**ABSTRACT:**

This paper provides a collection of basic programming tips and techniques that SAS® users can implement in their daily programming. This collection of tips should be useful immediately and will improve the efficiency of the programs. There are several examples organized by the following categories (Keywords: BASE STAT FUNCTIONS):

- Read and write data selectively
- Concise coding techniques
- Effective use of sorting techniques
- Data manipulation
- Macros (tips for the beginner)

**READ & WRITE DATA SELECTIVELY:**

**Example 1:**
Use the KEEP= or DROP= on the SET or MERGE statement to keep unneeded variables out of the Program Data Vector (PDV) (the storage area for variables, values, and attributes).

<table>
<thead>
<tr>
<th>Acceptable</th>
<th>More Efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>data small;</td>
<td>data small;</td>
</tr>
<tr>
<td>set large;</td>
<td>set large(keep=a b c);</td>
</tr>
<tr>
<td>keep a b c prod lrgst;</td>
<td>prod = a*b;</td>
</tr>
<tr>
<td>prod = a*b;</td>
<td>lrgst = max(a,b,c);</td>
</tr>
<tr>
<td>lrgst = max(a,b,c);</td>
<td>More SAS® statements;</td>
</tr>
<tr>
<td>More SAS® statements;</td>
<td>run;</td>
</tr>
</tbody>
</table>

**Example 2:**
Produce all subsets you require for further processing in one step to minimize the # of times a large dataset is read in.
### Acceptable:

```sas
data one;
set large;
if a < 10;
run;
data two;
set large;
if 9 < a < 90;
run; (similar for data three;)
```

### More Efficient:

```sas
data one two three;
set large;
if a < 10 then output one;
else if 9 < a < 90 then
  output two;
else if a > 89 then
  output three;
run;
```

### Example 3:

Read an existing SAS® dataset and subset it based on values of one (or more) of the variables. Use a where instead of an if....

### Acceptable:

```sas
data new;
set old;
if age > 65 and gndr = ‘F’;
More SAS® statements..;
run;
```

### More Efficient:

```sas
data new;
set old (where=( age > 65 and gndr = ‘F’));
More SAS® statements..;
run;
```

### CONCISE CODING TECHNIQUES:

#### Example 1:

Execute only the necessary statements – move a “subsetting” if before statements you only want to execute after the condition is met.

### Acceptable:

```sas
data elders;
set demog;
age = date2 – date1;
relday = date3 – date2;
tot_rslt =
  sum(rslt1,rslt2,rslt3);
More SAS® stmnts.;
if age > 65;
```

### More Efficient:

```sas
data elders;
set demog;
age = date2 – date1;
if age > 65;
  relday = date3 – date2;
tot_rslt =
  sum(rslt1,rslt2,rslt3);
More SAS® stmnts.;
```
Example 2:
Execute only the necessary statements – when “if” conditions are mutually exclusive, use if-then-else instead AND move most likely condition to top of logic.

**Acceptable:**

```sas
data rsn_dscn;
set pat_stat;
if upcase(resn_dsc) in ('Reason 1', 'Reason 2') then code_dsc = 1;
if upcase(resn_dsc) in ('Reason 3', 'Reason 4') then code_dsc = 2;
if upcase(resn_dsc) in ('Reason 5', 'Reason 6') then code_dsc = 3;
run;
```

**More Efficient:**

```sas
data rsn_dscn;
set pat_stat;
if upcase(resn_dsc) in ('Reason 5', 'Reason 6') then code_dsc = 3;
else if upcase(resn_dsc) in ('Reason 3', 'Reason 4') then code_dsc = 2;
else if upcase(resn_dsc) in ('Reason 1', 'Reason 2') then code_dsc = 1;
run;
```

Example 3:
Take advantage of boolean logic [expressions evaluate to true (0) or false (1)] to assign values instead of using if-then-else logic.

**Acceptable:**

```sas
data survey;
set survey;
if age <= 25 then agegr = 1;
else if age <= 40 then agegr = 2;
else agegr = 3;
run;
```

**More Compact:**

```sas
data survey;
set survey;
agegr =
  (age <= 25) +
  2*(((age > 25) and (age <=40)) +
  3*(age > 41));
run;
```

Example 4:
Use select statements instead of if-then-else when many mutually exclusive conditions exist.
**Method 1 (no select expression):**

```sas
data new;
set old;
selct;
  when (temp < 32 )
    code = 1;
  when (temp < 80 )
    code = 2;
  when (temp < 110 )
    code = 3;
  otherwise code = 4;
end;
More SAS® statements ...;
run;
```

**Method 2 (a select expression):**

```sas
data new;
set old;
select (cat);
  when ('A' ) code = 1;
  when ('B') code = 2;
  when ('C') code = 3;
  otherwise code = 4;
end;
More SAS® statements; run;
```

---

**EFFICIENT SORTING:**

**Example 1:**

Use the WHERE= option on the PROC SORT to reduce unnecessary observations.

*NOTE:* By default, the dataset in the data = option is replaced by the sorted version. Therefore, to keep the integrity of your original dataset, get into the habit of using the OUT = option.

### Acceptable:

```sas
proc sort data = adverse_events;
  By intensity;
run;
```

### More Efficient:

```sas
proc sort data = adverse_events
  where = (relation = 'Y'))
  out = relatedAE;
  By intensity;
run;
```

**Example 2:**

Use the NODUPKEY= option on PROC SORT to eliminate duplicate output observations with the same values for the BY variables.

*NOTE:* Useful in situations when you have multiple observations for each individual and you only want, for example, the first or last (use DESCENDING on sort) adverse event in your dataset.
### Example 3:
Use the TAGSORT= option on the PROC SORT to reduce amount of temporary disc spaced used.

**NOTE:** Best performance is gained with this option when the total length of the BY variables is short when compared to the record length.

<table>
<thead>
<tr>
<th>Acceptable:</th>
<th>More Efficient:</th>
</tr>
</thead>
</table>
| **proc sort** data = adverse_events  
out = lastAE;  
by an_num **descending** intensity;  
Run;  
data keeplast;  
Set lastAE;  
by an_num **descending** intensity;  
If first.intensity;  
Run; | **proc sort** data = adverse_events  
NODUPKEY  
out = lastAE;  
By an_num **descending** intensity;  
Run; |

### Example 4:
Use the CLASS statement with your procedure (MEANS or SUMMARY) This eliminates the need to do a PROC SORT prior to your procedure.

<table>
<thead>
<tr>
<th>Acceptable:</th>
<th>More Efficient:</th>
</tr>
</thead>
</table>
| **proc sort** data = adverse_events;  
By intensity;  
Run;  
**proc summary** data=adverse_events;  
var duration;  
**by** intensity;  
output new = new MEAN = meandur;  
Run; | **proc summary** data=adverse_events;  
**class** intensity;  
var duration;  
output new = new MEAN = meandur;  
Run; |
DATA MANIPULATION:

Example 1:
By-Merging of Sorted Datasets
Merging without a BY merges observation by observation.
NOTE: If a variable is found in both datasets, the resultant value for this variable is pulled from the LAST dataset in the list.
By-Merging of Sorted Datasets using IN=
Allows specification of which input dataset contribute to the merge.

<table>
<thead>
<tr>
<th>Step 1: Sort</th>
<th>Step 2: Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc sort data = adverse_events;</td>
<td>data both;</td>
</tr>
<tr>
<td>By an code date;</td>
<td>merge adverse_events (in= ina)</td>
</tr>
<tr>
<td>run;</td>
<td>therapy (in=int) ;</td>
</tr>
<tr>
<td>proc sort data = therapy;</td>
<td>by an code date;</td>
</tr>
<tr>
<td>By an code date;</td>
<td>if ina;</td>
</tr>
<tr>
<td>run;</td>
<td>run;</td>
</tr>
</tbody>
</table>

Example 2:
Labels
When printing the contents of a SAS® dataset created with labels, add readability to your output by printing the variable label.

Create your SAS® dataset with Labels
Proc SQL;
    create table adverse_events (label="Adverse Event") as
    select   prsc_tm label= "PRESCRIBED TIME",
              strltrdy label= "START_DAY_REL_TO_TRIAL",
              dur_un1 label= "DURATION UNITS",
              intn_c label= "INTENSITY_CODE",
        from ae;
quit;

<table>
<thead>
<tr>
<th>Proc Print - No Labels</th>
<th>Proc Print - with Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc print data = adverse_events;</td>
<td>proc print data adverse_events label;</td>
</tr>
<tr>
<td>title 'WITHOUT Label Option';</td>
<td>title 'WITH Label Option';</td>
</tr>
<tr>
<td>run;</td>
<td>run;</td>
</tr>
</tbody>
</table>
Example 3:
Use of Formats

<table>
<thead>
<tr>
<th>Create Formats</th>
<th>Print using Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>proc format;</strong></td>
<td><strong>Proc print data=adverse_events label;</strong></td>
</tr>
<tr>
<td><strong>value codes</strong></td>
<td><strong>format intn_cd codes.;</strong></td>
</tr>
<tr>
<td>1 = 'High'</td>
<td>Run;</td>
</tr>
<tr>
<td>2 = 'Medium'</td>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>3 = 'Low';</td>
<td><strong>WITH Format on Intensity Code</strong></td>
</tr>
<tr>
<td>run;</td>
<td>PRESCRIBED_</td>
</tr>
<tr>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td>Post-dose</td>
<td>High</td>
</tr>
<tr>
<td>Post-dose</td>
<td>Medium</td>
</tr>
<tr>
<td>Post-dose</td>
<td>High</td>
</tr>
</tbody>
</table>

Example 4:
Renaming Variables.
Variables are renamed with either the RENAME= data set option or the RENAME statement.

<table>
<thead>
<tr>
<th>RENAME= Data Set Option:</th>
<th>RENAME statement:</th>
</tr>
</thead>
<tbody>
<tr>
<td>data two; set one(rename=(x1=y z1=u) keep=y u total); total=y+u; run;</td>
<td>data two; set one; rename x1=y z1=u; total=x1+z1; keep=y u total; run;</td>
</tr>
</tbody>
</table>
Example 5:
Retaining Variables.
The RETAIN statement causes a variable to retain its value from the previous iteration of the DATA step.

RETAIN with first. and last:

```plaintext
proc sort data=new; by rating; run;
data count; set new; by rating;
retain cnt 0 howmany 0;
if first.rating then cnt=1; else cnt+1;
if first.rating then howmany+1;
if last.rating then output;
run;
proc print data=countem; run;
```

Here is the printout of data set countem:

<table>
<thead>
<tr>
<th>OBS</th>
<th>RATING</th>
<th>TX</th>
<th>CNT</th>
<th>HOWMANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>apparent</td>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>discrete</td>
<td>9</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>none</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

RETAIN without a variable list:

```plaintext
data group;
Input x;
retain;
if x=1 then gender='M';
if x=2 then gender='F';
cards;
data lines;
X is retained if value is other than 1 or 2
```

MACROS (tips for the beginner):

Example 1:
Creating a Macro Variable

```plaintext
%LET:
%let dsn=NEWDATA;
title "Display of Data Set &dsn";
TITLE "Display of Data Set NEWDATA";
```

```plaintext
CALL SYMPUT:
%macro create;
data temp;
set newdata end=final;
if age >=20 then do;
    N+1;
    output;
end;
if final then call symput('number',n);
run;
%mend create;
Footnote "&number Observations";
```
Example 2:
Concatenating Several datasets using a macro.

<table>
<thead>
<tr>
<th>Acceptable without macro:</th>
<th>Alternate Method:</th>
</tr>
</thead>
<tbody>
<tr>
<td>data x1; x=1; run;</td>
<td>data x1; x=1; run;</td>
</tr>
<tr>
<td>data x2; x=2; run;</td>
<td>data x2; x=2; run;</td>
</tr>
<tr>
<td>data x3; x=3; run;</td>
<td>data x3; x=3; run;</td>
</tr>
<tr>
<td>data final;</td>
<td>%macro test;</td>
</tr>
<tr>
<td>set x1 x2 x3;</td>
<td>data final; set</td>
</tr>
<tr>
<td>run;</td>
<td>%do i = 1 %to 3;</td>
</tr>
<tr>
<td></td>
<td>x&amp;i</td>
</tr>
<tr>
<td></td>
<td>%end;</td>
</tr>
<tr>
<td></td>
<td>run;</td>
</tr>
<tr>
<td></td>
<td>%mend test;</td>
</tr>
<tr>
<td></td>
<td>%test</td>
</tr>
</tbody>
</table>

Example 3:
Generating a series of DATA steps

<table>
<thead>
<tr>
<th>Invoke macro CREATE:</th>
<th>Produces:</th>
</tr>
</thead>
<tbody>
<tr>
<td>%macro create;</td>
<td>data month1;</td>
</tr>
<tr>
<td>%do i=1 %to 3;</td>
<td>infile in1;</td>
</tr>
<tr>
<td>data month&amp;i;</td>
<td>input product cost date;</td>
</tr>
<tr>
<td>infile in&amp;i;</td>
<td>run;</td>
</tr>
<tr>
<td>input product cost date;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td>data month2;</td>
</tr>
<tr>
<td>%mend create;</td>
<td>infile in2;</td>
</tr>
<tr>
<td>%create;</td>
<td>input product cost date;</td>
</tr>
<tr>
<td></td>
<td>run;</td>
</tr>
<tr>
<td></td>
<td>data month3;</td>
</tr>
<tr>
<td></td>
<td>infile in3;</td>
</tr>
<tr>
<td></td>
<td>input product cost date;</td>
</tr>
<tr>
<td></td>
<td>run;</td>
</tr>
</tbody>
</table>
Example 4:
Comment out code using a macro or use “HOT” key:

<table>
<thead>
<tr>
<th>“HOT” Key Method</th>
<th>Macro Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the “HOT” key <code>cntl+</code>/ to comment selected code.</td>
<td><code>%macro skipstep;</code></td>
</tr>
<tr>
<td>/<em>data small.subset;</em>/</td>
<td>data small.subset;</td>
</tr>
<tr>
<td>/*set big.dataset;</td>
<td>set big.dataset;</td>
</tr>
<tr>
<td>/<em>if mod(<em>n</em>, 50) = 28;</em>/</td>
<td>if mod(<em>n</em>, 50) = 28;</td>
</tr>
<tr>
<td>/* select every 50th record */</td>
<td>/* select every 50th record*/</td>
</tr>
<tr>
<td>/<em>run;</em>/</td>
<td>run;</td>
</tr>
<tr>
<td></td>
<td><code>%mend skipstep;</code></td>
</tr>
</tbody>
</table>
REFERENCES:

1. In the Know…SAS® Tips & Techniques From Around the Globe, Phil Mason
2. SAS® Guide to Macro Processing, SAS Institute, Inc.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

CONTACT INFORMATION:

Kathy Harkins  
Merck & Co. Inc.  
PO Box 1000  
North Wales, PA  19454-1099  
267-305-5533  
kathy_harkins@merck.com

Carolyn Maass  
Merck & Co. Inc.  
PO Box 1000  
North Wales, PA  19454-1099  
267-305-7145  
carolyn_maass@merck.com

Mary Anne Rutkowski  
Merck & Co. Inc.  
PO Box 1000  
North Wales, PA  19454-1099  
267-305-6925  
mary_anne_rutkowski@merck.com