CS-1 for Scientists

Panel Proposal

Greg Wilson*
Dept. of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 2E4
gvwilson@cs.toronto.edu

Categories and Subject Descriptors

K.3 [**K.3.2**]: Computer and Information Science Education

General Terms

Design

Keywords

CS-1, Computational Science

1. PROPOSAL

Students in science and engineering are poorly served by most general-purpose CS-1 courses, which rarely discuss scientific problems or applications. At the same time, fewer and fewer computer science students are exposed to scientific ideas or thinking in any of their introductory courses. This divide hurts both sides: scientists and engineers must struggle later in their careers to pick up the computing skills and mindset they need to cope with increasingly computational disciplines, while CS graduates lack the background knowledge needed to work in "relevant" domains ranging from health care to climate modeling.

The aim of this panel is to explore what each community thinks it and its students need, and to discuss what is being done to meet those needs at leading institutions. Topics will include:

1. What "CS-1 for Scientists" courses ought to look like, and how (or whether) this differs from the standard CS-1 curriculum.

2. How to teach computational thinking [1] to science students.

3. How the scientific approach to problem solving can be incorporated into mainstream computer science courses, and what the benefits of doing this are.

4. How to present scientific topics and examples to mainstream computer science students, many of whom are unfamiliar with advanced mathematics.

5. What effect such changes might have on enrollments in affected disciplines.

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.
ACM 978-1-59593-947-0/08/0003.

2. PANELISTS

Christine Alvarado, Assistant Professor, Computer Science Department, Harvey Mudd College, Claremont CA

Harvey Mudd College, a small liberal arts college focused on science and engineering, requires all of its students to take a single introductory computer science course in the fall of their freshman year. Thus approximately 85% of the students in this course will go on to study a scientific discipline other than computer science.

In 2005 we redesigned our traditional Java-programming introductory course to better serve the needs of our broad student body. We aimed to present a broad, representative view of the field of computer science, and to allow students to develop programming skills that would be both sufficient to continue on to a CS2 course and useful for any scientific field of study.

Our new course, which completed its second offering in the fall of 2007, takes a breadth-first approach comprising five major modules: functional programming, digital logic and circuit design, imperative programming, object-oriented programming, and concepts in computability. The emphasis is on conveying major concepts rather than training "industrial strength" programmers. We use Python for the functional, imperative and object-oriented parts of the course because Python is simple enough for an introductory course, versatile enough to illustrate a broad range of programming paradigms, and powerful enough to be used by scientists in many disciplines. Our weekly programming assignments offer a broad range of often real-world scientific problems. Examples include programs to simulate biological systems, programs to explore the Collatz problem, and a final project to build a virtual pool game, including modeling the physical interactions of the balls. To excite a broad range of students, we often allow students to select a subset of problems they find the most interesting.

Although we are still gathering and analyzing data, initial assessment suggests that, compared to students completing the previous offering of the course, students completing this course have a better understanding of the field of computer science, are equally proficient or superior programmers, and

have a better understanding of the relationship of computer science to other scientific disciplines.

Jennifer Campbell, Lecturer, Dept. of Computer Science, University of Toronto, Toronto ON

We recently introduced a first-year course aimed at science students. Our goals for this course include teaching enough programming and general technical material to demonstrate to students the practical value of CS in their scientific discipline, and introducing the field of CS to a larger number of science students to give them a path into, and possibly attract them to, our programs.

The course content overlaps significantly with traditional CS1 courses, but we also introduce atypical topics like databases and 2D plotting. All examples from the course are scientific, which gives students a sense of how the tools and skills they learn can be applied to their disciplines.

By creating a course that is specifically for students in the sciences, we are making computer science more accessible to students in other disciplines. Rather than take a course with computer science majors, the students can sign up for a course that seems more relevant to them. For students who take this course and then decide to become CS majors, there is a suitable follow-up course that allows them to enter second year with the same number of first-year CS courses as a typical CS major.

We have already seen positive signs that we are achieving our goals to some degree. Several students have expressed interest in learning more computer science and least one student applied his newly attained knowledge while working at his summer research position in chemistry. As an interesting side note, the percentage of women enrolled was 37%.

Rubin Landau, Professor, Dept. of Physics, Oregon State University, Corvallis OR

The first and second Scientific Computing courses at Oregon State are taken by freshmen and aim to give them a background in the computing skills and tools they can use throughout their college careers. Three subsequent courses are taken by all computational physics students, and many others as well (including some graduate students). All courses are project-oriented, and cover over 100 scientific problems.

Our experience is that science and engineering students learn computing best by first applying it to concrete science problems (our courses) and then learning the formal theory of computing (CS courses). We also find it important for science students to understand the basis of floating-point computations, the concordant round-off and algorithmic errors, and the use of large arrays of data. Although much scientific computation is done with problem-solving environments, programming remains a valuable skill for students. While many departments do offer courses in computation for their particular disciplines, there still is value in general courses that teach the theory underlying computing.

Robert Sedgewick, Professor, Dept. of Computer Science, Princeton University, Princeton NJ

Before the advent of computing, a standard science and mathematics curriculum was developed in secondary education, supported and expanded by first-year college courses, that serves as the technical basis for every student in science and engineering. For whatever reason, we have seen two unfortunate developments at the dawn of the new millenium. First, many computer science students have somehow been exempted from having to know basic precepts of science and math (instead, they learn to develop large programs). Second, many students in science and engineering have somehow been exempted from being exposed to basic precepts of computer science (instead, they learn specific programming tools). In both cases, students are being shortchanged. One solution to both of these problems is to integrate computer science into the standard curriculum, so that all students with a general interest in science and engineering develop a common technical basis in math, science, and computing before moving into discipline-specific studies. Providing such integration early in the curriculum is easier than it might seem.

3. REFERENCES

[1] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3), March 2006.