Developing and Supporting Computational Learning Objectives in Lower Division Science Courses

Saturnino Garcia - Associate Professor of Computer Science - University of San Diego
Eleanor Gillette - Assistant Professor of Chemistry - University of San Diego

Over the past decade computer science has seen a surge in popularity as computation has grown to impact nearly every aspect of our lives. Computational thinking is now widely viewed as a fundamental skill across many disciplines, from the natural sciences to business and beyond. Unfortunately, universities have struggled to keep up with the demand for computer science and those that do persist through a CS major are much less diverse and not representative of society as a whole.

In this study, we looked at the feasibility of integrating computational topics into discipline specific courses (e.g. chemistry or biology), which we refer to as the X^{CS} approach. Specifically, we looked to (1) develop a curricular map that could guide those considering this approach (as compared to a *CS+X* approach, which requires students to take full courses in both CS and another discipline) and, (2) to identify the major barriers to making a transition to X^{CS}. We believe that the X^{CS} approach will increase the capacity to train students in critical computational skills, while also leading to more diversity within computing by building computational self-efficacy in more diverse disciplines.

Curricular Maps for Computing In Discipline Specific Courses

We started the development of a curricular map by looking at the intersection of two of our 14-week courses, one a Python-based, depth-first CS1 course and the other a lower division analytical chemistry course that had recently integrated Python into their lab to aid with quantitative analysis.

Our analysis of the learning objectives in these two courses identified six key computational topics that could form the basis of a computational component in a discipline specific course. Below we list these topics and overview the level to which they are covered in both courses.

1. Data Types:

- **CS 1:** Students progress from simple numeric data types and booleans through collections of data (strings, lists, tuples, and dictionaries), including two-dimensional data (e.g. lists of lists).
- **Chemistry Implementation:** Students are introduced to the concepts of integers and floats when working with simple data, and use lists, arrays, and matrices to organize more complex data.

2. Conditionals:

- **CS 1:** Students progress from basic if-then conditionals, to chained conditionals, and finally nested conditional statements.
- **Chemistry Implementation:** Students use if-then statements for simple statistical tests (eg. F and t tests) as well as for making decisions during calculations (eg. pH and titration calculations).

3. Iteration:

- **CS 1:** Students progress from definite iteration using for loops using the range function, to iterating through collections by index or item, and then to indefinite iteration using while loops. Students are also introduced to recursion as another form of iteration.
- **Chemistry Implementation:** Students are introduced to for loops to automate simple statistical calculations (e.g. standard deviation), and then use nested for-loops to process large data sets.

4. File Reading/Writing:

- **CS 1:** Students progress from reading through a whole file line-by-line, to processing structured data files (e.g. CSV format), and finally to reading only parts of a file. Students also learn how to write data to a file rather than printing it to the screen.
- **Chemistry Implementation:** Students read in data from .csv or .txt files and export graphs as .png files, using built-in functions.

5. Pattern Matching:

- **CS 1:** Students progress from identifying/using a basic pattern (the accumulator pattern) to seeing more complicated patterns (e.g. filtering items from a list) and finally learning to combine multiple patterns to solve a problem.
- **Chemistry Implementation:** Students develop strategies for converting mathematical processes that they have worked by hand (e.g. pH calculations) into increasingly automated sections of code.

6. Reliability/Design:

- **CS1:** Students learn about using functional composition to break down complex problems into smaller parts and write functions to allow for code reuse and automated testing of code.
- **Chemistry Implementation:** Students learn to use print statements to check their work periodically during extended calculations.

While the depth at which the chemistry course covers any of these topics does not reach that of the CS1 course, this level of integration did not require sacrificing any of the course's chemistry learning objectives. In many cases, introducing a computational topic was simply a matter of updating existing tools, e.g. exchanging Excel with Python to compute linear regressions. In areas where it is more difficult to dedicate time to details of the CS topic (e.g. file reading and writing), students are still afforded a minimal introduction to the topic through the use of Python modules or built-in functions. This provides a conceptual foundation for later courses that may have specialized computational needs (e.g. dealing with irregular data file formats). We believe this to be a realistic model for how computing concepts can be integrated into a lower division science course.

Recommendation: Integrating Computation at Multiple Levels

Covering all six computational areas at the same depth as a CS1 course is not a realistic expectation for a discipline specific course, especially at the lower division. Instead, we believe that disciplines should consider a strategy where one of their foundational, lower division courses introduces computation while one or more advanced courses further develop students computational skills.

The foundational course should provide an introduction to all six of the computational areas with deeper examination of at least a few of the topics. This model provides flexibility for specific disciplinary courses to decide what areas to focus on based on their own content. For example, one discipline may have numerous examples where a complex set of decisions need to be made, calling for a further emphasis on chained and nested conditionals than we found in the analytical chemistry course. We believe this type of course is appropriate at the lower division level for majors and non-majors alike in these discipline specific courses.

Conversely, the advanced courses should assume students have completed a foundational course and therefore have *at least* a minimal understanding of all six computational areas. This advanced course should then aim to reach the same level of coverage of a CS1 course in at least half of the areas. This type of course would most likely be an upper division course but the computational components do not require any highly specialized background (e.g. upper division mathematics) so it could also be situated at the lower division, as a follow-up to the basic course.

Challenges to Implementation

Our second goal was to better understand the barriers which prevent further integration of computer science concepts into lower division science courses, given both the observed overlap of applicable concepts and the desire in most fields to strengthen computational skills within undergraduate education.

To better understand the current status of computation in science courses at USD, we surveyed our colleagues in Physics (n=2) and Biology (n=1). All three faculty members agreed with the assertion that computation is a topic best introduced in lower division courses. However, all three also acknowledge that computational topics are currently scattered through upper division courses and research

experiences in their majors and are rarely addressed at the lower division. The survey further indicated that among several possible barriers to introducing computational topics, lack of flexibility in lower division courses was most commonly cited as a major barrier. In each case, faculty did not feel that it was their expertise that was limiting their ability to implement computer science concepts at the levels we were describing in our curricular map. While this is a very small snapshot of the situation in a single institution, it suggests that the pressure on our lower chemistry course to incorporate computation only if it can maintain existing learning objectives is not unique to chemistry.

These disciplines' lower division courses are often highly structured 'service courses' which must cover content which supports students in a variety of majors and who are using the courses as preparation for standardized exams like the MCAT, DAT or PCAT. If computational skills are going to be integrated into the lower division in these disciplines, they must directly support existing learning objectives in the courses. As we show in the curricular map, it is possible to address all six key CS areas without any modifications to the existing curriculum of a course which already contains quantitative concepts like statistical tests, graphing and data analysis. These courses cannot replace introductory CS courses as prerequisites for upper division CS coursework. However, these courses may be appropriate prerequisites for upper division courses within their own disciplines which can build off of these concepts to achieve a more comprehensive introduction to computer science. Given that our colleagues in Biology and Physics as well as in Chemistry are already trying to implement computer science in their upper division courses, or through undergraduate research experiences, this provides further support for the model of discipline specific scaffolding of computer science throughout at least two levels of the curriculum. However, this requires support for this scaffolding, especially at the foundational level.

Recommendation: Supporting Adoption of X^{CS}

Based on the authors' experience and feedback received from a survey of several colleagues, it became clear that the major impediment to introducing computing topics into lower division, discipline specific courses is curricular inflexibility in these courses. Overcoming this impediment will require a multi-faceted approach.

First, we recommend the development of curricular case studies that examine how an existing discipline specific course introduced computing into their course. The focus of these case studies will be on the decision making process, including which stakeholders were involved in the decision and the criteria that guided which existing content could be changed or removed to allow for the introduction of computational topics. These case studies can serve as a template for community members to help guide them at their own institutions.

Second, we recommend identifying key areas that are common in lower division science courses and creating small scale examples of how they can be used to support teaching of computational topics. Common areas we have already identified are simple statistical calculations (e.g. t tests), data visualization, and simple data processing. Instructors can readily adopt these examples for their class, similarly to how many instructors adopt textbook questions as homework. By keeping these examples short and tied to basic quantitative concepts, they can be adapted to many contexts without extensive modification.

Finally, we recommend that guided inquiry learning be used as the predominant model for development of discipline-specific computational exercises. Guided inquiry is well known within the science education community and has recently gained traction within computer science, providing a natural crossover point. The CS-POGIL project already has many peer-reviewed, guided inquiry examples for introductory level computational topics, including those for a scientific computing course. These could be used as a starting point to develop a limited set of exercises that are tailored to specific disciplines.