3	6	220	17	26	3583	104	179
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3	6	220	18	25	3627	104	179
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3	6	220	20	27	3561	109	192
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3	6	220	18	25	3880	109	192
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3	6	220	20	27	3814	105	180
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3	6	220	18	25	4013	105	180
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7	6	250	18	25	3836	109	192
18	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2	8	300	17	24	4024	109	193
19	Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2	8	330	17	24	4399	121	204
20	Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2	8	340	14	20	3825	104	179
21	Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2	8	450	15	22	4024	109	191
22	Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	$35,940	$32,512	1.8	4	180	20	28	3131	95	159
23	Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	$37,390	$33,891	1.8	4	225	20	28	2921	96	159
24	Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	$40,590	$36,739	3.2	6	250	21	29	3351	96	159
25	Audi	A6 3.0 Avant Quattro	Wagon	Europe	All	$40,840	$37,060	3	6	220	18	25	4035	109	192
26	Audi	S4 Avant Quattro	Wagon	Europe	All	$49,090	$44,446	4.2	8	340	15	21	3936	104	179
27	BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3	6	225	16	23	4023	110	180
28	BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4	8	325	16	22	4824	111	184
29	BMW	325i 4dr	Sedan	Europe	Rear	$28,495	$26,155	2.5	6	184	20	29	3219	107	176
30	BMW	325Ci 2dr	Sedan	Europe	Rear	$30,795	$28,245	2.5	6	184	20	29	3197	107	177
31	BMW	325Ci convertible 2dr	Sedan	Europe	Rear	$37,995	$34,800	2.5	6	184	19	27	3560	107	177
32	BMW	325xi 4dr	Sedan	Europe	All	$30,245	$27,745	2.5	6	184	19	27	3461	107	176

Best Practice Concepts

What is a Best Practice?

- A best practice is a particular approach, method, or technique that has achieved some level of approval or acceptance by a professional association, authoritative entity, and/or by published research results.
- A best practice achieves status by being:
  - ✓ Measurable with quantifiable results
  - ✓ Successful in accomplishing stated goals and objectives
  - ✓ Reproducible or adaptable to specific needs

Naming Conventions

Purpose of Naming Conventions

- Simplify identification and categorization when naming conventions are adhered to
- Assigned to programs, libraries, catalogs, catalog entries, datasets, views, macro definitions, variables, macro variables, arrays, indexes, hash objects, subroutines, labels, user-defined formats, SAS/ACCESS access descriptors and view descriptors, external files, and SAS name literals
- Serve as self-documentation for the individual application components
- Permit others to better understand the application and code details

Poor Naming Conventions

Valid, but Confusing, Syntax and Naming Conventions:

```
OPTIONS  DATASTMTCHK = NONE ;
%LET  WHERE  =  SAN DIEGO ;
DATA  WORK.DATA ;
   SET  WORK.SET ;
   WHERE  UPCASE(VARIABLE4) = “&WHERE” ;
RUN ;
```

SAS Log:

```
NOTE: There were 1 observations read from the data set WORK.SET.
     WHERE UPCASE(VARIABLE4)='SAN DIEGO';
NOTE: The data set WORK.DATA has 1 observations and 5 variables.
```

“Good” naming conventions are a must and should always be adhered to whenever possible. To inforce the use of “good” naming conventions in the SAS environment, specify:

OPTIONS DATASTMTCHK = COREKEYWORDS ;

Better Naming Conventions

Valid and less Confusing Syntax:

```
OPTIONS  DATASTMTCHK = COREKEYWORDS ;
%LET  WHERE  =  SAN DIEGO ;
DATA  WORK.SanDiego_Employees ;
   SET  WORK.Employees ;
   WHERE  UPCASE(City) = “&WHERE” ;
RUN ;
```

SAS Log:

```
NOTE: There were 1 observations read from the data set
      WORK.EMPLOYEES.
      WHERE UPCASE(City)='SAN DIEGO';
NOTE: The data set WORK.SanDiego_Employees has 1 observations
      and 5 variables.
```

Naming Convention Rules

SAS Datasets and Variables:

1. Name that is assigned can be 1-32 positions in length.
2. 1st position can be a letter (A-Z, a-z) or underscore (_) only.
3. Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
4. Example:

```
DATA  WORK.PG_Rated_Movies ;
   SET  libref.Movies ;
   WHERE  Rating = “PG” ;
RUN ;
```

Rules (continued)

Librefs, Filerefs, Subroutines, and Labels:

1. Assigned name can be 1-8 positions in length.
2. 1st position can be a letter (A-Z, a-z) or underscore (_) only.
3. Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
4. Example:

```
LIBNAME  MYDATA  'C:\DataLibrary' ;
DATA  MYDATA.PG_Rated_Movies ;
   SET  MYDATA.Movies ;
   WHERE  Rating = "PG" ;
RUN ;
```

Rules (continued)

Macro Names and Macro Definitions:

1. Name that is assigned can be 1-32 positions in length.
2. 1st position can be a letter (A-Z, a-z) or underscore (_) only.
3. Remaining positions can be letters (A-Z, a-z), underscore (_), or numbers (0-9).
4. Example:

```
PROC SQL ;
 SELECT TITLE INTO :mtitle SEPARATED BY "* "
   FROM MYDATA.Movies ;
    WHERE UPCASE(Rating) = "PG" ;
QUIT ;
%PUT &mtitle ;
```

Comments and Documentation

Purpose

- Comments and documentation can assist in understanding what's in datasets and variables
- Provide descriptive information about the intricacies of a program
- Minimize confusion about what a program is doing
- Save time during support or maintenance of a program
- Document program run instructions:
  - ✓ the purpose of the program
  - ✓ how to run the program
  - ✓ what the input, processing and output is
  - ✓ Special operations that need additional understanding

Assign Dataset Labels

Dataset Labels are typically specified when saving permanent datasets and are displayed with PROC CONTENTS or PROC DATASETS output.

1.
```
DATA  libref.PG_RATED_MOVIES
          (LABEL="PG-Rated Movies") ;
     SET  libref.MOVIES ;
     WHERE  RATING = "PG" ;
RUN ;
```

2.
```
PROC CONTENTS
     DATA=WORK._ALL_ ;
RUN ;
```

Data Set Name	WORK.PG_RATED_MOVIES	Observations	6
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	Friday, October 03, 2014 02:54:23 AM	Observation Length	88
Last Modified	Friday, October 03, 2014 02:54:23 AM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label	PG-Rated Movies		
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Assign Variable Labels

Variable Labels are typically specified with permanent datasets and can be displayed as column headers using PROC PRINT.

```
1.  DATA  libref.PG_RATED_MOVIES
                (LABEL="PG-Rated Movies") ;
        SET  libref.MOVIES ;
        WHERE  RATING = "PG" ;
        NEW_LENGTH = LENGTH + 1 ;
        LABEL NEW_LENGTH = "Length with Trailer" ;
    RUN ;
```

Variable Labels (continued)

2. PROC PRINT DATA=libref.PG_RATED_MOVIES LABEL ;
   TITLE ;
   RUN ;

Obs	Title	Length	Category	Year	Studio	Rating	Length with Trailer
1	Casablanca	103	Drama	1942	MGM / UA	PG	104
2	Jaws	125	Action Adventure	1975	Universal Studios	PG	126
3	Poltergeist	115	Horror	1982	MGM / UA	PG	116
4	Rocky	120	Action Adventure	1976	MGM / UA	PG	121
5	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG	125
6	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	136

Comment Types

Comments can be added in the following ways:

1. *** (Asterisk) (aka Flower-box)**

```
**************************************************;
*****  This program selects PG movies.        *****;
**************************************************;
```

2. **COMMENT statement**

```
COMMENT  This program selects PG movies;
```

3. **In-stream comments can be added on any line of code**

```
DATA  PG_RATED_MOVIES;  /* Create temp dataset */
 SET  libref.MOVIES;          /* Read MOVIES dataset */
 WHERE  RATING = "PG";    /*Select PG Movies */
RUN;
```

Program Headers

Program Headers should be created at the beginning of each program. It serves to provide details about the name of the program, the program purpose, the names of the called and calling programs, who wrote the program, the date it was written, the input and output sources, and the modification history.

Program Header Information

- Program/Macro Name
- Author/Written By
- Date Written
- Purpose
- SAS Version
- Requirements
- Input Files
- Datasets Created
- Output Files
- Subroutines/Sub-macros
- Macro Variables Created
- Included Copybook Code
- Limitations
- Modification History

Consider incorporating this content into your programs . . .

Program Header Example

```
*************************************************************************************************;
***** Program Name: %SYSFUNC_Welcome_Screen.SAS                                              *****;
***** Purpose.....: Display the Welcome screen for the Building Reusable Tool using         *****;
*****               the SAS Macro Language and %SYSFUNC for accessing functions.           *****;
*****                                                                                        *****;
***** Author......: Kirk Paul Lafler, Software Intelligence Corporation                     *****;
***** Date Written: 06/13/2010                                                              *****;
***** SAS Version.: SAS 9.2, 9.3, 9.4                                                       *****;
***** Input Files.: Workshop Data                                                           *****;
***** Output Files: None                                                                    *****;
***** Subroutines.: None                                                                    *****;
***** Macro Variables: &sysver, &sysday                                                     *****;
***** Includes....: None                                                                    *****;
***** Modification History:                                                                 *****;
*****  01/20/2015  KPL  Added ODS HTML CLOSE to reflect changes in SAS software.           *****;
*************************************************************************************************;

options ls=100 source source2;
ods html close;
ods listing;
libname mydata 'e:\workshops\workshop data';

%window Window_Welcome color=white
   #5 @25 'Welcome to Building Reusable Tools using the SAS Macro Language' attr=highlight color=blue
   #9 @27 "You are executing SAS Release &sysver on &sysday, %sysfunc(date(),worddate18.)"
  #14 @40 "Press the enter key to continue.";
  %display Window_Welcome;
```

Section Headers

Section Headers should be created at the beginning of major sections of a program, routines, subroutines and/or complicated sections of code. Generally shorter and less detailed than the Program Header, the section header serves to provide some level of understanding about the purpose of specific program code, along with what the code is doing.

Section Header Example

```
libname mydata 'e:\workshops\workshop data';

/*************************************************************/
/** ROUTINE.....: FIRST-BY-GROUP-ROWS                       **/
/** PURPOSE.....: Derive the first (min) row within         **/
/**                each by-group using a subquery.          **/
/*************************************************************/
proc sql ;
  create table first_bygroup_rows as
    select rating,
           title,
           'FirstRow' as ByGroup
      from mydata.movies M1
        where title =
              (select min(title)
                 from mydata.movies M2
                   where M1.rating = M2.rating)
          order by rating, title ;

  select * from first_bygroup_rows ;

quit ;
```

Configuration Management

Configuration Management Purpose

1. Initialize “key” variables (columns)
2. Support structured and/or modular design
3. Construct and use shareable code libraries
4. Enable SAS to systematically handle changes so a program can maintain high integrity over the life of the program
5. Utilize pre-written source code as “callable” routines so that it can be included, as needed, by a calling program. This is sometimes referred to as a “Copybook” process.

Determining SAS Option Settings

SAS Option settings can be examined by:

- **Using the PROC OPTIONS statement**

```
PROC OPTIONS ;
RUN ;
```

- **Viewing the OPTIONS window**
- **Accessing the contents of DICTIONARY.OPTIONS using SQL**

```
PROC SQL ;
  SELECT  *  FROM DICTIONARY.OPTIONS ;
QUIT ;
```

- **Accessing the virtual table SASHELP.VOPTION**

```
PROC PRINT DATA=SASHELP.VOPTION ;
RUN ;
```

SOURCE / NOSOURCE Option

Tells SAS to write all in-stream SAS source code to the SAS Log.

System Option

```
options  SOURCE ;
data libref.processed_movies ;
  set libref.movies ;
  <  other SAS statements  > ;
run ;
```

NOSOURCE2 / SOURCE2 Option

Tells SAS to write all included source code to the SAS Log to produce a comprehensive audit trail.

System Option

```
options  SOURCE2 ;
data libref.processed_movies ;
  set libref.movies ;
  %include  'c:\include-sas-code.sas' ;
  <  other SAS statements  > ;
run ;
```

Coding Conventions

Purpose of Coding Conventions

1. Produces SAS code that is easier to understand.
2. Results in more maintainable code.
3. Produces higher quality and more efficient code.
4. Results in code with greater integrity over its productive life.
5. Enables code to be reused from project to project, department to department, within an organization.
6. Results in code with fewer defects providing greater effectiveness and value.
7. Reduces the time spent on debugging.

Read Only Fields and Data Needed

```
filename  rawdata
        'e:\workshops\workshop data\movies.dat' ;
data  PG_MOVIES ;
   infile  rawdata  missover ;
   input  @1 title $30.
          @32 length 3.
          @36 category $20.
          @88 rating $5. ;
   if rating = "PG" or
     rating = "PG-13" ;
run ;
```

To reduce the size of the input buffer and SAS dataset, read only the fields that are needed from the external file. Any field(s) not listed on the INPUT statement is/are automatically eliminated from being read.

KEEP= / DROP= Data Set Option

```
data  PG_MOVIES ;
   set  libref.movies (keep = title  rating  length) ;
 < other SAS statements > ;
run ;
```

< or >

```
data  PG_MOVIES ;
   set  libref.movies (drop = studio  category  year) ;
 < other SAS statements > ;
run ;
```

The specification of a KEEP= data set option on a SET statement tells SAS that you're only interested in reading the listed variables. Any variable(s) not listed is/are automatically eliminated from the read.

KEEP= / DROP= Data Set Option

```
proc  print  data=libref.MOVIES
                    (keep = title  rating  length) ;
 < other SAS statements > ;
run ;
```

< or >

```
proc  print  data=libref.MOVIES
                    (drop = studio  category  year) ;
 < other SAS statements > ;
run ;
```

A KEEP= data set option can also be specified with most PROCedures in the SAS System. As with the DATA step, any variable(s) not listed is/ are automatically eliminated from the read.

Subsetting Datasets with WHERE=

```
data  PG_MOVIES ;
   set  libref.movies (where=(rating = 'PG')) ;
   < other SAS statements > ;
run ;
```

< or >

```
proc  print  data=libref.movies (where=(rating = 'PG'))
                                                noobs ;
run ;
```

Specifying a WHERE= data set option on a SET statement or a DATA= parameter is a good way to efficiently subset a SAS dataset because it only reads observations into the Program Data Vector (PDV) that satisfy the WHERE-clause expression.

Types of IF Statements

- Subsetting IF
- Simple IF-THEN
- Chains of IF-THENs
- Chains of IF-THEN/ELSEs

Subsetting IF

```
data  PG_Movies ;
   infile  carsdata  'e:\workshops\workshop data' ;
   if rating = 'PG' ;
run;
```

A subsetting IF statement is best used for subsetting in-stream data from non-SAS input files.

Simple IF-THEN

```
data  PG_Movies ;
   set  libref.movies ;
   if rating = 'PG'  then  output  PG_Movies ;
run ;
```

A simple IF-THEN statement can be combined with other SAS statements and operations (e.g., DELETE, DO, OUTPUT, assigning values to variables, etc.).

Chains of IF-THENs

```
data  G_Movies  PG_Movies  PG13_Movies  R_Movies ;
   set  libref.movies ;
   if rating = 'G' then output  G_Movies ;
   if rating = 'PG' then output  PG_Movies ;
   if rating = 'PG-13' then output  PG13_Movies ;
   if rating = 'R' then output  R_Movies ;
run ;
```

Although coding chains of IF-THEN statements is syntactically correct, it forces the SAS System to incur additional processing costs because each IF-THEN statement is executed regardless if the condition was satisfied in an earlier IF-THEN statement.

Chains of IF-THEN/ELSEs

```
data  G_Movies   PG_Movies   PG13_Movies   R_Movies ;
   set  libref.movies ;
   if rating = 'G' then output  G_Movies ;
    else
     if rating = 'PG' then output  PG_Movies ;
      else
       if rating = 'PG-13' then output  PG13_Movies ;
        else
         if rating = 'R' then output  R_Movies ;
run ;
```

A more efficient technique of coding chains of IF-THENs is to insert an ELSE between each IF-THEN, because once a condition is satisfied it branches out of the IF-THEN/ELSE logic construct.

Ordering Chains of IF-THEN/ELSEs

```
data  G_Movies    PG_Movies  PG13_Movies   R_Movies ;
   set  libref.movies ;
   if rating = 'R' then output  R_Movies ;
    else
     if rating = 'PG-13' then output  PG13_Movies ;
      else
       if rating = 'PG' then output  PG_Movies ;
        else
         if rating = 'G' then output  G_Movies ;
run ;
```

Construct IF-THEN/ELSE statements in descending order of occurrence to reduces CPU costs.

OR Operator versus IN Operator

```
data  PG_Movies_with_OR ;
   set  libref.movies (where=(rating = 'PG'   OR
                              rating = 'PG-13')) ;
run ;
```

Versus

```
data  PG_Movies_with_IN ;
   set  libref.movies (where=(rating IN ('PG', 'PG-13'))) ;
run ;
```

Although the OR operator is an acceptable syntax, it isn't as efficient as using the IN operator. The IN operator automatically stops the evaluation of the expression as soon as one value is "true".

Assigning Many Values

```
data  Movies_with_IF_THEN_ELSE ;
   set  libref.Movies ;
   if rating = 'R' then age_grp = '>=17' ;
    else
     if rating = 'PG-13' then age_grp = '>=13' ;
      else
       if rating = 'PG' then age_grp = '>=13' ;
        else
         if rating = 'G' then age_grp = 'All Ages' ;
run ;
```

Although using a series of IF-THEN/ELSE statements to assign many values is acceptable, it can be harder to maintain and increases CPU costs.

Assigning Many Values One Time

```
proc  format ;
  value  $age_grp  “G” = “All Ages"
                   “PG” = “>=13"
                “PG-13” = “>=13“
                     “R”  = “>=17” ;
run ;
data  Movies_with_Age_grp ;
   set  libref.Movies ;
   age_grp = put (rating, $age_grp.) ;
run ;
```

A more maintainable and efficient method to assign many values to other values is to use PROC FORMAT with the PUT function.

TRANSPOSE Procedure Overview

- PROC TRANSPOSE creates an output dataset
- Restructures the values in a dataset
  - ✓ Columns become rows
  - ✓ Rows become columns
- Transposes selected variables into observations
- Often used in lieu of DATA step programming
- PROC TRANSPOSE does not print output
- Can be a very handy method for restructuring the data in a dataset in preparation for use by an array.

Movies Dataset Before Transpose

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

Movies

Transpose Movies Dataset

```
proc transpose data=libref.movies
                        out=transposed_movies ;
run ;
```

Data can be restructured or transformed easily with the TRANSPOSE procedure. In this example, a default (ungrouped) transposition is performed on numeric variables.

Movies Dataset After Transpose

	NAME OF FORMER VARIABLE	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10
1	Length	177	103	97	116	130	105	142	127	125	127
2	Year	1995	1942	1989	1988	1993	1980	1994	1990	1975	1993

	COL11	COL12	COL13	COL14	COL15	COL16	COL17	COL18	COL19	COL20	COL21	COL22
1	110	106	98	115	120	170	118	124	135	108	101	194
2	1987	1997	1983	1982	1976	1983	1991	1977	1989	1984	1939	1997

Transposed_Movies

Without Array Processing

```
data IF_THEN_Example ;
  set transposed_movies ;
  if _NAME_ = 'Length' and col1 < 100 then col1 = . ;
  else
  if _NAME_ = 'Length' and col2 < 100 then col2 = . ;
  else
    . . .     . . .     . . .     . . .
  else
  if _NAME_ = 'Length' and col22 < 100 then col22 = . ;
run ;
```

Without arrays, some tasks require a series of IF-THEN/ELSE statements to be written, making the code harder to maintain.

Array Processing

```
data Array_Example;
  set transposed_movies;
  array a1 (22) col1-col22;
  do i = 1 to 22;
    if _NAME_='Length' and a1(i) < 100 then a1(i) = .;
  end;
run;
```

Arrays can be used to reduce the amount of code that is needed to perform a repetitive task. With arrays, actions on variables can be repeated any number of times and the code is typically easier to maintain.

Assigning a Value to Constant

```
data  PG_MOVIES ;
   set  libref.movies (where=(rating = 'PG')) ;
   group = 2 ;
run ;
```

Avoid using an assignment statement to assign a value to a constant, because it is executed for each iteration of the DATA step.

Assigning a Value to Constant Once

```
data  PG_MOVIES ;
   retain  group  2 ;
   set  libref.movies (where=(rating = 'PG')) ;
run ;
```

< or >

```
data  PG_MOVIES ;
   set  libref.movies (where=(rating = 'PG')) ;
   retain  group  2 ;
run ;
```

Instead, assign values to a constant using a RETAIN statement because it is executed only once.

Coding for Unknown Data Values

```
data  regrouped_movies ;
  set  libref.movies ;
  if rating="R" then group="1" ;
   else
    if rating="PG-13" then group="2" ;
     else
      if rating="PG"  then group="3" ;
       else
        if rating="G"  then group="4" ;
         else group="9" ;
run ;
```

Coding for unknown data values helps to isolate and assign user-defined value(s) to unknown or null data values.

Not Checking for Missing Values

```
data  Movies_with_Trailer ;
   set  libref.movies ;
   new_length = length + 5 ;
run ;
```

The likelihood of propagating missing values increases significantly when no check for missing values is performed before a computation.

Checking for Missing Values

```
data  Movies_with_Trailer ;
   set  libref.movies ;
   if  length NE  .  then  new_length = length + 5 ;
run ;
```

By checking for missing values before performing a computation, the likelihood of propagating missing values significantly decreases.

BY-group Processing with CLASS

```
proc means  data=libref.movies ;
   class  category ;
run ;
```

Specifying a CLASS statement with PROC MEANS is an efficient way to process BY-groups without the need to first perform costly sort operations.

NOTSORTED Option

```
proc print  data=libref.movies  noobs ;
    by category NOTSORTED ;
run ;
```

When the SORT procedure has not been used to sort data in ascending or descending order, but the data already is in the desired order then the NOTSORTED option can be specified in a BY statement.

OUT= Parameter and PROC SORT

```
proc  sort  data=libref.Movies  out=sorted_movies ;
   by  rating ;
run ;
```

To prevent writing over the original SAS dataset with PROC SORT, always specify an OUT= parameter with a different libref and/or a different dataset name.

Saving Labels in a DATA Step

```
data  movies ;
   set  libref.movies ;
   label title = 'Movie Title'
         year = 'Year Movie was Made' ;
run ;
```

Although this is an acceptable method to save labels, formats, informats, and other information as metadata in a SAS dataset to reduce programming time, particularly with ad-hoc scenarios and smaller datasets ... it forces SAS to perform excessive I/O operations and should be avoided.

Saving Labels with PROC DATASETS

```
proc  datasets  lib=work   memtype=data ;
  modify  movies ;
  label title = 'Movie Title'
        rating='Movie Rating'
        length='Movie Length in Minutes'
        year = 'Year Movie was Made' ;
quit ;
```

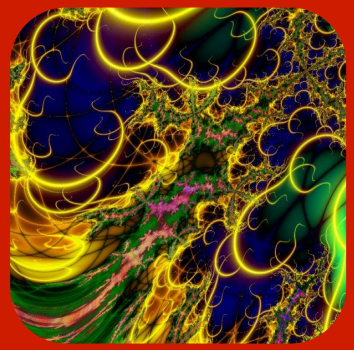
The data management procedure, PROC DATASETS, is a more efficient way to save, update and delete formats, informats, labels and other information as metadata in a SAS dataset.

Conclusion

Good Programming Concepts

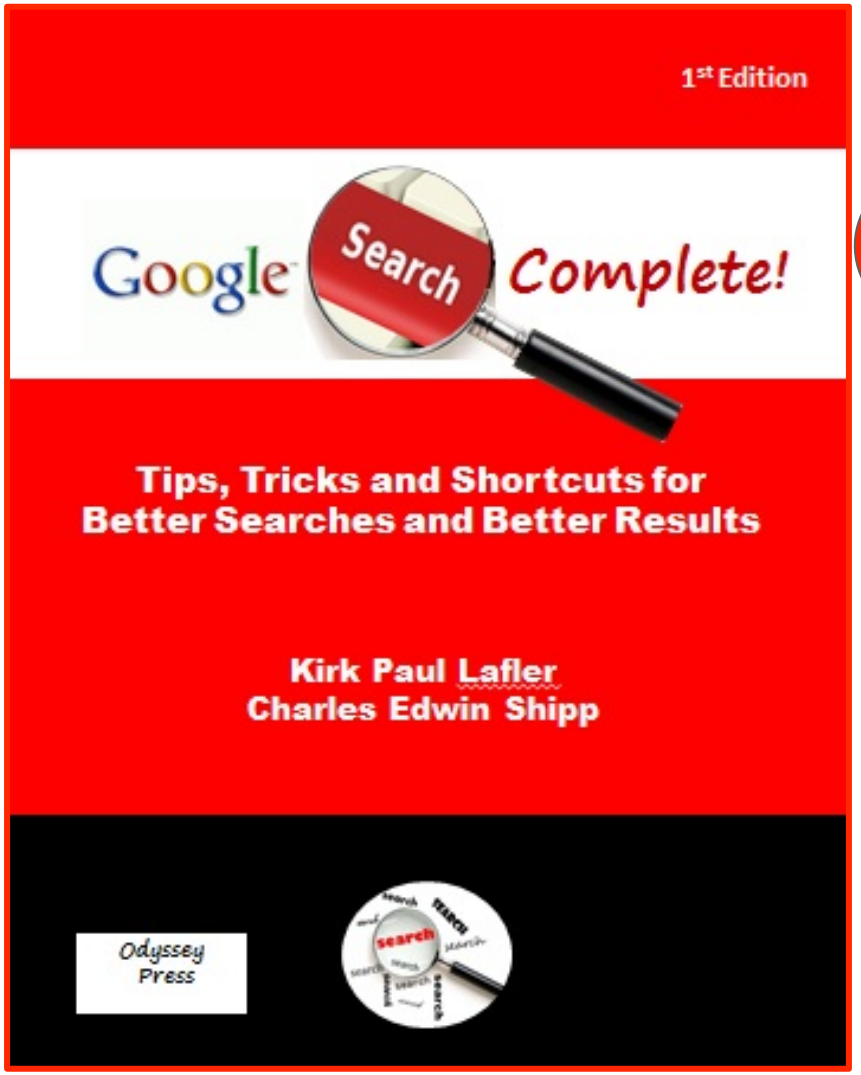
Naming Conventions

Comments and Documentation



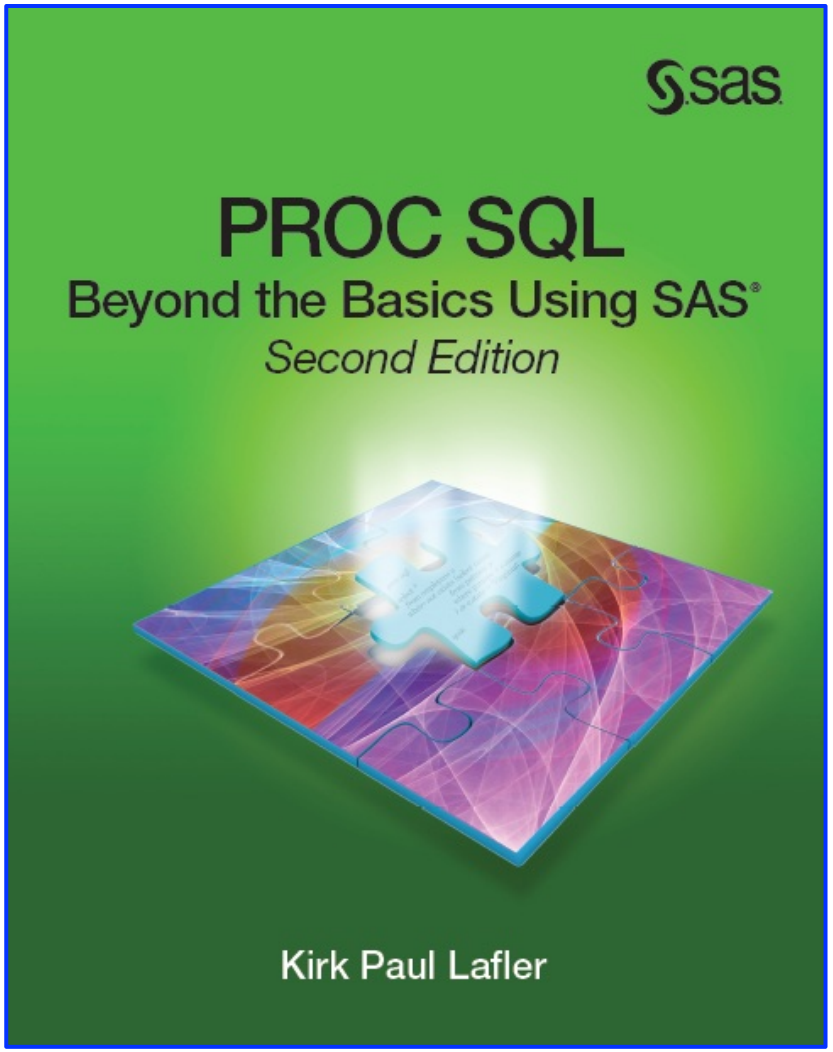
Configuration Management

Coding Conventions



A Google Search Book the Pros use for Better Searches and Better Results!

Available on www.Amazon.com!



An SQL Book with "under the hood" details, explanations and lots of examples

Available from SAS Press!

Thank You for Attending!

Questions?

Kirk Paul Lafler
Senior SAS® Consultant, Application Developer, Data Scientist, Educator and Author
https://www.linkedin.com/in/kirkpaullafler
@sasNerd