Preprint:

Lewis, Alan M orcid.org/0000-0002-3296-7203 Strategy before syntax: a new approach to programming instruction for science undergraduates. [Preprint] (Submitted)

v0

Strategy before syntax: a new approach to programming instruction for science undergraduates

Abstract

In most undergraduate science courses, programming teaching follows a "syntax-first" approach; students are first taught the basics of a programming language, and only later is programming incorporated into relevant scientific applications. While this seems a natural approach, it is not without risks: science students may not see the relevance of learning programming until late in a course and disengage as a result. This risk is more acute for students who initially see themselves as unable to learn to code; this self-perception is disproportionately prevalent in female students. We have investigated a novel "strategy-first" approach, beginning by exposing students to pre-written code which can be applied to problems they have seen in other parts of their undergraduate course, and then breaking down the code into sections for them to modify, teaching the relevant syntax along the way. This ensures students make connections between programming and scientific concepts and produce an output relevant to their scientific education from their first introduction to coding. We illustrate "strategy-first" programming instruction through two case studies, demonstrating how instructors can implement this approach. We show that these case studies produced effective learning outcomes, with strong improvements in students' understanding of both programming and chemistry. We also saw a statistically significant change in students' attitudes towards programming, becoming more confident and less apprehensive; this change is more pronounced in female students. Finally, we show that students identified a clear connection between their programming instruction and their broader scientific education, a key advantage of the "strategy-first" approach.

Keywords: programming knowledge; Python; science education, chemistry education; programming pedagogy

1 Introduction

Undergraduate science programmes around the world are increasingly recognising the need to incorporate programming teaching into their curricula (Fuchs et al., 2024; Guzman et al., 2019; Hambrusch et al., 2009; Mears et al., 2025; Ringer McDonald, 2021; Valle & Berdanier, 2012; Weiss, 2017). This reflects the expectation of both employers and students that science graduates will enter the workforce with some familiarity and competency with at least one programming language (Bright Network Research, 2023; Eurostat, 2023; Hooley, 2021; World Economic Forum, 2025).

Although the exact nature of programming teaching varies from institution to institution, the majority of published material for teaching programming to science undergraduates shares a broadly similar approach (Bravenec & Ward, 2023; Cawthorne, 2021; Cumby et al., 2023; Guzman et al., 2019; Lee et al., 2023; Mecca et al., 2021; Vallejo et al., 2022; van Staveren, 2022), even when novel delivery methods are introduced (Margulieux et al., 2020; Sun et al., 2024, 2021). This traditional approach begins by teaching students the basics of the programming language, defining variables, simple data structures, loops, if statements, and functions, and then combines those building blocks into more and more complex programmes until students are able to construct a program which can be applied to a scientific problem. We will call this a "syntax-first" approach, and will return to this terminology in more detail shortly.

While on first inspection a syntax-first approach appears to be the natural, and perhaps only, way to teach programming skills, it is not without drawbacks. For example, students may not see or understand the relevance of learning programming to their scientific education[1] until very late in a syntax-first course (Cawthorne, 2021), and as a result may disengage before reaching that point; similar risks are well known when teaching mathematics to chemistry and physics students (Bain et al., 2018; Haraldsrud & Odden, 2023; Redish & Kuo, 2015; Sherin, 2001). This risk is more acute for students who have previous bad experience of learning programming, or see themselves as unable to learn programming; this self-perception is disproportionately prevalent in female students (Koch et al., 2008; Mishkin, 2019). A syntax-first approach can also limit the number and breadth of scientific applications which can be introduced during a programming course, if basic syntax is first taught in the absence of scientific context. Therefore, in this paper we will introduce a novel alternative to the syntax-first approach, designed to avoid or mitigate these drawbacks, which we call a "strategy-first" approach.

The remainder of the paper is organised as follows: we first introduce some language describing programming knowledge, which will aid the comparison of these approaches, and then more formally describe the syntax-first approach and define our alternative strategy-first approach. We then describe two case studies in which this approach has been implemented to teach programming to chemistry undergraduate students. Finally, we use student surveys and instructor observations from these case studies to evaluate the effectiveness of strategy-first teaching, both as a tool to teach programming and chemistry, and in demonstrating strong student engagement and the relevance of programming teaching to scientific problems.

[1]For the purposes of this article, computer science students are excluded from the broader group of "science students".

2 Defining a strategy-first approach to programming teaching

In their influential article, McGill and Volet provide a framework for describing and analysing programming teaching and learning (McGill & Volet, 1997). They draw on three types of programming knowledge introduced by Bayman and Mayer: syntactic, conceptual, and strategic knowledge (Bayman & Mayer, 1988). These are defined as:

- Syntactic knowledge: knowledge of fundamental features of a programming language (e.g. how to define a string variable; how to add two integers together).
- Conceptual knowledge: an understanding of how individual pieces of syntax can be combined to achieve a specified goal.
- Strategic knowledge: the ability to deconstruct a large application into a series of smaller problems, which can be solved by the application of conceptual knowledge.

McGill and Volet combine these categories of programming knowledge with forms of knowledge defined in cognitive psychology literature, declarative and procedural knowledge (Anderson, 1982; Gagné et al., 1993). In doing so, they divide syntactic knowledge into two subcategories: declarative-syntactic knowledge, which consists of the ability to recognize basic syntactic facts of a particular programming language, and procedural-syntactic knowledge, which enables students to write syntactically correct lines of code (although not necessarily understand the concepts underpinning them). Similarly, conceptual knowledge is divided into declarative-conceptual knowledge, which allows students to recognize how combinations of programming statements in pre-written code combine to achieve a larger goal, and procedural-conceptual knowledge, which enables students to write such code themselves (McGill & Volet, 1997).

Having introduced this terminology, we can more formally define the idea of a syntax-first approach to teaching programming as one which expects a student to begin by learning declarative-syntactic knowledge, applying it to gain procedural-syntactic knowledge, then to synthesise their syntactic knowledge to develop declarative-conceptual and procedural-conceptual knowledge, before finally arriving at strategic knowledge. This is illustrated schematically in Figure 1. While this approach appears to be widespread, it is not necessarily optimal. Even as they introduced the ideas of syntactic and conceptual knowledge, Bayman and Mayer noted that teaching conceptual knowledge prior to syntactic knowledge produced significant improvements in students' knowledge transfer and problem solving, especially amongst students with a weaker programming background (Bayman & Mayer, 1988). Indeed, a number of studies have shown that an overemphasis on syntactic knowledge leads to what has been described as "fragile" knowledge, particularly among less confident students, resulting in students not being able to, or not believing that they are able to, apply their syntactic knowledge to solve conceptual problems (Linn & Clancy, 1992; McGill & Volet, 1997; Oliver & Malone, 1993; Snow & Lohman, 1984; Volet & Lund, 1994).

In this paper, we propose taking Bayman and Mayer's suggestion one step further: beginning with instruction of strategic knowledge, followed by conceptual and then

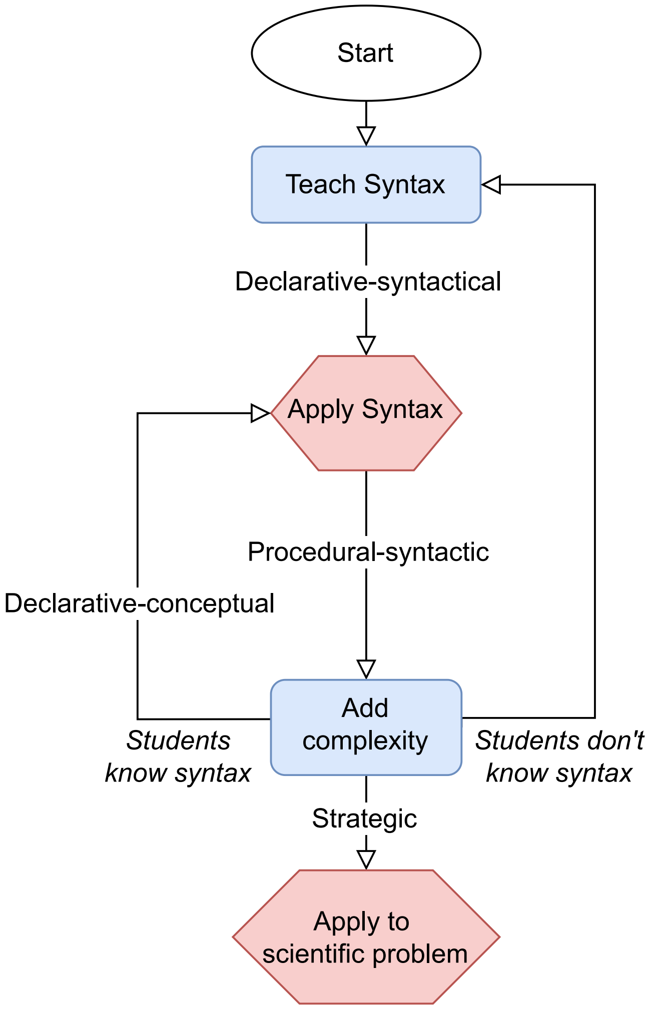

Fig. 1 A schematic representation of the syntax-first approach to programming teaching. Blue rectangles indicate instructor's actions; red hexagons are the student's actions. The lines are labelled with the category of programming knowledge developed following the action preceding it; italic text is used indicate the appropriate path at decision points

finally syntactic knowledge, following the diagram shown in Figure 2. This is possible thanks to the context of the programming instruction being undertaken: we will be considering scientific topics and concepts which students have already encountered elsewhere in their undergraduate programme, and introducing programming skills which can help them better understand and apply these concepts. As a result, students already have a measure of strategic knowledge about the problem to be solved, and some of the steps which must be undertaken to solve it. Therefore, strategy-first programming instruction focuses first on how these steps can be converted into a form which a computer can solve (introducing conceptual knowledge), and then finally gives

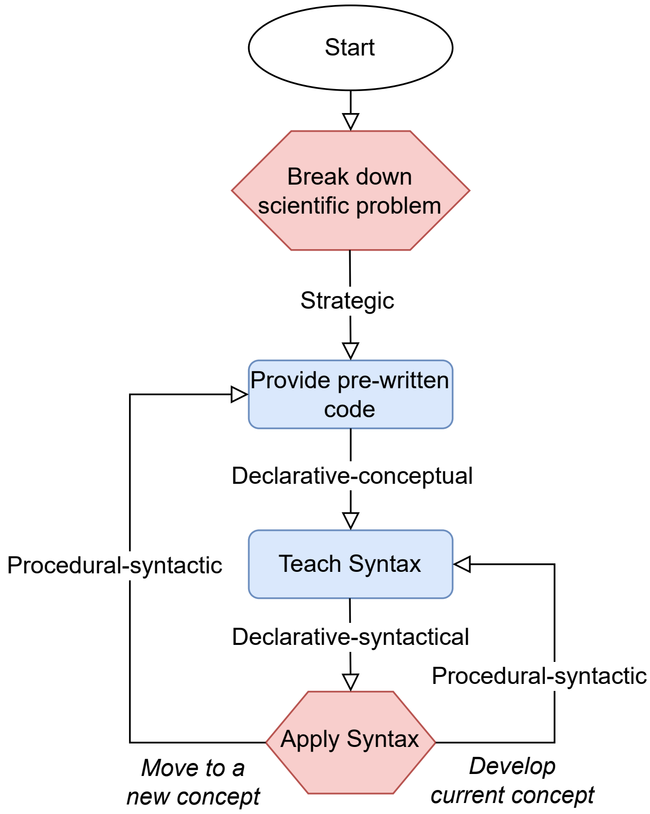

Fig. 2 A schematic representation of the strategy-first approach to programming teaching. Blue rectangles indicate instructor's actions; red hexagons are the student's actions. The lines are labelled with the category of programming knowledge developed following the action preceding it; italic text is used indicate the appropriate path at decision points

students the ability to modify parts of some of these steps to achieve specific goals (introducing syntactic knowledge). This is achieved by providing students with pre-written code which solves a simplified version of the scientific problem, and teaching students how each section of the code solves a particular part of the larger problem (developing declarative-conceptual knowledge). One or more of these sections are then discussed in further detail, describing the function of individual lines of code (teaching declarative-syntactic knowledge), and asking students to modify these lines of code to change the final output of the program (providing procedural-syntactic knowledge).[2] The strategy-first approach allows students to apply programming skills to scientific problems they are already familiar with from their very first exposure to programming teaching.

[2]Procedural-conceptual knowledge is the hardest for students to acquire, and can usually only be learned over the course of multiple teaching sessions; this is true for both syntax-first and strategy-first approaches (Kwon, 2017; McGill & Volet, 1997).

3 Case Studies of Strategy-First Teaching

To illustrate how strategy-first teaching can be implemented in practice, in this section we will describe two workshops designed to follow the approach illustrated in Figure 2. These workshops were delivered to second- and third-year Chemistry students at the author's University. The focus here is how the practice of the workshop aligns with the pedagogical approach outlined in the previous section; as such some details and background are omitted. The interested reader can find all of the workshop materials at the author's github page.

3.1 Case study 1: Atmospheric chemistry

This workshop is designed to help students apply theoretical knowledge recently acquired from a lecture course on atmospheric chemistry to a real-world dataset: hourly measurements of the concentration of three pollutants, NO, NO_2 and O_3, in the centre of Stoke, UK, between 2017 and 2020. This data is publicly available from the UK-AIR database (Department for Environment, Food and Rural Affairs, 2024). The goal of the workshop is to create a series of graphs plotting the concentrations of each of these pollutants, and for students to use their chemical knowledge to interpret the related changes in concentration of each pollutant revealed by those graphs.

At the start of the workshop students are provided with both the complete dataset and a Python script which initially produces a single graph. The students are encouraged to run the code, and to use the code structure and comments, the information provided in the dataset, and the graph produced by the code to deduce what they can about the overall purpose of the code. In doing so, they develop strategic knowledge: they can apply their chemical knowledge to understand the problem in general, and can begin to see how that is broken down into programming tasks (loading data, processing data, plotting a graph).

After obtaining their first graph, students are guided through a series of modifications to the script to produce more complex graphs using different parts of the dataset. At each stage, the same pattern was followed: the instructor identifies the part of the Python script relevant to the modification (teaching declarative-conceptual knowledge), and explains any new syntax the student would need to make these modifications (teaching declarative-syntactical knowledge). The students then make the appropriate modifications to the code, and run it to obtain a new or improved graph (developing procedural syntactic knowledge). Later in the workshop, less input is required from the instructor, as students become more able to identify the parts of the code they need to modify as their declarative-conceptual knowledge develops, and increasingly possess the declarative-syntactical knowledge needed to make the necessary modifications.

In the course of the workshop, students complete the following tasks, with each task introduced and completed following the pattern described above:

1. Identifying lines of code which decorate the plot, and modifying them to include axis labels describing the dependent and independent variables and their units.
2. Adding new data series corresponding to different pollutants to the plots, using existing code as a template.

3. Using a for loop to calculate averaged concentrations of pollutants over a number of days, and plotting the averaged data.
4. Using array indexing to access broader parts of the pollutant dataset, to plot and rationalise the impact of COVID on the concentration of pollutants.

After each task is completed, students are encouraged to consider the chemical implications and explanations for the data they have just plotted. In doing so, the focus of the workshop is always brought back to reinforcing chemistry knowledge they are already familiar with, and demonstrating the value of the programming skills they are learning to their chemical education.

3.2 Case study 2: Hückel theory

This workshop introduces third-year students to Hückel theory (Hückel, 1931; Nagaoka et al., 2018), which they are not expected to have encountered before, as a method of calculating molecular orbital coefficients and energies, which we expect students to have a strong understanding of from previous lecture courses. This workshop is delivered using a Jupyter notebook through Google Colaboratory (Kluyver et al., 2016), allowing the straightforward integration of chemistry and programming teaching notes into a single document; this is common practice in Python-based chemistry instruction (Bravenec & Ward, 2023; Heras-Domingo & Garay-Ruiz, 2024; Menke, 2020; Vallejo et al., 2022; van Staveren, 2022).

The workshop begins by describing the assumptions and mathematics of Hückel theory. Since this is a topic students are less familiar with, the strategic knowledge development, namely how useful chemical information can be obtained from the assumptions of Hückel theory, is more instructor-led than is suggested in Figure 2. However, it is still critical that students understand how the overall problem is broken down into component tasks before beginning the programming portion of the workshop, to enable them to identify sections of code which correspond to each task, and because the component tasks relate more closely to scientific concepts with which they are familiar (in this case molecular orbitals and their energies). Having established in principle the steps which must be taken to solve a problem using Hückel theory, the workshop then moves straight to an application, providing students with code which executes a Hückel theory calculation producing molecular orbital energies and coefficients for a particular molecule, 1,3-butadiene.

The remainder of the workshop follows the same repeating pattern described in Section 3.1. Students are asked to make a series of small modifications to the provided code to achieve some specific goals. To enable them to do this, the instructor first conveys declarative-conceptual knowledge, which in this context primarily means identifying how specific parts of the code correspond to the mathematical formalism of Hückel theory. They then provide declarative-syntactical knowledge, explaining enough syntax to enable students to achieve the set tasks. Students apply this syntax to complete these tasks, in so doing developing their procedural-syntactic knowledge. Following this pattern, students are expected to:

1. Use array indexing to print and manipulate orbital energies of 1,3-butadiene.

2. Use array indexing to print and interpret molecular orbital coefficients of 1,3-butadiene.
3. Use a for loop to calculate energetic properties of a series of linear conjugated hydrocarbons.
4. Create graphs to illustrate the trends in these energetic properties.
5. Extension material: Modify the provided code to produce a function which constructs a Hückel Hamiltonian for cyclic molecules, rather than linear ones.

The emphasis of the workshop remains on the chemical application of the results obtained from programming. For example, following task 2 students are encouraged to think about how their results related to the selectivity of an organic reaction they are familiar with from previous undergraduate lecture courses. This enables them to see the physical reason behind a reactivity "rule" which they have previously been taught, reinforcing the potential value of programming for a chemistry undergraduate.

4 Evaluating strategy-first teaching

The workshops described in the preceding case studies were used to evaluate the effectiveness of strategy-first teaching. To help determine this, students completed a survey measuring their attitudes towards programming and their aptitude in the topics covered in the workshop both immediately before and immediately after the workshop. The two-hour workshops were delivered to one quarter of a year group at a time (between 30 and 40 students) in a dedicated PC classroom, with the relevant software pre-installed on all machines to ensure a uniform programming environment.

These workshops were introduced at a time when programming teaching in the chemistry programme at the author's University was very limited: the students involved in the study had only attended one or two programming workshops in their previous year of study, which had introduced basic mathematical operations and simple data structures in Python in the context of analysing first-order kinetics and predicting isotope patterns of ionic fragments in mass spectrometry. As such, students entered these workshops with a wide range of abilities and interest in programming, depending on their previous educational background. With this in mind, the objectives of the workshops were to:

1. Demonstrate an application of programming relevant to chemistry.
2. Familiarise students with the operation of Python scripts/notebooks.
3. Teach students some basic Python syntax.
4. Enable students to modify Python code informed by their chemical knowledge to produce new outputs.

In turn, the primary objectives of this study are to:

1. Establish whether the pedagogical objectives described above were met.
2. Determine whether a strategy-first approach produces a measurable difference in students' attitudes towards programming.

Three approaches were used to evaluate the impact of these workshops: instructor perceptions from both the author and independent observers, a brief assessment of students' knowledge, and an analysis of the students' perceptions of and attitudes towards the workshop. The majority of survey questions were common to both workshops. The surveys were completed during the workshops, and response rates amongst the third-year students, who took the Hückel theory workshop, were very high, with 74 of the 78 students registered in attendance responding (95%); response rates amongst the second-year students, who took the atmospheric chemistry workshop, were significantly lower, with only 41 responses from 122 attending students (34%). Each question on the survey was optional, resulting in lower response rates for some survey questions.

4.1 Instructor observations

Students' engagement with the chemistry aspects of the workshops was very positive, with a clear interest and desire to progress with the programming activity in order to understand the chemical implications of the output they obtained. In particular, many students experienced a moment of realisation and satisfaction when they connected output from the code they were writing to chemistry they had learned in other lecture courses. In the atmospheric chemistry workshop they were able to interpret real-world data they had plotted using the simple reaction models covered in their atmospheric chemistry lectures, and in the Hückel theory workshop they used molecular orbital coefficients they had calculated to predict selectivity of simple organic reactions. Students' pleasure in making these connections was reflected in a number of their responses when asked in the survey what they enjoyed about the workshop: "writing code can be used to find out information about molecules and applying it to real chemistry examples"; "Learning how to use python in more advanced chemistry"; "I liked learning more about how programming can be applied to chemical contexts".

However, students were often initially reluctant to attempt programming problems, feeling unable to apply the examples of syntax provided to the problem they were attempting to solve. This is a potential weakness of the strategy-first approach: students lack the confidence, and in some cases the conceptual knowledge, to go through the process of trial and error sometimes needed to apply newly introduced syntax to the problem to be solved. After encouragement, their confidence grew throughout the workshop and they made much faster progress; this observation is consistent with students' reported self-perceptions, which are discussed later.

These perceptions were supported by the observing instructors, who provided written feedback on the same day as the workshop, noting that "students were very much engaged", and "even those who seemed relatively disengaged and uninterested at the start of the sessions seemed to get more engaged as the session progressed". One instructor attributed this to the strategy-first approach, noting that the "accessible nature of the material enabled even those who see themselves as 'not being good at computers' to see progress". Another observed the significant "challenge in carrying out the programming task at the same time as understanding chemistry", and that this challenge is often overcome by "effective interaction with students, through asking well-directed and thoughtful questions"; while this will always be a necessary part of programming education, it would be beneficial to improve the workshop design to reduce the need for intensive one-to-one interactions, to allow staff to interact with more students during the workshop.

Table 1 The average and change in students' marks awarded for the programming and chemistry questions, along with the associated Cohen's d value. Marks were given on a scale from zero to four. In the column headings, the values of n refer to the number of responses to the programming and chemistry questions, respectively. Responses which were missing an answer to the post-workshop question were excluded

		Atmospheric chemistry (n=27/26)	Hückel theory (n = 57/56)
Programming	Pre-workshop average	0.59	1.30
	Post-workshop average	1.56	2.39
	Change	0.96	1.08
	Cohen's d	1.09	1.02
Chemistry	Pre-workshop average	2.15	0.77
	Post-workshop average	2.5	2.46
	Change	0.34	1.17
	Cohen's d	0.29	1.15

4.2 Assessing students' knowledge

To assess what students had learned during each workshop, students were asked one question about programming and one about chemistry as part of the survey, with the same question asked both before and after the workshop. To analyse these responses, we subsequently "marked" students' answers between zero and four, where zero indicates no meaningful response is given, four is a perfect answer, and partially correct answers were scored in between. Table 1 summarizes the responses to these questions, with the effect of the workshop described by Cohen's d statistic for paired samples (Cohen, 2013). This reveals a "large" effect ($d > 0.8$)(Sawilowsky, 2009) on students' understanding of both the programming and chemistry content covered in the workshop, with the exception of their understanding of the chemistry content of the atmospheric chemistry workshop. The smaller (but still positive) effect observed here is not surprising, given this workshop related to more familiar chemistry, as evidenced by the high pre-workshop average score for this question. Overall, these results suggest that the workshops are effective in achieving the pedagogical objectives described above.

4.3 Evaluating students' perceptions of programming

Finally, we investigated how students' perceptions of programming were impacted by the workshops, and in particular how gender affected these perceptions. For this analysis, the data from both workshops were combined (n = 115). The gender makeup of the respondents is shown in Figure 3; for clarity and to protect the anonymity of respondents the following histograms exclude non-binary respondents or those who preferred not to provide information about their gender, due to the small sample sizes from these groups.

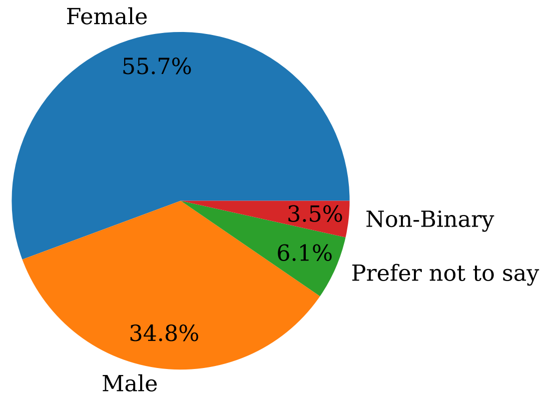

Fig. 3 The gender make-up of the respondents to the survey across all workshops (n = 115)

Students were asked to score their feelings towards using Python as a tool before and after the workshop on a seven-point Likert scale, with one corresponding to "Confident" and seven corresponding to "Apprehensive" (Figure 4). Prior to the workshop, participants were apprehensive, with the median score of female students being six and the median score of male students being five. After the workshop, students were significantly more confident, with the median score of both male and female students decreasing to four; the mean response fell from 4.95 to 4.06, indicating a significant increase in the confidence of the students after the workshop ($p < 10^{-5}$).

Interestingly, this shift was much more pronounced amongst female students: prior to the workshop, female students were much more likely to respond with a score of six or seven than male students, whereas after the workshop this trend was reversed. One possible explanation for this observation is that female students were much more likely than male students to strongly agree with the statement "I was able to connect today's workshop to chemistry topics I have studied before" (Figure 5). This aligns with one motivation for introducing strategy-first programming teaching for science undergraduates - students' motivation to learn programming, and their confidence in their ability to write code, is enhanced when they can clearly see the value of programming to the rest of their undergraduate programme.

5 Conclusions

In this work we have introduced an alternative approach to programming instruction for science undergraduates. This strategy-first approach begins not from the fundamentals of programming syntax, but from working code which tackles a scientific problem familiar to undergraduate students from other parts of their undergraduate studies, and breaks it down into programming concepts and then specific syntactic knowledge which students need to use to modify the code to achieve the goals of the workshop. We have provided a template of a pedagogical approach too rarely adopted

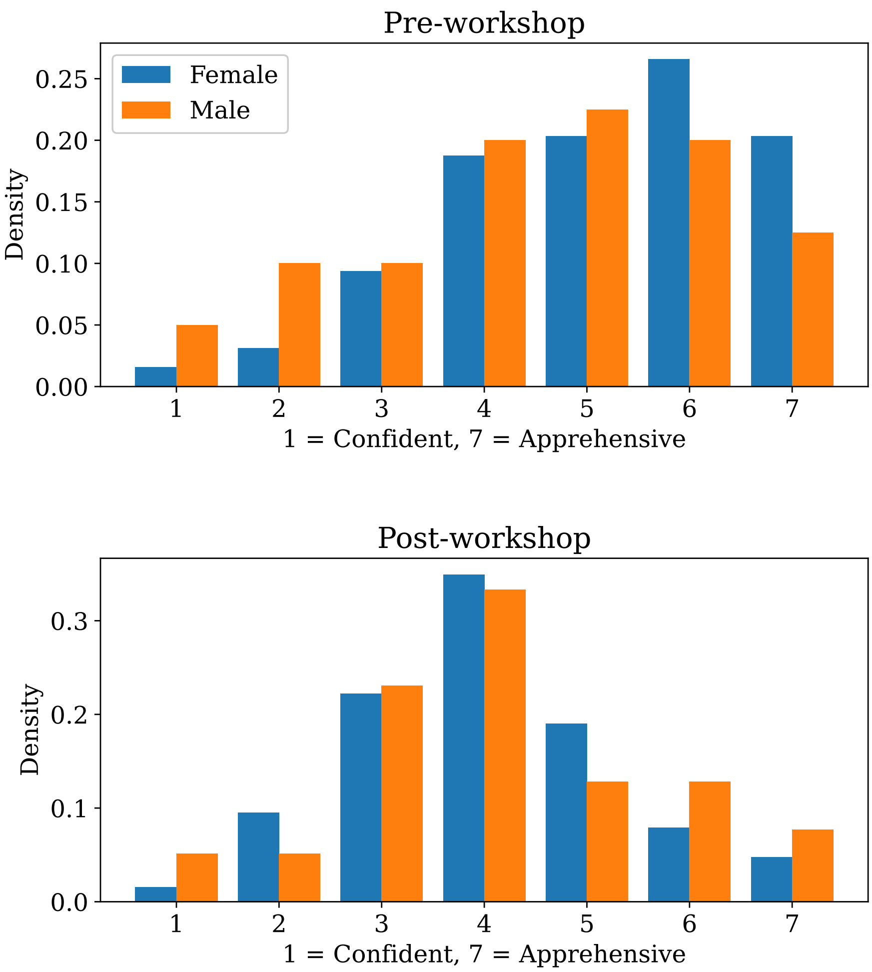

Fig. 4 Histograms showing the response of students to the question "How would you describe your feelings about using Python as a tool?" on a seven-point Likert scale, with one corresponding to "Confident" and seven corresponding to "Apprehensive". Above: responses collected before the workshop. Below: responses collected after the workshop. In total, there are 64 female respondents, and 40 male respondents

in the science education literature; even those workshops which do follow a problem-oriented structure are usually not intended for use by programming novices (De Haan et al., 2021; Hirschi et al., 2023; Menke, 2020), notwithstanding a small number of exceptions (Valle & Berdanier, 2012). The assumption that participants already have syntactic knowledge excludes these studies from being truly strategy-first approaches. We hope that the outline and cases studies of strategy-first teaching described here will equip other educators to adopt this approach.

We have provided evidence from student surveys that the strategy-first workshops introduced here are effective in educating students about both programming

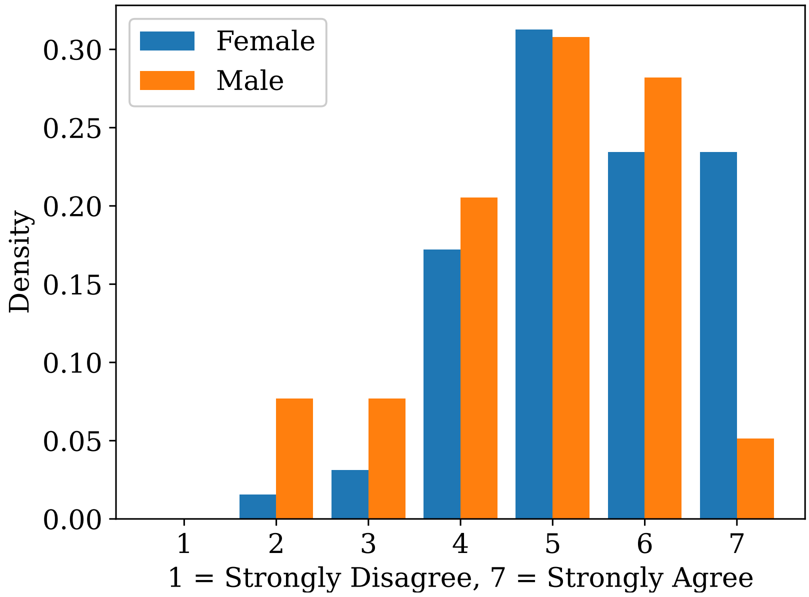

Fig. 5 Histogram showing the agreement of students to the statement "I was able to connect today's workshop to chemistry topics I have studied before" on a seven-point Likert scale, with one corresponding to "Strongly Disagree" and seven corresponding to "Strongly Agree". In total, there are 64 female respondents, and 40 male respondents

and chemistry, and in developing students' confidence with programming and engaging them with the workshops. Students highlighted being able to make connections between programming and chemistry as a part of the workshop they particularly enjoyed, which is one of the key goals of the strategy-first approach, and stands in direct contrast to traditional syntax-first approaches to introductory programming teaching. We found that female students in particular responded well to this style of workshop, with female students being more likely than males students to describe themselves as apprehensive about programming before the workshop, but less likely to do so after the workshop. While much has been written about the gendered attitudes towards programming (Koch et al., 2008; Mishkin, 2019), it appears that very little evidence has been collected about gendered perceptions of programming education specifically in a science context. Therefore, future work will investigate in more detail the gendered impact of the strategy-first approach to programming-based chemistry education. In addition, further strategy-first workshops will be developed; for example, this approach will enable us to introduce machine learning concepts to students with limited programming experience.

6 Declarations

6.1 Availability of data and materials

All of the teaching materials and survey questions described in this article can be found at the author's github page. Survey data is not publicly available to protect the anonymity of respondents; queries about the survey data should be directed to the corresponding author.

6.2 Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Anderson, J.R. (1982). Acquisition of Cognitive Skill. *Psychological Review*, *89*(4), 369–406, https://doi.org/10.1037/0033-295x.89.4.369

Bain, K., Rodriguez, J.-M.G., Moon, A., Towns, M.H. (2018, April). The characterization of cognitive processes involved in chemical kinetics using a blended processing framework. *Chemistry Education Research and Practice*, *19*(2), 617–628, https://doi.org/10.1039/C7RP00230K

Bayman, P., & Mayer, R.E. (1988). Using Conceptual Models to Teach BASIC Computer Programming. *Journal of Educational Psychology*, *80*(3), 291–298, https://doi.org/10.1037/0022-0663.80.3.291

Bravenec, A.D., & Ward, K.D. (2023, February). Interactive Python Notebooks for Physical Chemistry. *Journal of Chemical Education*, *100*(2), 933–940, https://doi.org/10.1021/acs.jchemed.2c00665

Bright Network Research (2023). *What do graduates want?* Retrieved 2025-01-14, from https://employers.brightnetwork.co.uk/sites/default/files/2024-02/BN%20Research%20Report%202023%20digital.pdf

Cawthorne, L. (2021, October). Invited viewpoint: Teaching programming to students in physical sciences and engineering. *Journal of Materials Science*, *56*(29), 16183–16194, https://doi.org/10.1007/s10853-021-06368-1

Cohen, J. (2013). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). New York: Routledge.

Cumby, J., Degiacomi, M.T., Erastova, V., Güven, J.J., Hobday, C.L., Mey, A.S.J.S., . . . Szabla, R. (2023, May). Course Materials for an Introduction to Data-Driven Chemistry. *Journal of Open Source Education*, *6*(63), 192, https://doi.org/10.21105/jose.00192

De Haan, D.O., Schafer, J.A., Gillette, E.I. (2021, October). Using a Modular Approach to Introduce Python Coding to Support Existing Course Learning Outcomes in a Lower Division Analytical Chemistry Course. *Journal of Chemical Education*, *98*(10), 3245–3250, https://doi.org/10.1021/acs.jchemed.1c00456

Department for Environment, Food and Rural Affairs (2024). *Data Archive - Defra, UK.* Retrieved 2025-01-08, from https://uk-air.defra.gov.uk/data/

Eurostat (2023). *ICT specialists - statistics on hard-to-fill vacancies in enterprises.* Retrieved 2025-04-16, from https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT_specialists_-_statistics_on_hard-to-fill_vacancies_in_enterprises

Fuchs, W., McDonald, A.R., Gautam, A., Kazerouni, A.M. (2024, August). Recommendations for Improving End-User Programming Education: A Case Study with Undergraduate Chemistry Students. *Journal of Chemical Education*, *101*(8), 3085–3096, https://doi.org/10.1021/acs.jchemed.4c00219

Gagné, E.D., Yekovich, C.W., Yekovich, F.R. (1993). *The Cognitive Psychology of School Learning.* HarperCollins College Publishers.

Guzman, L.M., Pennell, M.W., Nikelski, E., Srivastava, D.S. (2019, December). Successful Integration of Data Science in Undergraduate Biostatistics Courses Using Cognitive Load Theory. *CBE—Life Sciences Education*, *18*(4), ar49, https://doi.org/10.1187/cbe.19-02-0041

Hambrusch, S., Hoffmann, C., Korb, J.T., Haugan, M., Hosking, A.L. (2009, March). A multidisciplinary approach towards computational thinking for science majors. *SIGCSE Bull.*, *41*(1), 183–187, https://doi.org/10.1145/1539024.1508931

Haraldsrud, A., & Odden, T.O.B. (2023, May). From Integrated Rate Laws to Integrating Rate Laws: Computation as a Conceptual Catalyst. *Journal of Chemical Education*, *100*(5), 1739–1750, https://doi.org/10.1021/acs.jchemed.2c00881

Heras-Domingo, J., & Garay-Ruiz, D. (2024, November). Pythonic Chemistry: The Beginner's Guide to Digital Chemistry. *Journal of Chemical Education*, *101*(11), 4883–4891, https://doi.org/10.1021/acs.jchemed.4c00840

Hirschi, J.S., Bashirova, D., Zuehlsdorff, T.J. (2023, November). Opening the Density Functional Theory Black Box: A Collection of Pedagogic Jupyter Notebooks. *Journal of Chemical Education*, *100*(11), 4496–4503, https://doi.org/10.1021/acs.jchemed.3c00535

Hooley, T. (2021). *A mixed bag: Employer perspectives on graduate skills.* Retrieved 2025-01-14, from https://luminate.prospects.ac.uk/a-mixed-bag-employer-perspectives-on-graduate-skills

Hückel, E. (1931, March). Quantentheoretische Beiträge zum Benzolproblem. *Zeitschrift für Physik*, *70*(3), 204–286, https://doi.org/10.1007/BF01339530

Kluyver, T., Ragan-Kelley, B., Perez, F., Granger, B., Bussonnier, M., Frederic, J., . . . Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (pp. 87–90). IOS Press.

Koch, S.C., Müller, S.M., Sieverding, M. (2008, December). Women and computers. Effects of stereotype threat on attribution of failure. *Computers & Education*, *51*(4), 1795–1803, https://doi.org/10.1016/j.compedu.2008.05.007

Kwon, K. (2017, October). Novice programmers' misconception of programming reflected on problem-solving plans. *International Journal of Computer Science Education in Schools*, *1*(4), 14–24, https://doi.org/10.21585/ijcses.v1i4.19

Lee, H.-Y., Lin, C.-J., Wang, W.-S., Chang, W.-C., Huang, Y.-M. (2023, August). Precision education via timely intervention in K-12 computer programming course to enhance programming skill and affective-domain learning objectives. *International Journal of STEM Education*, *10*(1), 52, https://doi.org/10.1186/s40594-023-00444-5

Linn, M.C., & Clancy, M.J. (1992, March). The case for case studies of programming problems. *Communications of the ACM*, *35*(3), 121–132, https://doi.org/10.1145/131295.131301

Margulieux, L.E., Morrison, B.B., Decker, A. (2020, May). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, *7*(1), 19, https://doi.org/10.1186/

s40594-020-00222-7

McGill, T.J., & Volet, S.E. (1997, March). A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education*, *29*(3), 276–297,

Mears, M., Dash, L., Galloway, R., Karpenko, C., Labrosse, N., Mason, V., Quinn, M. (2025, April). Mixed-Methods Study of First-Year Physics Students: Soft Barriers to Coding. *Journal of Science Education and Technology*, *34*(2), 420–435, https://doi.org/10.1007/s10956-025-10198-0

Mecca, G., Santoro, D., Sileno, N., Veltri, E. (2021, March). Diogene-CT: Tools and methodologies for teaching and learning coding. *International Journal of Educational Technology in Higher Education*, *18*(1), 12, https://doi.org/10.1186/s41239-021-00246-1

Menke, E.J. (2020, October). Series of Jupyter Notebooks Using Python for an Analytical Chemistry Course. *Journal of Chemical Education*, *97*(10), 3899–3903, https://doi.org/10.1021/acs.jchemed.9b01131

Mishkin, A. (2019, February). Applying Self-Determination Theory towards Motivating Young Women in Computer Science. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 1025–1031). New York, NY, USA: Association for Computing Machinery.

Nagaoka, S.-i., Kokubo, T., Teramae, H., Nagashima, U. (2018, September). Practical Training in Simple Hückel Theory: Matrix Diagonalization for Highly Symmetric Molecules and Visualization of Molecular Orbitals. *Journal of Chemical Education*, *95*(9), 1579–1586, https://doi.org/10.1021/acs.jchemed.8b00244

Oliver, R., & Malone, J. (1993, June). The Influence of Instruction and Activity on the Development of Semantic Programming Knowledge. *Journal of Research on Computing in Education*, *25*(4), 521–533, https://doi.org/10.1080/08886504.1993.10782071

Redish, E.F., & Kuo, E. (2015, July). Language of Physics, Language of Math: Disciplinary Culture and Dynamic Epistemology. *Science & Education*, *24*(5), 561–590, https://doi.org/10.1007/s11191-015-9749-7

Ringer McDonald, A. (2021, July). Teaching programming across the chemistry curriculum: A revolution or a revival? *Teaching programming across the chemistry curriculum* (Vol. 1387, pp. 1–11). American Chemical Society.

Sawilowsky, S.S. (2009, November). New Effect Size Rules of Thumb. *Journal of Modern Applied Statistical Methods*, *8*, 597–599, https://doi.org/10.56801/10.56801/v8.i.452

Sherin, B.L. (2001, December). How Students Understand Physics Equations. *Cognition and Instruction*, *19*(4), 479–541, https://doi.org/10.1207/S1532690XCI1904_3

Snow, R.E., & Lohman, D.F. (1984). Toward a theory of cognitive aptitude for learning from instruction. *Journal of Educational Psychology*, *76*(3), 347–376, https://doi.org/10.1037/0022-0663.76.3.347

Sun, D., Boudouaia, A., Zhu, C., Li, Y. (2024, February). Would ChatGPT-facilitated programming mode impact college students' programming behaviors, performances, and perceptions? An empirical study. *International Journal of Educational Technology in Higher Education*, *21*(1), 14, https://doi.org/10.1186/s41239-024-00446-5

Sun, D., Ouyang, F., Li, Y., Zhu, C. (2021, September). Comparing learners' knowledge, behaviors, and attitudes between two instructional modes of computer programming in secondary education. *International Journal of STEM Education*, *8*(1), 54, https://doi.org/10.1186/s40594-021-00311-1

Valle, D., & Berdanier, A. (2012). Computer Programming Skills for Environmental Sciences. *The Bulletin of the Ecological Society of America*, *93*(4), 373–389, https://doi.org/10.1890/0012-9623-93.4.373

Vallejo, W., Díaz-Uribe, C., Fajardo, C. (2022, March). Google Colab and Virtual Simulations: Practical e-Learning Tools to Support the Teaching of Thermodynamics and to Introduce Coding to Students. *ACS Omega*, *7*(8), 7421–7429, https://doi.org/10.1021/acsomega.2c00362

van Staveren, M. (2022, July). Integrating Python into a Physical Chemistry Lab. *Journal of Chemical Education*, *99*(7), 2604–2609, https://doi.org/10.1021/acs.jchemed.2c00193

Volet, S.E., & Lund, C.P. (1994, June). Metacognitive Instruction in Introductory Computer Programming: A Better Explanatory Construct for Performance than Traditional Factors. *Journal of Educational Computing Research*, *10*(4), 297–328, https://doi.org/10.2190/9A08-Y2Q0-6AER-6KLQ

Weiss, C.J. (2017, September). Perspectives: Teaching chemists to code. *C&EN Global Enterprise*, *95*(35), 30–31, https://doi.org/10.1021/cen-09535-scitech2

World Economic Forum (2025). *The Future of Jobs Report 2025.* Retrieved 2025-04-16, from https://www.weforum.org/publications/the-future-of-jobs-report-2025/in-full/3-skills-outlook/