Natalie Collier, Kate Henry, Rebecca Tootell

‘The Good, the Bad and the Ugly’ (of Programming Standards)

Introduction

While most programmers strive to write neat or “whizzy” code there are some basic principles that underpin the success of a piece of code. We would like to use this poster to highlight what we believe is fundamental to the writing of successful code and to raise the lid on the good, the bad and the ugly of programming standards. We have concentrated on 3 main areas to achieve clarity, maintainability and efficiency in code, and have included some examples of what can happen when SAS seems to go wrong!

The Good

Clarity:

- **Comments:** This helps the user understand and follow the logic and reasoning behind the code
- **Unnecessary code:** Remove code that is commented out and not required in the final program
- **Header:** Use to provide up to date general information about the program, e.g. author, purpose, etc
- **Spacing:** At least 1 blank line between each PROC/DATA step and only 1 SAS statement per line
- **Lower/Upper case:** Be consistent – lower case is easier to read
- **Indent code:** For logical code like DO loops and IF THEN DO statements ensure they begin and end in the same position
- **Dataset and variable labels:** Use meaningful labels
- **Input/output datasets:** Ensure each PROC/DATA step has a separate input and output dataset – easier to follow and debug

Efficiency:

- **KEEP/DROP statements:** Use appropriately to keep variables relevant to the datastep. DROP any intermediate variables used for calculations
- **WHERE/IF statements:** Use WHERE rather than IF (WHERE will subset the data before entering the Program Data Vector)
- **IF/THEN/ELSE statements:** When stepping through an IF condition check the most likely condition first. Use IF/ELSE for mutually exclusive conditions. Add ELSE OUTPUT <*dataset*> to catch all other scenarios
- **Temporary datasets:** Clear any temporary datasets throughout a program
- **LENGTH/ATTRIB:** Use to specify the optimum length of character variables to avoid truncating any data
- **DATA _NULL_:** Use for steps that are not creating a dataset, e.g. when creating macro variables
- **Master programs/Batch runs:** Use to control the run order of programs, create a single log for checking and to initiate the set-up information, e.g. options, macro variables, libnames, etc

Maintainability:

- **Program flow:** Do not overwrite temporary datasets, e.g. DSET1 -> DSET2-> DSET3 not DSET1 -> DSET2-> DSET1
- **Specify the dataset:** Use data = ‘dataset’ in PROC statements (do not let it default to last dataset)
- **Macros:** Use for repetitive programming or where the routine requires programming logic that cannot be included in a data step
  - Macros should be simple, necessary, easily understood and clearly documented
  - Use %let to specify an operational value for easy re-use of code, e.g. %let DCO=21Jun07
- **Naming conventions:** Use standard abbreviations (e.g. –dtc for date fields) to allow easy identification of different groups of programs or variables
- **Use run; or quit;** appropriately at the end of all DATA/PROC steps
- **Hardcoding:** Avoid at all costs for the changing of data values as this will impact on the re-usability
- **Formats:** Ensure format lengths are sufficient for all formatted values and value lists are complete

The Bad

SAS Log:

- **‘Bad’ Log messages to look out for:**
  - `ERROR` & `WARNING` messages – all errors and warnings must be avoided
  - `NOTE: Variable XXX is uninitialized` – check variable exists
  - `NOTE: Character values have been converted to numeric values` – use INPUT function
  - `NOTE: Merge statement has more than one data set with repeats of by values` – may indicate a merging problem
  - `NOTE: Observation(s) outside the axis range`, or similar, when creating plots – may indicate the axis range is not wide enough
  - `NOTE: Missing values were generated as a result of performing an operation on missing values` – indicates missing values may need to be handled in the code
  - `NOTE: At least one w.d format was too small for the number to be printed. The decimal may be shifted by the “best” format` – numeric format is not suitable
  - `NOTE: Multiple lengths were specified for the BY variable` may indicate truncation of variables has occurred – use LENGTH statements to avoid
  - Other `NOTE` messages of concern – confirm that all messages are acceptable and commented appropriately
- **Number of observations and variables:** `NOTE: The data set WORK.XX has N observations and N variables` – an unexpected number of observations or variables after a datastep may indicate a coding error
- **PUT statements:** Use to output to the log/file any potential data errors/inconsistencies that may impact on programming, e.g. missing values, incomplete dates

Unexpected Results:

- **Numerical precision:** SAS uses floating point or real binary representation to store all numeric values (finite precision)
  e.g. $0.2 \approx \frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^7}+\frac{1}{2^8}+\frac{1}{2^{11}}+\frac{1}{2^{12}}+\frac{1}{2^{15}}+\ldots = 0.199981689$ use FUZZ/ROUND functions as appropriate to your data
- **Informats:** When using an x.y informat a decimal place must be present within all data values (e.g. 1 is formatted to 0.0001 if informat8.4 is applied) – instead use a bestx. informat to prevent spurious results
- **Division by 0:** Include code to prevent this, e.g. if b ne 0 then x=a/b; else x=.;
- **SAS version:** Statistics procedures may change with different versions of SAS, e.g. ODS objects
- **Missing values:** These may influence the statistical output, e.g. SUM(X,Y) ? X+Y if X=.
- **Date functions:** Leap years may affect the outcome of date functions, e.g. YRDIF
- **Statistical assumptions:** Check any statistical assumptions to ensure analyses being done are appropriate

The Ugly

Only use 1 SAS statement / line

```
proc sort data = x.lab; by subject visit;run;
```

Use a unique dataset name to clarify program flow

```
DATA lab (drop=labval) dset1;
label newvar ‘Another variable’;
set lab;
```

Use a meaningful dataset name, variable name and label

Use WHERE instead of IF for efficiency

```
if labcode in (‘1234’,’5678’,’9999’);
/* if subject=12 then hardcode LABVAL to
10.204 */;
```

Provide useful comments with explanation of reason or logic

Need to consistently reference the LABVAL variable as numeric or character

```
if subj=12 and labval=10,204 then do;
labval=10.204; end;
```

Lab value should not be hardcoded

```
if labval = 0 then newvar = 2;
if LABVAL NE 0 then newvar = 6;
```

Should use ELSE IF for mutually exclusive conditions

Use Best format instead of 8.

```
labvalun = input(labval,8.);
```

Use lower and upper case consistently

```
ARRAY old {3} x y z;
array new {3} var1 vard var4;
```

Provide comments to explain code

Use indentation & spacing

```
do i = 1 to 3; IF OLD{I} = ‘xyz’ THEN DO;
new{i} = ‘No’; end;
else do; new{i} = ‘Yes’;
end;
ELSE DO;
output dset1;
end;
```

Need LENGTH/ATTRIB statement for new variables to avoid truncation

Temporary variable i should be dropped

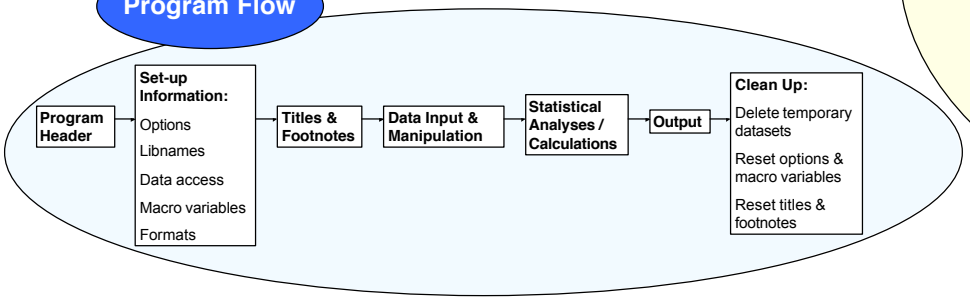
Add RUN statement

```
END;
proc sort;
  by labcode;
run;
```

Include spacing between data step & sort

Add ‘DATA =’ to avoid defaulting to last datasetcreated

Program Flow



AstraZeneca