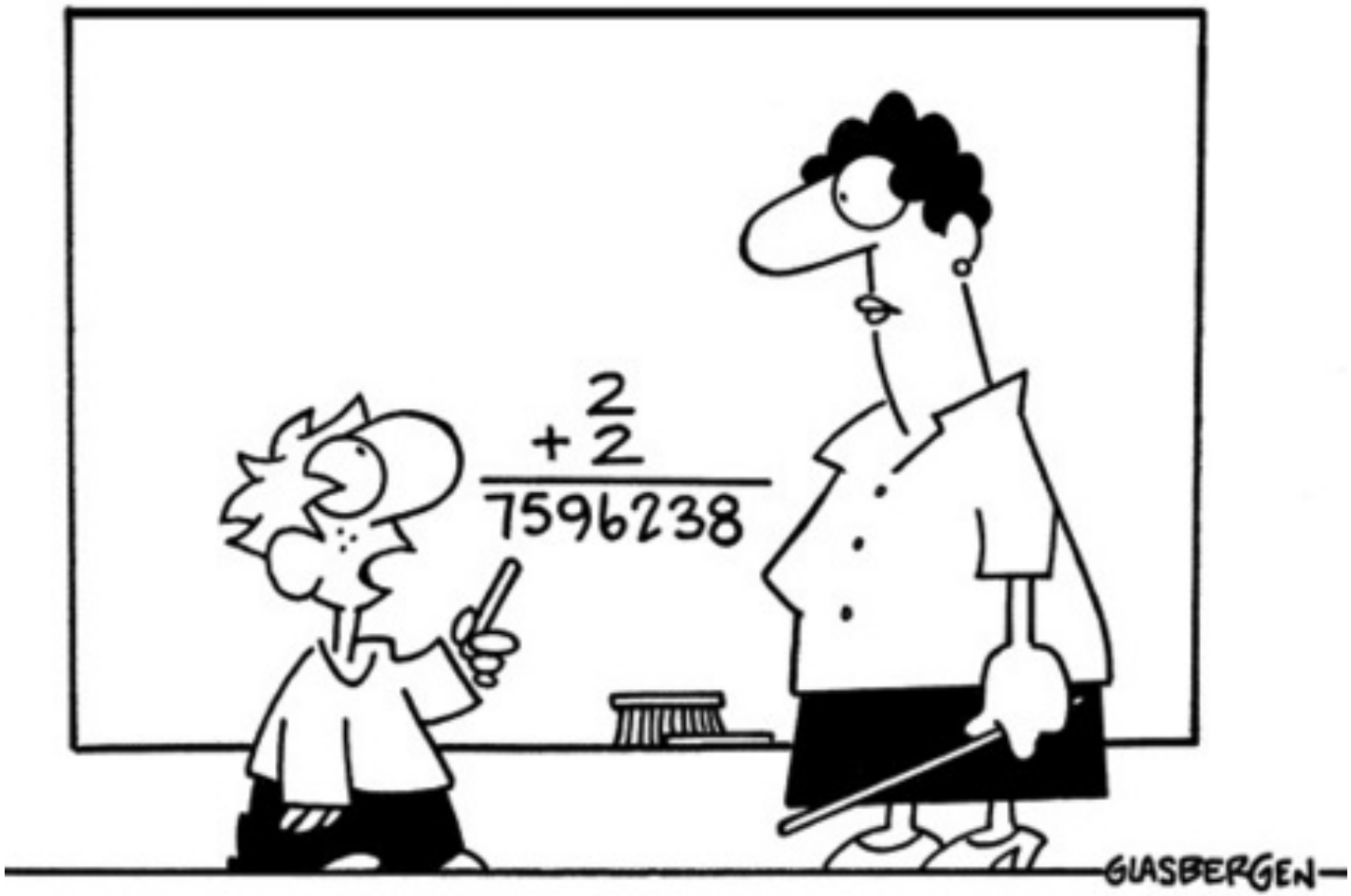
EFFICIENT SAS PROGRAMMING - ARE YOU DOING IT??

Ms. Pratibha Jalui
Manager-Biostatistics & Programming
SIRO Clinpharm Pvt. Ltd



"In an increasingly complex world, sometimes old questions require new answers."

Importance of Efficiency

What is Efficient coding?

- Code a program that will run the quickest - or??
- Completing a task while utilizing the minimum resources.

→ Continuing process of reaching an OPTIMAL BALANCE between competing resources and activities!!!

Two Classes of Resources

Computer Resources

CPU Time, Data Storage, Input/Output Time, Memory

Human Resources

Human efficiency - programmer time & experience level-priority attached to producing results quickly rather than inexpensively -expenditure of programmer time for design, coding, and maintenance

Efficiency Techniques

Computer Resources

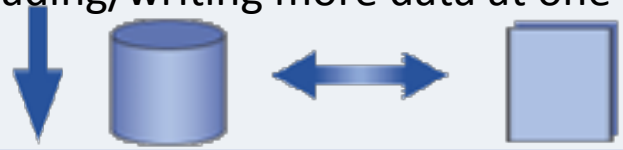
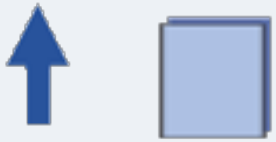
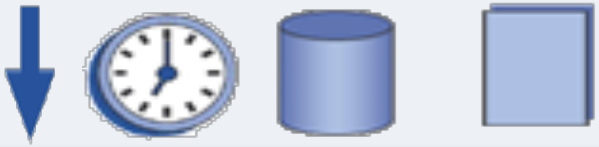
- ❑ **Read** and **write** only the **data** that **you require**.
- ❑ **Execute** only **necessary** statements.
- ❑ **Eliminate unnecessary passes** of the data.
- ❑ **Store data in SAS** data sets.
- ❑ **Avoid unnecessary procedure invocation.**

Efficiency Techniques

Human Resources

- ❑ **Think – What matters the most?**
- ❑ **Plan** & follow **sound development** practices
- ❑ **Document** your program
- ❑ Apply **Good Programming Practices**
- ❑ **Organize** data steps consistently
- ❑ **Talk to your** data
- ❑ **Plan & code for unknown data**
- ❑ Have a **test phase** to identify and repair issues
- ❑ **Keep Learning !!**

Efficiency Trade-offs

Decreasing usage of this resource ...	Might increase usage of this resource ...
Disk space 	**CPU time** 
I/O (reading/writing more data at one go) 	**Memory** 
Real time (by enabling threading in SAS 9.1) 	**CPU time** 
Any computer resource (by increasing program complexity) 	**Programmer Time** 

Examples - Using Computer Resources efficiently

Read and write only the data that you require.
Example: *Only Keep Necessary Variables*

Acceptable:

```
data small;
    set large;
        keep a b c prod lrgst;
        prod = a*b;
        lrgst = max(a,b,c);
*More SAS® statements..;
run;
```

More Efficient:

```
data small;
        set large (keep=a b c);
            prod = a*b;
            lrgst = max(a,b,c);
*More SAS® statements..;
run;
```

Eliminate unnecessary passes of the data.
Example: *Produce Datasets Efficiently*
Using a Single DATA Step to Create Multiple Output Data Sets

Acceptable:

```
data one;
    set large;
        if a < 10;
run;

data two;
    set large;
        if 9 < a < 90 ;
run;
```

More Efficient:

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

Avoid unnecessary Procedure Invocation.
Example : Using the DATASETS procedure to modify variable attributes of SAS dataset.

Acceptable:

```
proc datasets lib=company;
    modify newcustomer;
        rename Country_ID=Country;
        label birth_dt='Date of Birth';
quit;

proc datasets lib=company;
    change newitems=Neworder;
        format birth_date date9.;
quit;
```

More Efficient:

```
proc datasets lib=company;
    modify newcustomer;
        rename country_ID=Country;
        label birth_dt='Date of Birth';
    change newitems=Neworder;
        format birth_date date9.;
quit;
```

Execute only Necessary Statements
Example: *Produce Datasets Efficiently*
Read an existing SAS® dataset and subset it based on values of variables. Use a WHERE instead of an IF statement.

Acceptable:

```
data new;
    set old;
    if age > 65 and gender = 'F';
*More SAS® statements..;
run;
```

More Efficient:

```
data new;
    set old (where=( age > 65 and
                     gender ='F'));
*More SAS® statements..;
run;
```

Execute only Necessary Statements

Move a "sub-setting" IF before the statements you only want to execute after the condition is met.

Acceptable:

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

More Efficient:

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

Execute only Necessary Statements

When "IF" conditions are mutually exclusive, use IF-THEN-ELSE.

Move most likely condition to top of logic of the variables

Acceptable:

```
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;
 run;
```

More Efficient:

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

Example: Efficient Sub-setting of Data

Take advantage of Boolean Logic [Expressions evaluate to true (0) or false (1)] to assign values instead of using IF-THEN-ELSE logic.

Acceptable:

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

More Compact:

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

EFFICIENT SORTING:
Example 1: *Use the WHERE= option*

Acceptable:

```
proc sort data = adverse_events;
     by intensity;
run;

data relatedAE;
     set adverse_events;
      where relation = 'Y' ;
run;
```

More Efficient:

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

EFFICIENT SORTING:

Example 2: *NODUPKEY Option*

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

Acceptable:

```
proc sort data = adverse_events
                out = lastAE;
by an_num intensity;
run;
data keeplast  (keep = an_num
                       intensity);
    set lastAE;
        by an_num intensity;
    if last.intensity;
run;
```

More Efficient:

```
proc sort data = adverse_events
            out = keeplast NODUPKEY
          (keep = an_num intensity);
    by an_num descending intensity;
run;
```

EFFICIENT SORTING:
Example 3: *TAGSORT= option*
Use the TAGSORT= option on the PROC SORT to reduce amount of temporary disc space used.

Acceptable:

```
proc sort data = adverse_events
                out = TaggedAE;
    by intensity;
run;
```

More Efficient:

```
proc sort data = adverse_events
                out = TaggedAE tagsort ;
    by intensity;
run;
```

Example: *CLASS statement (MEANS or SUMMARY)*

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

Acceptable:

```
proc sort data = adverse_events;
     by intensity;
run;
proc summary data=adverse_events;
     var duration;
     by intensity;
     output new = new
               MEAN = meandur;
run;
```

More Efficient:

```
proc summary data=adverse_events;
     class intensity;
      var duration;
               output new = new
                         MEAN = meandur;
run;
```

DATA MANIPULATION

Example: *By-Merging of Sorted Datasets*

By-Merging of Sorted Datasets using IN=

Step 1: *Sort*

```
proc sort data = adverse_events;
     by an code date;
run;

proc sort data = therapy;
     by an code date;
run;
```

Step 2: *Merge*

```
data both;
     merge adverse_events (in = ina)
                          therapy (in = int);
     by an code date;
          if ina;
run;
```

Example: Use the Specific Procedure for the job

Using the specialized procedures, in place of the data step, will improve the efficiency in most of the cases

Set Option:

```
data  datset1;
    set datset1 datset2;
run;
```

Both the datasets are read into the PDV, before the final dataset is created

*(datset1 is larger dataset)

Append Procedure:

```
proc append  base = datset1
                    data= datset2;
run;
```

ONLY the **dataset in "data=" is read** into **buffer**. **=>** if we **use larger dataset in** the **"base="** option, we can **cut** the **processing time by more than 50%. !!!**

Example: Choose the right tool for the job

Choosing the optimal procedure helps in getting overall efficiency!!

SQL Procedure:

```
proc sql ;
  create table lb as
    select lb.*, sv.visit, sv.visitdt
      from lab as lb left join visit as sv
      on lb.subjid = sv.subjid and lb.visitnum =
       sv.visitnum
      order by lb.subjid, lb.visitnum, lb.lbtestn ;
quit ;
```

Real time : 0.42 sec
CPU time: 0.63 sec

HashProcedure:

```
data lb ;
  retain subjid …….. lborresu ;
       length lbtest $15 lborresu ..visitdt $10 ;
        if _n_ = 1 then do ;
                declare hash sv(dataset: 'visit') ;
                 sv.definekey('subjid', 'visitnum') ;
                sv.definedata(all:'Y') ;
                sv.definedone() ;
        call missing (visit, visitdt) ;
  end ;
  set lab ;
        if sv.find() = 0 then output ;
run ;
```

Real time : 0.26 sec
CPU time: 0.26 sec

Example: Use SAS Functions instead of Expressions

SAS functions have been designed to perform the specific task and are more faster and sophisticated than the raw expressions

Raw Expression:

```
data  totsal;
      set inpdata;
      Total = Salary + Incentive;
run;
```

Problem: If **missing value** either for Salary or for the Incentive column

SUM Function:

```
data  totsal;
      set inpdata;
      Total = sum(Salary , Incentive);
run;
```

"SUM" function : **Executes faster -** considers **missing** values as well.

Example: Complex derivation of a subpopulation of patients – how to transfer sub-population efficiently into analysis programs?
Dataset-controlled format!

1st Step: Identify patients in subpopulation:

E.g. in a data step that may be complex due to complex selection conditions

```
data subpop;
    merge ae lab medhist  . . . . ;
*(complex code);
run;
```

→ Dataset with 1 record for each subject in subpopulation.

Example: Complex derivation of a subpopulation of patients – how to transfer sub-population efficiently into analysis programs?
Dataset-controlled format!

2nd Step: Implement subpop in analysis programs

Acceptable:

```
/*in each analysis program*/
data analysis; /*existing code*/
    set . . . . ;
run;
data analysis2; /*additional code*/
 merge analysis subpop (in = sub);
    by . …. ;
 if sub;
 run;
```

More Efficient:

```
data fmt;
    set subpop (keep = subjid);
          fmtname =“subpop”;
           start = subjid;
          label = “Y”;
 run;
proc format cntlin = fmt ;
 run;
```

Example: Complex derivation of a subpopulation of patients – how to transfer sub-population efficiently into analysis programs?
Dataset-controlled format!

2nd Step: Implement subpop in analysis programs

Only a simple, safe statement to be added in analysis program (Human Resources)

No additional temporary datasets, data steps (Computer Resources)

More Efficient (continued):

```
/*in each analysis program*/
data analysis; /*existing code*/
 set . . . . (where=(put(subjid,subpop.)
     ="Y"));
run;
```

Contact information

- Your comments /suggestions / questions are valued and encouraged.

- Contact the author at:

- **Pratibha Jalui**
  - SIRO Clinpharm Pvt. Ltd.
  - Email id: pratibha.jalui@siroclinpharm.com


danke
spasiba
merci
Thank you
gracias
domo arrigato