Ten Simple Rules for running a Code Club for increasing computational skills among biologists

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

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 the ability to generate data and the ability analyze data, 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 to develop the data analysis skills needed to analyze one's data. In our experience performing microbiome research, it is more common to find exceptional bench scientists who are inexperienced at analyzing large datasets than to find the reverse. Of course this raises a challenge: How do we train bench scientists to analyze datasets 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. Within our research group we have leveraged this similarity to address the challenges of training bench scientists to engage in reproducible data analysis practices. Our group and others hold a regular Journal Club to stay current on the literature and model to junior scientists how to engage that literature [1]. Over the past 4 years we have experimented with using a Code Club to introduce new software packages, solve specific problems, and evaluate each other's code.

Initially, the Code Club was created as a way to review code that was being used in different projects. We would project someone's script onto a screen and go line by line through the code with each person in the room stating the logic behind the code. This approach emphasized the importance of readability and it gave beginners the opportunity to see the real, "smelly" code of more experienced people in the group. Unfortunately, the format only allowed us to review a small fraction of a project's code base and it became difficult to integrate the programmer's logic across their full project. We quickly realized that although beginners could contribute to the discussion, the more senior group members dominated conversation. This frequently created an adversarial environment, which led to the presenter not offering to present again. From this experience we learned that it is critical for the presenter to clearly articulate their goals for the Code Club and that

the participants needed to engage with the code in that context.

Our Code Club sessions are each an hour long and are held roughly every other week as part of a weekly two hour lab meeting. Presenters voluntarily sign up to lead a session. The current format of our Code Club generally takes one of two approaches. The first is a more humane version of a group code review. The presenter will clearly state the problem they would like to solve, break the participants into smaller groups, and then ask each group to solve the problem. For example, someone may have an R script where a chunk of code is repeated throughout the script. The challenge for the session would be to convert the chunk into a function and call it 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 everyone relearns the importance of writing DRY code. The second is a tutorial approach. The presenter introduces a new package or way of doing things. They then give the group an activity that forces them to replace their current approach with a new approach. For example, one of the more memorable activities was giving participants raw data and a finished plot and asking two pairs of participants to generate the plot using base R and asking the other two pairs to generate the plot with ggplot. The stipulation was that base R users had to build the plots in ggplot and ggplot users had to build the plot in base R. Each Code Club ends with an opportunity to report back to the presenter and larger group. Our Code Clubs typically have 7 to 10 participants. Because of the "think-pair-share" approach that is baked into each session it could likely be scaled to larger groups [3].

We continue to experiment with approaches to running Code Club. Although any individual Code Club session may be be more experimental than others, on the whole they have been a critical tool to providing the much needed training to use reproducible practices for analyzing data. Examples of successful code club topics are described in Table 1. These Ten Rules summarize what we have learned as presenters and participants.

Rule 1: Reciprocate respect

It is critical that the presenter and participants respect each other and that there be an individual (e.g. the lab director) who enforces a code of conduct. Everyone in the session needs to have

the humility to acknowledge that they still have more to learn about any topic. We have found that many problems are avoided when the presenter takes charge of the session by having a clear lesson plan, thoughtfully creating groups, and giving 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 it can lead to a toxic environment. All parties in a Code Club are responsible for preventing this toxicity.

Rule 2: Set specific goals

In our early Code Clubs we noticed that if the presenter did not clearly state their goals for the session, it would often lead to frustration for the presenter and participants. If the presenter shared their own code, did they want us to focus on their coding style or did they want help with incorporating a new package into their workflow? Participants will always notice or ask about things that are not the focus of the exercise. Be specific about the goals and bring them back to the task if they are distracted. Where possible, creating a simplified scenario (i.e. minimal, reproducible example [4]) can be helpful to remove distractions. The presenter should make sure that the simplified example works and behaves the way they expect before coming to the Code Club. Beyond the content, participants should know what they can be expected to have completed by the end of the session.

Rule 3: Keep it simple

Our Code Club needs to fit within an hour time slot. The presenter should anticipate enough time for an introduction and brief instruction, time for participants to engage the material, and time for everyone to report back. A typical schedule for Code Review 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 a task that you can accomplish in 10 minutes may take a more junior member of your group 30 minutes. The goal of the Code Club should be to help that junior member to learn the topic and to reinforce the topic for the more senior person

and help them see the concept in a new context. One possibility would be to create "stretch goals" for those that complete the task more quickly than everyone else. We once had a presenter try to teach the group some basic Julia syntax. By the time people installed the interpreter, the time was up. Alternatively, a session where participants were asked to make code DRY gave them multiple stopping points where they could feel successful (e.g. documenting code, writing a function signature, implementing a function, refactoring code).

Rule 4: Give participants time to prepare

Similar to a Journal Club, the presenter should give participants a few days to prepare for the Code Club. Be sure to tell the participants what you will be covering. Provide them with instructions on how to install dependencies, download data, and get the initial code. Make sure that the materials work together as intended. Considering the compressed schedule, asking participants to download materials beforehand is helpful. We have found that using separate GitHub repositories for each session can be a helpful way to get information to participants. Of course, using git and GitHub to engage in collaborative coding should probably be its own Code Club before using the tools more broadly with the group. An easier entry point is to have groups post their code to a Slack channel that we use to announce materials for our meetings.

Rule 5: Prepare in advance to maximize your participation

It is the rare scientist that can fully participate in a Journal Club discussion about a paper that they have not read. Coming to Code Club without installing a dependency is like going to Journal Club and asking a simple question about the paper. It shows a lack of respect and preparation. Following Rule 4, review the code in advance, download the datasets, install the necessary packages, and perhaps read up on the topic. If your Code Club is reading a paper or chapters in a data science book, be sure to read them before the session (e.g. [5–7]). Consider how you could use the material for your own work.

Rule 6: Don't give your participants busy work

Participants want to learn something because it will be useful to them or because they think it will help the presenter to improve their code. Presenters should do their best to satisfy those desires. It would not make much sense to hold a Code Club on downloading stock market data if the group was not interested in those data. At the same time, participants should not be asked to help the presenter improve their code if the presenter has no intention of incorporating the suggestions. It would be useful to keep a list of packages or tasks that members of the group want to learn about. That way if a presenter is having a hard time coming up with ideas, they can take something from the list and present it.

Rule 7: Include all levels of participants

A significant challenge to presenting at Code Club is to pick topics and activities that appeal to a critical mass of the participants. For example, if there has recently been a large turn over in group membership, it can be useful to go back to the basics and cover commonly used tools. We have identified several strategies to overcome the challenges of having varying skill levels. As an alternative to letting participants form their own pairs, the presenter can be purposeful in how they form pairs. Depending on the goals of the session, forming pairs between participants that have similar or different skills can be effective. If pairs are formed where partners have similar skill levels, then the presenter needs to design activities that are appropriate to different levels of participants. We have found that a good activity for beginners is to have them comment code, which forces them to dissect a code chunk and understand what each line does. An advantage of forming pairs with disparate skill levels is that all of the pairs can be given the same task leading to a diversity of methods to achieve the same result. This approach to pairing also helps graft new members that have emerging programming skills into the group. Regardless of how partners are selected, asking the pairs to identify a navigator and driver is helpful [8]. The driver types at the computer while the navigator tells them what to type. Midway through the activity, the partners can switch roles. Intentionally forming pairs can also be a mechanism to engineer interactions by breaking up cliques, avoiding potentially toxic combinations of members, or pairing reliable role models with new members of the group.

Rule 8: Participate

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. Allow your partner to contribute. Speak up for yourself and force your partner to let you contribute. Even if the material seems too advanced for you, programming languages like R and Python are generally expressive and should allow you to engage with the logic, even if the syntax is too hard. Frankly, understanding the logic of *when* to use one modeling approach over another is more important than knowing *how* to use the modeling approach. If you understand the "why", the "how" will generally quickly follow. If you are a more experienced programmer, give your feedback and coding suggestions at a level that seeks to inform everyone. Regardless of your skill level, your partner and the presenter are putting themselves in a vulnerable position by revealing what they do or do not know. Encourage them and show your gratitude for helping you learn something new.

Rule 9: Let the material change you

Part of the humility required to participate in a Code Club is both acknowledging your incomplete training and that learning something new is possible. Assume that the presenter has followed the other rules and that they think you should be using what they presented. After the Code Club, try to incorporate that material into your new code or by refactoring your old code. By practicing the material in a different context, you will learn the material better. You may identify concepts that you are unsure of, which gives you another opportunity to ask the presenter for help.

Rule 10: Experiment!

The selection of content and structure for each Code Club is determined by the presenter, making the process democratic and distributed. If someone thinks something 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 turn over in a research

group and thus it is important to make sure that there are opportunities to go back to things people consider “simple” to teach the content to new people or provide a refresher to people that perhaps have been at the lab bench more than at the computer recently. The group should also feel free to experiment with the format. If we try a different format, then it is useful to give everyone an opportunity to debrief on what they did or did not like with the new approach.

Conclusion

The most important rules are the first and last. Members of the Code Club need to feel comfortable with other members of the group and sufficiently empowered to try something new or to ask for help. Aside from growing in our programming skills, we have noticed two other benefits that point to the efforts of creating a positive culture. First, we have intentionally interviewed postdoc candidates on the days we have held Code Club. Although it is important to make it clear that they are not being assessed on their coding skills, it is useful 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. Second, members of other research groups where they are isolated in their desire to grow in these skills have integrated themselves into our Code Club. This speaks to the broader need for Code Club and the likelihood that it would work when expanded to a larger group of individuals that are not necessarily in the same research group. Ultimately, our Code Club has helped researchers feel more comfortable seeking out their colleagues outside of Code Club sessions for help in improving their code.

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

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

7. 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

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

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 Snakemake workflow (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