

theory

adam okulicz-kozaryn

adam.okulicz.kozaryn@gmail.com

this version: Friday 5th December, 2025 12:35

outline

key scientific computing rules to get your started

and yet one more variation on general rules

Know Your Data!

- can't use it well if you don't know it well
 - (not just data; the field: theory, lit, method, etc)
 - that's how you beat IT folks (MS/PhD just in IT)
- again, invest a lot of time into your data
 - use data that you're passionate about
 - or that can make \$ (now or in future career)
- think about it! don't be mindless!
 - ask questions, be investigative, be critical
- double check, cross check, give to others to check

outline

key scientific computing rules to get your started

and yet one more variation on general rules

Wilson put it well

- print out Box 1 from these 2 art
 - hang it at your office, home, and elsewhere
- <https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745>
- <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510>
- and Know Your Data (2nd slide)
- and variations on these, and other general rules follow

simplicity, cleanliness, and organization!

- keep it as simple as possible esp if overwhelmed/struggling
- say retain only 5vars and 25obs
- have chunks of code only once
- code it all from raw to final (replication principle)
- organize: sections, comments, and logical order
(eg rewrite, move code around)

outline

key scientific computing rules to get your started

and yet one more variation on general rules

more principles

- from 2 books about general programming (classics and free!)
 - <http://catb.org/esr/writings/taoup/>
 - <http://www.htdp.org/2003-09-26/Book/curriculum-Z-H-1.html>

clarity

- “design for transparency and discoverability”
 - write clean code [eg split 1 fn over many l for readability]
 - avoid fancy code
 - fancy code is buggier
 - clarity is better than cleverness
- eg:
 - group logical chunks together

modularity

- “write simple parts that are cleanly connected”
- “controlling complexity is the essence of computer programming”
 - debugging dominates development
- eg:
 - clear sections of one file
 - or many files instead of one file without sections
- organized code logically, not chronologically
 - do free writing, but then reorganize/rewrite
(like with papers writing)
 - no data management in data vis part
 - move rename, replace, etc to earlier

composition

- “design programs to be connected to other programs”
- notebook or its sec will produce output for another notebook or sec
- eg: you clean up data in one file to make data ready for another one to vis
 - or just have one big file
- but the workflow needs to be logically organized

optimization (fancier, fewer lines)

- yes, but “get it working before optimizing” !
- eg:
 - first make mpl hist for one var, make it working
 - and then deploy it for 10 vars with a loop

extensibility

- “design for the future because it will be sooner than you think”
 - you will reuse your code in the near future
 - so write it clean
 - have sections, etc
 - use lots of comments
 - reorganize, rewrite
 - optimize

silence

- “when a program has nothing surprising to say, it should say nothing”
- drop unnecessary code
 - if you think it may be useful in the future comment it out
- do not generate unnecessary output, do not lose your reader in unnecessary clutter
 - if the output has nothing useful to say it should be dropped
 - (or commented out)

automation (again)

- “rule of generation: avoid hand-hacking”
- because humans make mistakes and computers don’t, computers should replace humans wherever possible
- automate anything that you can
- but stay human, focus on fun creative part, eg vis
- dont automate everything; eg dont crank out bunch of vis mindlessly

reuse (copy-paste), don't reinvent the wheel

- if someone has already solved a problem once, reuse it!
- very unlikely doing something completely new
 - google what you are mapping and see images
- often all you need is to adjust somebody else's map
- ask for code: supervisor, article authors, colleagues, etc
- share your code: github, your website
 - (may protect critical/innovative parts)
- acknowledge others' work

defensive programming

- “people are dumb-make program bullet-proof”
- so important to interpret critically your maps
 - if sth looks weird/unlikely, there's probably mistake
- check double check
- google your maps see images
- go to conferences, publish