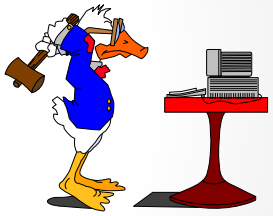
Quick Hits:
My Favorite SAS® Tricks



Marje Fecht
Senior Partner, Prowerk Consulting
SAS Global Forum 2014 Conference Chair

Are you time-poor and code-heavy?

- It's easy to get into a rut with your SAS code and it can be time-consuming to spend time learning and implementing improved techniques.
- This presentation is designed to show you quick improvements that take 5 minutes to learn and about the same time to implement. The quick hits are applicable across (most) versions of SAS and require only BASE SAS knowledge.

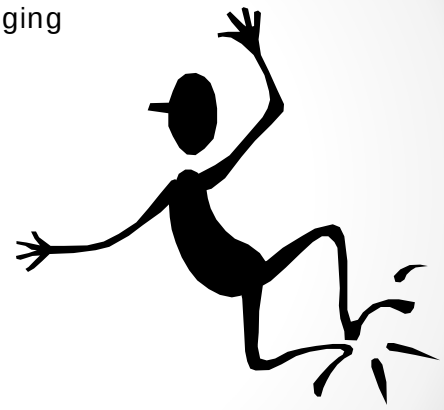
Topics

- Code Reduction
- Code Generalization
- Output Customization
- Data Summarization
- Data and Space Management
- Testing and Validation

Code Reduction: Functions

CAT

- Concatenation and managing *blanks* can be difficult.
- A lot of steps are needed
  - Trim
  - Left
  - Strip
  - | |
  - Adding delimiters
- Replace the messy code - use **CAT** functions.



Code Reduction: Functions

CAT for conCATenation

- Replace ***messy code***

```
old =
   put(n,1.)  || ' ' || trim(a) || ' ' ||
      trim(b) || ' ' || c;
```

- With simplified, easier to read code

```
new = CATX ( ' ',  n , a, b, c);
```

Note: If any of n, a, b, c are ***blank***, since CATX includes ***Strip*** – New will not include the blanks.

CATs – Concatenate easily

The SAS 9 family of CAT functions reduces complexity when concatenating strings!

- CAT concatenates multiple strings in one function call
- CATT same as CAT but also TRIMs
- CATS same as CAT but also strips leading and trailing blanks
- CATX same as CATS but you can specify a delimiter

Code Reduction: Functions

IFC and IFN for Conditional assignment

Consolidate sets of **`if-then-else`** code with a single function call

```
If grade ge 70 then status = "Pass";
  else if grade = . then status = "Pending";
     else status = "Fail";
```

IFC (**IFN**) returns a character (numeric) value based on whether an expression is

- true *result of expression NOT 0 or .*
- false *result of expression is 0*
- missing *result of expression is .*

Code Reduction: Functions

IFC for Conditional assignment

IFC returns a character value based on whether an expression is true(**not 0 or .**), false(**0**), or missing(**.**).

```
Length Status $7;
Status = IFC( grade ge 70 ,
              "Pass" ,   /* TRUE */
              "Fail" ,   /* FALSE = 0 */
              "Pending"  /* Missing */
            );
```

Define Length of new variable when using IFC. Default is $200.

IFC for Conditional assignment

The previous example will only result in Pass and Fail, since the logical expression will only result in True (1) or False(0). BEWARE!!

The below works since

- when grade is missing, the expression is missing
- when grade lt 70, the expression is 0.

```
Length Status $7;
Status = IFC( grade * (grade ge 70) ,
               "Pass" ,   /* TRUE */
               "Fail" ,   /* FALSE = 0 */
               "Pending"  /* Missing */
            );
```


Code Reduction: Functions

Proper Names

Names are notoriously problematic…

mARje fEcHt

O'Neill or o'Neill or O'neill

introducing. . . PROPCASE

Functions - PROPCASE

```
/* default = upcase the first character after
      a blank, forward slash, hyphen,
      open parenthesis, period, or tab
*/
data names1;
   input CapsName $ 1 - 50;

   PropcaseName = propcase(capsname);

   datalines;
EUGENE DELACROIX
GEORGIA O'KEEFFE
NORMAN ROCKWELL
EDWARD BURNE-JONES
INTRODUCTION TO THE SCIENCE OF ASTRONOMY
VIRGIN ISLANDS (U.S.)
SAINT KITTS/NEVIS
WINSTON-SALEM, N.C.
;
```


PROPCASE – default results

CapsName	PropcaseName
EUGENE DELACROIX	Eugene Delacroix
GEORGIA O'KEEFFE	Georgia O'keeffe
NORMAN ROCKWELL	Norman Rockwell
EDWARD BURNE-JONES	Edward Burne-Jones
INTRODUCTION TO THE SCIENCE OF ASTRONOMY	Introduction To The Science Of Astronomy
VIRGIN ISLANDS (U.S.)	Virgin Islands (U.S.)
SAINT KITTS/NEVIS	Saint Kitts/Nevis
WINSTON-SALEM, N.C.	Winston-Salem, N.C.

Functions - PROPCASE

```
/* specify characters (blank, hyphen, single quote)
           for argument 2 */

data names2;
   input CapsName $ 1-50;

   PropcaseName=propcase(capsname, " -'");

   datalines;
EUGENE DELACROIX
GEORGIA O'KEEFFE
NORMAN ROCKWELL
EDWARD BURNE-JONES
INTRODUCTION TO THE SCIENCE OF ASTRONOMY
VIRGIN ISLANDS (U.S.)
SAINT KITTS/NEVIS
WINSTON-SALEM, N.C.
;
```


PROPCASE – override defaults

CapsName	PropcaseName
EUGENE DELACROIX	Eugene Delacroix
GEORGIA O'KEEFFE	Georgia O'Keeffe
NORMAN ROCKWELL	Norman Rockwell
EDWARD BURNE-JONES	Edward Burne-Jones
INTRODUCTION TO THE SCIENCE OF ASTRONOMY	Introduction To The Science Of Astronomy
VIRGIN ISLANDS (U.S.)	Virgin Islands **(u.s.)**
SAINT KITTS/NEVIS	Saint **Kitts/nevis**
WINSTON-SALEM, N.C.	Winston-Salem, **N.c.**

More functions for your toolkit!

Some other functions you may find handy to reduce code:

- **TRIMN** ➔ removes trailing blanks – returns **null** for missing
- **Count**, **CountC** ➔ counts # of occurrences of a string or **C**haracter
- **Index**, **IndexC**, **IndexW** ➔ locates position of first occurrence of string, **C**haracter, or **W**ord
- **Length** (min=1) , **LengthN** (min=**0**) ➔position of last non-blank
- **Largest**(***k*** , *var1, var2, …)* ➔kth largest non-missing value
- **Smallest**(***k*** , *var1, var2, …)* ➔kth smallest non-missing value

Code Generalization

Best Practices: *Reusable Code*

According to wikipedia,

In computer science and software engineering, **reusability** *is the likelihood that a segment of source code can be used again to add new functionalities with slight or no modification.*

Reusable modules

- reduce implementation time,
- increase the likelihood that prior testing and use has eliminated bugs
- localizes code modifications when a change in implementation is required.

Code Generalization

Do your programs include:
*/***** NO CHANGES BELOW HERE *****/*

- Pass all parameters to your source via
  - User Defined Macro variables for "static" information
  - Data-Driven values via metadata or functions
- Generalize location / naming / etc to enable easy changes
  - File locations
  - File names
  - System locations
- Maintain source code that stays "un-touched"

Code Generalization: Log and Output

```
%let requestID = HR123;
%let filepath  = mygroup\myprojectfiles ;
%let stage = prod;            ** could be devl, test, prod **;
... <other program logic – following parameter def'n> ...
%let dir = \&filepath.\&requestID.;
%let filename = &requestID._extract1;
%let datetime = %sysfunc(compress(%sysfunc(today(),
                    yymmddN8.)_%sysfunc(time(),hhmm6.), ': '));
** route Log and Listing to permanent location **;
proc printto
       log ="&dir.\logs\&stage.\&filename._&datetime..log"
       print="&dir.\output\&stage.\&filename._&datetime..lst";
run;
... < other program logic > ...
** reroute to default locations **;
proc printto; run;
```


Code Generalization: Log and Output

The resulting "Versioned" file names would be

- *HR123_20120507_1329.log*
- *HR123_20120507_1329.lst*

Note: The N in **yymmddN8** *requests NO separators (Dashes, Slashes, etc)*

Generalized Code: Dynamic Variable Names

Suppose you need to summarize the latest 3 months of data. And you want monthly variables representing the totals for each month of data.

Your Variable names should reflect the month

- TotAmt_**1205** represents Current Month (May2012)
- TotAmt_**1204** represents one month ago (Apr2012)
- TotAmt_**1203** represents two months ago (Mar2012)

Dynamic Variable Names

You plan to use SQL with CASE to create the monthly buckets.

```
sum(case when txn_date between "&M1_beg"d
                           and "&M1_end"d
         then txn_amt
         else 0 end ) as Tot_Amt_&m1
```

You need macro variables for:

- Beginning Date of each month – **M1_beg**
- Ending Date of each Month – **M1_end**
- YYMM for each month – **M1**

Date Functions

INTNX = *move in intervals*

%sysfunc together with **INTNX** is handy to dymanically create different variations of dates.

INTNX *increments dates by intervals*

INTNX (*interval*, *from*, *n* < , *alignment* >) ;

- ***interval*** - interval name eg: 'MONTH', 'DAY', 'YEAR'
- ***from*** - a SAS date value (for date intervals) or datetime value (for datetime intervals)
- ***n*** - number of intervals to increment from the interval that contains the *from* value
- ***alignment*** - alignment of resulting SAS date, within the interval.

  Eg: **B**EGINNING, **M**IDDLE, **E**ND.

Dynamic Variable Names

Create 3 macro variables that contain **yymm**

```
%let M0 = %sysfunc( today() , yymmN4.);

%let M1 = %sysfunc( intnx( MONTH ,
                  %sysfunc( today() ) ,
                  -1) , yymmN4.);

%let M2 = %sysfunc( intnx( MONTH ,
                  %sysfunc( today() ) ,
                  -2) , yymmN4.);
```

Note: when used in %sysfunc, do not use quotes for the arguments of INTNX.

Dynamic Date values

Create 2 macro variables per month for 1st and last day of month

```
%let M2_beg = %sysfunc( intnx( MONTH
                  , %sysfunc( today() ) , -2 , B)
                  , date9.);
%let M2_end = %sysfunc( intnx( MONTH
                  , %sysfunc( today() ) , -2 , E)
                  , date9.);
```

Note: Use whatever date format is appropriate for your data.

Dynamic Date Values

```
%let M0 = %sysfunc( today() , yymmN4.);
%let M0_beg = %sysfunc( intnx( MONTH , %sysfunc( today() ) , 0 , B)
                        , date9.);
%let M0_end = %sysfunc( intnx( MONTH , %sysfunc( today() ) , 0 , E)
                        , date9.);
%let M1 = %sysfunc( intnx( MONTH     ,  %sysfunc( today() ) , -1)
                        , yymmN4.);
%let M1_beg = %sysfunc( intnx( MONTH , %sysfunc( today() ) , -1 , B)
                        , date9.);
%let M1_end = %sysfunc( intnx( MONTH , %sysfunc( today() ) , -1 , E)
                        , date9.);
%let M2 = %sysfunc( intnx( MONTH     , %sysfunc( today() ) , -2)
                        , yymmN4.);
%let M2_beg = %sysfunc( intnx( MONTH , %sysfunc( today() ) , -2 , B)
                        , date9.);
%let M2_end = %sysfunc( intnx( MONTH , %sysfunc( today() ) , -2 , E)
                        , date9.);
```


Monthly Computations

```
select
     sum(case when txn_date between
                    "&M0_beg"d and "&M0_end"d
               then txn_amt else 0 end ) as Tot_Amt_&m0
    ,sum(case when txn_date between
                    "&M1_beg"d and "&M1_end"d
               then txn_amt else 0 end ) as Tot_Amt_&m1
    ,sum(case when txn_date between
                    "&M2_beg"d and "&M2_end"d
               then txn_amt else 0 end ) as Tot_Amt_&m2
  from amts
  where txn_date between "&M2_beg"d and "&M0_end"d
```

Generalized Code:
Dynamic Code Generation

The code just developed IS dynamic and DOES work.

But – what if we now want 12 months of results?

COPY – PASTE . . .

Notice that the code could easily be generated in a macro loop as long as MaxMonth is defined.

Dynamic Code Generation

The **Macro** variables needed:

Name	**Value**
M0	1205
M0_Beg	1May2012
M0_End	31May2012
M1	1204
M1_Beg	1Apr2012
M1_End	30Apr2012
. . .	
M11	1106
M11_Beg	01Jun2011
M11_End	30Jun2011

***SAS** variables:*

Name

Tot_Amt_1205

Tot_Amt_1204

. . .

Tot_Amt_1106

Dynamic Code Generation

```
%let MaxMonth=11;   /** Months start at ZERO **/
%macro Monthly;
%do Num = 0 %to &MaxMonth;
/*** use %global if macro variables needed outside macro ***/
  %global M&Num. M&Num._beg M&Num._end;
  %let M&Num. = %sysfunc( intnx( MONTH ,%sysfunc( today() )
                         , -&Num.) , yymmN4.);
  %let M&Num._beg = %sysfunc( intnx( MONTH , %sysfunc( today() )
                         , -&Num. , B) , date9.);
  %let M&Num._end = %sysfunc( intnx( MONTH , %sysfunc( today() )
                         , -&Num. , E) , date9.);
%end;
  /*** Similar approach with SQL CASE Statements ***/
%mend Monthly;
%Monthly
%put _user_;
```


Code Generalization: Using Source Modules

Do you suffer from ***Copy-Paste-syndrome?***

Do you find the code you want to use, copy it, and paste it into a new Program?

What do you do when changes are needed? Finding EVERY place you used the code can be tough.

Consider creating SOURCE MODULES that can easily be re-used.

Drivers and Source

Solution: A driver program specifies parameters and calls the standard source modules.

```
*** Driver Program – specify parameters;

%let prestart  = 01SEP2009;
%let prestop   = 31DEC2009;
%let poststart = 01JAN2010;
%let poststop  = 30JUN2010;
%let title = "January 2010 Introduction of
  Claims Changes";
%let plans = ('78','7X','7R','55','85','18');
%let codes = ('015', '119', '214');

*** run standard reporting programs;
%include 'ClaimsReportExtract_20100310.sas';
%include 'ClaimsReportOutput_20100113.sas';
```


Focus on Minimizing Changes!

Only one problem remains...

when the source code changes, you have to find ALL the drivers that call it to change the version.

Solution:

Create a program that calls the latest source!

ClaimsReportExtract_CurrentPgm.sas

```
*** call latest version of source program;
%include 'ClaimsReportExtract_20100310.sas';
```


Focus on Production Code!

When source changes, your drivers always call the **most current version** ☺ .

```
*** Driver Program – specify correct parameters;
%let prestart  = 01SEP2009;
%let prestop   = 31DEC2009;
%let poststart = 01JAN2010;
%let poststop  = 30JUN2010;
%let title = "January 2010 Introduction of
  Claims Changes";
%let plans = ('78','7X','7R','55','85','18');
%let codes = ('015', '119', '214');

*** run standard reporting programs;
%include 'ClaimsReportExtract_CurrentPgm.sas';
%include 'ClaimsReportOutput_CurrentPgm.sas';
```


Reusable Code

This driver-source example is a simple one, but demonstrates modularization and "hands-off" code.

A few further notes:

- There are no limits to the # of %includes you can use
- If you want the code from a %include to display in your log, use the **SOURCE2** option
- In addition to %include, reusable modules may be
  - Macros
  - Format Libraries.

Output Customization: Value Recoding

Multipurpose user-defined Formats
- Handle special cases
- Use SAS-supplied formats otherwise

```
PROC FORMAT;
  value neg_miss
    LOW - <0 = "    Negative"
    .        = "    No Data"
    OTHER    = [dollar12.2]
;
  run;
```

Amt	Fmt_Amt
25000.0	$25,000.00
-5000.0	Negative
.	No Data
12.5	$12.50

Space Management: Best Practice

REUSE SPACE !!

- Your use of SAS Work Space could crash other jobs (and yours too)!
- If you are processing large data files, delete intermediate datasets when they are no longer needed
  - eg: once joins are complete with other data, delete the intermediate data sets.

Space Management:
Delete Data when not needed

```
proc datasets lib = work
              memtype = data  details;
  /** delete old data **/
  delete ChqReport_1a
         ChqReport_1b
         ChqReport_1c ;
quit;
```

Note: Before deleting intermediate data, you should confirm there are no error conditions, etc.

Data Management:
Changing Attributes

You do not have to read data values to

- Rename Datasets or variables
- Change variable labels, formats, names...

```
proc datasets lib = project memtype = data  details;
  /** rename dataset and delete old data **/
  change ChqReport = ChqReport_&lastMonth ;

  /** change variable attributes in ChqExtract **/
  modify ChqExtract;
      rename txns = Transactions
             Date = Txn_Date;
      format TotalAmt Dollar12.;
quit;
```

Testing: Compare Before and After

COMPARE *before and after* data sets to confirm comparability of solutions!

```
*** Useful to compare 2 solutions;
proc compare data=HeartReport
             compare=HeartReport2
;
/** NOTE –compares Obs 1 to 1, 2 to 2, etc **/
run;
```

If you see the following result – Be Happy ☺

```
NOTE: No unequal values were found.
      All values compared are exactly equal.
```


Testing: Isolate variables to compare

```
%let old_dataset = report_20120430;
%let new_dataset = report_20120430;
%let not_same = var19 var20;
%let dir      = mygroup\myprojectfiles;


libname old "&dir.\data\prod";
libname new "&dir.\data\qa";


** compare columns EXCEPT those we expect
     to be different **;
proc compare
     data    = new.&new_dataset.
               (drop = &not_same.)
     compare = old.&old_dataset.
               (drop = &not_same.);
run;
```

Other favorites . . .

- Do you create Excel reports?
  - o Consider **ODS** and **ExcelXP Tagsets** reference papers by **Vince DelGobbo**
- Have you inherited poorly FORMATTED programs?
  - o See SAS Enterprise Guide - **FORMAT CODE**

Thank You !

Marje Fecht
marje.fecht@prowerk.com