PharmaSUG 2017 - Paper BB06

Good artists copy; Great artists steal; (implement pseudo R ,Jquery code in Base SAS® without installing R or other Applications

Hui Liu, Merck Research Laboratories (China)

ABSTRACT

There are some syntax sugars in other languages we are dreaming of using in SAS. Why? Because they are more expressive. To borrow these syntax sugars usually we install other languages and use them in the background by passing X command to them or relying on some modern SAS modules such as PROC IML to link them together.

The downside of these ways is that the code is no longer the pure SAS code and when we pass SAS code mixed with R code to FDA they have to use a computer with both SAS/IML and R installed to regenerate results. We aren't supposed to carry a cow everywhere when we only want to drink a cup of milk.

So, our goal is to STEAL these syntax features from other languages and implement them in pure SAS BASE/macro as pseudo code. The philosophy of pseudo code is "what you see is what you get, almost".

Examples of code syntax we want to steal (include but not limited to)

a. R-like "by" function to execute BY group analysis on macros (without BY option inside/ without source code).

```
%_(%str( by(sashelp.class,sex,MacroWithNoByOption(sashelp.class))));
```

b. Jquery style wildcard to manipulate variables.

```
Data new;
   %_(%str(
   set old;
   /*Impute value for variables ended with "TERM"*/
   if $("*term") ='' then $("*term") ='N/A';

   /*drop all variables look like ae*flag*/
   drop #("ae*flag");

   /*rename variables with a pattern */
   rename $("gr*n")=group_$("gr*n");

/*inherit format/label from variable in other dataset*/
   format money %_($(sashelp.cars.invoice.format));
   Label size=%_($(sashelp.cars.engineSize.label));
   ));
run;
```

INTRODUCTION

This paper is going to help SAS programmers to understand how to steal syntax from other languages and implement in SAS BASE. And also the SAS macro knowledge is required. After the reading of this paper you should learn how to extend the SAS syntax by pure SAS BASE and Macro utilities and to gain the benefits of these syntax sugars.

SAS LANGUAGE SYNTAX HISTORY

SAS language's syntax is in a long history. During the forty years since its birthday there have been many other programming languages appearing or evolving. Some of them have attractive syntax sugars and we are dreaming of using them in SAS. Let's ask ourselves a question first. Why? Why do we need something new? The answer is : Because they are more expressive therefore it is more productive. It is the ultimate goal for any programming language that is the reason we need them in the first place.

Like other natural languages such as English or Chinese there is supposed to have new words added into vocabulary from other cultures gradually. But also as natural languages these words mainly are not added by authorities or educational committee. It is decentralized and emerging non-forcefully. Therefore we cannot expect every movement comes from SAS Institute and on the other hand the new features added by SAS Institute might not act exactly as you expect it should.

WHAT ARE THE PROBLEMS WE ARE TALKING ABOUT?

Now let’s take a look of what are the problems we are aiming to resolve.

PROBLEM 1

There is a SDTM dataset and you want to create a temp dataset with only subjid and all numerical group variables and rename them to newXXXX. In CDISC standard the variables should all look like *gr*n such as agegr1n, sexgr2n.

The first instinct is to use SAS array, however there is no syntax in SAS to create an array with a rule instead of using plain variable names unless there is a fixed string in the beginning of variables like aeXXXX then you can use ae: to represent them in an ARRAY. But in our case these variables do not start with the same string. And even though you still cannot rename array as ARR=somethingARR.

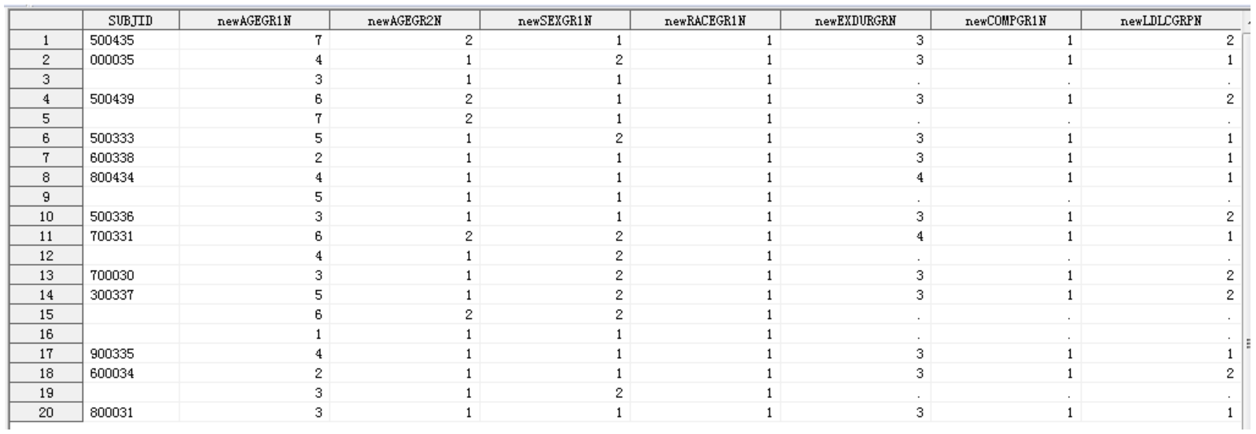
```
data group;
  set sasuser.adsl;
  array grpArr *gr*n;  %* illegal statement;
  keep grpArr ;  %* illegal statement;
  rename grpArr=newGrpArr;%* illegal statement;
run;
```

So, usually a SAS programmer will do the following things.

1. Get variable information from sashelp.vcolumn or dictionary datasets.
2. Filter out variable names in this pattern.
3. Construct some SAS statements in macro variables and use them back in the code.
   **** I will not paste these code here as you all know how to do or at least you can google them out.

But the code can be very neat like below if you steal the Jquery syntax and implement into SAS.

```
%* trust me, they are legal statements;
data group;
  %_(%str(
  set sasuser.adsl;
  keep subjid #("gr*n");
  rename $("gr*n")=new$("gr*n");
  ));
run;
```



	SUBJID	newAGEGR1N	newAGEGR2N	newSEXGR1N	newRACEGR1N	newEXDURGRN	newCOMPGR1N	newLDLCGRPN
1	500435	7	2	1	1	3	1	2
2	000035	4	1	2	1	3	1	1
3		3	1	1	1	.	.	.
4	500439	6	2	1	1	3	1	2
5		7	2	1	1	.	.	.
6	500333	5	1	2	1	3	1	1
7	600338	2	1	1	1	3	1	1
8	800434	4	1	1	1	4	1	1
9		5	1	1	1	.	.	.
10	500336	3	1	1	1	3	1	2
11	700331	6	2	2	1	4	1	1
12		4	1	2	1	.	.	.
13	700030	3	1	2	1	3	1	2
14	300337	5	1	2	1	3	1	2
15		6	2	2	1	.	.	.
16		1	1	1	1	.	.	.
17	900335	4	1	1	1	3	1	1
18	600034	2	1	1	1	3	1	2
19		3	1	2	1	.	.	.
20	800031	3	1	1	1	3	1	1

PROBLEM 2

inherit format,label from another variable in another dataset.

If you are working on CDISC data for SDTM and ADAM datasets you must have the headache when you are working on the manipulation of them by creating and merging multiple datasets, from time to time you are annoyed by the warning like this.

```
data adae;
   merge adsl adae;
   by subjid;
run;
```

```
WARNING: Multiple lengths were specified for the BY
variable subjid by input data sets. This might
cause unexpected results.
```

So, we’d better to make the format and length and even label consistent between datasets.

You might need to use format or label statement in a dataset but how to get this information? One way if manually hardcode it like `format subjid $8.;` but if this attribute information is still under development and not finalized then it is troublesome you need to modify it again and again in your code if it is changed. And another cliché approach is to use a template dataset when creating target dataset but if any of the dataset is not a standard dataset you might not have a template dataset to use. Such as you are merging adsl with a tempAE there is no template tempAE dataset out there. So, the normal practice is still using dictionary or sashelp dataset to get variables format/label and apply them on dataset through macro variables. The below is the traditional approach.

```
proc sql;
  select LENGTH INTO :subjid_length
  from sashelp.vcolumn
  where libname='WORK' and memname='ADSL' and UPCASE(name)='SUBJID'
  ;
  select LENGTH INTO :trtp_length
  from sashelp.vcolumn
  where libname='WORK' and memname='ADSL' and UPCASE(name)='TRTP'
  ;
quit;
```

```
data adae;
  length subjid $ &subjid_length  trtp $ &trtp_length;
  set adae;
run;
```

However if you can steal syntax from object-oriented language you can make a short version without repeat PROC SQL again and again on different variables and attributes.

```
/*inherit format/label from variable in other dataset*/

data adae;
  length subjid $%_($(adsl.subjid.length)) trtp $%_($(adsl.trtp.length));
  set adae;
run;
```

PROBLEM 3

There is a macro you want to use but this macro has no “by option” and you don’t have source code or are not allowed to modify it. So, should we give up other's efforts and re-invent the wheel? Let’s see what we can learn from other languages. R is famous as vectorized operation.

For example R function “by” can apply a mean function on data by a variable BYVAR. That looks straightforward.

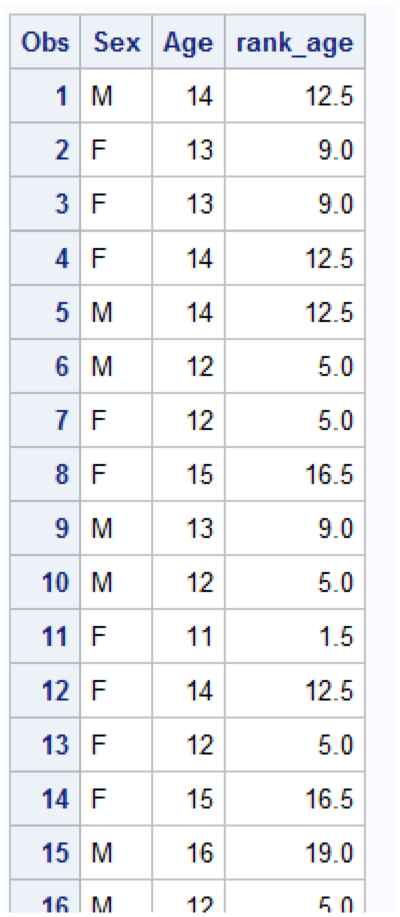
by(data, byvar, mean);

So let's take it from there and implement in SAS.

```
/*This is a macro without any by option in it*/
%macro complicatedMac(dsn,var);
proc rank data=&dsn(keep=&var sex) out=rank;
  var &var;
  ranks  rank_&var;
run;

proc print;run;
%mend;

%complicatedMac(sashelp.class,age);
```



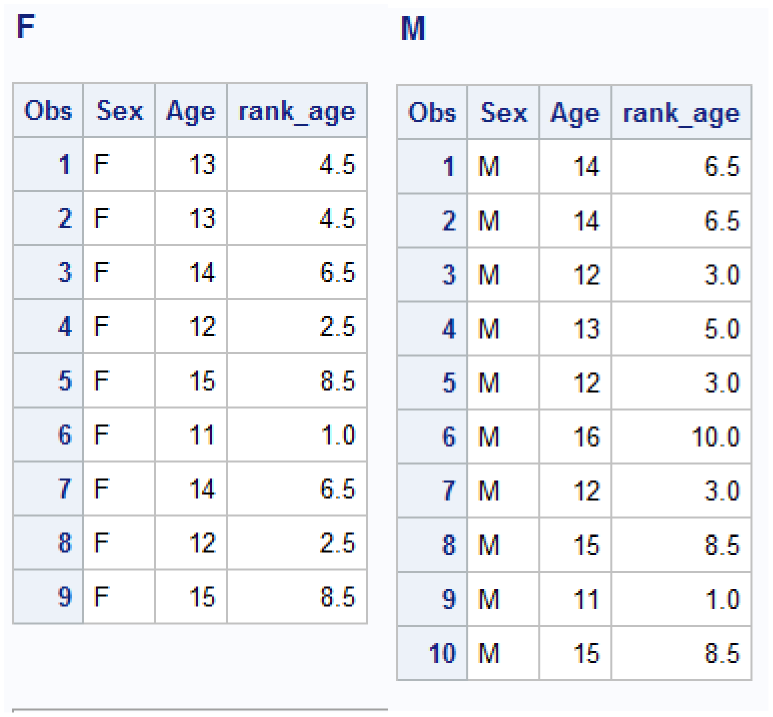
Obs	Sex	Age	rank_age
1	M	14	12.5
2	F	13	9.0
3	F	13	9.0
4	F	14	12.5
5	M	14	12.5
6	M	12	5.0
7	F	12	5.0
8	F	15	16.5
9	M	13	9.0
10	M	12	5.0
11	F	11	1.5
12	F	14	12.5
13	F	12	5.0
14	F	15	16.5
15	M	16	19.0
16	M	12	5.0

After the steal of the syntax we can use the below call to get the BY analysis done.

a. You do not have to modify the original macro with or without source code.
b. BY style syntax applies to any macro on single dataset.

```
%_(by(sashelp.class,sex,%str(complicatedMac(sashelp.class,age))));
```

This is the BY group output without too much effort.



F

Obs	Sex	Age	rank_age
1	F	13	4.5
2	F	13	4.5
3	F	14	6.5
4	F	12	2.5
5	F	15	8.5
6	F	11	1.0
7	F	14	6.5
8	F	12	2.5
9	F	15	8.5

M

Obs	Sex	Age	rank_age
1	M	14	6.5
2	M	14	6.5
3	M	12	3.0
4	M	13	5.0
5	M	12	3.0
6	M	16	10.0
7	M	12	3.0
8	M	15	8.5
9	M	11	1.0
10	M	15	8.5

Conclusion

Why is this macro UNDERSCORE so omnipotent?

It is because " We are standing on the shoulders of giants " what features other languages have we take. Whatever they grow we steal. I call it another version of "Winner takes all".

Get back to the code itself.

How do we construct this interesting macro?

1. Choose a good name for your macro. _ is not interesting but short and non-distracting.
2. Make it accepts arbitrary strings as parameter, not any fixed number of normal positional or keyword-style parameter.
3. Create a series of rules (features stolen from R, Jquery, object-oriented programming language)
4. Detect the pattern and decide which rule to apply.
5. Parse the key parameters and pass it to the REAL SAS code behind of them.
6. Enjoy the saved hours.

EXAMPLE CODE OF MACRO _ FOR PROBLEM #2

There is a macro you want

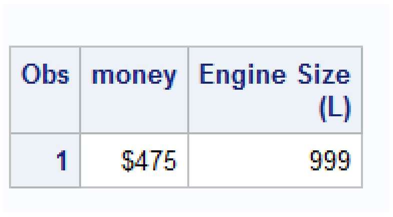
```
%macro _(str)  ; %* #1 name ;
  %if
%sysfunc(prxmatch(m/^\s*\$\((\w+\.)?(\w+)\.(\w+)\.(\w+)\)\s*$/i,%bquote(&str))) %then
    %do;  %* #2 #3 #4;
    %local   dsn var prop myprop;
    %let dsn=
%sysfunc(prxchange(s/^\s*\$\((\w+\.)?(\w+)\.(\w+)\.(\w+)\)\s*$/\1\2/i,1,%bquote(&str))
) ;
    %let var= %sysfunc(prxchange(s/^\s*\$\((\w+\.)?(\w+)\.(\w+)\.(\w+)\)\s*$/\3/i,1
,%bquote(&str))) ;
    %let prop=%sysfunc(prxchange(s/^\s*\$\((\w+\.)?(\w+)\.(\w+)\.(\w+)\)\s*$/\4/i,1
,%bquote(&str))) ;
    %let myprop=%AHGprop(&dsn,&var,&prop)  ; %*   #5   #6;
    &myprop
    %end;
%mend;

%macro AHGprop(dsn,var,prop);
%local i tableid propvalue rc;
%macro AHGeq(one,two);
  %upcase(&one) eq %upcase(&two)
%mend;
%if %sysfunc(exist(&dsn)) %then
%do;
%let tableid=%sysfunc(open(&dsn,i));

%do i=1 %to  %sysfunc(attrn(&tableid,nvars));
   %if %AHGeq(&var,%sysfunc(varname(&tableid,&i))) %then
     %do;
     %if %AHGeq(&prop,format) %then  %let propvalue=%sysfunc(varfmt(&tableid,&i));
     %else %if %AHGeq(&prop,label) %then  %let
propvalue=%sysfunc(varlabel(&tableid,&i));
     %else %if %AHGeq(&prop,length) %then
        %do;
        %let propvalue=%sysfunc(varlength(&tableid,&i));
        %if %sysfunc(vartype(&tableid,&i))=C %then %let propvalue=$&propvalue;
        %end;
     %end;
%end;
%let rc=%sysfunc(close(&tableid));
%end;
&propvalue
%mend;


data ahuige;
  format money %_($(sashelp.cars.invoice.format));
  Label size=%_($(sashelp.cars.engineSize.label));
  money=475;
  size=999;
run;
proc print label;run;
```

The below result is the output of example code for referring attributes in other datasets. It is part of the code in the omnipotent macro and trust me: if you can write stolen syntax features beautifully one by one, you can build a magnificent macro.



Obs	money	Engine Size (L)
1	$475	999

REFERENCES

SAS HELP for BASE

SAS HELP for Macro Utility

ACKNOWLEDGMENTS

Greg Zhou: my boss at Merck who sponsors my trip to this conference from China.

Peng Wan: my colleague who reminded me about PharmaSUG 2017 and suggested that I submit a paper.

Everyone who discussed SAS programming with me during the years.

CONTACT INFORMATION

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

Hui Liu
Assoc. Dir, Stat. Programming, Merck Research Laboratories (China)
ahuige@ymail.com hui.liu10@merck.com
Facebook: https://www.facebook.com/hui.liu.169
Wechat: dokomono