ARTICLE

Integrating Programming into Neuroscience Courses

Ashley L. Juavinett
Neurobiology Department, University of California, San Diego, La Jolla, CA 92037.
https://doi.org/10.59390/PYYP5010

Programming is a useful skill for students, both in neuroscience research and in the broader economy. Many instructors, however, may wonder how and when they should integrate it into their coursework, especially if they themselves have limited computational training. The suggestions offered here aim to help a wide range of educators — even those who have minimal coding experience — who wish to expose their students to the intersection of neuroscience and programming. Throughout, I provide examples of how I have weaved coding into various elements of neuroscience courses, even those without a computational focus. I also discuss the rich landscape of low-cost, accessible programming tools as well as how generative AI can (and should) impact the way that we are teaching programming. Ultimately, the goal is not to insist that *all* our students learn how to code, but rather to lower barriers and provide exposure and opportunity to any student who wishes to integrate programming into their research or careers.

Key words: programming, computational, quantitative, laboratory courses

Like most STEM fields as well as the broader economy, neuroscience research is increasingly reliant on programming. Datasets are larger and more complex, computational models contain multitudes, and open science is making it possible for anyone to dig in (Sejnowski et al., 2014; Paninski and Cunningham, 2018). As myself and others have argued over the years, these changes are necessitating changes in both undergraduate and graduate education (Akil et al., 2016; Grisham et al., 2016, 2017; Juavinett, 2022). To work in these rapidly expanding intersections, students need strong quantitative foundations, such as statistics and data science, alongside practical skills such as programming. In this article, I focus on the latter, exploring three different ways that educators can integrate programming into their neuroscience curricula.

Despite the clear importance of offering exposure to coding, there are a variety of barriers, from limited course offerings to instructor knowledge of coding (Juavinett, 2022; Casimo, 2023). In addition, there are attitudinal barriers, especially from students who have been historically (and continue to be) excluded from coding (Margolis and Fisher, 2003; Cheryan et al., 2009, 2015; Lewis et al., 2016). To ensure that neither instructors nor their students are left behind, we need to consider each of these challenges as we build innovative programming curricula. The hope here is to provide a starting guide for instructors who wish to integrate programming into their courses but are not sure where to start.

TOOLS FOR INTEGRATING PROGRAMMING

While pervasive stereotypes about who codes as well as access to computing courses continue to interfere with inclusion, other more logistical barriers have been significantly reduced in recent years. Specifically, there are many different platforms on which students and educators can learn how to code, and these environments support a variety of languages. Thus, educators have two choices when getting started: programming language and platform.

Choice of Programming Language

Programming languages differ in their syntax (the rules of the language), speed, and usage. While there is not one best programming language to learn for neuroscientists, there are important considerations: ease of readability, cost and accessibility, and use within our discipline. There are lower-level, more difficult to read languages (such as Java) and languages which are higher-level and easier to read (such as Python, R, and MATLAB). For programming beginners who are budding neuroscience researchers, these higher-level languages are generally a better choice. Second, some computing platforms (e.g., MATLAB) are expensive and therefore cost prohibitive for institutions or students without licenses. On the other hand, languages such as Python and R are open source and free for students to use.

An additional consideration is the prevalence of a language in neuroscience research. While this is a tough feature to assess quantitively, MATLAB and Python are both commonly used for data analysis, visualization, and computational modeling, while R is often used for statistics as well as bioinformatics (Muller et al., 2015; Schlafly et al., 2020; Grisham et al., 2021). Relatedly, we can consider the utility of a language in the broader workforce — another tough aspect to assess and constantly changing — however, Python and R are both defensible choices when it comes to data science or related fields.

A final and important consideration in this context is an instructor's experience with a given language. It is a very big ask for someone to switch languages entirely for the purposes of teaching, especially without time or incentives. It is therefore understandable for an instructor to choose the language they know best. Instructors seeking to improve their own programming knowledge are strongly encouraged to seek out training such as that offered through The Carpentries (https://carpentries.org/), which also offers instructor training, or other online resources (e.g., DataCamp).

Considering each of these factors, I predominantly teach Python in my classes while many of my colleagues in biology focus on R. That said, I will not begrudge any instructor for choosing the language that is most comfortable for them — in reality, any coding language is a great place to start. In my opinion, the most important consideration is that instructors do not engender fear about programming, but rather empower their students to learn, to make mistakes, and to ask questions, so that they too can build self-efficacy in programming.

Choice of Programming Environment

Educators also need to decide on the best programming environment for their students. Programming can be taught using offline or online tools. Traditional computing education teaches line-by-line coding in the terminal or by running short scripts, while more modern computing education takes advantage of free online tools or user-friendly integrated development environments (IDEs). These IDEs (e.g., MATLAB, R Studio, or VS Code), usefully contain a command prompt, an area to write scripts, and windows to show outputs or the contents of current variables.

In the past decade, there has also been a proliferation of free online platforms (essentially lightweight IDEs) which have made it possible for anyone with an internet connection to try programming. For example, free cloud computing tools such as Google Colab (http://colab.research.google.com) and Binder (http://mybinder.org) have made it such that students and departments do not need to download hefty software onto their computers. Google Colab and Binder allow students to both review explanatory markdown text and interact with live code which can be pre-written. In this article, I refer to such cloud computing notebooks, regardless of where they are hosted, as simply "coding notebooks." Even MATLAB has developed an online computing environment, a clever workaround for what is otherwise a large program to install. Further easing logistical barriers, such online platforms run in a browser and therefore work on tablets as well as laptops.

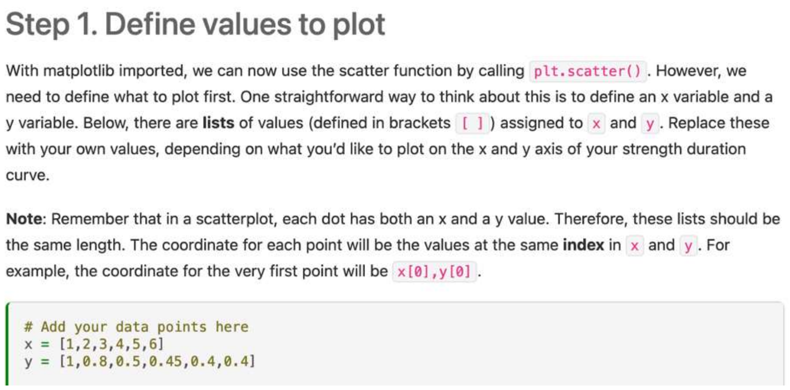
Inspired by other instructors at my institution, I primarily teach programming in coding notebooks, which have partially written code that is completed with live coding in class. These notebooks are hosted on GitHub (www.github.com), a common place for professionals to post their code, and run on our institution's JupyterHub or run via Google Colab.

THREE INTEGRATION STRATEGIES

With this motivation in mind and using widely available tools, here I offer three possible ways of integrating programming into neuroscience curricula, with concrete examples (Table 1). Each of these strategies are designed for students with limited or no programming backgrounds, though some statistical understanding will be useful for programming for data analysis. These approaches can be implemented using free computing platforms such as Google Colab, which are especially useful without institutional support for a locally hosted JupyterHub (or similar computational platform).

	Exposure to coding in coursework	**Integration of coding in coursework**	**Discipline-based coding classes**
Learning Outcomes *Students will be able to:*	• Recognize use cases for programming in neuroscience • Edit provided code examples, often to generate figures or analyze data	*Previous column, plus:* • Write additional code based on provided examples • Edit code to address a research question	*Previous column, plus:* • Design coding workflows to address a data analysis challenge *and/or* • Write code to illustrate a computational model
Time required	< 1 hour	> 1 hour	One semester/ quarter
Instructor comfort with coding	Novice: Instructors can troubleshoot basics of coding notebooks and simple examples	Novice: Instructors can troubleshoot more complex code examples	Expert: Instructors can explain mechanics of code, troubleshoot complex code, write additional examples and problem sets for students, and write code to analyze neuroscience data
Examples	• Implementation in a laboratory course: https://bipn145.github.io/	• Analysis of Allen Institute cell types data: Juavinett, 2020; http://github.com/ajuavinett/CellTypesLesson/ and Ho et al., 2021 • Bioinformatics module: Madlung, 2018	• Introductory computing for biologists: http://www.github.com/BILD62 or Libeskind-Hadas and Bush, *Computing for Biologists* (see also Dodds et al., 2012) • "Case studies in Python": https://mark-kramer.github.io/Case-Studies-Python • Advanced coursework: Neuromatch Academy https://compneuro.neuromatch.io

Table 1. Three different strategies to integrate programming into neuroscience courses, along with their learning objectives, time required, the recommended level of instructor coding background, and resources with examples.



Step 1. Define values to plot

With matplotlib imported, we can now use the scatter function by calling `plt.scatter()`. However, we need to define what to plot first. One straightforward way to think about this is to define an x variable and a y variable. Below, there are **lists** of values (defined in brackets `[ ]`) assigned to `x` and `y`. Replace these with your own values, depending on what you'd like to plot on the x and y axis of your strength duration curve.

Note: Remember that in a scatterplot, each dot has both an x and a y value. Therefore, these lists should be the same length. The coordinate for each point will be the values at the same **index** in `x` and `y`. For example, the coordinate for the very first point will be `x[0],y[0]`.

```
# Add your data points here
x = [1,2,3,4,5,6]
y = [1,0.8,0.5,0.45,0.4,0.4]
```

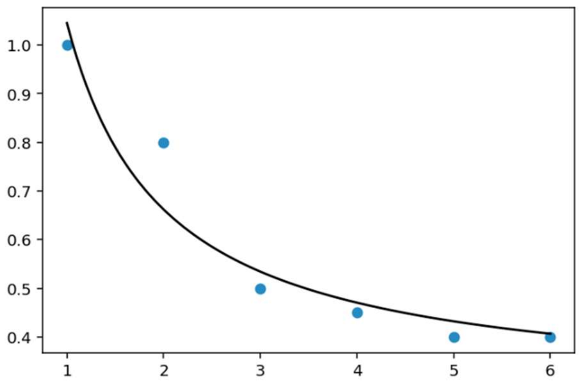


Figure 1. Examples of output from a curve-fitting notebook. *Left*: Screenshot of the prompt to input data. *Right*: Real student data and the figure, with fitted hyperbolic curve, that is initially generated. Students are required to input their own axes labels onto the figure.

Exposure to Coding in Coursework
Many neuroscience instructors do not have the time in their courses nor the training themselves to lead in-depth programming exercises. There are, however, several lightweight ways to integrate coding in such a way that students are at least exposed to the intersection of programming of neuroscience and related fields.

First, instructors can pull back the curtain on how particular data analyses are done, the workflows in their own lab, or their own experience with coding. For example, a common example in neuroscience might be showing tuning curves and illustrating how they are computed. While these tuning curves were done by hand for many decades, now they are generated using custom-written software programs. Even explicitly making this point to students can be a way to illustrate how the "everyday neuroscientist" is using programming tools – a point that may be lost on students. Similarly, when introducing techniques like electroencephalography (EEG) or functional magnetic resonance imaging (fMRI), instructors may highlight that researchers use large, custom toolboxes written in various programming languages to analyze these kinds of data.

Instructors may also choose to take this one step further by inviting students to analyze their own data using programming notebooks. This is most appropriate in a laboratory course but could also be a very short example in other contexts. For example, instructors that are teaching about reaction times or simple psychophysics could spend 20 minutes in class asking students to assess their own auditory and visual reaction times and input these into a class dataset and ultimately a coding notebook for data visualization and hypothesis testing. Relatedly, coding notebooks can be great sandboxes to introduce statistical concepts such as the central limit theorem.

In a laboratory course, instructors can provide pre-written notebooks that students use to input their data, generate visualizations, and run statistics. In a Neurobiology Laboratory that I teach, we collect data from the medial giant fiber of the earthworm, a common model for teaching extracellular electrophysiology and fundamental properties of neurons, especially cable theory (Kladt et al., 2010; Bähring and Bauer, 2014). In this experiment, students are tasked with generating a strength-duration curve, which illustrates that nerves require longer stimulus durations when less current is administered. Students also use these data to estimate a rheobase and chronaxie. The strength-duration curve follows a hyperbolic function, which can be used to derive the rheobase and chronaxie directly (Bähring and Bauer, 2014).

Microsoft Excel, however, does not offer a hyperbolic curve in their curve fitting toolbox. Therefore, we have generated a notebook where students input their data and fit this curve (Figure 1; a complete notebook can be found at https://bipn145.github.io/Python/FitCurve.html). This curve fitting notebook also illustrates the curve with ideal, hypothetical data, helping students build an intuition for curve fitting more broadly. In our implementation, students only modify two parts of the code: a list containing their own data and the axes labels. Even with this small example they can identify the limitations of Excel and the utility of using programming to analyze their data.

If there are enough use cases for such notebooks, instructors can even build a full, online book with different pages for different exercises. An example of such a book, built using JupyterBook (https://jupyterbook.org/), can be found at http://bipn145.github.io. Instructors can use the source code for this book to build their own.

Integration of Coding into Coursework
Alternatively, instructors may feel empowered to integrate programming directly into their coursework, even expecting students to write their own code from scratch. Instructors could spend one or more course or laboratory sessions dedicated to introducing students to fundamental coding concepts, such as creating variables or generating plots. The goal here is not to teach students everything they need to know about programming, but rather to illustrate its use and empower students to learn more through carefully chosen examples.

An example of such a strategy has been illustrated in a number of publications, for example in using open datasets to teach coding alongside neurobiology concepts or to teach bioinformatics (Juavinett, 2020; Ho et al., 2021; Madlung, 2018). In each of these, instructors should define the codingspecific learning objectives for students, who may be concerned about how much coding they are expected to do on their own. One benefit of spending this amount of time on programming, even in a neuroscience course, is that it can allow students to directly interact with real neuroscience data to pose their own research questions and connect real-world data to course material. For example, in Juavinett et al. (2020), students are invited to compare two different cell types of their choosing. This kind of activity could be tailored for different lesson plans – for example, a discussion of comparative neuroscience (looking at human versus mouse cells) or an exploration of different inhibitory cell types.

Discipline-Based Coding Classes

Finally, instructors can develop full courses that are contextualized in neuroscience or biology. Such discipline-based introductory programming classes are particularly effective for students who are ineligible for computer science courses at their institutions, anxious about coding, or late to realize its utility in biological research. Inspired by other successful models of such courses (Dodds et al., 2012), I teach an introduction to Python course for biology students at my institution. Importantly, courses such as these are particularly effective at recruiting women and students of color (Dodds et al., 2021; Zuckerman and Juavinett, 2024).

Relatedly, many institutions are now developing introductory data science classes, which teach programming and statistics with broad applicability (Donoghue et al., 2021, 2022; Çetinkaya-Rundel and Ellison, 2021). These are powerful courses for neuroscience students to develop intuitions around data and foundational programming skills and can be more accessible (both conceptually and logistically) than introductory computer science courses.

There are also endless opportunities to develop and teach more advanced computational classes in neuroscience, for example with a focus on data science or computational modeling. In fact, these classes are arguably more common than introductory discipline-specific classes. Usefully, the availability of online coding environments, as noted above, has led to a proliferation in free online coding textbooks and even full courses in advanced neuroscience topics (van Viegen et al., 2021; see Examples in Table 1). Instructors and departments should think carefully about prerequisites for such courses and ensure that if programming is *not* a prerequisite, ample time is spent bringing all students up to speed. The alternative — where instructors assume the programming skills of their students — is exceptionally damaging to students who feel like they do not have an entry point into these skillsets.

LOOKING FORWARD

As I write this, computational education is changing under our feet. These changes are largely driven by technological advances in our ability to live with and learn from large language models.

Considerations in the Age of Generative AI

Like all fields of education, computational education is also changing in the wake of generative AI tools (e.g., chatGPT, Google Bard) as well as more established AI assistants such as GitHub Copilot, which offers predictive suggestions for code and is free to both educators and students. Innovators in computer science education are now integrating advanced artificial intelligence (AI) assistants such as Copilot to teach students how to work alongside them (see Porter and Zingaro, 2023).

These are particularly interesting developments especially when it comes to computational skill building in the context of neuroscience or biology, where computing is a *tool* rather than a focus of study. In practice, many neuroscience researchers and industry professionals will turn to generative AI tools for support with coding as well as many other tasks (Guo, 2023). Instructors should also consider using generative AI to support their own learning — personally, I have found it incredibly useful. Specifically, I use chatGPT for suggestions when writing my own code, to help generate problem sets, and to test whether a prompt for a programming assessment is interpretable in the way I expect. These days, if I have a programming question, I am more likely to turn to chatGPT than to a generic Google search. Thus, it would not make sense for us to teach our neuroscience students to avoid such tools entirely when they are learning how to code.

For the purposes of exposing our students to coding and encouraging them to give it a try, generative AI can lower barriers to entry — students may be more willing to ask chatGPT than an instructor. Further, generative AI tools can provide very helpful explanations of code. We should therefore encourage our students to include generative AI as an additional learning resource.

That said, we should also advise our students on how to effectively prompt generative AI and assess its output. If students simply enter a dataset and say, “analyze this data,” a generative AI may interpret that in any number of ways. Relatedly, a user could give a very detailed and considered prompt, but the code may still misinterpret the structure of the data. It is essential that we teach students to be critical of AI-written code and resulting processed outputs, for example, by asking questions such as, “What is the new mean, minimum, and maximum of the processed data? Does that make sense?” These concerns become especially important in the context of full-blown discipline-based coding classes, where instructors should be very clear about how students can use such tools. Finally, and importantly, students should be advised that such tools are built on the collective, open information on the internet, which has inherent, deep-seated biases as well as copyright concerns (Reynolds, 2023).

Concluding Thoughts

It is *not* essential that *every* neuroscience student learn how to code. There are many vibrant and productive areas of research that do not rely on programming but instead are powered by longstanding, fundamental approaches in biology research. That said, given the equity considerations and the increasing role of computing in research and in the broader economy, it may be strategic for neuroscience instructors to at least expose their students to these skill sets.

REFERENCES

Akil H, Balice-Gordon R, Cardozo DLL, Koroshetz W, Posey Norris SMM, Sherer T, Sherman SM, Thiels E (2016) Neuroscience Training for the 21st Century. Neuron 90:917–926. doi: 10.1016/j.neuron.2016.05.030

Bähring R, Bauer CK (2014) Easy method to examine single nerve fiber excitability and conduction parameters using intact nonanesthetized earthworms. Advances in Physiology Education 38:253–264. doi: 10.1152/advan.00137.2013

Casimo K (2023) Teaching and Training with Open Science: From Classroom Teaching Tool to Professional Development. Neuroscience 525:6–12. doi: 10.1016/j.neuroscience.2023.07.013

Çetinkaya-Rundel M, Ellison V (2021) A Fresh Look at Introductory Data Science. Journal of Statistics and Data Science Education 29:S16–S26. doi: 10.1080/10691898.2020.1804497

Cheryan S, Master A, Meltzoff AN (2015) Cultural stereotypes as gatekeepers: increasing girls' interest in computer science and engineering by diversifying stereotypes. Frontiers in Psychology 6:1–8. doi: 10.3389/fpsyg.2015.00049

Cheryan S, Plaut VC, Davies PG, Steele CM (2009) Ambient Belonging: How Stereotypical Cues Impact Gender Participation in Computer Science. Journal of Personality and Social Psychology 97:1045–1060. doi: 10.1037/a0016239

Dodds Z, Libeskind-Hadas R, Bush E (2012) Bio1 as CS1: evaluating a crossdisciplinary CS context. In: Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education, pp 268–272. New York, NY: Association for Computing Machinery. doi: 10.1145/2325296.2325360

Dodds Z, Morgan M, Popowski L, Coxe H, Coxe C, Zhou K, Bush E, Libeskind-Hadas R (2021) A Biology-based CS1: Results and Reflections, Ten Years in. In: SIGCSE 2021 - Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, pp 796–801. New York, NY: Association for Computing Machinery. doi: 10.1145/3408877.3432469

Donoghue T, Voytek B, Ellis S (2022) Course Materials for Data Science in Practice. Journal of Open Source Education 5:121. doi: 10.21105/jose.00121

Donoghue T, Voytek B, Ellis SE (2021) Teaching Creative and Practical Data Science at Scale. Journal of Statistics and Data Science Education 29:S27–S39. doi: 10.1080/10691898.2020.1860725

Grisham W, Abrams M, Babiec WE, Fairhall AL, Kass RE, Wallisch P, Olivo R (2021) Teaching Computation in Neuroscience: Notes on the 2019 Society for Neuroscience Professional Development Workshop on Teaching. J Undergrad Neurosci Educ 19:A185–A191.

Grisham W, Brumberg JC, Gilbert T, Lanyon L, Williams R, Olivo R (2017) Teaching with Big Data: Report from the 2016 Society for Neuroscience Teaching Workshop. J Undergrad Neurosci Educ 16:68–76.

Grisham W, Lom B, Lanyon L, L. Ramos R (2016) Proposed Training to Meet Challenges of Large-Scale Data in Neuroscience. Frontiers in Neuroinformatics 10:1–6. doi: 10.3389/fninf.2016.00028

Guo P (2023) Six Opportunities for Scientists and Engineers to Learn Programming Using AI Tools such as ChatGPT. Computing in Science and Engineering 25:73–78. doi: 10.1109/MCSE.2023.3308476

Ho Y-Y, Roeser A, Law G, Johnson BR (2021) Pandemic Teaching: Using the Allen Cell Types Database for Final Semester Projects in an Undergraduate Neurophysiology Lab Course. J Undergrad Neurosci Educ 20:A100–A110.

Juavinett A (2020) Learning How to Code While Analyzing an Open Access Electrophysiology Dataset. J Undergrad Neurosci Educ 19(1):A94-A104.

Juavinett AL (2022) The next generation of neuroscientists needs to learn how to code, and we need new ways to teach them. Neuron 110:576–578. doi: 10.1016/j.neuron.2021.12.001

Kladt N, Hanslik U, Heinzel H-G (2010) Teaching Basic Neurophysiology Using Intact Earthworms. J Undergrad Neurosci Educ 9:A20–A35.

Lewis CM, Anderson RE, Yasuhara K (2016) "I Don't Code All Day": Fitting in computer science when the stereotypes don't fit. In: ICER 2016 - Proceedings of the 2016 ACM Conference on International Computing Education Research, pp 23–32. New York, NY: Association for Computing Machinery. doi: 10.1145/2960310.2960332

Madlung A (2018) Assessing an effective undergraduate module teaching applied bioinformatics to biology students. PLOS Computational Biology 14:e1005872. doi: 10.1371/journal.pcbi.1005872

Margolis J, Fisher A (2003) Unlocking the Clubhouse. Cambridge, MA: The MIT Press.

Muller E, Bednar JA, Diesmann M, Gewaltig M-O, Hines M, Davison AP (2015) Python in neuroscience. Frontiers in Neuroinformatics 9:11. doi: 10.3389/fninf.2015.00011

Paninski L, Cunningham JP (2018) Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience. Current Opinion in Neurobiology 50:232–241. doi: 10.1016/j.conb.2018.04.007

Porter L, Zingaro D (2023) Learn AI-assisted Python Programming. Shelter Island, NY: Manning Publishing.

Reynolds E (2023) Of Chatbots and Colonizers. J Undergrad Neurosci Educ 21(2): E8-E9. doi: 10.59390/YLHJ6332

Schlafly E, Cheung A, Michalka S, Lipton P, Moore-Kochlacs C, Bohland J, Eden U, Kramer M (2020) Python for the practicing neuroscientist: an online educational resource. eLife Available at: https://elifesciences.org/labs/f779833b/python-for-the-practicing-neuroscientist-an-online-educational-resource.

Sejnowski TJ, Churchland PS, Movshon JA (2014) Putting big data to good use in neuroscience. Nature Neuroscience 17:1440–1441. doi: 10.1038/nn.3839

van Viegen T et al. (2021) Neuromatch Academy: Teaching Computational Neuroscience with Global Accessibility. Trends in Cognitive Sciences 25:535–538. doi: 10.1016/j.tics.2021.03.018

Zuckerman AL, Juavinett AL (2024) When Coding Meets Biology: The tension between access and authenticity in a contextualized coding class. In: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA. New York, NY: Association for Computing Machinery. doi: 10.1145/3626252.3630966

Received October 10, 2023; revised February 8, 2024; accepted February 10, 2024.

ACKNOWLEDGEMENTS
The author would like to thank her colleague Dr. Marc Marino for helping to develop some of the coding exercises referenced here (particularly the strength-duration curve notebook), as well as the attendees at the "Integrating Programming into Neuroscience Courses" workshop at the 2023 Faculty for Undergraduate Neuroscience Conference, for which these ideas were originally compiled.

Address correspondence to: Dr. Ashley Juavinett, Neurobiology Department, UC San Diego, La Jolla, CA 92037. Email: ajuavine@ucsd.edu