Ten Simple Rules to increase computational skills among biologists with Code Clubs

Ada K. Hagan[2], Nick Lesniak, *Author names here, order TBD*

Patrick D. Schloss[1†]

† To whom correspondence should be addressed: pschloss@umich.edu

1 Department of Microbiology and Immunology, University of Michigan, Ann Arbor, MI 48109

2 Alliance SciComm & Consulting, Linden, MI 48418

Ten Simple Rules

Introduction

For most biologists, the ability to generate data has outpaced the ability to analyze those data. High throughput data comes to us from DNA and RNA sequencing, flow cytometry, metabolomics, molecular screens, and more. Although some accept the approach of compartmentalizing data generation and data analysis, we have found scientists feel empowered when they can both ask and answer their own biological questions. Yet, the standard undergraduate and graduate training in the biological sciences is insufficient for developing these advanced data analysis skills. In our experience performing microbiome research, it is more common to find exceptional bench scientists who are inexperienced at analyzing large data sets than to find the reverse. Of course this raises a challenge: How do we train bench scientists that analyze data sets to answer biological questions?

The ever-growing ability to generate data and constant feeling of helplessness in analyzing it is analogous to the struggles we also face with engaging the voluminous scientific literature. A common strategy to keeping up with the literature are journal clubs, which involve group discussion of a pre-selected paper. These range from informal discussions to PowerPoint presentations and course credit. In addition to staying current on the literature, journal clubs help strengthen skills in critical thinking, communication, and literature review [1]. Our research group has leveraged the similarity between the overwhelming natures of scientific literature and data analysis to address the challenge of teaching reproducible data analysis practices to bench scientists. Over the past 4 years we have experimented with a Code Club model to improve data analysis skills in a community environment.

Our Code Club sessions are an hour long and alternate with Journal Club as the second part of weekly two hour lab meetings. Presenters volunteer to lead a session with the expectation that they alternate leading Journal and Code Clubs. Initially, the Code Club was used to review code from trainee projects. Instead of a presentation, the presenter would project their code onto a screen and the participants would go line by line through the code, stating the logic behind each line. This approach emphasized the importance of code readability and gave beginners the opportunity to see the real-life, messy code of more experienced peers. Unfortunately, the format

only allowed us to review a fraction of a project's code base, making it difficult to integrate the programmer's logic across their full project. A major issue with this model was that sometimes beginners couldn't contribute to improving the code, and even when they could, more senior group members would eventually dominate the discussion. This led to a lack of participation by beginners, who would mentally check out to focus on other activities, and frequently resulted in an adversarial environment between the presenter and senior group members. As a result, presenters rarely offered to present Code Club again.

From these experiences, we began a collective conversation to improve our Code Club model and have evolved it into two approaches. The first is a more constructive version of a group code review. The presenter clearly states the problem they want to solve, breaks the participants into smaller groups, and then asks each group to solve the problem, or a portion of it. For example, someone may have an R script with a repeating chunk of code. The challenge for the session would be to convert the code chunk into a function to be called throughout the script to make it "DRY" (i.e. Don't Repeat Yourself)[2]. The presenter leaves the session with several partial or working solutions to their problem and the importance of writing DRY code is reinforced. The second approach is a tutorial. The presenter introduces a new package or technique and assigns an activity to practice the new approach. For example, a more memorable activity was giving participants raw data and a finished plot. Paired participants were tasked with generating the plot using either base R or ggplot, but base R users had to use ggplot and vice versa. In either approach, the Code Club ends with a report back to the presenter and larger group describing the approach each pair took. Our Code Clubs typically have 7 to 10 participants, but the inherent "think-pair-share" strategy should allow it to be scaled to larger groups [3].

We continue to experiment with approaches to running Code Club, but have learned that it is critical for the presenter to clearly articulate their goals and facilitate participant engagement. Although any individual Code Club session may be more experimental than another, on the whole they are a critical tool to train bench scientists in reproducible data analysis practices. We have provided some examples of successful code club topics in Table 1 and summarized the results of our own experimentation as Code Club presenters and participants into Ten Rules. The first 3 rules apply to all participants at every Code Club, while Rules 4 through 8 are targeted to Code Club presenters

and the last two for participants.

Rule 1: Reciprocate respect

It is critical that the presenter and participants respect each other and that a designated individual (e.g. the lab director) enforces a code of conduct (see X for an example). Each member of the Code Club must have the humility to acknowledge that they have more to learn about any given topic. We have found that many problems are avoided when the presenter takes charge of the session with a clear lesson plan, thoughtful group creation, and constructive encouragement. Similarly, participants foster a positive environment by remembering that the task is not a competition, focusing on the presenter's goals, allowing their partner to contribute, asking clarifying questions when appropriate, and avoiding distractions (e.g. email, social media). Learning to program is challenging and too often results in a toxic environment of nitpicking and nonconstructive criticism. All parties in a Code Club are responsible for preventing this toxicity by demonstrating respect for themselves and their colleagues.

Rule 2: Let the material change you

Part of the humility required to participate in a Code Club is acknowledging that your training is incomplete and that it is possible for you to learn something new. For participants, assume that the presenter has a plan and follow their presented approach. After the Code Club, try to incorporate that material into new code or refactor old code. By practicing the material in a different context you will learn the material better. For presenters, incorporate suggested changes into your code. Either party may identify concepts that they are unsure of, presenting opportunities for further conversation and learning.

Rule 3: Experiment!

The selection of content and structure for each Code Club is democratic and distributed. If someone thinks a technique is worth learning and wants to teach it, they have that power. If they want to experiment with a different format, they are free to try it out. Members of the Code Club need to feel like they have the power to shape the direction of the group. If members are following these Rules, they will naturally reflect on the skills and interests of the other members in the group. For example, there is always turnover in a research group, making it important to revisit basic concepts to teach it to new people and provide a refresher to others. The group should also feel free to experiment with the format and incorporate group feedback by ending with a debrief discussing the pros and cons of new formats.

Rule 4: Set specific goals

In our early Code Clubs we noticed that if the presenter did not clearly state their goals for the session, it often led to frustration for both the presenter and participants. If the presenter shared their own code, did they want participants to focus on their coding style or did they want help incorporating a new package into their workflow? Participants will always notice or ask about code concepts that are not the focus of the exercise; a presenter with a specific goal can bring tangential conversations back to the planned task. Where possible, presenters should create a simplified scenario (i.e. minimal, reproducible example [4]), which can be helpful in focusing the participants. The presenter should verify ahead of time that the simplified example works and behaves the way they expect. Beyond the content, clear goals for participant activities will help both parties stay on task and avoid frustration. For more advanced learners, the presenter can create stretch goals or give an activity with multiple stopping points where participants would feel successful (e.g. commenting code, creating function, implementing function, refactoring function). Accomplishing specific goals is more likely to result in a positive outcome for presenters and participants.

Rule 5: Keep it simple

Our Code Club needs to fit within an hour time slot. When considering their Code Club activity, the presenter should plan for an introduction and brief instruction, time for participants to engage the material, and time for everyone to report back within that hour. A typical schedule for Code Club is 10 minutes of introduction and instruction, 30 minutes of paired programming, 5 minutes to get groups to wrap up, and 10 minutes to report back to the group. Remember that learners may need up to three times as long to complete a task that is simple for the presenter, so Code Club is best kept simple. We once had a presenter try to teach basic Julia syntax, but time was up by the time participants installed the interpreter. Some tips to help keep it simple are to limit the presented code chunks to less than 50 or, conversely, consider the number of lines that might be required to accomplish a solution.

Rule 6: Give participants time to prepare

Similar to a Journal Club, the presenter should give participants a few days (ideally a week) to prepare for the Code Club. Considering the compressed schedule described in Rule 5, asking participants to download materials beforehand is helpful to ensure a quick start. The presenter should provide the participants with instructions on how to install dependencies, download data, and get the initial code. This might also uncover weak points in the presenter's plan and enable them to ensure that the materials work as intended before the Code Club. We have found that using GitHub repositories for each Code Club can help make information, scripts, and data easily available to participants. Perhaps a first Code Club could be using git and GitHub to engage in collaborative coding. A lower barrier entry point is posting their code, data, and information in a lab meeting-dedicated Slack channel or sharing via email. Whatever method is used, the presenter should be sure to communicate the topic and necessary materials with the participants ahead of time.

Rule 7: Don't give participants busy work

Participants want to learn topics that will either be useful to them or help their colleague (i.e., the presenter). Presenters should do their best to satisfy those motivations, whether it is through the relevancy of the concept or the data. It does not make sense to present a Code Club on downloading stock market data if it is not useful or interesting to the group. Similarly, participants should not be tasked with improving the presenter's code if the presenter has no intention of incorporating the suggestions. A list of packages or tasks that group members are interested in could help an uninspired presenter choose a topic that will make a rewarding Code Club.

Rule 8: Include all levels of participants

As suggested by Rule 7, a significant challenge to presenting at Code Club is selecting topics and activities that appeal to a critical mass of the participants. This is particularly difficult if participants have a wide range of coding experience, such as follows a turnover in group membership. In these circumstances it can be useful to cover commonly used tools but that might result in disinterest of more experienced participants. We have identified several strategies to overcome the challenges presented by participants at varying skill levels. Instead of letting participants form their own pairs, the presenter can select pairs of participants with either similar or differing skill levels, depending on their goals. Partnerships between those with similar skill levels requires the presenter to design appropriate activites each skill level. We have found that commenting code is a good skill for beginners since it forces them to dissect and understand a code chunk line-by-line. An advantage of partnerships with disparate skill levels is assigning a single task, which provides the presenter with a diverse range of methods that achieve the same result. This approach to pairing also helps to graft new members into the group that have emerging programming skills. Regardless of how partners are selected, consider asking the pairs to identify a navigator and a driver [5]. The driver types at the computer while the navigator tells them what to type, thus ensuring participation of both partners. Midway through the activity, the presenter can have the partners switch roles. Intentionally forming pairs can also engineer group interactions by breaking up cliques, avoiding

potentially disruptive partnerships, or pairing reliable role models with new group members.

Rule 9: Prepare in advance to maximize participation

It is not possible to fully participate in a Journal Club discussion about a paper that the participant has not read. In that context, coming to Code Club without having installed a dependency is similar to asking a simple question about the Journal Club paper. Both instances show a lack of respect and preparation. Just as a presenter must follow Rule 6 to provide materials ahead of time, participants must review the code in advance, download the data sets, install the necessary packages, and perhaps read up on the topic. If the Code Club is based on a paper or chapters in a data science book (e.g., [6–8]) the participants should read them before the session and consider how they might incorporate the concepts into their own work.

Rule 10: Participate

An essential ingredient of any Code Club is active participation from all parties. Having an open laptop on the table and permission to use it can feel like an invitation to get distracted by other work, emails, and browsing the internet; fight that urge and focus on the presenter's goals. Be respectful and allow your partner to contribute. Speak up for yourself and force your partner to let you contribute. If the material seems too advanced for you, it can be frustrating, and tempting to mentally check out. Fortuntately, programming languages like R and Python are generally expressive, which should allow you to engage with the logic, even if the syntax is too advanced. Frankly, understanding the logic of when to use one modeling approach over another is more important than how to use it. If you understand the "why", the "how" will quickly follow. More experienced participants should aim to communicate feedback and coding suggestions at a level that all participants can understand and engage in. Regardless of skill levels, your partner and the presenter put themselves at risk by revealing what they do or do not know. Encourage them, and show your gratitude for helping you learn something new, by fully participating in each Code Club.

Conclusion

The most important rules are the first and last. Members of the Code Club need to feel comfortable with other group members and sufficiently empowered to try something new or ask for help. Aside from growing in our programming skills, we have noticed two other benefits that help create a positive work culture. First, we have intentionally interviewed postdoc candidates on the days we hold Code Club. We make it clear that they are not being assessed on their coding skills, but instead use it as an opportunity to see how the candidate interacts with other members of the research group in a relatively low stakes environment. At the same time, the candidate can learn about the culture of the research group through active participation. Second, members of other research groups that feel isolated in growing their skills have integrated themselves into our Code Club. This speaks to both the broader need for Code Club and the likelihood of success when expanded to include a larger group of individuals with broader research interests. Ultimately, Code Club has improved the overall data analysis skills, community, and research success of our lab by empowering researchers to seek help from their colleagues.

Acknowledgements

All co-authors have been participants in the Code Clubs run as part of PDS's lab meetings. They have had a critical role in shaping the evolution of the sessions. AKH led a lab discussion of Code Clubs to identify the key concepts presented, which were supplemented by contributions from NL. The order of co-authors was determined by. . .

References

1. Lonsdale A, Penington JS, Rice T, Walker M, Dashnow H. Ten simple rules for a bioinformatics journal club. Lewitter F, editor. PLOS Computational Biology. 2016;12: e1004526. doi:10.1371/journal.pcbi.1004526

2. Wilson G, Aruliah DA, Brown CT, Hong NPC, Davis M, Guy RT, et al. Best practices for scientific computing. Eisen JA, editor. PLoS Biology. 2014;12: e1001745. doi:10.1371/journal.pbio.1001745

3. Kothiyal A, Murthy S, Iyer S. Think-pair-share in a large CS1 class. Proceedings of the 2014 conference on innovation & technology in computer science education - ITiCSE 14. ACM Press; 2014. doi:10.1145/2591708.2591739

4. Stack Overflow. How to create a minimal, reproducible example. 2020. Available: https://stackoverflow.com/help/minimal-reproducible-example

5. Böckeler B, Siessegger N. On pair programming. 2020. Available: https://martinfowler.com/articles/on-pair-programming.html

6. Wickham H, Grolemund G. O'Reilly Media; 2016. Available: https://r4ds.had.co.nz

7. Schloss PD. The riffomonas reproducible research tutorial series. Journal of Open Source Education. 2018;1: 13. doi:10.21105/jose.00013

8. Noble WS. A quick guide to organizing computational biology projects. Lewitter F, editor. PLoS Computational Biology. 2009;5: e1000424. doi:10.1371/journal.pcbi.1000424

Table 1. Examples of successful Code Club topics.

Title	Description
base vs. ggplot	Given input data and a figure, recreate the figure using R's base graphics or ggplot syntax
Snakemake	Given a bash script that contains an analysis pipeline, convert it to a series of recipes (can also be done with GNU Make)
DRYing code	Given script with repeated code, create functions to remove repetition
mothur and Vegan	Given a pairwise community dissimilarity matrix, compare communities using the `adonis` function in the Vegan R package
tidy data	Given a wide-formatted data table, convert it to a long, tidy-formatted data table using tools from R's tidyverse
GitFlow	Participants file and claim an issue to add their name to a README file in a GitHub-hosted repository and file a pull request to complete the issue
Integrating R and Google docs	Scrape a Google docs workbook and clean the data to identify previous Code Club presenters