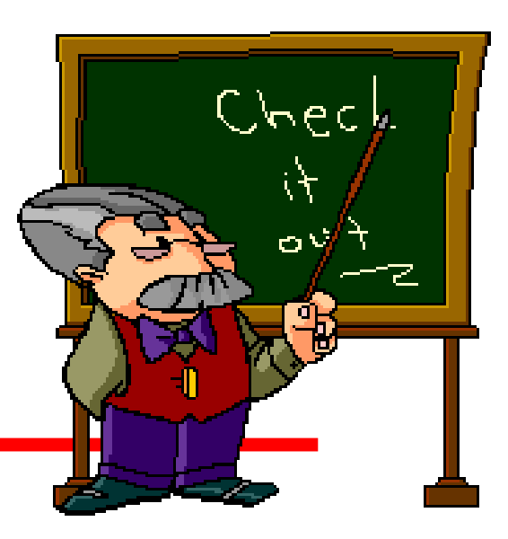
PhUSE GPP Steering Board
Good Programming Practice at a Glance



The GPP Steering Board is a voluntary industry group with representatives from a diverse array of health and life sciences organizations

- Working to develop common good programming practices applicable to statistical programming for the development of high quality and efficient programming code
- Focus on analysis and reporting of clinical trial data; data integration; preparation of clinical data for e-submissions
- This Poster contains an overview of PhUSE GPP guidance Version 1: Consolidated industry guideline with potential to replace/augment individual company guidance and serve as a set of business rules when working on standard scripts for public use.

PhUSE Wiki
Welcome to the phuse Wiki
CSS Working Groups
Annual Conference Brussels
Good Programming Practice Guidance

PhUSE GPP Guidance is found on PhUSE Wiki

The Wiki is a place to share your ideas and post your contributions.
Please provide your comments!

http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice_Guidance

Before you begin

Understand what you need to do; plan how to get there; collect what you need

- SOPs & training
- Study Protocol
- Standards
- Annotated CRF
- SAP & Shells
- Specifications
- Tools and macros

Have a look at the Wiki, there is more …

GPP Principles

Write code that is

- Clear and easy to read and review
- Easy to maintain, modify and debug
- Robust and needs minimal code maintenance
- Can be reused or easily adapted.
- Efficient and uses minimal computer processing resource
- Written in such a way to reduce logical and syntax errors

Coding conventions

- Do not overwrite existing datasets
- Use of all uppercase should be avoided
- Separate data steps and procedures with at least one blank line – Use whitespace
- Use 'data=dataset' option in procedure statements
- End data steps and procedures with run or quit
- Put each statement on a separate line
- Indent statements belonging to a level by 2 to 5 columns (use the same number of spaces throughout the program), i.e. every nesting level should be visibly indented from the previous level.
- Do loops and if then else statements start and end with same level of indentation

Examples of Good and bad code **Review the 2 code snippets below: Both produce the same output. Why is code on the right better?**

```
data alb_tmp ;
set &derived_target..alb;
if saffl = 'Y' ;
if aval is missing then do;
tempflag1='x';
tempflag2='y';
end;
run;
data alb_tmp( where = (avisitn is not missing));
set alb_tmp;
where (aval is not missing or chg is not missing);
run;

proc sql;
select armcd,count(distinct subjid) into :arm1 - :arm2 , :trt1 - :trt2
from &derived_target..asl
where saffl = 'Y'
group by armcd;
create table comb_alb as
select subjid,avisit,avisitn,"a" as groupby,paramcd,armcd,aval as val from
alb_tmp
where aval ne .
union all
select subjid,avisit,avisitn,"b" as groupby,
paramcd,armcd,chg as val from alb_tmp
where chg ne . ;
quit;

* Sort the data ;
proc sort
data = comb_alb;
by groupby paramcd armcd avisitn subjid;
run;
```

```
*******************************************************************;
*** Get external data ***;
*******************************************************************;
DATA alb
    (KEEP=studyid subjid usubjid avisit avisitn paramcd aval chg armcd tempflag);
    SET &derived_target..alb;
    WHERE saffl = 'Y';
    IF aval IS MISSING THEN  tempflag='x';
    IF avisitn is missing then put "Note: Missing avisitn for " usubjid= paramcd= aval= chg= ;
RUN;

PROC SQL NOPRINT;
    SELECT armcd, COUNT(DISTINCT subjid)
        INTO  :arm1 - :arm99, :trt1 - :trt99
    FROM &derived_target..asl
    WHERE saffl = 'Y'
    GROUP BY armcd;
QUIT;
*******************************************************************;
*** Remove missing values ***;
*** Create standard structure for summarising ***;
*******************************************************************;
DATA alb_nomiss;
    SET alb;
    WHERE (aval IS NOT MISSING or chg IS NOT MISSING) AND (avisitn IS NOT MISSING);
RUN;

PROC SQL NOPRINT;
    CREATE TABLE comb_alb AS
        SELECT  subjid, avisit, avisitn, "Actual" AS groupby
                , paramcd, armcd, aval AS val
        FROM alb_nomiss
        WHERE aval NE .
     UNION all
        SELECT subjid, avisit, avisitn, "Change_baseline" AS groupby
               , paramcd, armcd, chg AS val
        FROM alb_nomiss
        WHERE chg NE .;
QUIT;
*******************************************************************;
*** Sort and summarise the data ***;
*******************************************************************;
PROC SORT DATA = comb_alb
    OUT = sort_Alb;
    BY groupby paramcd armcd avisitn subjid;
RUN;
```

- Comments explain the purpose of what is being done
- Indented Code
- Keep only the variables you need
- Data sets not overwritten
- Where clauses do not overwrite
- *What else?*

Layout & Header

1. Always have a program header!
2. Clean up work area from previous program
3. Read in external data
4. For multiple data sources, pre-process , then combine before common derivations are applied
5. Process and manipulate data, merge and derive variables
6. Perform analysis
7. Report data: Produce tables, figures and listings
8. Clean up work area

```
/*=============================================================================
*
* STUDY                 : xxxxx.xx          CREATION DATE: YYYY-MM-DD
*
* PROGRAM               : xxxx.sas
*
* PROGRAM LOCATION      : yyyy\yyyy\yxy
*
* PROGRAMMER            :
*
* DESCRIPTION           : <Describe the purpose of the program>
*
* FINAL PROGRAM DATE    : < Date the program was considered final/validated>
*
* SOFTWARE VERSION      : <State sotware name, version, operating system>
*
*-----------------------------------------------------------------------------
* REVISION HISTORY (COPY FOR EACH REVISION):
*
* DATE                  :
* PROGRAMMER            :
* REASON FOR CHANGE     :
*
*=============================================================================*/
```

DO	DON'T
• Use a program Header • Use Informative comments in appropriate places • Use unique and informative dataset names • Use one statement per line • Indent consistently • Program defensively for missing and unexpected data • Check log for ERRORS, WARNING, *and NOTES* that indicate data issues	• Use a header but forget to fill in or update! • Use comments that simply say what the code does (not Why) • Use inconsistent or random naming for datasets and variables • Program around data issues or "to the data" • Ignore Notes in the Log that indicate potential issues E.G. Automatic Number/character conversions, Merge by multiple records

Check your Log!

Not(e): just ERROR:s and WARNING:s !

```
not referenced
never been refere
not resolved
is not in the report def
has more than one data set with repeats of by values
current word or quoted string has become more than 200
observation(s) outside the axis range
is unknown
uninitialized
cannot be determined
extraneous information
missing values were generated
observation(s) contained a MISSING value
Invalid arguments
truncated to 32 characters
overwritten by
can't modify it at this time
have been converted to
```

Many thanks to the GPP Steering Board for their constructive reviews and contributions.

Mark Foxwell, PRA Health Sciences, Reading, UK