JOURNAL OF CHEMICAL EDUCATION

Open Access

Modular Integration of Python Programming in Undergraduate Physical Chemistry Experiments

Derri J. Hughes* and Samuel C. Perry*

Cite This: J. Chem. Educ. 2025, 102, 4005−4016 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Programming is a key transferable skill within the chemical sciences with applications supporting data acquisition, as a tool for chemical and spectroscopic analysis and as an environment for theoretical modeling. Of the many available programming languages, Python stands out due to its broad functionality and open-source structure. However, introducing any programming training to an undergraduate chemistry curriculum can be challenging due to students' lack of previous experience and limited time in pre-existing curricula for dedicated training. Here, we present a modular approach to introducing undergraduate students to Python programming through a series of taught undergraduate physical chemistry laboratory experiments. Students are first provided with a carefully scaffolded approach to basic Python syntax before enhancing the student skill set through context-based learning integrated with practical chemistry challenges. In this way, we demonstrate how a modularly integrated approach can provide a complete introduction to Python programming regardless of previous experience and without needing dedicated training time.

KEYWORDS: *Python programming, curriculum design, scaffolded learning, active learning, undergraduate experiments, physical chemistry*

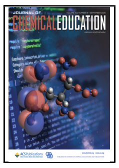
1. INTRODUCTION

Programming has become increasingly essential in the chemical sciences, enabling data analysis,[1] automation of laboratory processes,[2−4] chemical reaction modeling,[5−7] and materials design.[8−10] As this trend accelerates, there is a growing demand for programming proficiency in both industrial and academic research.[11,12] Among all programming languages, Python has gained significant traction due to its ease of use, extensive libraries, and ability to handle large data sets for data analysis and visualization.[13] In response to this shift, numerous publications have explored the integration of programming into chemistry degree programmes. Efforts have focused so far on general curriculum design,[14−16] the development of specialized Python libraries and codes in areas such as quantum chemistry,[5,17−19] spectroscopy,[20−24] cheminformatics,[25,26] and electrochemistry,[27−29] and the creation of scientific programming resources for students.[30,31] While several resources and studies discuss the integration of programming into chemistry education, there remains a need for detailed and proven, curriculum-focused examples that demonstrate how programming can be embedded into existing undergraduate curricula.

While many Python libraries offer prebuilt functions for visualizing concepts or calculating chemical properties, few encourage students to develop their own code from scratch. This hands-on experience is invaluable for processing raw data,[32,33] thinking computationally and algorithmically,[32,34,35] and bridging the gap between laboratory work, classroom learning, and real-world applications.[14,32,34,35] Undergraduate chemistry laboratory experiments play a crucial role in providing context for often abstract concepts taught in core modules and thus stand as an ideal point to integrate a programming aspect into the curriculum.

This paper outlines a curriculum-focused modular approach to incorporating Python programming into undergraduate physical chemistry experiments. We focus on first- and second-year students enrolled in either the three-year Bachelor of Science (BSc) course or the integrated four-year Master of Chemistry (MChem) course. Both degree programs share an identical curriculum during the first two years, comprising core and optional modules. The programming activities described are embedded within the core practical chemistry laboratory modules, which are compulsory for all students. Over the two years, students complete four core laboratory modules, and Python-based programming exercises are integrated into these as part of the practical tasks. In total, eight practicals across these modules include a programming component, with additional opportunities for students to apply their coding skills in nonprogramming practicals. Here, we present six of these programming-based practicals.

Received: May 28, 2025
Revised: August 9, 2025
Accepted: August 11, 2025
Published: August 20, 2025



For all lab practicals involving Python, students use lab-owned laptops with Python preinstalled via the Anaconda distribution. The Introduction to Python lab is completed in Spyder, with all required libraries (Matplotlib, NumPy, and SciPy) readily available. Recognizing that programming preferences vary, we also offer alternative environments such as Jupyter Notebooks, allowing students to explore and choose the interface that best suits their workflow and productivity in the later lab practicals.

All practicals discussed here that include a wet chemistry aspect should be performed wearing standard personal protective equipment (lab coat, nitrile gloves, and splash goggles) and in a well-ventilated space or fume cupboard to mitigate any hazards. Chemical-specific hazards associated with an experiment, ways to mitigate the risk to health, and disposal methods are discussed in Section 1 of the Supporting Information. However, there are no unexpected hazards associated with any of the wet chemistry practicals reported here.

2. INTRODUCTION TO PYTHON

Python is integrated throughout our curriculum for first- and second-year undergraduates, with an emphasis on coupling programming with chemistry to create a holistic learning experience. Embedding programming challenges within relevant laboratory contexts has been shown to enhance student engagement and underscore the practical relevance of coding skills.[36]

In our revised first-year course, we assume no prior experience with Python or significant hands-on laboratory work, acknowledging the variability in students' educational backgrounds.[37] Introducing both new chemistry skills and a programming language simultaneously presents a cognitive burden that can hinder learning, particularly for neurodiverse students.[38−40] Therefore, we implemented a scaffolded approach to Python instruction that supports incremental learning and promotes accessibility. This approach is embedded within a spiral learning model, wherein each subsequent practical introduces new Python functionality while reinforcing previously acquired skills.[41,42] The iterative structure promotes knowledge retention and incremental skill development, aligning with a constructivist framework, in which students engage with project-based tasks. These tasks encourage active exploration of code, experimentation with different solutions, and reflection on errors as part of the learning process.[16,43] We outline the skills introduced in each of the laboratories presented here and those reinforced in Table S1 of the Supporting Information.

The introductory Python module focuses on foundational concepts such as syntax, data types (e.g., strings, floats, variables, and lists), and user inputs. Instruction is delivered through a blended learning model, incorporating prelab tasks supported by written materials and accompanying "code-along" video tutorials.[44,45] These resources feature worked examples and high levels of instructional guidance.[46] This format fosters active learning by allowing students to engage with the material at their own pace, building both confidence and familiarity.[47]

Debugging is introduced early to demystify Python error messages, which are often verbose and intimidating.[48] Students are given functional code and are then asked to intentionally introduce an error to observe the resulting message. This promotes early and structured exposure to common debugging strategies that are linked to the concepts being taught.

Successful completion of prelab activities equips students to undertake the first contextualized coding challenge in the lab: building a temperature conversion calculator between Celsius and Kelvin. Previously introduced skills (inputs, floats, arithmetic operations) are extended to include conditional logic (`if`, `elif`, or `else`) to allow user selection of conversion direction. Pseudocode for this exercise is shown in Algorithm 1 (Supporting Information).[16]

This scaffolded foundation enables the introduction of more advanced Python techniques within a contextualised framework. For this purpose, we use the Belousov−Zhabotinsky (BZ) reaction—a well-established oscillating reaction between bromate and bromide species, and a metal catalyst—as a thematic case study.[49,50] Students complete the BZ experiment the week prior, recording oscillation rates (via red-blue color changes) as a function of temperature. This data are used to calculate activation energy using the Arrhenius equation:

$$\ln(k) = \ln(A) - \frac{E_a}{RT} \tag{1}$$

where k is the rate constant (inverse of the oscillation period), A is the pre-exponential factor, E_a is the activation energy, R is the ideal gas constant, and T is the absolute temperature. Students are already familiar with this analysis from previous coursework using Excel, ensuring the data and methodology are accessible.

Python is then used to revisit the same analysis. Students begin by compiling a list of reagents and progress to import their own experimental data. They use `numpy.polyfit()` to compute the linear fit and extract the slope corresponding to $-E_a/R$, from which E_a is calculated. Algorithm 2 (Supporting Information) outlines this process.

The final task integrates all skills developed during the session: list handling to organize data, plotting to visualize the Arrhenius relationship, and use of print functions to report E_a (Figure 1).

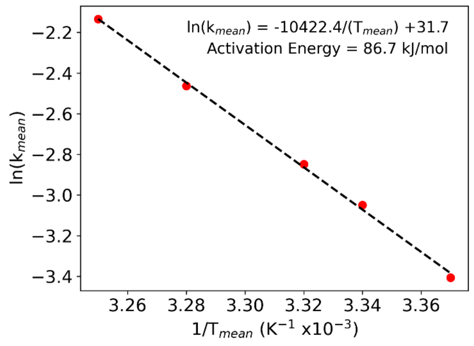


Figure 1. Student-generated figure from the kinetic analysis of the Belousov−Zhabotinsky reaction, reporting the rate constant (k) as a function of temperature (T). Analysis using the Arrhenius eq (eq 1) allows calculation of the activation energy (E_a).

This capstone activity enables students to apply new programming skills to a familiar chemical system, reinforcing learning through contextualiztion while minimizing cognitive overload.

After completing the introductory session, students gain confidence in essential Python programming skills, including basic syntax, data handling, mathematical operations, and the use of core libraries, such as NumPy and Matplotlib. This foundational knowledge equips them to progress through the remainder of the Python-integrated curriculum.

Formal assessment of the programming elements in each practical is based on students' code outputs and results, which provides a direct measure of the code functionality, rather than the specific structure of their code. This approach accommodates variation in coding styles, which we have informally observed to evolve as students engage with more complex tasks. However, we currently do not have formal data evaluating its impact on the learning objectives of each practical. Our indication of increasing Python literacy and competence comes from informal observations—specifically, students' ability to complete more advanced programming tasks, producing the expected code output within the allocated time frame, and relying less on supplementary Python instruction as the lab practicals progress from highly scaffolded activities (Sections 2–4.2) to independent, project-focused challenges (Sections 4.3–5.3).

3. CREATING KINETIC MODELS AND SIMULATIONS

The next set of practical experiments incorporating Python-based programming focuses on the derivation and simulation of chemical kinetic models. So far, we run three different laboratories: two in the first semester of the first year, and one in the first semester of the second year. In our example, we introduce students to the physics and chemistry of nuclear processes—a topic not formally covered by any core or optional Chemistry modules at Southampton. Our aim was to adopt an active learning, "learn as you go" approach, allowing students to explore a new area of chemistry while solving problems within it[51] and continuing to develop their Python programming skills introduced in Section 2.

3.1. Kinetics of Nuclear Decay

The first programming-based objective of the lab is for students to generate model data for the first-order β^+ decay of the ^{15}O isotope, extract its half-life ($\tau_{0.5}$), and compare this to the extracted $\tau_{0.5}$ of the ^{18}F isotope. Both decay processes follow the same general reaction scheme:

$$^{A}_{Z}X \rightarrow {}^{A}_{Z-1}X + {}^{0}_{+1}e + {}^{0}_{0}\nu \tag{2}$$

where X represents the isotopic element of interest, A is the mass number, Z is the atomic number, $^{0}_{+1}e$ is a positron, and $^{0}_{0}\nu$ is a neutrino.[52,53]

To begin, model experimental data for the decay of ^{18}F are provided to students. They are guided to plot this data appropriately (i.e., plotting data points as markers and fits or continuous data as a line, and with informative axis titles and unit), apply a sorting operation based on the absolute difference from a reference value to extract $\tau_{0.5}$, and print the half-life in the terminal. We explicitly define the concept of half-life within the lab script to support students in identifying the correct parameters for extracting the appropriate value. The pseudocode for the sorting algorithm is presented in Algorithm 3 (Supporting Information).

The extracted half-life for ^{18}F is 6588 s (or 109.8 min). Following this and using the universal rate law for radioactive decay, students are introduced to the β^+ decay of ^{15}O and instructed to generate model data based on the known decay constant of this isotope. The universal rate law for radioactive decay is given by

$$N_t = N_0 e^{-kt} \tag{3}$$

where N_t is the number of radioactive particles at time t, N_0 is the initial number of radioactive particles, and k is the decay constant for the isotope. For ^{15}O, the decay constant is 0.00568 s^{-1}. Using this equation, students generate model data following the pseudocode provided in Algorithm 4 and extract $\tau_{0.5}$ using Algorithm 3—both presented in the Supporting Information.

At the end of the program, the output plot will consist of both the experimental ^{18}F data and the model ^{15}O data as shown in Figure 2 alongside the extracted $\tau_{0.5}$ in the terminal.

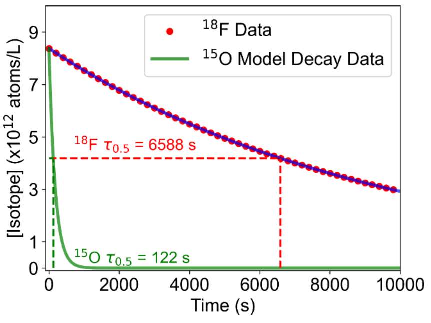


Figure 2. Modified student-generated output nuclear decay profile showing the variations in the concentration of isotopes as a function of time. Red data points (every 200 data points plotted) and blue line denote the ^{18}F decay data. The green line denotes the model ^{15}O data generated in the lab practical. $\tau_{0.5}$ values for each isotope are given.

Educators can generate their own model data by generating a list of times using `numpy.linspace(0, 10000, 10000)` and the data using eq 3 and the extracted decay constant (1.52×10^{-4} s^{-1}).

3.2. Basic Simulations of Nuclear Fusion

The second part of the lab practical involves simulating the deuterium (^{2}H)–tritium (^{3}H) fusion reaction, a key process in stellar nucleosynthesis, to form helium (^{4}He) and a neutron (^{1}n).[52,53] The reaction proceeds in two sequential steps:

$$^{2}H + {}^{2}H \xrightarrow{k_1} {}^{3}H + {}^{1}H \tag{4}$$

$$^{2}H + {}^{3}H \xrightarrow{k_2} {}^{4}He + {}^{1}n \tag{5}$$

where k_1 and k_2 are the respective rate constants for each reaction step. Students are provided with a worked example as guidance to derive indexed numerical rate equations (explicit rate calculations at each discrete time step, indexed by the time step counter) for each species and reaction rates for each overall reaction step. This enables the calculation of concentration changes and rates of reaction at each time step of the simulation (see Section 4.2.1 in the Supporting Information). These rate equations are thus solved iteratively rather than via direct integration.

Students then develop their own Python code to perform the simulation, supported by code snippets and instructions within the lab script. A *for* loop is introduced to the students and used to control the necessary calculations at each time step. The workflow of this code is given in Algorithm 5 (Supporting Information).

Using a given set of initial values for [^{2}H], [^{3}H], [^{4}He], and k_1, students run the initial simulation and plot how the concentration of each species changes over time. The initial parameters and resulting plot are shown in Figure 3a. The simulation spans 100 billion years, in increments of 2 billion years per time step.

To extend the exercise, students modify the initial parameters to investigate how changes in [^{2}H] and [^{3}H] affect the simulation results. This enables them to determine the rate order for each species in the reaction and to identify the rate-

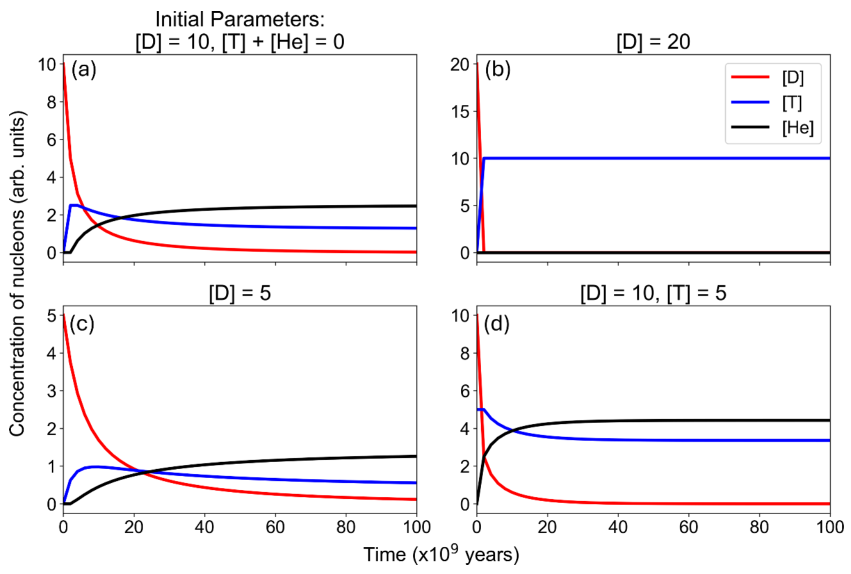


Figure 3. A kinetic model of deuterium−tritium fusion using the initial model parameters (a), where [D] is doubled (b), where [D] is halved (c), and where there is a starting amount of [T] (d). [D] is the concentration of deuterium (red), [T] is the concentration of tritium (blue), and [He] is the concentration of helium (black). The values for k_1 and k_2 are 1.25×10^7 and 2.50×10^7 years^{-1} respectively.

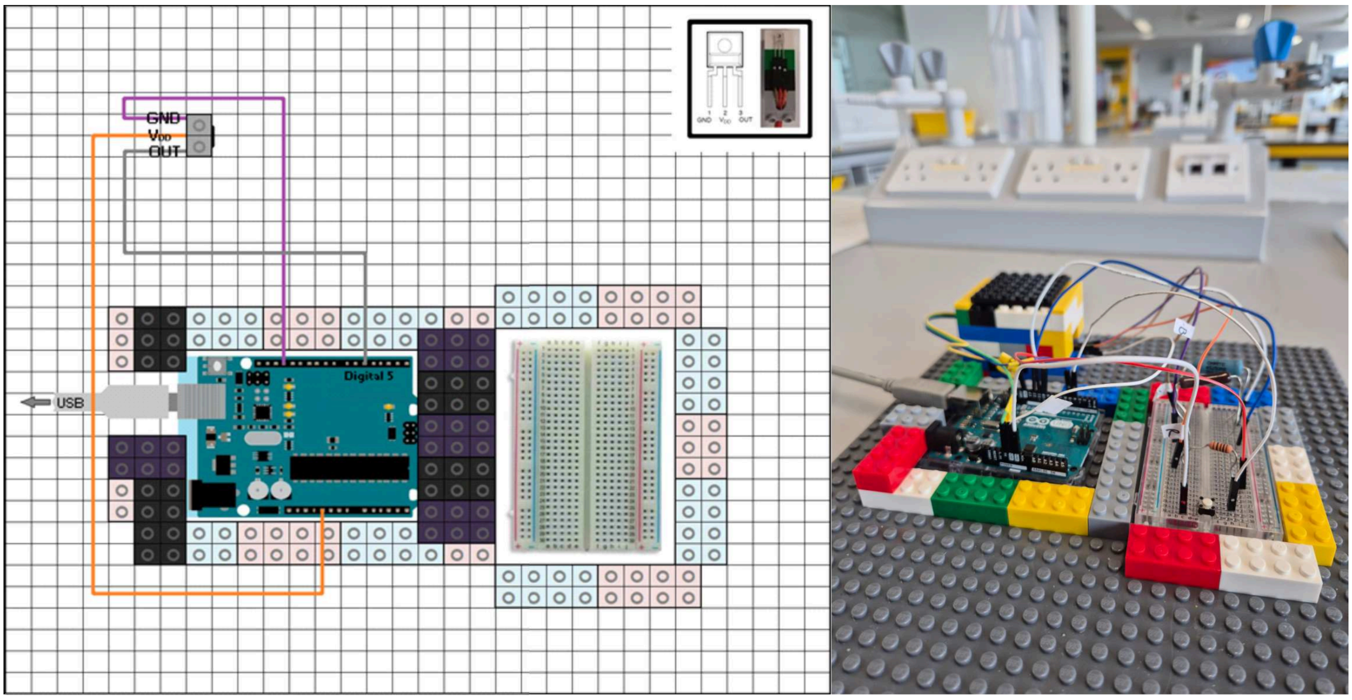


Figure 4. Left: Schematic provided to students demonstrating how to correctly connect a light sensor to an Arduino board. Further steps direct students to connect a RGB LED via the breadboard. Right: Photograph of the Arduino colorimeter inside its LEGO scaffold. The glass cuvette is contained in the LEGO box at the back left to provide a dark environment.

limiting step. The modified parameters and the corresponding plots are presented in Figures 3b−d, respectively.

Although this model is crude, overly simplistic, and heavily dependent on the chosen rate constants, we found during the first delivery of the lab practical that it provides a valuable opportunity for students to freely modify the system and critically engage with its limitations. These include the absence of temperature and pressure effects, side reactions, the complete lifecycle dynamics of a star, and additional mechanisms such as quantum tunnelling.

4. DATA ACQUISITION WITH AN ARDUINO

A key motivation for incorporating Python coding as a skill in chemistry laboratories is that it is open source, proving free access to powerful computational functions. This concept can be extended to data acquisition using open-source electronics, such as Arduino boards. These microcontrollers can be coupled to commercial or homemade equipment to facilitate remote or automated data acquisition.[54] Educators may use this concept to provide prebuilt equipment at greatly reduced costs[55,56] and or may incorporate construction of the device into the lesson plan to incorporate electronics as an additional skill.[57,58]

4.1. General Arduino Light Sensor Setup

In our laboratories, we use an Arduino board coupled to a simple light sensor as the basis for colorimetric experiments. Students follow a step-by-step guide for constructing the electronic components themselves by connecting the light sensor to an Arduino via a breadboard. An exception is that the Arduino board itself is provided with a preassembled C++ microcontroller code. Students are also provided with a simple python code, written using the Serial library, that forms the basis of their data acquisition code. Both codebases are freely available for educators online,[59] with a summary of the repository's content in Section 5 of the Supporting Information. Students also construct the basic architecture of the equipment out of either cardboard or LEGO to provide an engaging experience (Figure 4). A full component list and wiring diagram for the Arduino setups presented here are given in Section 5 of the Supporting Information.

We present these practicalities within the context of automated data acquisition. Students are provided with a scaffolded introduction to the new Arduino integration while being given the opportunity to independently construct analysis and plotting codes based on previous exercises. This is reinforced through two separate sessions built upon popular teaching lab experiments: Colorimetry[60] and Iodine Clock.[61]

4.2. Colorimetry of Dyes

Colorimetry relates the absorbance of a species in solution to its concentration according to Beer–Lambert's law

$$A = \log_{10}\left(\frac{I_0}{I}\right) = \varepsilon cl \tag{6}$$

where A is the absorbance, I_0 is the background intensity, I is the intensity through the sample, ε is the molar extinction coefficient and l is the path length of the colorimeter cell.

Students are guided through the process of coding data acquisition code for the colorimeter by being presented with the mathematical steps and then independently writing the code to achieve each calculation stage. For absorbance measurements, this involves recording the intensity detected at the light sensor when using the solvent (I_0), then the intensity through the sample (I) to determine the absorbance. This reinforces the origin of the absorbance value (eq 6), which is advantageous over standard colorimeters that simply report a value.

Students then use their Arduino-based colorimeter that they build to determine first the value of ε for Allura Red, a red food dye, and then build a calibration curve to quantify the amount of that dye in a commercial cough sweet. The analysis techniques required to achieve this build on skills first introduced in the Introduction to Python lab. Concentration and absorbance values are stored in lists in the code, Matplotlib and NumPy modules are used to plot graphs and perform a linear regression on the data respectively, and native Python functions are used to calculate values and print outputs. The workflow of the generated code is given in Algorithm 6, and the experimental procedure for this practical is given in Section 5.3.1 in the Supporting Information.

We add wider thinking and experimental design into the lab practical by requiring students to first select the appropriate color LED for the absorption of Allura red. This incorporates a discussion on complementary colors and encourages students to verify their answer experimentally.

4.3. Iodine Clock

The iodine clock is a cyclical reaction mechanism consisting of simultaneous oxidation and reduction steps. Different variants to the mechanism are used in various teaching laboratories, but the core concept centers on the reaction between iodide, hydrogen peroxide, and thiosulfate. Peroxide can oxidize iodide into iodine

$$H_2O_2 + 2H^+ + 2I^- \rightarrow I_2 + OH^- \tag{7}$$

The resultant iodine can be reduced back to iodide through a reaction with thiosulfate.

$$I_2 + 2S_2O_3{}^{2-} \rightarrow 2I^- + S_4O_6{}^{2-} \tag{8}$$

Once all of the thiosulfate is consumed, iodine can be observed as a persistent brown color, or blue if used in combination with a starch indicator.[62] This can be detected and quantified through absorbance measurements.

For this experiment, we advance the student experience with automation further. Students are instructed to produce a code that produces full automation of both the experimental procedure and data analysis. The automation is built around input statements. Connection to the Arduino light sensor measures the intensity of the reaction through the working solution. When the thiosulfate is consumed and the solution turns yellow, the intensity drops due to light absorption. Students are introduced to the concept of a while loop and Boolean operators (True, False) and, using these, are guided to construct the code to continually record the measured intensity until it drops below a threshold value, thereby automatically detecting the reaction end point.

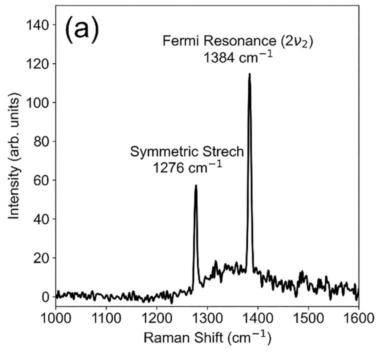
The code then prompts the user to add aliquots of thiosulfate until the solution turns colorless, indicating all iodine has been consumed. The input statement then records the number of aliquots added, and the time module records the time at which the threshold was breached, which are both added to lists for later analysis.

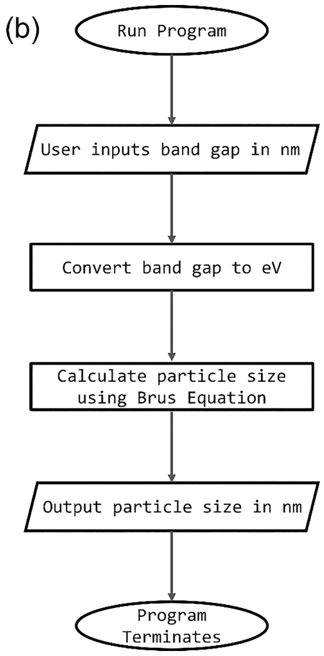
The code continues until prompted by the student to break the loop, at which point students use their data analysis and visualization to plot the trend in consumed thiosulfate vs time in order to extract a pseudo first order rate coefficient. Student outputs of the plotted linear trend and extracted parameters are presented in a similar way to that demonstrated in Figure 1.

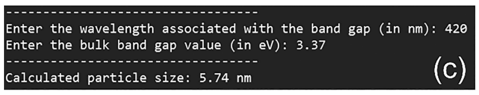
In this way, students produce code that can fully automate both data acquisition and analysis, whereby their constructed code prompts the user how to perform the experiment and then returns an extracted kinetic parameter. The workflow of the generated code is given in Algorithm 7 in the Supporting Information, alongside the experimental procedure in Section 5.4.1.

5. INCORPORATION OF CHEMICAL DATA ANALYSIS

One of the main challenges we encountered when designing new undergraduate physical chemistry experiments was striking a balance between the increasing complexity of Python programming and the demands of wet chemistry. The most effective solution was to completely shift the analysis component of the practicals to Python. Traditionally, our laboratory sessions have relied on spreadsheet software for data analysis, plotting, and annotation and performing extensive calculations. While this approach is convenient and often less time-consuming, we have found that students develop a stronger understanding of data handling and analysis when working programmatically. They




(b)
Run Program
User inputs band gap in nm
Convert band gap to eV
Calculate particle size using Brus Equation
Output particle size in nm
Program Terminates



```
---------------------------------
Enter the wavelength associated with the band gap (in nm): 420
Enter the bulk band gap value (in eV): 3.37
---------------------------------
Calculated particle size: 5.74 nm
```

Figure 5. Basic implementations of Python-based programming in chemical data analysis. In (a), we plot the annotated Raman spectrum of dry ice obtained in out astrochemical spectroscopy practical. In (b), we present a flow diagram to show the operation of a particle size calculator based on the Brus equation from our quantum dots practical, and in (c) we show the output of the code for ZnO quantum dots.

also become more critically aware of how data should be plotted and whether calculated values are physically reasonable.

To support this, we provide several example practicals in this section and the next section, in which these tasks are carried out exclusively in Python, concentrating on the context-learning aspect and building further on fundamental programming skills with more advanced techniques.

5.1. Basic Implementations: Plotting Data and Equation Calculators

Our most fundamental applications of Python programming in chemical data analysis involve plotting, annotating experimental data, and developing simple equation calculators.

For data visualization, we routinely use the Matplotlib library in our undergraduate laboratories as it offers straightforward and versatile tools for plotting chemical data. This also encouraged students to reuse their own codes to save time. As an example, Figure 5a shows the Raman spectrum of dry ice (CO_2) obtained during an astrochemical spectroscopy practical class. In this experiment, students are tasked with acquiring both infrared and Raman spectra of dry ice, alongside a micrometer-wavelength spectrum, and determining which vibrational mode of CO_2 is responsible for its detection in the GW Lup protoplanetary disk.[63] Detailed information about each plotting function used to create Figure 5a is provided in Section 6.1 of the Supporting Information.

Over the past decade, there has been a significant increase in the development and use of online calculators, often employed by students to avoid lengthy or repetitive calculations. Common examples used in our teaching laboratories include conversions between wavelength (nm) ↔ photon energy (eV), or between moles (mol) and concentration (M). However, one important point we emphasize to students is that no online calculator exists for every possible equation they might encounter. As such, we actively encourage students to develop their own custom calculators for laboratory practicals involving multiple, laborious calculations. In Figure 5b, we present a flow diagram and example terminal output for a Brus equation calculator, which is used to determine nanoparticle particle size from its measured band gap. This activity is part of our quantum dots laboratory, where students synthesize and dope ZnO quantum dots via microwave (oven) irradiation. The terminal output from this program is shown in Figure 5c. The underlying code can be easily adapted to calculate particle sizes for a range of band gap values using a while loop with a user-defined break condition to terminate the program when desired.

5.2. Particle in a Box: Electronic Structure of Naturally-Occurring Chromophores

In this example, we developed a particle in a box experiment that integrates chromophore extraction, UV−visible absorption spectroscopy, and Python-based data analysis to facilitate student exploration of quantum mechanical phenomena in conjugated systems. Students examine electronic transitions in naturally occurring chromophores, calculate the conjugation lengths, and visualize corresponding wave functions and probability amplitudes.

The experimental procedure, adapted from ref 64, involves extracting lutein, β-carotene, and lycopene from fresh spinach, carrot, and deseeded tomato, respectively, using 10 mL of methanol. These plant-derived chromophores were selected to underscore the relevance of quantum mechanics in everyday materials and to minimize laboratory costs. The Python analysis tools described below are equally applicable to synthetic dyes commonly employed in traditional particle in a box exercises.[65]

Students record UV−vis absorption spectra (400−700 nm) using a PerkinElmer Lambda XLS spectrometer, identify absorption maxima (λ_{max}), and export the data as `.csv` files for further analysis. As part of the prelaboratory preparation, students research expected λ_{max} values: 470−480 nm for lycopene,[66] 430−450 nm for lutein, and 440−460 nm for β-carotene.[67] Co-extracted chlorophyll a from spinach typically yields an additional absorption feature near 650−670 nm.[64]

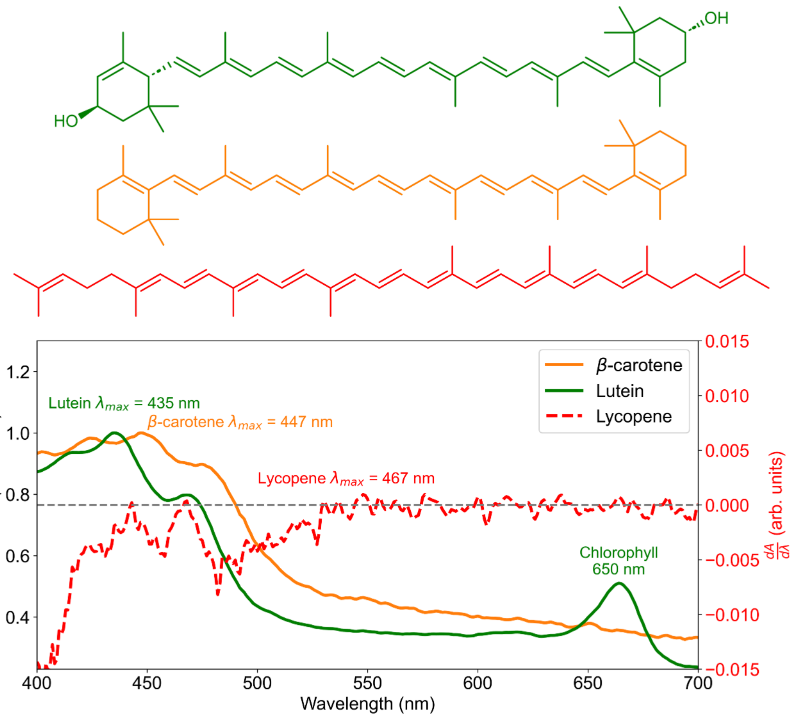


Figure 6. UV–visible absorption spectra for lycopene extracted from tomato (red, dashed), lutein and chlorophyll extracted from spinach (green), and β-carotene extracted from carrot (orange). We present the lycopene spectrum as dA/dλ and determine $\lambda_{\max}$ via the zero crossing at 467 nm. The gray dashed line denotes when dA/dλ = 0.

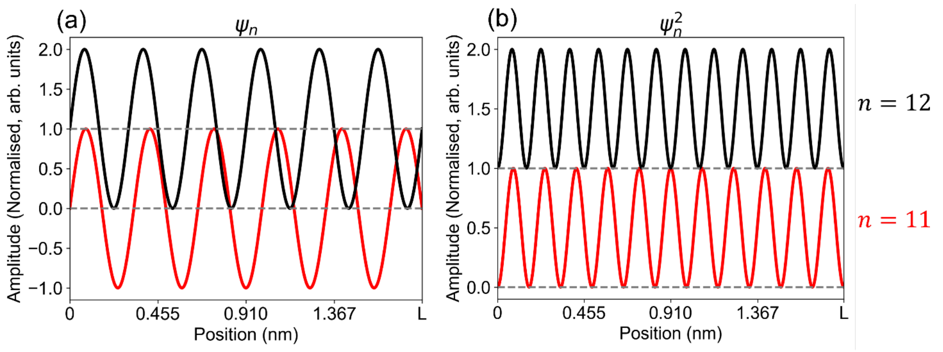


Figure 7. Simulated electronic wave functions (a) and probability amplitudes (b) for lycopene with a calculated box length of 1.82 nm at n = 11 (red) and n = 12 (black). As each sinusoid is normalized to the maximum in its respective list and offset, the gray dashed lines denote the zero point for each quantum number.

Representative overlaid spectra and associated $\lambda_{\max}$ values are presented in Figure 6.

To analyze their UV–visible spectra, students import the absorbance data into separate lists. For lutein and β-carotene, they identify the maximum absorbance and its corresponding index, which directly maps to $\lambda_{\max}$ in the wavelength list. This exercise reinforces basic Python skills, including list manipulation and indexing. The corresponding pseudocode is presented in Algorithm 8 (Supporting Information).

While this method is effective for lutein and β-carotene, the lycopene spectrum is complicated by overlap with other extractable compounds from tomato. To resolve this, students compute the first derivative of absorbance with respect to wavelength (dA/dλ) and plot it as a function of λ. This derivative can be obtained either iteratively or via `numpy.gradient()`. Although the iterative method is detailed in the lab script, students are encouraged to use NumPy to improve the efficiency. This step strengthens their computational proficiency and spectroscopic interpretation. As shown in Figure 6, the resulting plot reveals zero crossings, aiding in the visual identification of $\lambda_{\max}$ for lycopene.

Next, students assign quantum numbers (n) to each chromophore's transition and calculate the effective box length (L) using the particle in a box model:

$$\Delta E = \frac{h^2}{8m_e L^2}(2n + 1) \tag{9}$$

where ΔE is the transition energy (converted from $\lambda_{\max}$), h is Planck's constant, and m_e is the mass of an electron. Further problems may include comparing calculated L values with those from electronic structure calculations or from applying the law of cosines, considering model limitations, and exploring the effects of replacing the particle with a muon or antiproton. Students may also examine how chromophore structure affects absorption characteristics.

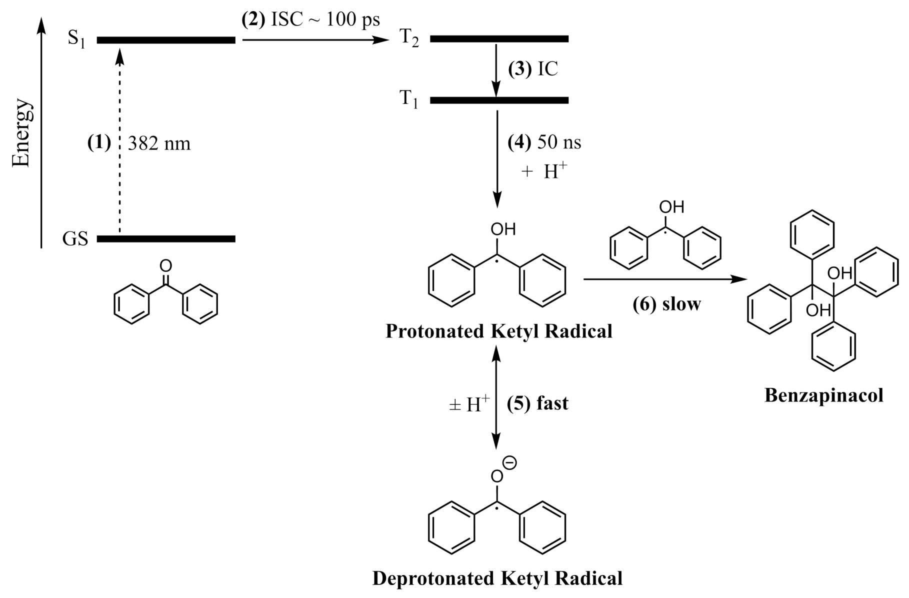


Figure 8. Schematic diagram of the photochemistry of benzophenone leading to the deprotonated ketyl radical and benzapinacol. GS is the ground state, ISC is intersystem crossing, and IC is internal conversion. The Jablonski diagram is reproduced from ref 68. Available under a CC-BY 4.0 license. Copyright 2017 Esther K. Riga, Julia S. Saar, Roman Erath, Michelle Hechenbichler, and Karen Lienkamp. Key time scales and state transitions are reported in refs 69 and 70. The reaction scheme portion is reproduced from ref 71. Copyright 2006 American Chemical Society.

Finally, students calculate and plot the electronic wave functions (ψ_n) and corresponding probability amplitudes (ψ_n^2) for each transition. For a 1D box, the wave function is defined as

$$\psi_n = \sqrt{\frac{2}{L}}\sin\left(\frac{n\pi x}{L}\right) \tag{10}$$

with x denoting the electron's position. Squaring ψ_n yields the probability amplitude. Students generate these plots using a custom Python script, iterating over x to compute ψ_n and ψ_n^2, as detailed in Algorithm 9 (Supporting Information). Figure 7 shows the results for lycopene with $L = 1.82$ nm and quantum numbers $n = 11$ and $n = 12$. The x values are generated using `numpy.linspace()`, producing 1000 evenly spaced points. Students can then estimate the electron's probability distribution by analyzing the peaks in the probability amplitude plot.

5.3. Flash Photolysis of Benzophenone

In our final example, we demonstrate how Python programming can be used to fit time-resolved spectroscopic data and extract meaningful error estimates from the fitting process. In this laboratory practical, students explore how the concentration of sodium hydroxide (NaOH) in solution influences the lifetime of the deprotonated ketyl radical (DKR), using a simplified transient absorption spectroscopy setup. The DKR is generated photochemically from benzophenone upon excitation with a 382 nm light. The key photochemical steps and subsequent reactions are summarized in Figure 8.[68−71]

The sample preparation procedure was adapted from literature.[72] Students begin by preparing a 5 mM solution of benzophenone in 2-propanol (250 mL), along with a series of sodium hydroxide (NaOH) solutions at varying concentrations (10−50 mM, 100 mL each). For each measurement, 25 mL of the benzophenone solution is combined with 25 mL of a selected NaOH solution. The resulting mixtures are deoxygenated by bubbling nitrogen (N_2) through them for 30 min.

Following deoxygenation, students assembled the flash photolysis apparatus, shown in Figure 9. The custom-built

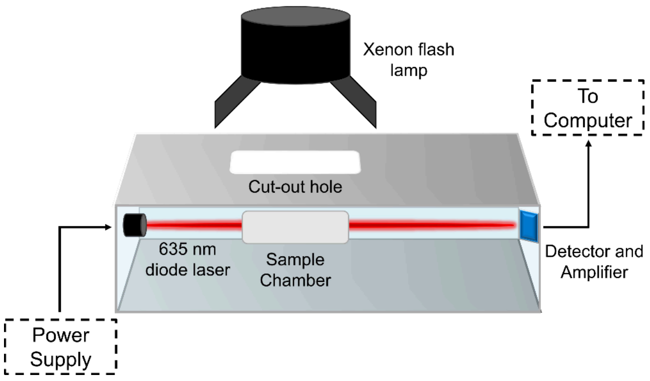


Figure 9. Experimental setup to study the lifetime of the deprotonated ketyl radical in solution. A description of the setup is given in the main text.

setup features a 635 nm laser, detector, and amplifier housed within a sample chamber, with a xenon flash lamp mounted above a cut-out . To operate the system, students activate the laser, amplifier (set to 9 V), detector, and flash lamp. Absorbance at 635 nm is monitored continuously, producing clear transient signals upon flashing the xenon lamp. While the apparatus is specific to this laboratory, similar designs may be adapted from other sources.[73,74]

Transient absorption data are collected using Picoscope 6 software, with 20−30 single-shot transients recorded for each NaOH concentration. These data are exported as `.csv` files for further processing. Students average the transients and identify the time zero (reaction start) and time end (reaction completion) points. While conceptually straightforward, data processing can be tedious. To streamline this step, a custom Python-based graphical user interface (GUI) is provided, allowing students to load raw traces, interactively define time zero and time end, compute an average trace, and convert the voltage to absorbance. The GUI, developed using the Tkinter library, is supplied with raw Python code and a flowchart to

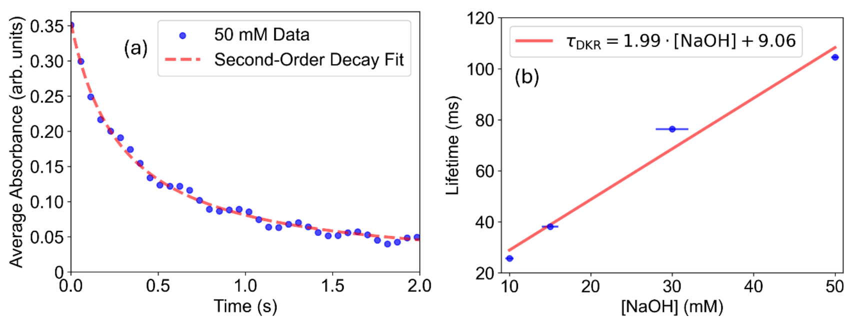


Figure 10. Student-generated figures showing, in (a), the transient data in blue dots and the fit of eq 11 as a red dashed line, and in (b) the effect of increasing [NaOH] on the lifetime with error bars along each data point (blue) in both axes. The straight line fit is shown in red with the full equation ($y = mx + c$) and fitted parameters given in the legend. We note that the extracted errors for each lifetime are on the order of 0.1 ms and are not visible on the plot unless zoomed in.

support student understanding of the data workflow. Although GUI programming is not formally taught, this activity provides an accessible introduction to graphical programming concepts. The app, alongside example data, is freely available online for educators to use and adapt in their own laboratories.[75] We also include a summary of the repository's contents in Section 6.3 of the Supporting Information.

Processed data are analyzed by fitting the averaged transients to a second-order rate equation:

$$A_t = \frac{A_0}{1 + kA_0t} \tag{11}$$

where A_t is the absorbance at time t, A_0 the fitted initial absorbance, and k the second-order rate constant. This model is implemented in Python as a function receiving initial parameter guesses and arrays of time and absorbance values. Fitting is performed using a least-squares fit via the `scipy.optimize.curve_fit()`, function which returns the best fit parameters (`popt`) and the covariance matrix (`pcov`). The standard errors of the fitted parameters (`perr`) are obtained from the square root of the diagonal elements of `pcov`.

The lifetime τ of the deprotonated ketyl radical is then calculated as

$$\tau = \frac{1}{k} \tag{12}$$

The modular Python code enables students to repeat the analysis across data sets by loading new `.csv` files. Pseudocode for the full workflow is provided in Algorithm 10 (Supporting Information). After fitting, students evaluated the effect of NaOH concentration on radical lifetime by plotting τ versus [NaOH], using `numpy.polyfit()` to perform a linear fit and including error bars for both variables. The slope of this line indicates the change in lifetime (ms) per mM increase in NaOH. Representative kinetic fits and concentration−lifetime trends are shown in Figure 10. Finally, students critically assess the robustness of the observed trend, especially in the context of the limited data set.

6. CONCLUSIONS

We have presented a curriculum-focused, modular approach to integrating Python coding as a key transferable skill into undergraduate physical chemistry experiments. Python is first introduced with a carefully scaffolded approach that assumes no previous coding or practical chemistry experience. As students gain basic skills and confidence, the course design shifts to project-based learning, where coding skills are integrated into chemical tasks to provide an engaging and relatable learning experience.

Although not demonstrated in this work, the core concept has the scope to extend the student skill set into powerful computational operations through Python, including development of bespoke GUIs and machine learning. We recommend care is taken in the introduction of coding as a concept, with a carefully scaffolded approach through worked examples and "code-along" exercises prioritizing support and confidence in early learning environments. This provides students with a sufficient foundation in coding to benefit from, and enjoy contextualized learning opportunities, as they develop their skills toward exciting higher order Python operations.

Finally, a brief note on the use of large language models (LLMs). The use of LLMs in programming has grown significantly in recent years. In our courses, we discourage their use during the foundational learning stages (sections 2−4). This is because the fundamental understanding of core Python concepts from our introduction is essential in both understanding LLM replies and engineering a prompt capable of producing a helpful response. At more advanced stages (sections 5.2 and 5.3), we encourage responsible use of LLMs for code debugging and general support, provided students do not use them to generate complete, AI-generated code and submit this as their own work.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available at https://pubs.acs.org/doi/10.1021/acs.jchemed.5c00677.

Chemical-specific hazards for each experiment reported here, table of programming skills taught for each practical presented, Algorithms 1−10 of pseudocode workflow for the Python exercises described throughout the main text, components list and wiring diagram for the Arduino practicals, experimental procedures for the colorimetry and iodine clock practicals, student guidance for kinetic model derivation in basic simulations of nuclear fusion, summary of contents for the GitHub repositories (Arduino codes and Transient Processing App, respectively)[59,75] (PDF)

AUTHOR INFORMATION

Corresponding Authors

Derri J. Hughes − *School of Chemistry and Chemical Engineering, University of Southampton, Southampton SO17 1BJ, United Kingdom*; orcid.org/0000-0002-7307-5690; Email: D.J.Hughes@soton.ac.uk

Samuel C. Perry − *School of Chemistry and Chemical Engineering, University of Southampton, Southampton SO17 1BJ, United Kingdom*; orcid.org/0000-0002-6263-6114; Email: S.C.Perry@soton.ac.uk

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jchemed.5c00677

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

D.J.H. would like to thank ELI Beamlines and the University of Southampton for a studentship. Both authors would like to thank Kat Swindells-Ridley, Jess Silvester and Jake Lewis for their continual support and invaluable assistance in the development and running of the L4 laboratories at Southampton.

REFERENCES

(1) McCluskey, A. R.; Dean, J. M.; Squires, A. G.; Meli, R.; Glass, W. G.; Symington, A. R.; Morgan, B. J. An Introduction to Python for Chemists Zenodo. DOI: DOI: 10.5281/zenodo.3718970.

(2) Hielscher, M. M.; Dörr, M.; Schneider, J.; Waldvogel, S. R. LABS: Laboratory Automation and Batch Scheduling − A Modular Open Source Python Program for the Control of Automated Electrochemical Synthesis with a Web Interface. *Chemistry − An Asian Journal* **2023**, *18*, No. e202300380.

(3) Seifrid, M.; Strieth-Kalthoff, F.; Haddadnia, M.; Wu, T. C.; Alca, E.; Bodo, L.; Arellano-Rubach, S.; Yoshikawa, N.; Skreta, M.; Keunen, R.; Aspuru-Guzik, A. Chemspyd: an open-source python interface for Chemspeed robotic chemistry and materials platforms. *Digital Discovery* **2024**, *3*, 1319−1326.

(4) 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.

(5) Sun, Q.; Berkelbach, T. C.; Blunt, N. S.; Booth, G. H.; Guo, S.; Li, Z.; Liu, J.; McClain, J. D.; Sayfutyarova, E. R.; Sharma, S.; Wouters, S.; Chan, G. K.-L. PySCF: the Python-based simulations of chemistry framework. *WIREs Computational Molecular Science* **2018**, *8*, No. e1340.

(6) Dahlgren, B. ChemPy: A package useful for chemistry written in Python. *Journal of Open Source Software* **2018**, *3*, 565.

(7) Malpica Galassi, R. PyCSP: A Python package for the analysis and simplification of chemically reacting systems based on Computational Singular Perturbation. *Comput. Phys. Commun.* **2022**, *276*, 108364.

(8) Ong, S. P.; Richards, W. D.; Jain, A.; Hautier, G.; Kocher, M.; Cholia, S.; Gunter, D.; Chevrier, V. L.; Persson, K. A.; Ceder, G. Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis. *Comput. Mater. Sci.* **2013**, *68*, 314−319.

(9) Broberg, D.; Medasani, B.; Zimmermann, N. E.; Yu, G.; Canning, A.; Haranczyk, M.; Asta, M.; Hautier, G. PyCDT: A Python toolkit for modeling point defects in semiconductors and insulators. *Comput. Phys. Commun.* **2018**, *226*, 165−179.

(10) Summers, A. Z.; Gilmer, J. B.; Iacovella, C. R.; Cummings, P. T.; MCabe, C. a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *J. Chem. Theory Comput.* **2020**, *16*, 1779−1793.

(11) Weiss, C. J. Perspectives: Teaching Chemists to Code. *Chem. Eng. News* **2017**, *95*, 30.

(12) Cropper, C. Why should chemistry students learn to code?.*RSC Education* **2017** (accessed 2025-03-22) https://edu.rsc.org/opinion/why-should-chemistry-students-learn-to-code/3008177.

(13) Ryzhkov, F. V.; Ryzhkova, Y. E.; Elinson, M. N. Python in Chemistry: Physicochemical Tools. *Processes* **2023**, *11*, 2897.

(14) Van Staveren, M. Integrating Python into a Physical Chemistry Lab. *J. Chem. Educ.* **2022**, *99*, 2604−2609.

(15) De Haan, D. O.; Schafer, J. A.; Gillette, E. I. Using a Modular Approach to Introduce Python Coding to Support Existing Course Learning Outcomes in a Lower Division Analytical Chemistry Course. *J. Chem. Educ.* **2021**, *98*, 3245−3250.

(16) Campbell, E. C.; Christensen, K. M.; Nuwer, M.; Ahuja, A.; Boram, O.; Liu, J.; Miller, R.; Osuna, I.; and, S. C. R. Cracking the code: An evidence-based approach to teaching Python in an undergraduate earth science setting. *Journal of Geoscience Education* **2025**, *73*, 239.

(17) 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.

(18) Fransson, T.; Delcey, M. G.; Brumboiu, I. E.; Hodecker, M.; Li, X.; Rinkevicius, Z.; Dreuw, A.; Rhee, Y. M.; Norman, P. eChem A Notebook Exploration of Quantum Chemistry. *J. Chem. Educ.* **2023**, *100*, 1664−1671.

(19) Stippell, E.; Akimov, A. V.; Prezhdo, O. V. PySyComp: A Symbolic Python Library for the Undergraduate Quantum Chemistry Course. *J. Chem. Educ.* **2023**, *100*, 4077−4084.

(20) Ayeni, O. M.; Migliaro, I.; Omary, M. A.; Atkinson, M. B. SymmSpec: An Interactive Python Tool for Predicting IR and Raman Activity for Undergraduate Inorganic Chemistry. *J. Chem. Educ.* **2025**, *102*, 644−653.

(21) Rehn, D. R.; Rinkevicius, Z.; Herbst, M. F.; Li, X.; Scheurer, M.; Brand, M.; Dempwolff, A. L.; Brumboiu, I. E.; Fransson, T.; Dreuw, A.; Norman, P. Gator A Python-driven program for spectroscopy simulations using correlated wave functions. *WIREs Computational Molecular Science* **2021**, *11*, No. e1528.

(22) Zlatković, D.; Đorđević Zlatković, M.; Radulović, N. Problem-Solving with Python: Modeling of Lanthanide-Shift Reagent Complexes. *J. Chem. Educ.* **2023**, *100*, 3620−3625.

(23) Zhang, Z.; Gautam, A.; Lim, S.-M.; Hilty, C. Analysis of Large Data Sets in a Physical Chemistry Laboratory NMR Experiment Using Python. *J. Chem. Educ.* **2023**, *100*, 4109−4113.

(24) St James, A. G.; Hand, L.; Mills, T.; Song, L.; Brunt, A. S. J.; Bergstrom Mann, P. E.; Worrall, A. F.; Stewart, M. I.; Vallance, C. Exploring Machine Learning in Chemistry through the Classification of Spectra: An Undergraduate Project. *J. Chem. Educ.* **2023**, *100*, 1343−1350.

(25) Mahjour, B.; McGrath, A.; Outlaw, A.; Zhao, R.; Zhang, C.; Cernak, T. Interactive Python Notebook Modules for Chemoinformatics in Medicinal Chemistry. *J. Chem. Educ.* **2023**, *100*, 4895−4902.

(26) Cao, D.-S.; Xu, Q.-S.; Hu, Q.-N.; Liang, Y.-Z. ChemoPy: freely available python package for computational biology and chemoinformatics. *Bioinformatics* **2013**, *29*, 1092−1094.

(27) Zheng, W. Python for Electrochemistry: A Free and All-In-One Toolset. *ECS Advances* **2023**, *2*, 040502.

(28) Qiu, J.; Moeller, A.; Zhen, J.; Yang, H.; Din, L.; Adelstein, N. Teaching Heterogeneous Electrocatalytic Water Oxidation with Nickel- and Cobalt-Based Catalysts Using Cyclic Voltammetry and Python Simulation. *J. Chem. Educ.* **2023**, *100*, 3036−3043.

(29) Wang, X.; Wang, Z. Animated Electrochemistry Simulation Modules. *J. Chem. Educ.* **2022**, *99*, 752−758.

(30) 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.

(31) Weiss, C. J. Scientific Computing for Chemists with Python. (accessed 2025-03-22), https://weisscharlesj.github.io/SciCompforChemists/notebooks/introduction/intro.html.

(32) Heras-Domingo, J.; Garay-Ruiz, D. Pythonic Chemistry: The Beginner's Guide to Digital Chemistry. *J. Chem. Educ.* **2024**, *101*, 4883−4891.

(33) Bazargan, G. Why chemists should learn to code. *Chemistry World* **2021** (accessed 2025-05-12) https://www.chemistryworld.com/careers/why-chemists-should-learn-to-code/4013922.

(34) Bravenec, A. D.; Ward, K. D. Interactive Python Notebooks for Physical Chemistry. *J. Chem. Educ.* **2023**, *100*, 933−940.

(35) Ringer McDonald, A. Teaching Programming across the Chemistry Curriculum. *ACS Symposium Series* **2021**, *1387*, 1−11 Chapter.

(36) Ahmad, A.; Ray, S.; Nawaz, A.; Hin, Hua., *2024 9th International STEM Education Conference (iSTEM-Ed)*; Cha-am: Thailand, 2024; pp 1−6.

(37) Hyde, J. A New Perspective on Chemistry Foundation Level Students Laboratory Skill Development using Reciprocal Peer-Teaching, Laboratory Simulations, and Practical Skills Portfolio (PSP) during COVID-19 and Post-Pandemic in 2024. *J. Chem. Educ.* **2025**, *102*, 984−1003.

(38) Yang, A. C.; Lin, J.-Y.; Lin, C.-Y.; Ogata, H. Enhancing python learning with PyTutor: Efficacy of a ChatGPT-Based intelligent tutoring system in programming education. *Computers and Education: Artificial Intelligence* **2024**, *7*, 100309.

(39) Paas, F.; Renkl, A.; Sweller, J. Cognitive Load Theory and Instructional Design: Recent Developments. *Educational Psychologist* **2003**, *38*, 1−4.

(40) Le Cunff, A.-L.; Martis, B.-L.; Glover, C.; Ahmed, E.; Ford, R.; Giampietro, V.; Dommett, E. J. Cognitive load and neurodiversity in online education: a preliminary framework for educational research and policy. *Frontiers in Education* **2025**, *9*, 1437673.

(41) Grove, N. P.; Hershberger, J. W.; Bretz, S. L. Impact of a spiral organic curriculum on student attrition and learning. *Chem. Educ. Res. Pract.* **2008**, *9*, 157−162.

(42) Campbell, C. D.; Midson, M. O.; Mann, P. E. B.; Cahill, S. T.; Green, N. J. B.; Harris, M. T.; Hibble, S. J.; O'Sullivan, S. K. E.; To, T.; Rowlands, L. J.; Smallwood, Z. M.; Vallance, C.; Worrall, A. F.; Stewart, M. I. Developing a skills-based practical chemistry programme: an integrated, spiral curriculum approach. *Chemistry Teacher International* **2022**, *4*, 243−257.

(43) Bada, S. O.; Olusegun, S. Constructivism learning theory: A paradigm for teaching and learning. *Journal of Research & Method in Education* **2015**, *5*, 66−70.

(44) Seery, M. K.; Agustian, H. Y.; Christiansen, F. V.; Gammelgaard, B.; Malm, R. H. 10 Guiding principles for learning in the laboratory. *Chem. Educ. Res. Pract.* **2024**, *25*, 383−402.

(45) Duda, M.; Sovacool, K.; Farzaneh, N.; Nguyen, V.; Haynes, S.; Falk, H.; Furman, K.; Walker, L.; Diao, R.; Oneka, M.; Drotos, A.; Woloshin, A.; Dotson, G.; Kriebel, A.; Meng, L.; Thiede, S.; Lapp, Z.; Wolford, B. Teaching Python for Data Science: Collaborative development of a modular interactive curriculum. *Journal of Open Source Education* **2021**, *4*, 138.

(46) Kirschner, P. A.; Sweller, J.; Clark, R. E. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist* **2006**, *41*, 75−86.

(47) Shin, Y.; Jung, J.; Zumbach, J.; Yi, E. The Effects of Worked-Out Example and Metacognitive Scaffolding on Problem-Solving Programming. *Journal of Educational Computing Research* **2023**, *61*, 1312−1331.

(48) Phung, T.; Cambronero, J. P.; Gulwani, S.; Kohn, T.; Majumdar, R.; Singla, A. K.; Soares, G. Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models. *ArXiv* **2023**.

(49) Barzykina, I. Chemistry and Mathematics of the Belousov−Zhabotinsky Reaction in a School Laboratory. *J. Chem. Educ.* **2020**, *97*, 1895−1902.

(50) Sobel, S. G.; Hastings, H. M.; Field, R. J. Oxidation State of BZ Reaction Mixtures. *J. Phys. Chem. A* **2006**, *110*, 5−7.

(51) Armbruster, P.; Patel, M.; Johnson, E.; Weiss, M. Active Learning and Student-centered Pedagogy Improve Student Attitudes and Performance in Introductory Biology. *CBE—Life Sciences Education* **2009**, *8*, 203−213.

(52) Mcpherson, P. A. C. *Principles of nuclear chemistry*; World Scientific Europe, 2017.

(53) Williams, W. S. C. *Nuclear and particle physics*; Clarendon Press: 1991.

(54) Grinias, J. P.; Whitfield, J. T.; Guetschow, E. D.; Kennedy, R. T. An Inexpensive, Open-Source USB Arduino Data Acquisition Device for Chemical Instrumentation. *J. Chem. Educ.* **2016**, *93*, 1316−1319.

(55) Gomes, V. V.; Cavaco, S. C. F.; Morgado, C. P.; Aires-de-Sousa, J.; Fernandes, J. C. B. An Arduino-Based Talking Calorimeter for Inclusive Lab Activities. *J. Chem. Educ.* **2020**, *97*, 1677−1681.

(56) Arrizabalaga, J. H.; Simmons, A. D.; Nollert, M. U. Fabrication of an Economical Arduino-Based Uniaxial Tensile Tester. *J. Chem. Educ.* **2017**, *94*, 530−533.

(57) Reivanth, K.; Priya, A.; Nataraj, D. A Low-Cost Arduino and Python Based Gas Sensing Setup: Bridging Theory and Practice in Educational Environments. *J. Chem. Educ.* **2024**, *101*, 5361−5368.

(58) Mabbott, G. A. Teaching Electronics and Laboratory Automation Using Microcontroller Boards. *J. Chem. Educ.* **2014**, *91*, 1458−1463.

(59) Perry, S. *ArduinoConfig*, v1.0.0; https://github.com/Perry-SC/ArduinoConfig/tree/v1.0.0 (Accessed 08/08/2025).

(60) Kulkarni, P. S.; Watwe, V. S.; Nirmal, R. S.; Jagdale, H. N.; Kulkarni, S. D. Simultaneous Determination of Ni(II) and Co(II) in Aqueous Solution Using an Image-Based Do-It-Yourself Photometer. *J. Chem. Educ.* **2024**, *101*, 3451−3458.

(61) Barrera, L. A.; Escobosa, A. C.; Alsaihati, L. S.; Noveron, J. C. Conducting a Low-Waste Iodine Clock Experiment on Filter Paper To Discern the Rate Law. *J. Chem. Educ.* **2019**, *96*, 165−168.

(62) Sattsangi, P. D. A Microscale Approach to Chemical Kinetics in the General Chemistry Laboratory: The Potassium Iodide Hydrogen Peroxide Iodine-Clock Reaction. *J. Chem. Educ.* **2011**, *88*, 184−188.

(63) Grant, S. L.; van Dishoeck, E. F.; Tabone, B.; Gasman, D.; Henning, T.; Kamp, I.; Gudel, M.; Lagage, P.-O.; Bettoni, G.; Perotti, G.; Christiaens, V.; Samland, M.; Arabhavi, A. M.; Argyriou, I.; Abergel, A.; Absil, O.; Barrado, D.; Boccaletti, A.; Bouwman, J.; o Garatti, A. C.; Geers, V.; Glauser, A. M.; Guadarrama, R.; Jang, H.; Kanwar, J.; Lahuis, F.; Morales-Calderon, M.; Mueller, M.; Nehme, C.; Olofsson, G.; Pantin, E.; Pawellek, N.; Ray, T. P.; Rodgers-Lee, D.; Scheithauer, S.; Schreiber, J.; Schwarz, K.; Temmink, M.; Vandenbussche, B.; Vlasblom, M.; Waters, L. B. F. M.; Wright, G.; Colina, L.; Greve, T. R.; Justannont, K.; Ostlin, G. MINDS. The Detection of 13CO2 with JWST-MIRI Indicates Abundant CO2 in a Protoplanetary Disk. *Astrophysical Journal Letters* **2023**, *947*, L6.

(64) Wimpfheimer, T. A. Particle in a Box Laboratory Experiment Using Everyday Compounds. *Journal of Laboratory Chemical Education* **2015**, *3* (2), 19−21.

(65) Anderson, B. D. Alternative Compounds for the Particle in a Box Experiment. *J. Chem. Educ.* **1997**, *74*, 985.

(66) Cámara, M.; de Cortes Sánchez-Mata, M.; Fernández-Ruiz, V.; Cámara, R. M.; Manzoor, S.; Caceres, J. O. *Studies in Natural Products Chemistry; Studies in natural products chemistry*; Elsevier, 2013; pp 383−426.

(67) Domenici, V.; Ancora, D.; Cifelli, M.; Serani, A.; Veracini, C. A.; Zandomeneghi, M. Extraction of Pigment Information from Near-UV Vis Absorption Spectra of Extra Virgin Olive Oils. *J. Agric. Food Chem.* **2014**, *62*, 9317−9325.

(68) Riga, E.; Saar, J.; Erath, R.; Hechenbichler, M.; Lienkamp, K. On the Limits of Benzophenone as Cross-Linker for Surface-Attached Polymer Hydrogels. *Polymers* **2017**, *9*, 686.

(69) Beckett, A.; Porter, G. Primary photochemical processes in aromatic molecules. Part 9.—Photochemistry of benzophenone in solution. *Trans. Faraday Soc.* **1963**, *59*, 2038−2050.

(70) Venkatraman, R. K.; Orr-Ewing, A. J. Photochemistry of Benzophenone in Solution: A Tale of Two Different Solvent Environments. *J. Am. Chem. Soc.* **2019**, *141*, 15222−15229.

(71) Sakamoto, M.; Cai, X.; Kim, S. S.; Fujitsuka, M.; Majima, T. Intermolecular Electron Transfer from Excited Benzophenone Ketyl Radical. *J. Phys. Chem. A* **2007**, *111*, 223−229.

(72) Churio, M. S.; Grela, M. A. Photochemistry of Benzophenone in 2-Propanol: An Easy Experiment for Undergraduate Physical Chemistry Courses. *J. Chem. Educ.* **1997**, *74*, 436.

(73) Tran, J. B.; McCoy, J. C.; Bailey, L. M.; McDaniel, B. P.; Simon, R. L.; Marchetti, B.; Karsili, T. N. V. Affordable Setup for Studying Photochemistry in Action in Undergraduate Teaching Laboratories: Principles and Applications. *J. Chem. Educ.* **2020**, *97*, 2203−2211.

(74) Larsen, M. C.; Perkins, R. J. Flash Photolysis Experiment of o-Methyl Red as a Function of pH: A Low-Cost Experiment for the Undergraduate Physical Chemistry Lab. *J. Chem. Educ.* **2016**, *93*, 2096−2100.

(75) Hughes, D.; Perry, S. *Transient Processing App*, v1.0.1; https://github.com/Perry-SC/Transient_Processing_App/tree/v1.0.1 (Accessed 08/08/2025).