PhUSE 2012

Paper PP22

Understanding SAS®..... the SAS-sy way!

Amar Nayak, Synergy, Langley, United Kingdom

ABSTRACT

"A picture is worth thousand words".

I could relate to this adage during my earlier years as I developed logical approaches to SAS® programming to gain strong understanding of it. Creative imagination becomes a powerful way of learning a programming language. We create pictures, objects, shapes, flying or flowing from one place to another, chopping and changing objects to understand the challenging concepts. If a trainer explains the concepts stepwise via reference materials, our micro processing mind continuously thinks of various ways to understand and absorb the theory until it sticks to our memory forever with a 'EUREKA' effect. We have such experiences while working or mulling over a complicated approach. Training materials have a traditional approach but it's rare that the same concepts are explained the non-traditional way.

This paper attempts to provide an illustrative approach to people naive to SAS® and can understand the concepts in layman's language.

INTRODUCTION

Since 2006, Synergy has been proud to run a SAS® Life Sciences Academy graduation programme in collaboration with SAS®, Marlow, UK where in people with very little to absolutely no programming background were interviewed and selected through a personal interview, logical and parity tests. The intention was to provide the pharmaceutical industry people with the required level of skills who are fully trained in SAS® and have built good experience on Statistical/SAS® programming techniques by being part of Synergy for the initial year as soon as they have successfully completed their SAS® certification.

SO WHERE DOES IT ALL START ..

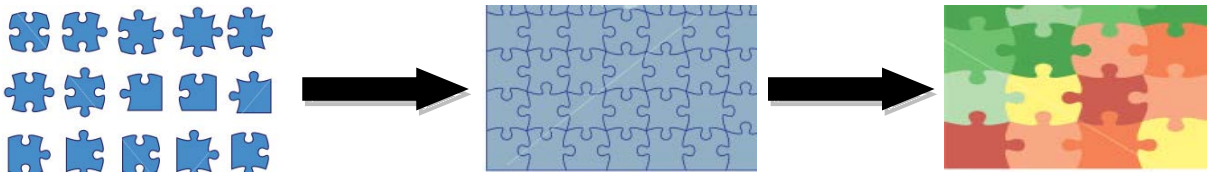
Being novice to the (big) world of SAS®, all of us would have our own way of understanding new concepts or learning a new skill as a professional. Some people are quick enough to grasp and understand the exciting concepts whereas for some a little more effort is required. I am sure every one of us who at some point would have been new to SAS® programming or who have been part of Synergy's SAS® graduation program would have gone through a range of emotions as they took their first step to be a SAS® programming professional.

Curious at DATA steps ➔ **Head-Scratch -ing PROC's** ➔ **Biting nails off at ARRAYS** ➔ **Worrying with MACROS** ➔ **Going pale with SQL** ➔ **and finally!!**

As we always say there is light at the end of the gloomy tunnel, what does the last picture tell you? We all know this to be a 'EUREKA' moment. It all comes into place after a while!

I have often experienced this spontaneous moment as soon as I have hit bull's eye around the wide range of SAS® topics or found a solution to a complicated and innovative approach to programming. Whether it is using DATA steps, PROCs, ARRAY's or SQL, the key to building a good SAS® program was through logic and robustness. Our programs are a mixture of PROCs and DATA steps combined with various statements, arrays or functions for data manipulation. If necessary, the data is chopped and changed using MERGE, SET statements or even transformed using TRANSPOSE procedure or ARRAY's. The intention is here is to explain most frequently used steps in SAS® in a more illustrative way.

Being part of the Statistical Programming industry, day in and day out, we deal with a variety of raw data to create derived/analysis datasets and finally creating tables (either a summary or inferential analysis), listings and figures to make sense of the raw data collected for any clinical trial. For me it is always been like solving a jigsaw puzzle.



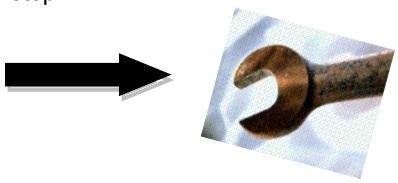
We could only wish that we had a magic wand and say the famous words 'Abracadabra' and hey presto!!... we have raw data (A) transformed to analysis/derived datasets (B) and repeat the same to get all nice and fancy outputs (C) at the end.... job done with a breeze!

However, the reality is that we have to use SAS®. That is the version of a magic wand in reality but not without using our brains ☺ ☺. Right, so where do we begin then,,,, with the first (DATA) step.

THE DATA STEP .

What is really a DATA step?

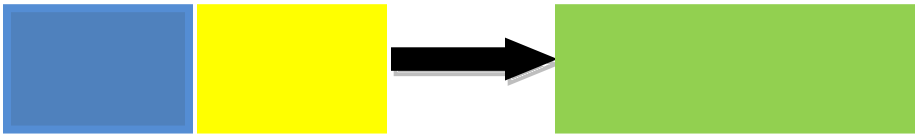
```
data new;
   set old;
run;
```



I relate this to fixing any given dataset using various tools and techniques either using various statements, functions or arrays (i.e. nuts and bolts). Now talking further on DATA steps, until the final derived dataset is achieved, the raw data undergoes chopping and changing either in terms of the number of variables or number of observations.

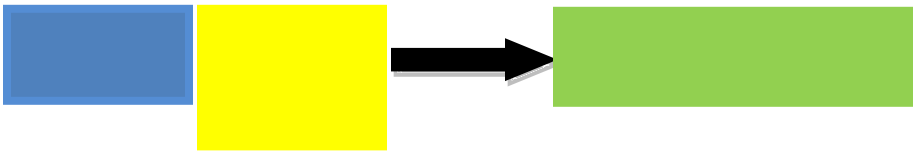
Now the following examples illustrate the execution of a MERGE statement within a DATA step resulting in the changes to the structure of the merging datasets.

```
DATA green
  MERGE blue yellow;
  BY vars...;
RUN;
```



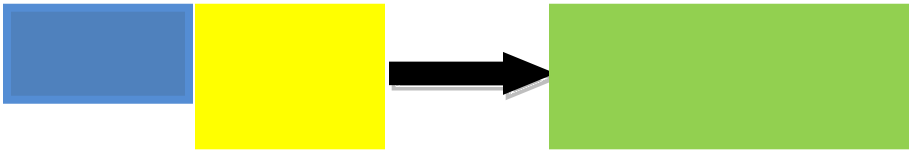
Here BLUE and YELLOW have the same number of observations and hence the resultant GREEN has the same number of observations and variables of BLUE and YELLOW.

```
DATA green
  MERGE blue (in=b)
        yellow;
  BY vars...;
  IF b;
RUN;
```



Here the number of observations in BLUE is less than the number of observations in YELLOW. However as the IN operator is used conditionally against BLUE, GREEN has the same number of observations as BLUE and variables of BLUE and YELLOW.

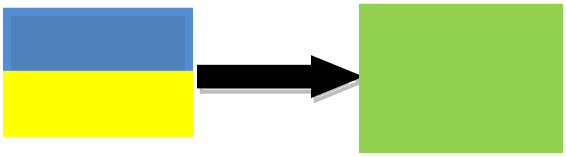
```
DATA green
  MERGE blue
        yellow (in=y);
  BY vars....;
  IF y;
RUN;
```



Here the number of observations in BLUE is less than the number of observations in YELLOW. However as the IN operator is used conditionally against YELLOW, GREEN has the same number of observations as YELLOW and variables of BLUE and YELLOW.

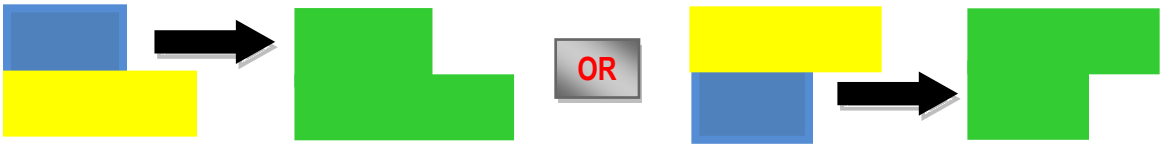
The following example illustrates the execution of a SET statement within a DATA step resulting in the changes to the structure of the union of two datasets with common/identical variables in both.

```
DATA green
  SET blue
      yellow;
RUN;
```



Here BLUE and YELLOW have identical variables and hence the resultant GREEN has similar number of variables and all observations of BLUE and YELLOW.

However, any difference in the number of variables across the two datasets can result in the following shapes. The variables not present in either of the datasets (BLUE or YELLOW) will result in missing values within the resultant GREEN.

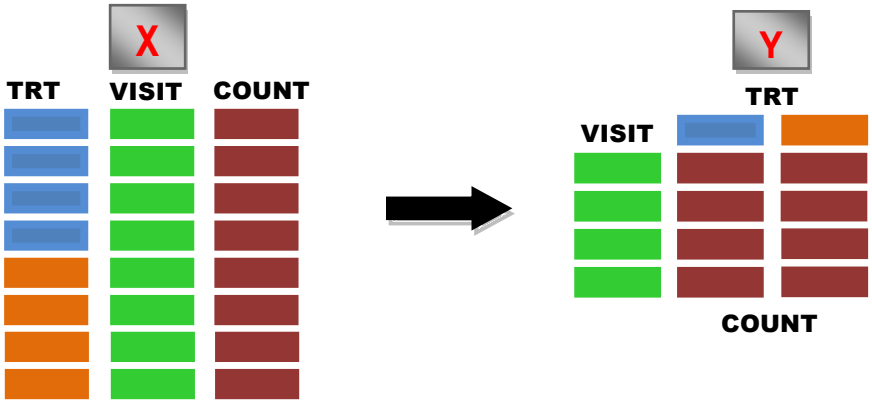


SAS® PROCEDURES

PROC TRANSPOSE

Next moving on to SAS® procedures, a commonly used procedure for data transformation is the **TRANSPOSE** procedure (**PROC TRANSPOSE**). You can see how the data gets transformed by using the example code.

```
PROC transpose data=x out=y
  ID trt;
  BY visit;
  VAR count;
RUN;
```



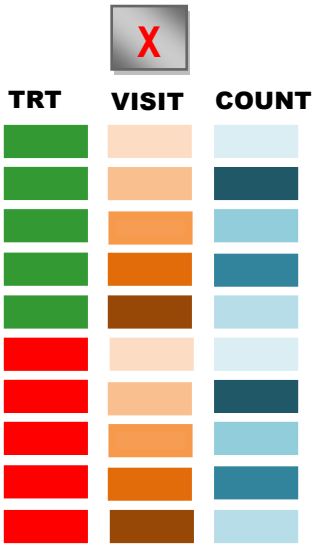
The ID statement within the TRANSPOSE procedure transforms the value of TRT observations in dataset X into two distinct variables in dataset Y, the BY statement retains the order of the VISIT variable and the VAR statement assigns the value of COUNT against the corresponding VISIT observations across the two TRT variables.

One Friday many years ago, a SAS® development manager walked into the office of SAS® president Jim Goodnight and said, "I want ***a raise***!" Jim misunderstood the request and by Monday morning had completed the design and coding for the ARRAY statement. So, what is the moral of the story? The above data transformation can also be achieved by saying "I want ***a raise***!" to do it ☺ ☺

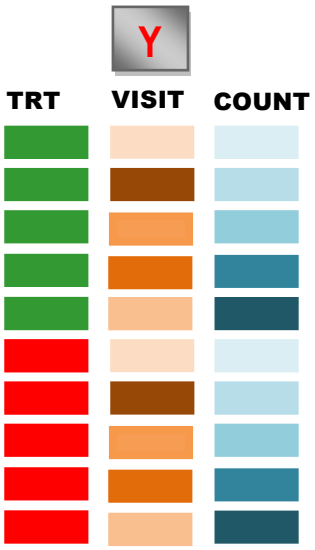
PROC SORT

Let us take an example of **SORT** procedure (**PROC SORT**). Let us assume the first set of blocks to be study treatments (there are two colours), the second set of blocks to be study visit time points (five each for a given treatment) and the third set of blocks to be number of subjects at each of the visits.

The following example demonstrates the changes to appearance of the data when one use the SORT procedure to sort a SAS® dataset with the following variables TRT, VISIT and COUNT. For the latter two variables, as the colour intensity increases so does the value of the variable. Unless specified otherwise, by default, all variables stated within the BY statement of the SORT procedure are arranged in the ascending order.




```
PROC SORT DATA=x;
  BY trt visit count;
RUN:
```




```
PROC SORT DATA=y;
  BY trt count visit;
RUN:
```


```
PROC SORT DATA=z;
  BY trt descending count visit;
RUN:
```

PROC FORMAT

All of us would be able to identify the following pictures and would have certainly associated these when we all started learning the alphabet, our first step to learning the English language. As every alphabet (code) gets associated to an object (decode), the **FORMAT** procedure (**PROC FORMAT**) works on similar lines.

So, if we have understood this in a form of SAS® code it would be follows. As both the code and decode are character values the format name is pre-fixed by a '$' (dollar) sign.

```
PROC FORMAT;
  VALUE $objectf
    'A' = 'APPLE'
    'B' = 'BALL'
    'C' = 'CAT'
    'D' = 'DOG'
    'E' = 'ELEPHANT'
    'F' = 'FOX'
    'G' = 'GLASS'
    ;
RUN;
```

Apart from the alphabet, something more we would have started learning by using our fingers at a very tender age.

Now if we have to represent these numbers within a format then it would be as follows. For a numeric (code) to character (decode) conversion, the format name does not require the '$' dollar (pre-fix) sign.

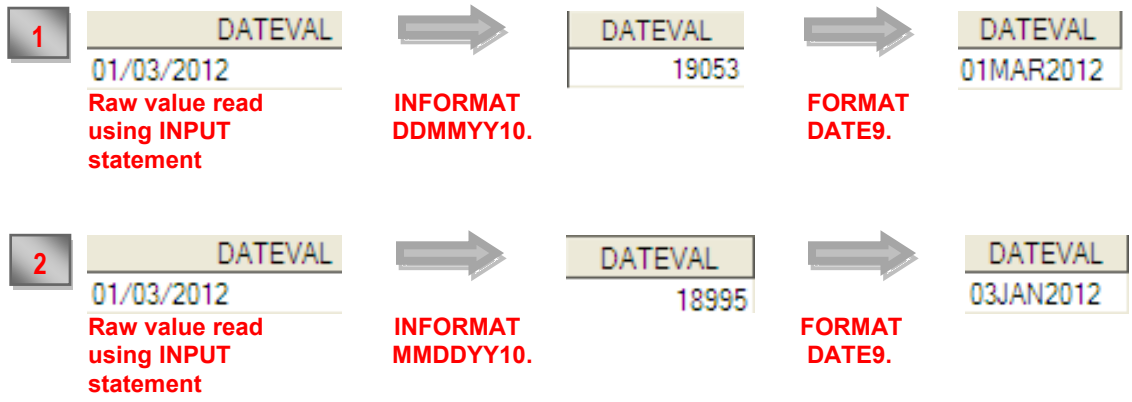
```
PROC FORMAT;
  VALUE numf
    1 = 'ONE'
    2 = 'TWO'
    3 = 'THREE'
    4 = 'FOUR'
    5 = 'FIVE'
    6 = 'SIX'
    ;
RUN;
```

There is a greater scope of using the FORMAT procedure within SAS®. The examples discussed above are to understand the very basic usage of this procedure.

SAS® STATEMENTS

INPUT, INFORMAT and FORMAT

Say a value is put on the drawing board as **'01/03/2012'**. Most people will interpret this value as a date of 1st March 2012. However, few others would say it is 3rd Jan 2012 huh?? How can the same value have two different interpretations? Based on a given geographical location, a date can be defined in British or US format. All of us as SAS® programmers would have come across this situation. So, while working on any clinical trial data, how can we be sure about the date format while interpreting the values similar to stated above. The clinical trial CRF which records all clinical data should be able give us the answer whether it is being recorded as DD/MM/YYYY and MM/DD/YYYY . which leads me to the point of how SAS® will be able create a variable (**INPUT**) by reading such value (**INFORMAT**), and represent it in a meaningful way (**FORMAT**). The pictures below will help understand it.



As you can see from the images above, for one value of **'01/03/2012'**, after reading the source value, it ends up having two different resultant numeric values depending on the INFORMAT definition (DDMMYY10. or MMDDYY10.). However, if you notice when the INFORMAT statement is applied, a numeric value gets created. FORMAT statement is then used to give a standard definition to the date value in DDMMMYYYY (DATE9.) format.

The example code of 1 and 2 is as follows

1
```
DATA x;
  INPUT dateval;
  INFORMAT dateval DDMMYY10.;
  FORMAT dateval DATE9.;
  DATALINES;
  01/03/2012
  ;
RUN;
```

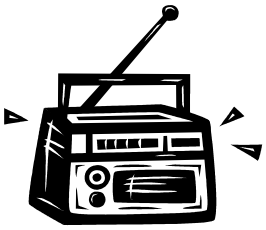
2
```
DATA y;
  INPUT dateval;
  INFORMAT dateval MMDDYY10.;
  FORMAT dateval DATE9.;
  DATALINES;
  01/03/2012
  ;
RUN;
```

So to summarise, the INPUT statement creates a SAS® variable and the INFORMAT statement reads the external data into the SAS® variable and instructs SAS® to interpret the data value. The FORMAT statement then instructs SAS® to write data values and control written appearance of it.

Raw value ⇒ **Create a SAS® variable using INPUT statement** ⇒ **Reading raw value using INFORMAT statement** ⇒ **Once value is read standardizing it with FORMAT statement**

DO LOOPS

Given that we touched base on ARRAYS earlier, it cannot function without the use of **DO LOOPS**. One of the most under-rated and underutilized are **DO UNTIL** and **DO WHILE**. So what are the functionalities of both and the key difference between the two?



What action will be performed repeatedly if I say DO WHILE you hear the music on the radio and DO UNTIL the radio is switched off? Yes, dancing on the floor which becomes an iterative step of the DO LOOP.

In simple words, for the DO WHILE loop, dancing on the floor will only happen if the condition (of the music on the radio) is true. So while the music is on, dancing on the floor will continue to happen. Whereas in the case of DO UNTIL, dancing on the floor will continue to happen until the condition (of the music on the radio) is not true i.e. until the radio is switched off. So essentially.....

Do While The Condition = True

is same as

Do Until The Condition = False

The following examples should help understand the concepts from SAS® code point of view.

DO WHILE
```
DATA x;
  x=1;
  DO WHILE (x le 1);
    x=;
    x + 1;
  END;
  PUT 'DO WHILE x= ' x=;
RUN;
```

DO UNTIL
```
DATA y;
  y=1;
  DO UNTIL (y gt 2)
    y=;
    y + 1;
  END:
  PUT 'DO UNTIL y= ' y=;
RUN;
```

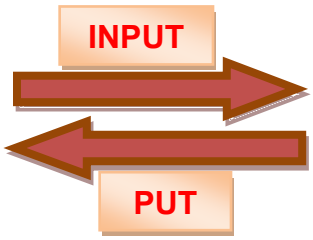
SAS® FUNCTIONS

Now let us talk about a bit about functions. The most common ones that are used in any data manipulation techniques are the **PUT ( )** (numeric to character conversion) and **INPUT ( )** (character to numeric).

Now you might be aware that using the former function one can store a numeric value as character within a SAS® dataset. Say if we have a dataset with just two variables as X and Y, how can one identify which of these variable is numeric or character without even looking at the variable properties. The hint is the alignment of the values. Any SAS® dataset has only two possibilities of storing any value i.e. as numeric or character. Any numeric variable will have its value right aligned and any character variable will have its value left aligned within a SAS® dataset. Hence, personally I have been able to associate the INPUT and PUT functions with the 'shift' in the alignment of values when the data conversion happens, a right directed (character to numeric) and a left directed arrow (numeric to character) for INPUT and PUT function respectively. I could clearly remember this pair of functions and their functionalities by relating to the following road hazard sign a.k.a, the numeric to character and character to numeric conversion sign.

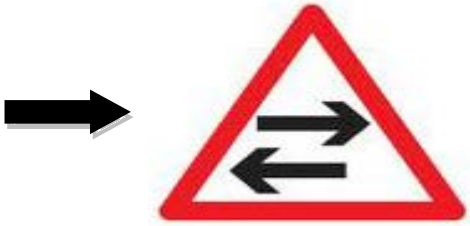
```
y = PUT(x, best12.);
```

Y
1
2
3
4
5




```
x = INPUT(y, best12.);
```

X
1
2
3
4
5



Now can anybody guess which SAS® functions can mend the broken heart?


```
brokehrt
mendhrt = COMPBL(brokehrt);
hrtbeat = COMPRESS(mendhrt);
```

Looking at the second picture we are almost but not quite there, the **COMPBL( )** function eliminates extra spaces between any two character values but leaves one space at the end between any two character values. So what function can actually make the heart beat again? Yes!!.... **COMPRESS ( )**. The function by default if used without any arguments will eliminate all spaces existing between any two character values.

ONE THING SAS® CAN'T DO WITHOUT

We have discussed several functionalities of SAS®. Now let me test your knowledge with just one question. What is the lone thing that SAS® can't do without? You would have guessed it from the 'running' image. Any DATA STEP or a SAS® procedure cannot execute without a '**RUN;**' statement. It would be like using a gun without a bullet. You cannot hit bullseye!!

CONCLUSION

Learning a programming language is never easy. It takes time and lots and lots of practice as it is a gradual learning curve to gain expertise. As a beginner, there are several ways to get familiar and raise confidence within yourself 1) by going through sample codes within reference materials or even on previous projects 2) not just reviewing or looking at the sample codes but by actually running them and observing how data changes step by step 3) creating your own program from scratch, the first one will always be a humongous task but pass the first test and you will conquer the rest.

I am sure there would be several different and better ways to explain SAS® programming concepts. However, this paper is limited to the basic steps, procedures, statements and functions of SAS® to give a flavor of the programming language. Also it is purely based on how I went around searching for simple practical examples that helped me to understand SAS® in the best possible way and never to forget any of these concepts in my lifetime!

REFERENCES

http://www.globalstatements.com/back/saswho.html

http://www2.sas.com/proceedings/forum2007/067-2007.pdf

http://www.sas.com/offices/NA/canada/downloads/presentations/Calgary2008/SAS_Formats.pdf

ACKNOWLEDGMENTS

I am grateful to my non-stop processing brain for churning out practical ways in an attempt to simplify the challenging concepts in the hope that anybody who might read this paper don't risk thinking non-stop to understand whatever has been said in this paper! Excuse me for my SAS-syriority complex ☺ ☺

Last but not the least, I would like to thank my wonderful colleagues who reviewed the paper and provided valuable inputs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Amar Nayak
Biometrics Manager
Synergy
10, Waterside Drive,
Buckland House,
Langley Business Park,
Langley, SL3 6EZ,
Berkshire, United Kingdom
Email: amar.nayak@synergyoutsourcing.com
Web: www.synergyoutsourcing.com