Approachable case studies support learning and reproducibility in data science: An example from evolutionary biology

Luna L. Sanchez Reyes*
School of Natural Sciences, University of California, Merced
and
Emily Jane McTavish
School of Natural Sciences, University of California, Merced

February 15, 2022

Abstract

Research reproducibility is essential for scientific development. Yet, rates of reproducibility are low, especially in the natural sciences. As increasingly more research is relying on computing tools and software, efforts for improving reproducibility rates have focused on making available research workflows as computer code, as well as raw and processed data in computer readable form. However, research products that are digitally available are not necessarily friendly for learners and interested parties with little to no experience in the field. This renders research products unapproachable, which counteracts availability, and hinders reproducibility short and long term. To improve long term adoption of reproducible workflows in research, they need to be made approachable for learners, the researchers of the future.

Using a case study within evolutionary biology, we identify aspects of research workflows that make them unapproachable to the general audience: use of highly specialized language is intimidating; unspecified, unclear or lengthy goals lead to high cognitive load; content-focused descriptions instead of user-focused; and little to no diversity of representation of information. Then, we propose a set of principles to improve the unapproachable aspects of research workflows, and illustrate their application in a case study from evolutionary biology that we used as teaching material. Finally, we elaborate on the general application of these principles for documenting research workflows and products, to provide present learners and future researchers with tools for successful scientific reproducibility.

Keywords: open science, R, phylogenetics, Open Tree of Life, pedagogy

*The authors gratefully acknowledge "Sustaining the Open Tree of Life", NSF ABI No. 1759838, and ABI No. 1759846. They also acknowledge "The Carpentries" project, without it this project could not have been possible

Introduction

Research reproducibility –the extent to which consistent results are obtained when a scientific experiment or research workflow is repeated (Curating for Reproducibility Consortium 2017)– is a key aspect of the advancement of science, as it constitutes a minimum standard that allows understanding research products, i.e., methods, data, analysis, results, etc. (Piwowar 2013), to determine their reliability and generality, and eventually build up scientific knowledge and applications based on those products (King 1995, Peng 2011, Powers & Hampton 2019). In the natural sciences, rates of reproducibility are low (Ioannidis 2005, Prinz et al. 2011), which has elicited concerns about a crisis in the field (Baker 2016).

In response, the scientific community has been developing new principles and standards to incentivize cultural changes that support a long term improvement of reproducibility rates in the natural sciences (Peng 2015, Wilkinson et al. 2016, Miyakawa 2020). A standard for reproducibility that has received much attention is availability, which we define as a property denoting that a research product can be reached (acquired, copied, analyzed, processed and/or reused) at no financial, legal or technical cost (Arnold et al. 2019), and without geographic, demographic, social or temporal barriers for the population (Fecher & Friesike 2014).

In this paper, we argue that research products that are digitally available are often unapproachable in practice, because they are not friendly for learners and interested parties with different levels of experience in the field. Research products that are unapproachable counteract availability, and hinder reproducibility short and long term. To support long term adoption of reproducible practices in the natural sciences, research workflows need to be made approachable for learners, the researchers of the future (Roland et al. 2002).

To elaborate on our thesis, we designed a case study within the research field of phylogenetics, a discipline within evolutionary biology. We use our case study to identify barriers that have made research workflows largely unapproachable to a general audience in the natural sciences. Then, we propose some principles for researchers to address these barriers, and create research workflows that are reproducible by a larger audience. The principles proposed here can be generalized and integrated into the undergraduate and graduate school STEM curriculum, either for courses specialized in reproducibility or within other

subject areas, as a necessary component of successful and impactful science.

A case study from phylogenetics

Phylogenetics is a key discipline within evolutionary biology (Dobzhansky 1973). It focuses on investigating the history of shared ancestry of living and extinct organisms using biological data, and represents this evolutionary history with a diagram known as a phylogeny or phylogenetic tree (because it grows through time and appears to have branches; Figure 1). Phylogenies provide the basis to study and understand all biological processes in an evolutionary context (Dobzhansky 1973). Hence, it appears that improving reproducibility rates in phylogenetics has the potential to positively impact research across the natural sciences.

To explore barriers to approachable phylogenetics, we develop a case study that touches on three common problems within the field: standardizing organism names in phylogenies, obtaining current phylogenetic knowledge for a group of organisms, and summarizing this phylogenetic knowledge in a meaningful way. To address these problems, we propose a research workflow that relies on resources from the Open Tree of Life (OpenTree), an open source project that provides digital availability of phylogenetic results from published, peer-reviewed research, which is considered as vetted and state-of-the-art knowledge in the field. OpenTree phylogenies are stored in a public database, the Phylesystem (McTavish et al. 2015), and are downloadable as various computer-readable file types, which is key for reusability and reproducible workflows (Wilson et al. 2017). OpenTree also provides access to a single standard for organism names (taxonomic standard) that is applied to the stored phylogenies (Rees & Cranston 2017), which are then used to summarize a single phylogenetic tree encompassing all life (Open Tree Of Life et al. 2019).

All of these resources are available for download and use from OpenTree free of cost to any user. One way to access OpenTree resources is through it Graphical User Interface (GUI; aka, a website or application that allows users to access computer functionalities, in this case OpenTree resources, with mouse or keyboard clicks). However, reducing as many manual steps as possible in research workflows is key for reproducibility, as manual data manipulation scales poorly and is prone to error (Bakken 2019). OpenTree's resources

are also programmatically available through its Application Programming Interface services (APIs; aka, computer code that implements computer functionalities, in this case OpenTree resources, that can be used by programmers to build more functionalities), which provide scalability and reproducibility (Open Tree Of Life et al. 2016). However, this comes at a high cost for the user, which requires considerable more computer programming experience and literacy to be able to successfully use APIs. The `rotl` R package (Michonneau et al. 2016) and the `opentree` Python module (McTavish et al. 2021) have been developed to wrap OpenTree's API services and make them available through the R and Python programming languages. R and Python are open source and free of cost, and represent two of the most widely used programming languages in the sciences today (Eglen 2009, Baker 2017). As such, `rotl` and `opentree` software packages should contribute to making OpenTree resources more accessible to a wider programming user audience.

However, while learners in the natural sciences have been engaging independently with R and Python programming languages, computer programming is not traditionally a core skill formally taught to biologists and naturalists (Sayres et al. 2018, Wright et al. 2019, Williams et al. 2019). As computers continue to play a larger role in most scientific disciplines (Piccolo & Frampton 2016), higher baseline computational skills are required across all natural sciences not only to develop an original research workflow, but to be able to follow and reproduce research workflows from other researchers.

Thus, efforts to increase reproducibility rates long term in the natural sciences would benefit from addressing specific barriers for learners in the field, to support them in acquiring the skills needed to reproduce research workflows that rely heavily on computer code (Peng 2011, Sandve et al. 2013, Powers & Hampton 2019).

In the next section, we describe (in no particular order) the barriers to approachable research workflows that we identified on our case study. Then we develop a set of principles to address these barriers, and apply the latter to a set of teaching materials that are available at https://mctavishlab.github.io/R_OpenTree_tutorials/.

Identifying barriers to approachable research workflows

The main goal of our case study is to obtain a single phylogeny summarizing data from a set of published phylogenies for the canids (the family of dogs, coyotes, wolves, etc.), our organism of study. All analysis for our case study can be completely accomplished using functions from the R package `rotl` or the Python module `opentree`. If a researcher were to use the proposed analysis workflow in a publication, they would typically describe it in the methodology section as "The canid summary phylogeny was obtained using functions from X package, details are available as supplementary files". This is usual practice, mainly because journals do not have space to publish all code used for an analysis in the methods section. Yet, supplementary files and data have the misfortune to not be peer-reviewed as thoroughly (or at all) as the main manuscript (Pop & Salzberg 2015). They are also prone to the dreaded promise "available upon request", which has very low rates of fulfillment (Krawczyk & Reuben 2012). Without the primary data and code that was used to perform an analysis it is impossible to reproduce said analysis (Miyakawa 2020). When the code is available, other issues can complicate reproduction of the analysis, to the point of completely obstructing reproducibility. For example, some questions that are often unaddressed in research workflows are: what software do I need to open the code files? Can I run the code with the same software I used to open the files, or do I need a different one? Do I need additional software that the analysis depends on? What does the code even mean?

These questions can usually be answered by referring to the software documentation, which are usually publicly available and can be accessed by any potential user. As opposed to code, software documentation is written in natural language (i.e., any known human language, e.g., English, Spanish, Chinese), and is considered a key element for successful adoption of software by target users (Karimzadeh & Hoffman 2018). This might explain why documentation for software addressed to academic users is also usually written using highly specialized computational language or jargon (i.e., computationally specific concepts, words, and phrases) as well as formal scientific and academic language. We identify this as *barrier 1* to approachable research workflows.– *Specialized language is intimidating.* While scientific jargon might have an important role for formal acceptance of software by

the scientific and academic community, it can be perceived as cold and/or intimidating language, that often slows down or even obstructs examination, application, and adoption of code by a wider audience (Ball 2017). In contrast, introducing information without the use of jargon, supports learners' conceptual understanding of new ideas and concepts (McDonnell et al. 2016, Pan et al. 2019).

Another element of good software documentation is that it has to be thorough (Karimzadeh & Hoffman 2018). Meaning that it should describe general usage of individual functions, as well as arguments and variables that said function can take (Karimzadeh & Hoffman 2018). Individual documentation for each function is usually presented in alphabetic order, and does not have a specific analysis goal. Moreover, most software has numerous functions, so documentation is usually very lengthy and it is hard to navigate. This can have the effect of increasing the amount of information that needs to be simultaneously processed by the users, which can lead to overload of the finite amount of working memory any one possesses, known as cognitive load (Sweller 1988). In this context, identifying connections across functions that are meant to work on the same analysis workflow can become a very difficult task. We recognize this as *barrier 2.– Lack of specific goals leading to high cognitive load*. High cognitive load is know to have a negative effect in learning software (Chandler & Sweller 1996, Van Merriënboer & Ayres 2005, Lambert et al. 2009).

A third important aspect of software documentation are examples that demonstrate usage of individual functions (Karimzadeh & Hoffman 2018). Examples presented in software documentation are usually worked to perfection, as they are intended to showcase the ideal or minimal case in which a function works well. Perfectly worked examples ignore the user experience by maintaining focus on the software content, and fail to provide users with expert and clear advice on how to troubleshoot if needed. We identify this as *barrier 3.– Content-focused examples*. Error management training is an approach that focuses on framing mistakes as beneficial to learning complex tasks, to give learners the opportunity to actively explore a task (Frese 1995). Providing examples that showcase potential errors supports users performance (Steele-Johnson & Kalinoski 2014), and can greatly improve learners ability to troubleshoot outside the classroom (Shannon & Summet 2015, Nederbragt et al. 2020).

Last but not least, software documentation is usually explained verbally in a document, very rarely is it approached using diagrams or other allegories. We recognize this as *barrier 4.– Little to no diversity of representation of information.*

In sum, best practices for good software documentation are not enough to promote reproducibility of research workflows that rely heavily on code. In the following section we describe some principles that can help to reduce or remove the identified barriers, to create research workflows that are more approachable and hence more reproducible by a larger audience.

Principles for approachable research workflows

Principle 1. Use friendly, relatable and respectful language

Besides avoiding formal language, and incorporating elements of pop culture, such as picture character icons known as "emojis", to make the language more familiar to a broader target audience (see Figure 2), we made an effort to specifically complement the primary documentation by identifying computational concepts that were assumed or were not explained in depth. We vetted the tutorials through feedback from workshop participants as well as individual users to identify such specialized concepts.

Principle 2. Reduce cognitive load by providing specific and clear goals with literate programming

Cognitive load can be greatly reduced for learners by applying an active learning strategy such as linking usage to a "real world" or "human" application (Felder & Brent 2009). Programming computer languages are by themselves quite abstract and represent a learning subject with a potentially high cognitive load for most learners. Pedagogical research shows that active learning practices are one of the most effective ways to take on abstract subjects (Freeman et al. 2014).

A story-like narrative that links code usage in an integrative example, invites learners to try the code, which can lead them to remember what they are doing and why they are doing it. This "literate programming" paradigm (Knuth 1984, Fritzson et al. 2002)

makes code more approachable, as it integrates narratives with computer code in the same document, supporting learners in actively following the code usage, supporting memory and understanding (Piccolo & Frampton 2016).

We propose that documents developed with "literate programming" can be made more accessible by choosing narratives that are relatable to a more general audience. An easy way to do this in biology is choosing a charismatic taxon as model organism. For a research group this can be the group they are studying. For the general audience, a highly charismatic group such as dinosaurs will do the trick.

We examined available primary documentation for the package `rotl`, and designed a narrative that required the usage of as many functions as possible. We demonstrate code applications that are commonly requested by OpenTree users, but that are not demonstrated in the primary documentation of the R package. By framing the function workflow using highly requested uses, the documentation acquires a narrative arc that is easier to follow and remember by users. This can also facilitate the application of code to other use cases in biology of interest for the users.

Principle 3. Provide examples that are user-focused by demonstrating errors and warnings

A practice that has become more and more widespread in programming-language pedagogical practices is live programming. One benefit of live programming is that typos and mistakes occur – which normalizes them for learners, and shows them how to solve them when they are outside the classroom (Shannon & Summet 2015, Nederbragt et al. 2020). When co-author McTavish was a postdoc, teaching an introductory programming workshop as a volunteer with the Carpentries (a non-profit group that teaches foundational coding and data science skills to researchers worldwide) (Wilson 2006, 2022), a senior faculty member taking the workshop complained that the typos were slowing things down and interfering with the pedagogy. McTavish replied "the typos ARE the pedagogy". This has become a slogan of sorts at the Carpentries, capturing the idea that embracing and discussing mistakes is essential to teaching programming (Wilson 2019). Yet, working through mistakes is rarely done on written pedagogical materials. Software documentation focuses

on demonstrating usage function with examples that work seamlessly, without errors. We argue that the opposite is needed to support adoption of reproducible workflows and support long term independence in learners and users performance (Gaspar & Langevin 2007, Steele-Johnson & Kalinoski 2014). In our tutorials, we apply this principle by demonstrating examples that do not work as expected, and exemplifying ways to address them (Figure 2).

We identified inputs that would give a wide range of warnings and errors, and support learners on reading the information provided by the message. This helps users to not be afraid of errors and warnings, but instead use them as an informative measure. We also identify effects of warnings and errors downstream of the workflow.

We identify ways to evaluate inputs to know if they will produce an error, and design alternatives on what to do when faced with an error or warning, and demonstrate these alternatives. One of the most essential skills in programming is interpreting and moving forward from errors.

In our tutorial, we focus on explaining the meaning of warnings and errors, and showcase ways to detect them before they are triggered (i.e., before using an input that would elicit a warning or error). This has two pedagogical benefits: 1) it provides users/students with the means to troubleshoot their own warnings and errors, and 2) it allows users/students to understand with more depth what the function is doing.

Principle 4. Demonstrate one thing in different ways

Conclusion

Response from the community has been invaluable in gauging success of our teaching materials. Senior researchers often comment on the usefulness of the tutorials for their research, as well as how they have supported students in using the R packages with less help from them as PIs.

One key aspect for success of our tutorials is that we applied all rules for successful reproducibility, in particular making our tutorials available through time in a public website.

Making approachable research workflows has several advantages:

* save explanation/training time when analyses are run again by students and collaborators; * save research time for yourself when analyses are run again with more data, a* different dataset, a different organism or biological model; * scientific efforts can build off of each other.

Ultimately, the long term improvement of reproducibility rates in science will depend on our ability to intentionally integrate reproducibility into the undergraduate curriculum, so college learners and future researchers have the basis to develop the fundamental skills needed to successfully create reproducible scientific workflows and materials.

Some universities have been incorporating the subject in their classes (see University of Washington Libraries - (2022), NIGMS Career Curriculum Development - (2015)). The focus of these resources has been for students to develop skills to document their work. The principles identified and outlined here can be used to set learning goals and outcomes on new reproducibility syllabi.

The principles to create tutorials described here facilitate adoption of software and analysis workflows among researchers at different academic levels, from undergrads to established researchers. It will also help close the gap between students that had access to computational resources (and computational training) from an early age and students that did not. Late access to computational resources and training can occur due to lack of economic resources, often occurring in households from underrepresented communities and minorities (Google Inc. & Gallup Inc. 2016, Warner et al. 2021). It can also be due to gender-biased parental and community pressures, in which male individuals are more often encouraged to perform activities related to computers, while female individuals are discouraged (Warner et al. 2021). These principles can be used to improve not only reproducibility practices, but also software adoption in the natural sciences.

SUPPLEMENTARY MATERIAL

Title: Website and GitHub repository containing the complete teaching materials developed and demonstrated here.

GitHub repository link: `https://github.com/McTavishLab/R_OpenTree_tutorials`

```R
ape::plot.phylo(my_tree, cex = 2) # or just plot(my_tree, cex = 2)
```

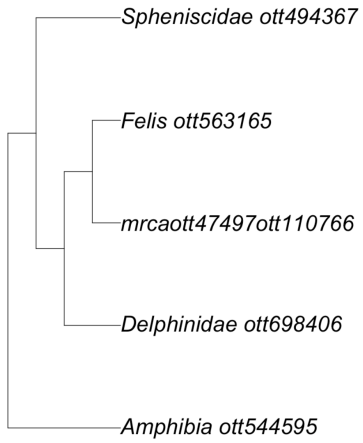

Figure 1: A phylogenetic tree from our tutorial. It was extracted using OpenTree of Life resources (Open Tree Of Life et al. 2019) wrapped in the `rotl` R package (Michonneau et al. 2016).

Website link: `https://mctavishlab.github.io/R_OpenTree_tutorials`

FIGURES

References

Arnold, B., Bowler, L., Gibson, S., Herterich, P., Higman, R., Krystalli, A., Morley, A., O'Reilly, M., Whitaker, K. et al. (2019), 'The turing way: a handbook for reproducible data science (version v1.0.1)', *Zenodo* .

Baker, M. (2016), 'Is there a reproducibility crisis?', *Nature* **533**(26), 353–66.

Baker, M. (2017), 'Scientific Computing: Code Alert', *Nature* **541**(7638), 563–565.

Bakken, S. (2019), 'The journey to transparency, reproducibility, and replicability'.

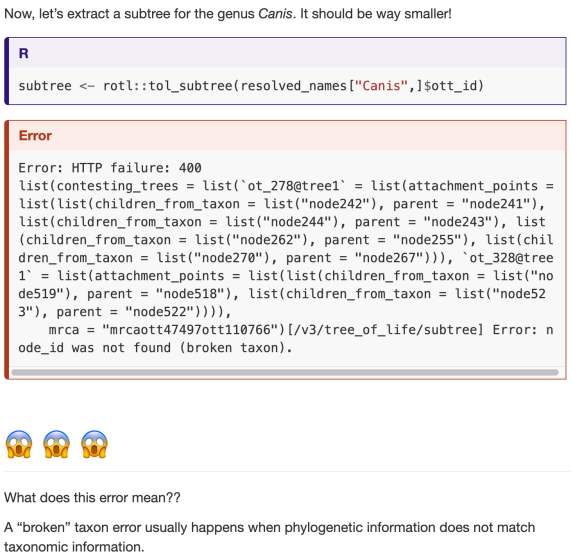

Figure 2: Snapshot of a section of the tutorial website, where we demonstrate a common error.

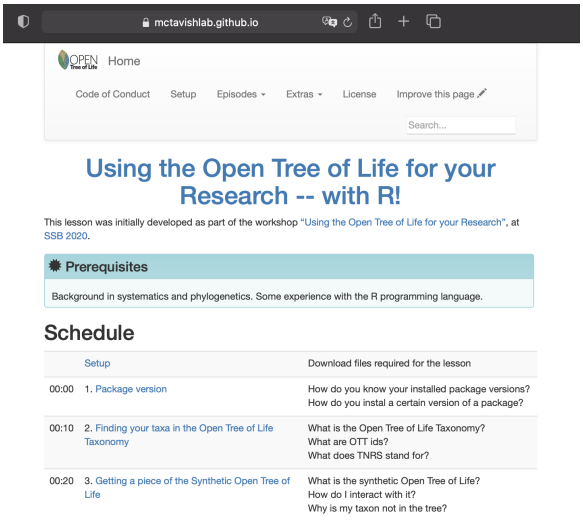

Figure 3: Snapshot of the home to our tutorial website, showing part of the schedule. Our tutorial website was constructed using The Software Carpentries workshop template (Wilson 2016).

Ball, P. (2017), 'It's not just you: science papers are getting harder to read', *Nature* **30**.

Chandler, P. & Sweller, J. (1996), 'Cognitive load while learning to use a computer program', *Applied Cognitive Psychology* **10**(2), 151–170.

Curating for Reproducibility Consortium (2017), 'Defining "reproducibility"'.
URL: *https://cure.web.unc.edu/defining-reproducibility/*

Dobzhansky, T. (1973), 'Nothing in biology makes sense except in the light of evolution', *The American Biology Teacher* **35**(3), 125–129.

Eglen, S. J. (2009), 'A quick guide to teaching r programming to computational biology students', *PLoS computational biology* **5**(8), e1000482.

Fecher, B. & Friesike, S. (2014), *Open Science: One Term, Five Schools of Thought*, Springer International Publishing, pp. 17–47.

Felder, R. M. & Brent, R. (2009), 'Active learning: An Introduction', *ASQ Higher Education Brief* **2**(4), 1–5.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H. & Wenderoth, M. P. (2014), 'Active learning increases student performance in science, engineering, and mathematics', *Proceedings of the National Academy of Sciences* **111**(23), 8410–8415.

Frese, M. (1995), Error management in training: Conceptual and empirical results, *in* 'Organizational Learning and Technological Change', Springer, pp. 112–124.

Fritzson, P., Gunnarsson, J. & Jirstrand, M. (2002), Mathmodelica - an extensible modeling and simulation environment with integrated graphics and literate programming, *in* '2nd International Modelica Conference, March 18-19, Munich, Germany'.

Gaspar, A. & Langevin, S. (2007), Restoring "coding with intention" in introductory programming courses, *in* 'Proceedings of the 8th ACM SIGITE conference on Information technology education', pp. 91–98.

Google Inc. & Gallup Inc. (2016), 'Diversity gaps in computer science: exploring the underrepresentation of girls, blacks and hispanics', *Retrieved from http://goo.gl/PG34aH* (Additional reports from Google's Computer Science Education Research are available at g.co/cseduresearch.).

Ioannidis, J. P. (2005), 'Why most published research findings are false', *PLoS Medicine* **2**(8), e124.

Karimzadeh, M. & Hoffman, M. M. (2018), 'Top considerations for creating bioinformatics software documentation', *Briefings in Bioinformatics* **19**(4), 693–699.

King, G. (1995), 'Replication, replication', *PS: Political Science & Politics* **28**(3), 444–452.

Knuth, D. E. (1984), 'Literate programming', *The Computer Journal* **27**(2), 97–111.

Krawczyk, M. & Reuben, E. (2012), '(un) available upon request: field experiment on researchers' willingness to share supplementary materials', *Accountability in Research* **19**(3), 175–186.

Lambert, J., Kalyuga, S. & Capan, L. A. (2009), 'Student perceptions and cognitive load: what can they tell us about e-learning web 2.0 course design?', *E-learning and Digital Media* **6**(2), 150–163.

McDonnell, L., Barker, M. K. & Wieman, C. (2016), 'Concepts first, jargon second improves student articulation of understanding', *Biochemistry and Molecular Biology Education* **44**(1), 12–19.

McTavish, E. J., Hinchliff, C. E., Allman, J. F., Brown, J. W., Cranston, K. A., Holder, M. T., Rees, J. A. & Smith, S. A. (2015), 'Phylesystem: a git-based data store for community-curated phylogenetic estimates', *Bioinformatics* **31**(17), 2794–2800.

McTavish, E. J., Sánchez Reyes, L. L. & Holder, M. T. (2021), 'OpenTree: A Python Package for Accessing and Analyzing Data from the Open Tree of Life', *Systematic Biology* .

URL: *https://doi.org/10.1093/sysbio/syab033*

Michonneau, F., Brown, J. W. & Winter, D. J. (2016), 'rotl: an R package to interact with the Open Tree of Life data', *Methods in Ecology and Evolution* **7**(12), 1476–1481.

Miyakawa, T. (2020), 'No raw data, no science: another possible source of the reproducibility crisis'.

Nederbragt, A., Harris, R. M., Hill, A. P. & Wilson, G. (2020), 'Ten quick tips for teaching with participatory live coding', *PLOS Computational Biology* **16**(9), e1008090.

NIGMS Career Curriculum Development - (2015), 'Rigor & Reproducibility, National Institute of General Medical Sciences'.
URL: *https://www.nigms.nih.gov/training/instpredoc/Pages/admin-supplements-prev.aspx*

Open Tree Of Life, Redelings, B., Cranston, K. A., Allman, J., Holder, M. T. & McTavish, E. J. (2016), 'Open Tree of Life APIs v3.0', *Open Tree of Life Project* (Online Resources).
URL: *https://github.com/OpenTreeOfLife/germinator/wiki/Open-Tree-of-Life-Web-APIs*

Open Tree Of Life, Redelings, B., Sánchez Reyes, L. L., Cranston, K. A., Allman, J., Holder, M. T. & McTavish, E. J. (2019), 'Open tree of life synthetic tree v12.3', *Zenodo* .
URL: *https://doi.org/10.5281/zenodo.3937742*

Pan, S. C., Cooke, J., Little, J. L., McDaniel, M. A., Foster, E. R., Connor, L. T. & Rickard, T. C. (2019), 'Online and clicker quizzing on jargon terms enhances definition-focused but not conceptually focused biology exam performance', *CBE—Life Sciences Education* **18**(4), ar54.

Peng, R. (2015), 'The reproducibility crisis in science: A statistical counterattack', *Significance* **12**(3), 30–32.

Peng, R. D. (2011), 'Reproducible research in computational science', *Science* **334**(6060), 1226–1227.

Piccolo, S. R. & Frampton, M. B. (2016), 'Tools and techniques for computational reproducibility', *Gigascience* **5**(1), s13742–016.

Piwowar, H. (2013), 'Value all research products', *Nature* **493**(7431), 159–159.

Pop, M. & Salzberg, S. L. (2015), 'Use and mis-use of supplementary material in science publications'.

Powers, S. M. & Hampton, S. E. (2019), 'Open science, reproducibility, and transparency in ecology', *Ecological Applications* **29**(1), e01822.

Prinz, F., Schlange, T. & Asadullah, K. (2011), 'Believe it or not: how much can we rely on published data on potential drug targets?', *Nature Reviews Drug discovery* **10**(9), 712–712.

Rees, J. A. & Cranston, K. (2017), 'Automated assembly of a reference taxonomy for phylogenetic data synthesis', *Biodiversity Data Journal* (5).

Roland, M.-C., Chèvre, A.-M., Chadoeuf, J., Hubert, B. & Bonnemaire, J. (2002), Think forward, act now: training young researchers for sustainability. reshaping the relationship between phd student and adviser, *in* '5. International COPERNICUS Conference', number 8, VAS Verlag für Akademische Schriften.

Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. (2013), 'Ten simple rules for reproducible computational research', *PLoS Computational Biology* **9**(10), e1003285.

Sayres, M. A. W., Hauser, C., Sierk, M., Robic, S., Rosenwald, A. G., Smith, T. M., Triplett, E. W., Williams, J. J., Dinsdale, E., Morgan, W. R. et al. (2018), 'Bioinformatics core competencies for undergraduate life sciences education', *PloS One* **13**(6), e0196878.

Shannon, A. & Summet, V. (2015), 'Live coding in introductory computer science courses', *Journal of Computing Sciences in Colleges* **31**(2), 158–164.

Steele-Johnson, D. & Kalinoski, Z. T. (2014), 'Error framing effects on performance: cognitive, motivational, and affective pathways', *The Journal of psychology* **148**(1), 93–111.

Sweller, J. (1988), 'Cognitive load during problem solving: Effects on learning', *Cognitive Science* **12**(2), 257–285.

University of Washington Libraries - (2022), 'Teaching Reproducibility'.
URL: *https://guides.lib.uw.edu/research/reproducibility/teaching*

Van Merriënboer, J. J. & Ayres, P. (2005), 'Research on cognitive load theory and its design implications for e-learning', *Educational Technology Research and Development* **53**(3), 5–13.

Warner, J. R., Childs, J., Fletcher, C. L., Martin, N. D. & Kennedy, M. (2021), Quantifying disparities in computing education: Access, participation, and intersectionality, *in* 'Proceedings of the 52nd ACM Technical Symposium on Computer Science Education', pp. 619–625.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E. et al. (2016), 'The fair guiding principles for scientific data management and stewardship', *Scientific Data* **3**(1), 1–9.

Williams, J. J., Drew, J. C., Galindo-Gonzalez, S., Robic, S., Dinsdale, E., Morgan, W. R., Triplett, E. W., Burnette III, J. M., Donovan, S. S., Fowlks, E. R. et al. (2019), 'Barriers to integration of bioinformatics into undergraduate life sciences education: A national study of us life sciences faculty uncover significant barriers to integrating bioinformatics into undergraduate instruction', *PLoS One* **14**(11), e0224288.

Wilson, G. (2006), 'Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive', *Computing in Science & Engineering* **8**(6), 66–69.

Wilson, G. (2016), 'Software Carpentry: Workshop Template v2016.06'.
URL: *https://github.com/carpentries/workshop-template*

Wilson, G. (2019), *Teaching Tech Together: How to Make your lessons work and build a teaching community around them*, CRC Press.

Wilson, G. (2022), 'The Carpentries', *Website* .
URL: *http://software-carpentry.org*

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L. & Teal, T. K. (2017), 'Good enough practices in scientific computing', *PLoS Computational Biology* **13**(6), e1005510.

Wright, A. M., Schwartz, R. S., Oaks, J. R., Newman, C. E. & Flanagan, S. P. (2019), 'The why, when, and how of computing in biology classrooms', *F1000Research* **8**.