

Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)  
<https://creativecommons.org/licenses/by-nc-sa/3.0/>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

**Please provide feedback**

Please support the ScholarWorks@UMBC repository by emailing [scholarworks-group@umbc.edu](mailto:scholarworks-group@umbc.edu) and telling us what having access to this work means to you and why it's important to you. Thank you.

# Integrating Python into Physical Chemistry Lab

Marie van Staveren\*

*Department of Chemistry and Biochemistry*

*University of Maryland Baltimore County, Baltimore, MD 21250;*

E-mail: [mvanstav@umbc.edu](mailto:mvanstav@umbc.edu)

## **Abstract**

This paper shows a method for integrating computer programming into a standard physical chemistry laboratory sequence to augment student 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 labs 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,<sup>1,2</sup> followed by incorporation of learned techniques into advanced laboratory and lecture courses might be appealing. However, already packed program requirements for a bachelors 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<sup>3,4,5,6,7</sup> or operate home-built instrumentation.<sup>8,9,10,11</sup> While these modules are relatively easy to include in a course, they are limited by how much programming the students (rather than the instructor) actual 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<sup>12,13</sup> which allow students to deeply explore concepts through the code they produce. A particularly interesting model<sup>14</sup> has students directly collaborating via Github, which is a current best-practice among professional programmers, though generally unfamiliar to students. 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.<sup>15</sup> Secondly, 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's 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 language for the last decade.<sup>16</sup> In addition to a robust standard library, Python has a robust ecosystem devoted to scientific computing, including Numpy<sup>17</sup> which adds in a powerful array function to handle numerical data, and Matplotlib<sup>18</sup> 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.<sup>19</sup>

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.<sup>20</sup> 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.<sup>21</sup>

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 is typed into a cell. Each cell can be executed individually, and it's 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 was split into three modules, consisting of seven lab periods

over two semesters. The first semester module, taking three lab periods, introduced students to programming in Python and taught them to interface with an instrument. The second term had a two day module on fitting data to a module, 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.

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

**Instructions**

**Loops**

Programming languages contain control structures, often called loops. You might find a `for` or a `while` loop useful at various times in the semester. Both of these loops have a similar structure. Loops work like this:

**Pseudocode**

```
In [ ]: while(something is true)
        do a thing
        do another thing
        end
```

**Instructions**

You're going to create a `while` loop. To decide how many times to run the loop, use an iterator: a variable that counts up (or down) each time the loop is run. The following code will run the loop several times, and print out successive values from `i=6` to `i=12`.

**Input**

```
In [1]: i=5

        while(i<12):
            i=i+1
            print(i)
```

**Output**

```
6
7
8
9
10
11
12
```

Figure 1: A screenshot taken from the first tutorial. Instructions for the student, including a snippet of pseudocode (plain language showing the structure), are displayed in markdown cells. Next is an input cell containing a `while` loop, and the loop output displays below.

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 <sup>18</sup>	Making prettier plots	Connecting the DAQ board
Numpy arrays <sup>17</sup>	Array manipulation	Advanced control structures
Accessing help	Saving data to a text file	
Random numbers	Imports	
Time functions		
Commenting		

A breakdown of the concepts covered in each of the three lab periods is shown in table 1. Students spend the lab period working through the Jupyter tutorial. At first, these tutorials ask students to alter instructor written code snippets. As they progress, students build up their own code snippet which is copied forward and added to. This code develops 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.

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.

As this sequence of tutorials is designed to help scientist 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 tuto-

rials from computational chemistry groups like PSI4Education<sup>22</sup> and the Molecular Sciences Software Institute.<sup>19</sup>

## Second Semester

The second semester laboratory course also meets twice a week for four hours. Second semester enrollment is 15 - 25 students, all chemistry majors. The accompanying lecture course covers quantum mechanics and spectroscopy. 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. These are required for assignments, but also show students how to use notebooks for their own reference.

### 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 one 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, which isn't 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<sup>23</sup> library. This is a non-linear 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 dataset and propose an appropriate model. We use the production of foam from the mixing of Diet Coke and Mentos, which piques student interest, and doesn't 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.<sup>24 25</sup> Students extract data of foam volume versus time from a video of the demonstration, and fit their data. The dataset 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 justify, based on available metrics, such as 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,<sup>26</sup> 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.<sup>27</sup> 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.

Table 2: Computational chemistry learning objectives

Computational Chemistry Learning Objectives
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’ <sup>28</sup> to visualize electronic surfaces and normal modes, allowing observation of symmetry
Collect generated data into plots that assist in justifying conclusions
Use the vocabulary of computational chemistry appropriately

Learning objectives are shown in table 2. 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.

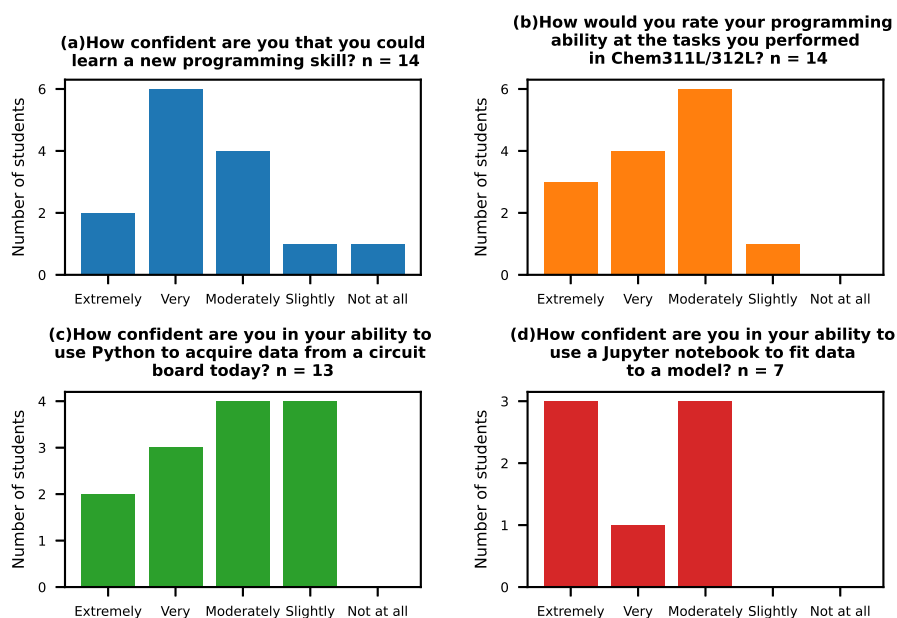


Figure 2: Likert-scale results to survey questions. Sample size varies because not every student had taken each course in the relevant semesters.

Overall, students show a high degree of confidence with both their ability to perform skills taught in the course 2(b) and at their ability to learn new skills 2(a). 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 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 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.”

## 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.

## Acknowledgement

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.

## Supporting Information Available

- Student survey questions.docx: Full text of questions in the student survey

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.

## References

- (1) Weiss, C. J. A Creative Commons Textbook for Teaching Scientific Computing to Chemistry Students with Python and Jupyter Notebooks. *Journal of Chemical Education* **2021**, *98*, 489–494.

- (2) Weiss, C. J. Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *Journal of Chemical Education* **2017**, *94*, 592–597.
- (3) Green, M.; Chen, X. Data Functionalization for Gas Chromatography in Python. *Journal of Chemical Education* **2020**, *97*, 1172–1175.
- (4) Möglich, A. An Open-Source, Cross-Platform Resource for Nonlinear Least-Squares Curve Fitting. *Journal of Chemical Education* **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. *Journal of Chemical Education* **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. *Journal of Chemical Education* **2017**, *94*, 813–815.
- (7) Weiss, C. J. Introduction to Stochastic Simulations for Chemical and Physical Processes: Principles and Applications. *Journal of Chemical Education* **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. *Journal of Chemical Education* **2018**, *95*, 326–330.
- (9) Navarre, E. C. Extensible Interface for a Compact Spectrophotometer for Teaching Molecular Absorption in the Undergraduate Laboratory. *Journal of Chemical Education* **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. *Journal of Chemical Education* **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. *Journal of Chemical Education* **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. *Journal of Chemical Education* **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. *Journal of Chemical Education* **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. *Journal of Chemical Education* **2020**, *97*, 689–694.
- (15) Menke, E. J. Series of Jupyter Notebooks Using Python for an Analytical Chemistry Course. *Journal of Chemical Education* **2020**, *97*, 3899–3903.
- (16) Tiobe Index. <https://www.tiobe.com/tiobe-index/>, Accessed: 2022-03-03.
- (17) Harris, C. R. et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362.
- (18) Hunter, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* **2007**, *9*, 90–95.
- (19) MolSSI – The Molecular Sciences Software Institute. <https://molssi.org/>, Accessed: 2021-10-14.

- (20) Anaconda: The World’s Most Popular Data Science Platform. <https://www.anaconda.com/>, Accessed: 2021-2-24.
- (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.; development team, J. 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) Psi4Education: Computational Labs Using Free Software | Posts. <https://psicode.org/posts/psi4education/>, Accessed: 2021-10-14.
- (23) Virtanen, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **2020**, *17*, 261–272.
- (24) 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. *Journal of Chemical Education* **2020**, *97*, 283–288.
- (25) Kuntzleman, T. S.; Johnson, R. Probing the Mechanism of Bubble Nucleation in and the Effect of Atmospheric Pressure on the Candy–Cola Soda Geyser. *Journal of Chemical Education* **2020**, *97*, 980–985.
- (26) Parrish, R. M. et al. Psi41.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *Journal of Chemical Theory and Computation* **2017**, *13*, 3185–3197.
- (27) Perri, M. J.; Weber, S. H. Web-Based Job Submission Interface for the GAMESS Computational Chemistry Program. *J. Chem. Educ.* **2014**, *3*.

(28) fortecteview. <https://github.com/evangelistalab/fortecteview>, Accssed:  
2021-10-19.



## Graphical TOC Entry

