JOURNAL OF CHEMICAL EDUCATION

pubs.acs.org/jchemeduc Article

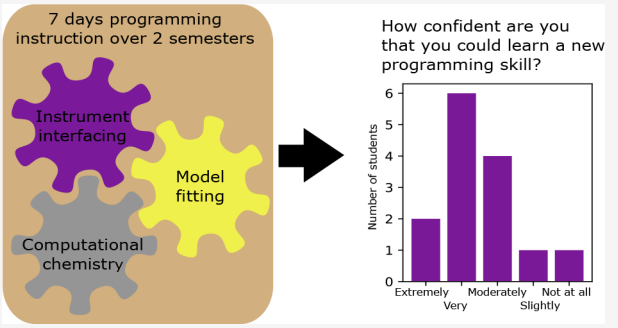
Integrating Python into a Physical Chemistry Lab

Marie van Staveren*

Cite This: J. Chem. Educ. 2022, 99, 2604−2609 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: This paper shows a method for integrating computer programming into a standard physical chemistry laboratory sequence to augment student data analysis abilities and allow them to carry programming skills forward to other courses. The Python programming language is used, taking advantage of the pedagogical benefits of Jupyter notebooks, primarily the ability to intersperse instructions and interactive code cells. A series of five notebooks, plus one traditional script exercise, are designed to teach basic techniques (e.g., loops, assignments, data types), instrument interfacing, model fitting, and introductory quantum chemistry. The skills are directly applicable to laboratories the students perform during the hands-on portion of the courses. A survey of students who have completed the course show high confidence in their ability to learn new skills, and student comments reveal they have used these skills in a variety of other contexts.



KEYWORDS: *Upper-Division Undergraduate, Curriculum, Laboratory Instruction, Physical Chemistry, Computer-Based Learning, Computational Chemistry, Laboratory Interfacing*

INTRODUCTION

The amount of programming instruction and course structure used to train undergraduate chemistry students varies widely between institutions. A stand-alone introductory course,[1,2] followed by incorporation of learned techniques into advanced laboratory and lecture courses might be appealing. However, the already packed program requirements for a bachelor's degree often leave no room for such a course. Additionally, as programming skills can be uneven among faculty, it is often easier to focus on a single course or course sequence. Often this takes the form of instructor-supplied code that students view and use to solve homework problems[3−7] or operate home-built instrumentation.[8−11] While these modules are relatively easy to include in a course, they are limited by how much programming the students (rather than the instructor) actually do.

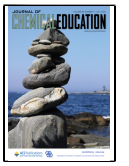
A key question to consider is the suitability of lab versus lecture instruction to achieve the goal of teaching students programming skills. Stand-alone modules can be incorporated into lecture courses,[12,13] which allow students to deeply explore concepts through the code they produce. A particularly interesting model[14] has students directly collaborating via GitHub, which is a current best-practice among professional programmers, though students are generally unfamiliar with this mode of collaboration. However, laboratory courses are an excellent fit for programming instruction for two reasons. First, the programming instruction can be made directly and consistently relevant, by using the programming language to perform experiments as well as aiding in data analysis and presentation.[15] Second, they have more contact hours, making it easier to spend precious class time doing the necessary introductory work for the new language.

This article describes a series of programming modules comprising seven lab periods spread over a two semester physical chemistry lab sequence. Students learn and then apply increasingly complex programming skills throughout the courses. The goal of these modules is to teach students immediately applicable programming skills, which aid them in their lab tasks. As the skills are directly relevant to their chemistry course work, students are able to carry forward their programming knowledge into future courses.

STRUCTURE AND IMPLEMENTATION

Python was chosen as the programming language for several reasons. It is free, open source software, which eliminates access barriers during coursework, as well as in the future where a student might wish to apply what they learned in this course. Python has been among the most popular programming languages for the past decade.[16] In addition to a robust standard library, Python has a robust ecosystem devoted to scientific computing, including NumPy,[17] which adds in a powerful array

Received: March 4, 2022
Revised: May 20, 2022
Published: June 6, 2022



https://doi.org/10.1021/acs.jchemed.2c00193
J. Chem. Educ. 2022, 99, 2604−2609

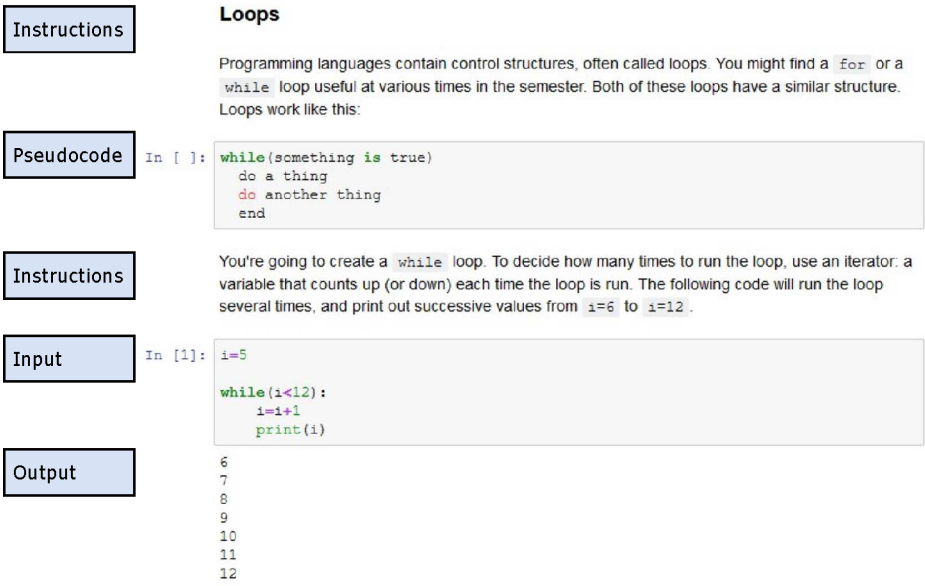


Figure 1. A screenshot taken from the first tutorial. Instructions for students are displayed in markdown cells, with a snippet of pseudocode (plain language showing the structure) placed in an input cell. Finally, an input cell contains a `while` loop, and the loop output displays below.

function to handle numerical data, and Matplotlib[18] for easy visualizations. Python is popular enough to have code already developed for most scientific problems, and several electronic structure packages run in Python or have Python interfaces.[19]

In teaching programming, getting the software onto student computers is often a significant barrier. To that end, the Anaconda Python distribution was chosen for this project.[20] Anaconda is a popular distribution of Python, which includes a large number of standard libraries, and is easy to install on PCs, Macs, and most Linux distributions. Additionally, Anaconda includes both an IDE (Spyder) and Jupyter notebooks.[21] The core libraries used in these modules will be familiar to many Python programmers: Matplotlib[18] for plotting and NumPy[17] for storing numerical data in arrays. In the first term, students use the standard Python libraries `random` (random number generation) and `time` (measuring acquisition time). In order to interface with our National Instruments data acquisition boards, the `nidaqmx`[22] package is used. Second term introduces Scipy,[23] which has a wide variety of scientifically useful functions, from which `curve_fit` is used. Psi4[24] is the library for electronic structure calculations, and `fortecube-view`[25] allows electronic surface and normal mode visualizations. Finally, the data science package Pandas[26] is introduced briefly as a way to output a neat table. With the exception of Psi4 and `fortecubeview`, all of these libraries are included in Anaconda.

Jupyter uses notebook (also called literate) programming, also seen in Mathematica and RStudio. A sample notebook, taken from the first tutorial, is shown in Figure 1. In a Jupyter notebook, a few lines of code are typed into a cell. Each cell can be executed individually, and its output is printed below the cell. A key feature of Jupyter notebooks for instruction is the use of markdown cells, which allow the display of rich text instead of text which executes as code. The instructor can create tutorials that switch between instructions typed as rich text in markdown cells, executable cells for the student to use to follow the instructions, and empty cells for students to add in their own code.

Programming instruction is split into three modules, consisting of seven lab periods over two semesters. The first semester module, taking three lab periods, introduces students to programming in Python and teaches them to interface with an instrument. The second term has a two day module on fitting data to a model, and another two day module solving computational chemistry problems. There is no required computer science requirement for these courses, and prerequisite courses primarily use Microsoft Excel for data analysis.

During lab periods, students work through the tutorial notebooks at their own pace, with the instructor answering student questions and checking their understanding throughout the lab period. First term students work in pairs, while they work individually second term. Peer interaction is encouraged, with the instructor highlighting students who have mastered a section as a resource for peers, encouraging peer learning.[27] Lab time is comparable in instructional intensity to a typical wet lab experiment. This allows these modules to be run in standard lab sections at normal class sizes. The one caveat here is that the instructor needs to be familiar with the programming language. The first author has found that the graduate teaching assistants who usually supervise lab sections tend to have a wide range of programming abilities, and has elected to teach these sections themselves in order to provide the required programming knowledge.

First Semester

The first semester lab course accompanies a standard physical chemistry lecture course that covers thermodynamics and kinetics. Lab enrollment is generally 30−60 students, split approximately evenly between chemistry and chemical engineering majors. The largest number of students in a section at any time is 18 students. The only required prerequisite lab course is general chemistry, though most of the students have taken at least one further chemistry or chemical engineering lab course. One experimental goal is to have students build their own temperature acquisition device and connect it to a computer for digital data acquisition. Programming goals include being familiar with basic language constructs, using Jupyter notebooks,

https://doi.org/10.1021/acs.jchemed.2c00193
J. Chem. Educ. 2022, 99, 2604−2609

and learning how to execute a script in an integrated development environment (IDE).

A breakdown of the concepts covered in each of the three lab periods is shown in Table 1. Before the first day, students are given information about how to download Python and optional tutorials. A sample homework assignment covering this material is given in the Supporting Information. Students spend their lab periods working through the Jupyter tutorials. At first, these tutorials ask students to alter instructor written code snippets. As they progress, students build up their own code snippet by copying it into a new notebook cell and adding to it. This process develops their code snippet into their temperature acquisition script. In the first implementation of these notebooks, students worked in the lab room, either alone or in pairs. During the second implementation, COVID-19 necessitated running the course primarily online. Students worked remotely on their own computers, with the instructor coordinating and offering assistance via online conferencing.

Table 1. Summary of First Semester Skills by Lab Period

First Lab Period	Second Lab Period	Third Lab Period
Hello world	User inputs	User interrupts
Loops	Strings, ints, and floats	Working in the Spyder IDE
Plotting in Matplotlib	Plot settings and annotations	Reading in a voltage
NumPy arrays	Array manipulation	Advanced control structures
Accessing help	Saving data to a text file	
Random numbers	Imports	
Time functions		
Commenting		

The goal of the first semester is for students to write a script that controls a student built digital thermometer. The three notebook tutorials take students through the programming of the script for their instrument. They alternate lab days building a circuit which connects a thermocouple to the computer for signal input. Both circuitry and programming are intimidating skills for many chemistry students. As such, using both while building a real instrument that students will use adds relevance that many students find motivating. Typical scripts are 40−80 lines of code, and almost every group has a functioning device at the end of the unit. The completed scripts and devices are used to measure temperature in a bomb calorimetry experiment and the measurement of the Joule−Thomson coefficient.

As this sequence of tutorials is designed to help science students create useful code as quickly as possible, minimal computer science theory is included. Instead, tutorials place heavy emphasis on accessing the documentation and getting regular feedback, whether from peers or the instructor. The skills acquired match well with introductory skill lists and tutorials from computational chemistry groups like PSI4Education[28] and the Molecular Sciences Software Institute.[19] The topics chosen—acquiring a voltage, plotting it real time, and saving the data—are generalizable to a variety of upper division chemistry laboratories. This deployment is in physical chemistry, as the department emphasizes that an input voltage is the heart of digitally acquired data in this lab course, but it would also be appropriate for an instrumental analysis course to illustrate the process of data acquisition that happens in the instruments we use.

Second Semester

The second semester laboratory course also meets twice a week for 4 h. Second semester enrollment is 12−24 students, all chemistry majors, enrolled in a single section. The accompanying lecture course covers quantum mechanics and spectroscopy. In addition to the first semester lab course, analytical chemistry lab is a prerequisite for this course. Approximately half of the laboratory term is spent on kinetics experiments, while the other half is spectroscopy experiments.

The notebooks in the second semester are aimed at increasing student independence. There are frequent references to appropriate documentation and help functions. Sample code is increasingly sophisticated. Theory is included where relevant, whether computer science, math, kinetics, or quantum mechanics. Finally students are expected to create their own notebooks and generate appropriate markdown cells to describe the code in rich text. Markdown cells allow a Jupyter notebook to function much like a traditional lab notebook. The scientist can include data as well as the meaning of that data and steps taken in analyzing that data all in one place. As this is a learning goal, the use of well formatted markdown cells is required for assignments this term.

Model Fitting Lab. During the kinetics unit, students see a variety of data sets showing decay that is either first or second order, and are asked to determine the reaction order with respect to the varying substance. In order to give students the computational tools to make this evaluation, the first experiment of the semester is a 1 week (two lab period) experiment on fitting data. The notebook contains three sections: import and cleanup of sample data, fitting sample data to several models, and instructions for students to fit their own data set.

Students begin by importing sample data using the NumPy `loadtxt` function. They see how to handle metadata without deleting it using the `skiprows` argument. Next, they see how to use array slices to trim off the initial rapid increase in the data values, which is not part of the region to be fit.

Next, students learn to write functions by defining the model they'll use to fit their data. To actually perform the fit, we use `curve_fit` from the Scipy library. This is a nonlinear least-squares function which allows several fitting methods and is flexible and reasonably robust. A major advantage of `curve_fit` is that it returns the covariance matrix, making extraction of uncertainties in the fit parameters straightforward, which allows comparison with how similar analysis is done in the analytical chemistry course.

The evaluation of goodness of fit introduces new plotting skills. For each model, students create a residual plot stacked on top of their fitted experimental data. Additionally, they make a bar chart of parameter errors. Multiple possible model functions are introduced and their suitability discussed.

The lab report has students analyze a new data set and propose an appropriate model. We use the production of foam from the mixing of Diet Coke and Mentos, which piques student interest, and does not have an obvious answer for the reaction order. Students view a live demonstration of the foam geyser using an apparatus that allows the measurement of foam volume as a function of time.[29,30] Students extract data of foam volume versus time from a video of the demonstration, and fit their data. The data set shows an initial rapid increase, followed by a slow decay; only the decay portion is fit. The student lab report has students submit two fits: one that they accept as the best model for their data, and one that they've rejected. Using markdown cells, students discuss the goodness of fit for each model, and

https://doi.org/10.1021/acs.jchemed.2c00193
J. Chem. Educ. 2022, 99, 2604−2609

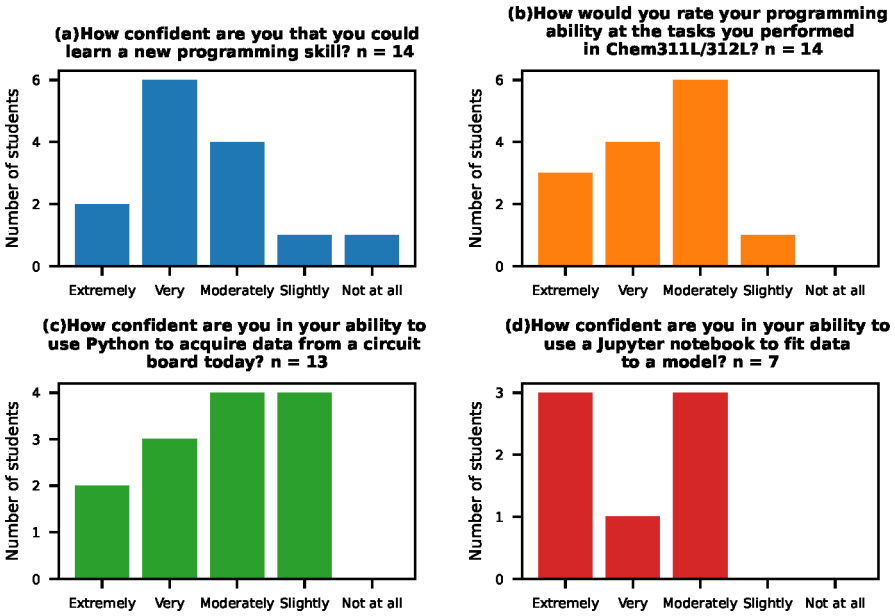


Figure 2. Likert-scale results to survey questions. Sample size varies because not every student had taken each course in the relevant semesters.

justify, based on available metrics, their residual plot and the uncertainties in the fit parameters.

Computational Chemistry Lab. The majority of students take this lab course alongside an inorganic lecture course, which covers molecular symmetry in great detail, as well as a physical chemistry lecture course on quantum mechanics. As such, there is a lot of synergy between the three courses in the area of applying quantum mechanics to spectroscopy of small molecules. To complement that, a computational chemistry experiment has been added to the physical chemistry lab.

In the computational chemistry notebook, students use Psi4, an open-source research-grade suite of computational chemistry software. While Psi4 is capable of using a traditional command-prompt interface for computations, it can also be run inside a Jupyter notebook. It is, however, difficult to get a stable compile of Psi4 on a variety of computer set-ups. In order to make this software stack more user-friendly, we used ChemCompute.[31] ChemCompute is a web service that allows students and educators to use a variety of computational software (including GAMESS, TINKER, and NAMD) free of charge. All that a user needs is an email address to register.

The learning objectives for this module are as follows:

- Pick an appropriate basis set for a given problem and justify the choice.
- Perform an energy minimization in Psi4, importing the molecule from PubChem.
- Perform a geometry optimization.
- Generate predicted frequencies of normal modes and see the irreducible representation for each mode.
- Use "fortecubeview" to visualize electronic surfaces and normal modes, allowing observation of symmetry.
- Collect generated data into plots that assist in justifying conclusions.

Students first read through an example tutorial in which a variety of computations are performed. They are then instructed to perform a similar series of calculations on a new set of molecules. The tutorial uses methane for basis set determination and energy minimization, and carbon dioxide for normal-mode analysis. The assignment uses chloromethane for basis set determination, and water for normal-mode analysis. Students submit a Jupyter notebook they create, which includes markdown cells as needed to explain their work.

■ ASSESSMENT OF EFFECTIVENESS

In order to evaluate the effectiveness of this programming instruction, students in the capstone instrumental analysis course were surveyed as to their programming abilities (UMBC IRB #513, exempt). Students take this course in the spring semester, either concurrently with the second term of physical chemistry lab, or a year after the physical chemistry lab sequence. Fifteen students gave informed consent to participate in the study; of those, 14 had taken one or both courses during the relevant semesters, 14 had taken the first term course, and 8 had taken the second term course. Not every student answered every question. The survey consisted of paired questions: one Likert-scale item and an accompanying free-response item to elaborate on their answer, as well as a final open-ended question. Figure 2 shows answers to the Likert-scale items.

Overall, students show a high degree of confidence with both their ability to perform skills taught in the course (Figure 2b) and at their ability to learn new skills (Figure 2a). This was reflected in their answers to the free-response questions. For example, one student wrote "I could definitely learn [a new skill] given the time. I think I need a lot of time, but I definitely have a foundation now." Similarly, another student wrote "At the start of [fall semester] Jupyter notebooks were completely new to me, but by the end of [spring semester] I felt extremely comfortable using it for my analysis." The collaborative nature of the method of instruction was also mentioned as increasing confidence: "It didn't feel like I was alone in all of it, which was reassuring, especially since I know that in the real-world, you're with a team of people working together to get things done."

Confidence with particular skills was mixed, and depended on the particular skill. Both programming and circuitry are new skills for most chemistry majors. Figure 2(c) shows confidence with ability to program a circuit interface. One student responded "I don't know if I can work well with circuits. I could probably do it, but it would take quite a bit of time to

https://doi.org/10.1021/acs.jchemed.2c00193
J. Chem. Educ. 2022, 99, 2604−2609

refamiliarize myself with it." In contrast, student confidence with using Python to fit data to a model was significantly higher, as shown in Figure 2(d). Because of the timing of the survey, every student who answered the question about model fitting was answering it a full year after they had taken that tutorial, and all were moderately to extremely confident in their ability to perform that skill. A typical quote describing this: "This was probably the skill I performed the most. I did it so much that I simplified it into a function that I could use again and again with modifications to help me perform it quicker."

Another goal of the Python instruction was to build skills that students could transfer to other situations. Several students mentioned skill transfer in their free-response answers. One student wrote "I used all of the instruction I got in those classes for [inorganic lab] to process all the data I got and make figures for the lab report." Another student, speaking of circuit interfaces, said "With the help from the Jupyter notebooks, I've been able to write a similar program at home." Another student compared the chemistry-specific programming instruction to what they learned in a computer science course: "At the end of the year, I knew so much more about skills necessary for chemistry analysis that a computer science class would not have primarily focused on."

In the instructor's interactions with students, several themes emerged. During the first term, chemistry majors realized quickly that most of their engineering classmates have some prior programming experience. Ideally, this would trigger peer learning, especially as chemistry majors know that they will continue on with Python into second term. Unfortunately, more often, chemistry students allow (or push) their engineering classmates to do much of the programming in the first semester. Consequently, many students struggle with the model fitting lab at the start of the second term, and need to spend time reviewing the first semester materials or online tutorials.

By the end of the two semester sequence, all of the students have basic fluency. Students can choose software for analyzing data for the wet chemistry experiments that make up the bulk of the two courses. Most students prefer Excel when the courses begin, but have switched to Python by the end of the sequence for peak finding and model fitting, as well as assorted general plotting tasks. Approximately a quarter of the class achieves enough proficiency to switch to Python for the majority of their analysis tasks. For students who take the sequence as juniors, many continue to develop their coding skills during their senior year. Perhaps a quarter of last year's class has asked the instructor for help on a coding task, or mentioned using it for a lab report, during this academic year. Several students have achieved advanced levels, through independent study or use in their research groups.

CONCLUSION

Computer programming in Python using Jupyter notebooks has been successfully implemented into a two semester upper division course. This method allows students to apply the programming they are learning to chemistry laboratory tasks. The focus on immediately relevant skills is designed to increase student confidence at attempting programming in the future. Survey results show students have transferred these skills to new contexts.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available at https://pubs.acs.org/doi/10.1021/acs.jchemed.2c00193.

Introductory assignment: Compilation of introductory information and tutorials for students new to Python (PDF) (DOCX)

Student survey questions: Full text of questions in the student survey (PDF) (DOCX)

AUTHOR INFORMATION

Corresponding Author

Marie van Staveren − *Department of Chemistry and Biochemistry, University of Maryland, Baltimore, Maryland 21250, United States;* orcid.org/0000-0002-4081-838X; Email: mvanstav@umbc.edu

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jchemed.2c00193

Notes

The author declares no competing financial interest.

Student versions of the Jupyter notebooks described, as well as additional data files as needed, can be found at https://github.com/chemdrv/python-for-pchem. Completed notebooks for instructor use can be requested by emailing the author.

ACKNOWLEDGMENTS

ChemCompute used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation Grant Number ACI-1548562. The author would like to thank Joseph Bennett, Clarissa Sorensen-Unruh, and Sarah Bass for their editorial support, and the students whose hard work learning to code made this project a success.

REFERENCES

(1) Weiss, C. J. A Creative Commons Textbook for Teaching Scientific Computing to Chemistry Students with Python and Jupyter Notebooks. *J. Chem. Educ.* **2021**, *98*, 489−494.

(2) Weiss, C. J. Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *J. Chem. Educ.* **2017**, *94*, 592−597.

(3) Green, M.; Chen, X. Data Functionalization for Gas Chromatography in Python. *J. Chem. Educ.* **2020**, *97*, 1172−1175.

(4) Möglich, A. An Open-Source, Cross-Platform Resource for Nonlinear Least-Squares Curve Fitting. *J. Chem. Educ.* **2018**, *95*, 2273−2278.

(5) Srnec, M. N.; Upadhyay, S.; Madura, J. D. Teaching Reciprocal Space to Undergraduates via Theory and Code Components of an IPython Notebook. *J. Chem. Educ.* **2016**, *93*, 2106−2109.

(6) Srnec, M. N.; Upadhyay, S.; Madura, J. D. A Python Program for Solving Schrödinger's Equation in Undergraduate Physical Chemistry. *J. Chem. Educ.* **2017**, *94*, 813−815.

(7) Weiss, C. J. Introduction to Stochastic Simulations for Chemical and Physical Processes: Principles and Applications. *J. Chem. Educ.* **2017**, *94*, 1904−1910.

(8) Jin, H.; Qin, Y.; Pan, S.; Alam, A. U.; Dong, S.; Ghosh, R.; Deen, M. J. Open-Source Low-Cost Wireless Potentiometric Instrument for pH Determination Experiments. *J. Chem. Educ.* **2018**, *95*, 326−330.

(9) Navarre, E. C. Extensible Interface for a Compact Spectrophotometer for Teaching Molecular Absorption in the Undergraduate Laboratory. *J. Chem. Educ.* **2020**, *97*, 1500−1503.

(10) Bougot-Robin, K.; Paget, J.; Atkins, S. C.; Edel, J. B. Optimization and Design of an Absorbance Spectrometer Controlled Using a Raspberry Pi To Improve Analytical Skills. *J. Chem. Educ.* **2016**, *93*, 1232−1240.
(11) Tan, S. W. B.; Naraharisetti, P. K.; Chin, S. K.; Lee, L. Y. Simple Visual-Aided Automated Titration Using the Python Programming Language. *J. Chem. Educ.* **2020**, *97*, 850−854.
(12) Fisher, A. A. E. Developing the Chemist's Inner Coder: A MATLAB Tutorial on the Stochastic Simulation of a Pseudo-First-Order Reaction. *J. Chem. Educ.* **2020**, *97*, 1476−1480.
(13) Fisher, A. A. An Introduction to Coding with Matlab: Simulation of X-ray Photoelectron Spectroscopy by Employing Slater's Rules. *J. Chem. Educ.* **2019**, *96*, 1502−1505.
(14) Vargas, S.; Zamirpour, S.; Menon, S.; Rothman, A.; Häse, F.; Tamayo-Mendoza, T.; Romero, J.; Sim, S.; Menke, T.; Aspuru-Guzik, A. Team-Based Learning for Scientific Computing and Automated Experimentation: Visualization of Colored Reactions. *J. Chem. Educ.* **2020**, *97*, 689−694.
(15) Menke, E. J. Series of Jupyter Notebooks Using Python for an Analytical Chemistry Course. *J. Chem. Educ.* **2020**, *97*, 3899−3903.
(16) Tiobe Index. https://www.tiobe.com/tiobe-index/ (accessed on March 3, 2022).
(17) Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357−362.
(18) Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **2007**, *9*, 90−95.
(19) MolSSI—The Molecular Sciences Software Institute. https://molssi.org/ (accessed on October 14, 2021).
(20) Anaconda: The World's Most Popular Data Science Platform. https://www.anaconda.com/ (accessed on February 24, 2021).
(21) Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; Ivanov, P.; Avila, D.; Abdalla, S.; Willing, C.; Jupyter development team. *Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows*; Positioning and Power in Academic Publishing: Players, Agents and Agendas: Netherlands, 2016; pp 87−90.
(22) NI-DAQmx Python Documentation. https://nidaqmx-python.readthedocs.io/en/latest/ (accessed on April 22, 2022).
(23) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261−272.
(24) Parrish, R. M.; Burns, L. A.; Smith, D. G. A.; Simmonett, A. C.; DePrince, A. E.; Hohenstein, E. G.; Bozkaya, U.; Sokolov, A. Y.; Di Remigio, R.; Richard, R. M.; et al. Psi41.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *J. Chem. Theory Comput.* **2017**, *13*, 3185−3197.
(25) fortecubeview. https://github.com/evangelistalab/fortecubeview (accessed on October 19, 2021).
(26) pandas—Python Data Analysis Library. https://pandas.pydata.org/ (accessed on April 22, 2022).
(27) Stanger-Hall, K. F.; Lang, S.; Maas, M. Facilitating Learning in Large Lecture Classes: Testing the "Teaching Team" Approach to Peer Learning. *CBE Life Sci. Educ.* **2010**, *9*, 489−503.
(28) Psi4Education: Computational Labs Using Free Software | Posts. https://psicode.org/posts/psi4education/ (accessed on October 14, 2021).
(29) Kuntzleman, T. S.; Annis, J.; Anderson, H.; Kenney, J. B.; Doctor, N. Kinetic Modeling of and Effect of Candy Additives on the Candy−Cola Soda Geyser: Experiments for Elementary School Science through Physical Chemistry. *J. Chem. Educ.* **2020**, *97*, 283−288.
(30) Kuntzleman, T. S.; Johnson, R. Probing the Mechanism of Bubble Nucleation in and the Effect of Atmospheric Pressure on the Candy−Cola Soda Geyser. *J. Chem. Educ.* **2020**, *97*, 980−985.
(31) Perri, M. J.; Weber, S. H. Web-Based Job Submission Interface for the GAMESS Computational Chemistry Program. *J. Chem. Educ.* **2014**, *91*, 2206−2208.

https://doi.org/10.1021/acs.jchemed.2c00193
J. Chem. Educ. 2022, 99, 2604−2609