

Code Like it Matters
Writing Code That's Readable and Shareable

Paul Kaefer
IT Data Analyst/SAS Developer
UnitedHealthcare Financial Data Management

MinnSUG: July 2017 meeting

UnitedHealthcare®



UnitedHealthcare®

Today's Agenda

1. Introduction
2. Why write **readable** and **shareable** code?
3. Best practices for writing
4. Code standards
5. Best practices for sharing
6. Conclusion
7. Discussion

You are encouraged to interrupt with questions.

Introducing me

- B.S., Computer Engineering, 2013
- M.S., Computational Sciences (a.k.a. applied math/data science), 2015
- Both degrees from Marquette University
- Pursued M.S. while working at Marquette University GasDay
  - Small business housed in research lab
  - Natural gas demand forecasting for about 21% of U.S. industrial, commercial, and residential gas consumption
- Five months volunteering at Ifakara Health Institute in Tanzania
  - Taught workshops on technical English writing, stats, and R code

Why can I be trusted?

- Recent degree in computer engineering
- I have experience with MATLAB, Java, C, python, R, SAS, and more → what I'm presenting can be translated to each
- Data Analysis Team Lead at GasDay
  - Maintained large research codebase
  - Helped with migration & training for version control
  - Interfaced with Software Development Team & learned from their best practices
- I use what I'm presenting daily at my current job

Coding Like It Matters

- Coming from a background in software engineering, I see gaps in the SAS field
- SAS programmers across the board who may work as sole developer or silo on larger team
- Many SAS contractors who also might work alone or deliver code they developed alone
- Let's bridge the gap!

software engineering best practices + writing SAS code

Disclaimers

- The aim of this paper and presentation is to start the discussion
- Some concepts presented should be treated as *suggestions*
- Your organization may have their own opinions or existing standards
- I am presenting what I consider best practices, but I acknowledge that there are other ways of doing things.

What do we mean by "readable" and "shareable"?

UnitedHealthcare®

readable – clarity in the structure and content of the code

shareable – how quickly and easily someone new can understand and modify your code

These have similar meanings

How "pretty" is your code? And is its function self-explanatory?

Why write readable and shareable code?

UnitedHealthcare®

- Software developers often share their code with others
  - Sometimes one of those "others" is your future self
  - Have you revisited code months or years later and wondered why you wrote it that way?
- Our code might be adapted or modified by a new team member, or someone on another team
  - Sometimes this is known in advance
  - Often we can't predict: move to another team, new project maintainer, etc.
- Things change
  - Database URL
  - Column/variable format
  - Structure of input/configuration file

Best practices for writing code

- Modularity: write code so it is multi-purpose and can be repurposed
  - e.g., macro variables for filenames and date values
- Comments, comments, comments!
  - I recommend using a header template
  - Explain concisely what functions and steps do
  - Use "sort by customer ID" instead of "do a sort"
  - Avoid explaining obvious lines
- Avoid keeping many lines of commented-out code
  - If expecting reuse later, move it to a macro function
  - If it's a change, use **version control** (stay tuned for later slides)

Functions/keyboard abbreviations

- If you write generic functions that can be reused in other processes, save them
  - For example, code to initialize variables based on dates, code to produce a summary report/file of output datasets, or code to send emails
  - Shared drive or common folder if others can use them

- Templates can be given keyboard abbreviations in SAS
  - I store a program header that my team uses
  - I store code to convert dates to numbers and vice-versa

- see, e.g. sasCommunity.org/wiki/Abbreviations/Macros

Coding Standards

- Open-source projects and big software companies have these
  - Can be quite strict (in some cases, can't commit code that doesn't conform)
  - Usually agreed upon early, but can be amended later
  - Learned through documentation and **code review** (to be discussed)

- Main points:
  - **Be consistent.** This enables unity across code and projects.
  - Follow examples from the field
  - Follow applicable standards, like ISO 8601 for dates
  - Be consistent. Your coding style is like a brand.

Code Standards: Tables and Variables

- useCamelCaseForVariables or use_underscores_to_separate_words
  - Pick a standard and go with it
  - Maybe camelCase for variables and underscores for tables?

- Use descriptive names
  - You get 32 characters; use them!
  - More flexibility for macro variables and functions
  - e.g., table: <data source>_<subset or area>_<specific purpose>_<datestamp>_<version>

Code Standards: Format Your Code

- No need to use all capitals (a.k.a., "angry text message")
- Use whitespace to make it easy to read:

```
1  proc sql;
2      create table new_table as
3      select e.firstName
4             ,e.lastName
5             ,e.age
6             ,s.salary format=comma18.2
7      from employee e
8      inner join salaries s
9          on e.employee_ID = s.employee_ID
10     where e.employee_ID is not missing
11         and s.salary not in (., 0);
12 quit;
```

- Put commas before variable names
  - There is some debate here; I find this is easier to line up, see differences with CASE WHEN, and see where commas are missing
- see, e.g., sasCommunity.org/wiki/Anatomy_of_a_SQL_query

Code Standards: Queries, continued

- In the same vein, don't list variables on one line
  - group by a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z
  - vs. separate lines, to match select clause

```
proc sql;
    select a format=date9.
          ,b
          ,c as different_name
          ,d label="test label" format=$50.
          ,e
    from dataset
    group by a
             ,b
             ,c
             ,d
             ,e;
quit;
```

```
proc sql;
    select a format=date9.,b,c as different_name,d label="test label" format=$50.,e
    from dataset
    group by a,b,c,d,e;
quit;
```


Check and X courtesy of the Wikimedia Foundation.

Code Standards: whitespace and font

- Back to whitespace: use spaces instead of tabs
  - Some debate and difference in preference
  - I recommend setting one tab = four spaces

    In EG, that's

    Tools >> Options >> SAS Programs >> Editor Options

    (insert spaces for tabs & set size = 4)
  - Spaces will preserve across editors (e.g., Notepad++ or UltraEdit) and when viewing code in terminal
  - Prevent formatting frustrations when you modify text and tabs change width
  - Helps if you code in multiple languages, e.g., python where spaces matter

- Also recommend `using a monospace font`
  - Default in EG and most code editors
  - Recommend across ANY programming language

Code Standards: Miscellaneous

- Enable line numbers
  - Tools >> Options >> SAS Programs >> Editor Options
  - Makes it MUCH easier to debug & discuss code
- Use semicolons at the end of the line
  - Having a line with only a semicolon is useless
  - Some languages error if it's on the next line
- Don't make your lines too long
  - Various sources recommend 80 characters, e.g. [1]
  - I recommend keeping it so you never have to scroll left-right using a standard monitor

Code Standards: Last Slide

- Order-of-operations
  - Equations, etc. will follow *P*arentheses, *E*xponents, *M*ultiplication, *D*ivision, *A*ddition, *S*ubtraction (PEMDAS)
  - If it is complicated (e.g., nested subqueries and complicated IF statements), be liberal with parentheses
  - indent further for each level of nesting

```
1 data test_dataset;
2
3     set numbers;
4
5     if (a = 4 or
6           (b = 3 and
7               (c = 5 and d = 7);
8
9 quit;
```

- Interim datasets should ALSO have descriptive names
  - Typically written to WORK library
  - May be stored in a data folder separate from main outputs

Best Practices for SHARING your code

- Good, concise documentation
  - Comments, as previously discussed
  - External guide/manual/architecture documentation
    e.g., might have a separate guide for users than for developers
  - Debugging code can count as documentation
    could have macro variables like debug = on;
    (when debug = off; certain code won't run)

- Meaningful variable names

- Version control!

Version Control

- How many of you use version control?

Version Control

- How many of you use version control?
  - ALL HANDS SHOULD BE UP!

Version Control

- All programmers should use version control
  - Whether you're on a team, or coding alone
  - Even useful in other fields (writing/editing, anybody who makes presentations or has a resume)
- Rudimentary strategy
  - code.sas, code_v2.sas, code_new.sas, code_old.sas
  - Highly discouraged.
- Use software like git, Mercurial, Subversion, etc.
  - Some learning curve, but the basics can be taught in < 1 hour
  - Makes sharing easier
  - Saves you over and over (story time…)

Version Control: The Basics

- Version control software saves *changes* instead of full copies
  - So ideal for code, not for binary files (.zip, .pdf, .egp, etc.), though binary files can be added
  - Lets you quickly compare any two versions, or see changes made since the last commit

- When you have saved changes, you prepare a **commit**. This is a set of changes that can be **pushed** to the master copy. Then other team members can **pull** the latest changes from the master.

- Depends somewhat on the software you use
  - Subversion is centralized, so everyone commits to the master
  - Git is decentralized; you can commit locally until ready to push

- See also sasCommunity.org/wiki/Version_control

Version Control: Best Practices

UnitedHealthcare

- Save code as raw .sas files, not in .egp
  - You *can* commit .egp files, but won't be able to compare changes
  - Use a master script with %INCLUDE instead of the graphical program flow
- Descriptive commit messages
  - Not "changed code"
  - Say "added filter on historical customer purchases"
  - Explain everything "fixed issue with … and wrote code to …"
  - Can be multiple lines
- Reference commit number (Subversion) or hash (git)
  - Documentation can say "This feature was added in commit 77" or "commit 0c93e1af6" (git hash)
  - Tags can be used to preserve code as it was on a particular date or for a release version (i.e., tag for v4.0)

Code Review

UnitedHealthcare

- This really helps with developing a culture of standards
- Best for teams collaborating on same codebase
- Can be a formal review, in a conference room with a projector
- Software tools allow asynchronous review
  - Log on to review site daily and spend some time reviewing code
- Helps encourage a spirit of collaboration and collective learning
  - Use constructive criticism
  - Critiques on code should not be personal attacks
  - We can all learn from each other

Conclusions

- Does this sound useful?
- I hope I haven't overwhelmed you…
  - Use this as a starting point
  - If you leave with one or two "gems", I'm happy
- You may disagree with me!
  - Different standards or existing practices by organization/field
- Above all, be consistent!

Further reading

- This presentation and paper will be available at sasCommunity.org/wiki/Code_Like_It_Matters:_Writing_Code_That's_Readable_and_Shareable
- *Clean Code: A Handbook of Agile Software Craftsmanship*, Robert Martin
- sasCommunity.org/wiki/Good_Programming_Practice_for_Clinical_Trials
- sasCommunity.org/wiki/Style_guide_for_writing_and_polishing_programs

Discussion

Does your team/organization have any standards or practices not mentioned here?

What ideas presented are new or unfamiliar to you? How can I help?