Teaching Computational Thinking to Science Majors *

Susanne Hambrusch
Dept. of Computer Sciences
Purdue University
W. Lafayette, IN 47907, USA
seh@cs.purdue.edu

Christoph Hoffmann
Dept. of Computer Sciences
Purdue University
W. Lafayette, IN 47907, USA
cmh@cs.purdue.edu

John T. Korb
Dept. of Computer Sciences
Purdue University
W. Lafayette, IN 47907, USA
jtk@cs.purdue.edu

Mark Haugan
Department of Physics
Purdue University
W. Lafayette, IN 47907, USA
mph@physics.purdue.edu

ABSTRACT

This paper describes the development and initial evaluation of a new course ”Introduction to Computational Thinking” taken by science majors to fulfill a college computing requirement. The course was developed by computer science faculty in collaboration with science faculty and it focuses on the role of computing and computational principles in scientific inquiry. It uses Python and Python libraries to teach computational thinking via basic programming concepts, data management concepts, simulation, and visualization. Projects drawn from different scientific disciplines are complemented with lectures from faculty in these areas. Our initial evaluation indicates that the problem-driven approach focused on scientific discovery and computational principles increases the student's interest in computing.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation

Keywords

Computational thinking, curriculum, multi-disciplinary, computing for scientists.

1. INTRODUCTION

Scientific research is becoming unthinkable without computing. The ubiquity of computerized instrumentation and detailed simulations generates scientific data in volumes that no longer can be understood without computation. For example, in high-energy Physics, the Large Hadron Collider (LHC) will soon generate data at a rate of 0.1 - 1 GB a second, accumulating about 8 PB of data a year in search of the Higgs boson. If one is found, it is expected to manifest only on 1 in 1013 recorded collision events [10]. Much of today's scientific research is computational in nature, evaluating scientific models by detailed simulations and generating data volumes that are often larger than can be analyzed in entirety [1]. In light of this evolution of science, future generations of scientists have to engage computing and have to understand what computer science can do for their work, much as they have to understand what mathematics already does for their work.

This paper describes a multi-disciplinary effort developing a course on computational thinking for science majors. At Purdue, science undergraduates have to fulfill a computing requirement, which is generally done by taking a CS course[2]. The new course was developed by CS in collaboration with faculty in Physics, Biology, Chemistry, and Statistics [4]. It uses a problem-driven approach allowing a focuses on scientific discovery through computational methods as well as computer science principles.

Related work. Introductory courses with a focus on science students have been described in recent papers, including [7, 8, 13, 11]. The concept of computational thinking introduces by Wing in [12] plays an important part in many of such new courses [9].

2. A COURSE FOR SCIENCE MAJORS

The principles underlying the course were developed by CS faculty with experience in teaching introductory courses in collaboration with science faculty. One simple principle we followed is that examples given should be couched in a language that is familiar to the student. A physics major would appreciate examples from mechanics using dimensional units, a chemistry major might be comfortable with balancing chemical formulae as example. After those examples have been understood, the student can proceed to abstracting the underlying formal structures. Similarly, it is our thesis that science majors, conversant in the basics of the classical disciplines, may apprehend computational concepts easier if those concepts can be motivated by ex-

*Work supported in part by National Science Foundation under Grant No. CCF-0722210.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE '09 Chattanooga, Tennessee, USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

amples from science. A second important principle was to use a language that allows students to quickly write meaningful programs and comes with useful libraries, and which is used by the scientific community. Not surprisingly, we used Python. A third principle was to teach in a problem-driven way. For example, viewing a thermodynamic system from the computational perspective, as opposed to a purely descriptive view, naturally led to randomized models and Monte Carlo techniques. This, in turn, motivated pseudo random number generation algorithms, discussions of where else Monte Carlo methods arise, and what the limitations of this computational paradigm are.

Three areas, physics, chemistry, and bioinformntics gave expectations on what they wanted students to learn:

- The Physics department uses the approach developed by Chabay and Sherwood in an introductory calculus-based physics course [5, 6]. In the lab, students run and modify Python programs to model and visualize mechanical systems and fields in 3D using VPython [3]. Teaching programming or computational principles is beyond the scope of the physics course. Physics faculty were interested in having their students take a CS course that could lead to a better integration with computation.
- The computational chemistry faculty are interested in students learning computational methods relevant in chemical research, in particular Monte Carlo and Simulated Annealing. In addition, being able to use and integrate existing Fortran programs was viewed as important.
- For the area of bioinformatics and statistical computing, there was an interest in teaching the use of R for statistical computing and visualization, followed by learning to program in a language for which bioinformatics software packages exist or can easily be integrated.

The following describes the material covered in the 15-week course which uses a two 1-hour lectures and one 2-hour lab per week format.

I. Basic Programming Tools (6 weeks)

- Introduction to Python. Elementary values and data types.
- Straight line programs, assignments to variables, type conversion, math library.
- Strings, lists, and tuples. Vectors and arrays.
- Conditionals and loop structures.
- Introduction to 3D visualization in VPython.
- Functions, parameters and scope. Recursion.

II. Computational Tools and Methods (6 weeks)

- Basic plotting using MatPlotLib and VPython.
- Arithmetic and random numbers. Using NumPy. Examples of numerical stability and problem stability.
- Introduction to simulations and Monte Carlo methods.
- Computational Physics: Ideal gas and Ising Spin simulations.
- Trees as a data structure, traversal and exploration.
- Introduction to graphs, graph operations using NetworkX, graphs in science applications.
- Bioinformatics: Analyzing protein interactions. Visualizing large graphs using Cytoscape
- Grand challenges in scientific computing

III. Looking Under the Hood at Computer Science (3 weeks)

- Object-oriented design. Use and design of classes, OO concepts. Dictionaries and spatial queries as examples
- History of computer science.
- Limits of computing, intractability, computability.
- Future models of computation: DNA computing, quantum computing.

Python was chosen because of its interactive environment, its ability to let novice programmers quickly write meaningful code, its adoption by many scientific communities, and the availability of numerous libraries. Python can be executed efficiently, making it a good vehicle not only for small-scale experimentation, but also for larger data sets and longer computational problems. The course included teaching basics of object-oriented code development. We found that this subject was quite natural to apprehend towards the end of the semester, after programming in Python had become fluent. On the other hand, recursion was considered challenging by the students.

Visualization is an important element in computing and brings many quantitative scientific facts to life. VPython and MatPlotLib were introduced early. VPython allows creating sophisticated 3D visuals without having to learn the complexities of traditional libraries, such as OpenGL. The graphical programming done with MatPlotLib will serve students well in other contexts. Visualization helped students understand the scientific questions asked in the projects, but it also helped them understand their code. It made it clear that visual computing is an engaging activity that is underutilized in many CS curricula.

As the course was developed jointly with faculty in the other science departments, it was natural for them to give guest lectures. The guest lectures showed how CS concepts arise when solving the disciplinary problems. These lectures included concepts such as Maxwell's Demon and used state-of-the-art software, such as NetworkX for graph manipulations and CytoScape for visualizing protein interactions. The course material covered was to a large extent driven by the projects described in more detail in the next section.

3. PROJECT OVERVIEW

The course assigned four small-scale programming assignments and four projects. Almost all questions on the smaller programming assignments were preparation for the larger projects. Each project consisted of a programming part and an experimental part (which for some projects used data

culled from research). The experimental part could be completed with the code the student wrote or with code provided by us. Interestingly, very few students decided to abandon their code, even when it proved to be incorrect and inefficient. This at times meant that experiments for larger data sets could not be completed, mainly due to excessive running times. All projects asked students to produce visualizations of computational results and provide a write-up on their observations.

1. Manipulating Digital Audio.
   Explore the generation and manipulation of digital sound. Students write and use several basic functions that represent sounds as a sequence of wave amplitudes. The project emphasizes arrays, loop structures, numeric data (including overflow and round off issues), and modularity through procedures. Experimentation encourages students to generate sounds with different kinds of waves (e.g., square and sawtooth) and to investigate music in different scales.

2. Computational Experiments on Percolation in Grids.
   Examine the spread of wild fire through a patchy field of dry grass, electricity through a surface of conductors and insulators, or how water soaks through a porous landscape. This project uses a two dimensional array to represent these physical scenes and a single parameter to represent the probability that any node in the grid "percolates", e.g., burns, conducts, or flows. By varying the parameter, generating random grids based on it, and simulating flow in those grids, students create plots to answer the question, "What is the smallest probability q at which a grid generated with probability q will percolate?" In addition to reinforcing loops, conditionals, and multi-dimensional arrays, this project uses random number generation and introduces recursive functions.

3. Simulating Physical Systems.
   This project elaborates on Monte Carlo methods as a way to understand the behavior of physical systems without resorting to a detailed, low-level simulation. It introduces the "demon algorithm" (from Maxwell's demon) and has students performing two experiments: (1) simulating an ideal gas to determine its average system energy (temperature), and (2) using the Ising model to analyze a grid of magnetic spins, determining the average magnetization of the grid. The projects use tools from the VPython and Matplotlib libraries to create visualizations of their experimental results.

4. Analyzing Protein-Protein Interactions.
   Analyze the results of large-scale experiments that characterize protein-protein interactions and predict the quality of the experimentally observed protein complexes. Goals are to determine functional modules (finding groups of interacting proteins that participate in the same or similar biological function) and to find novel protein complexes that have not been observed before. Students use the Python-based NetworkX and Markov Cluster (MCL) libraries to manipulate and cluster these large graphs, relate them to the publically available Gene Ontology (GO) database, and visualize the results using Cytoscape, a high quality, open-source graph visualization tool.

The two most popular projects were percolation and the simulation of physical systems. Seeing different percolation algorithms detect different type of flows through a grid while graphs plot experimental probability results played a role in making this a favorite project. The preference of the simulating physical systems projects seems to be related to the large number of physics majors who had seen this material in a physics course. The project on protein interactions required students to write specified algorithms operating on graphs (using NetworkX) which was a new and for some unusual level of abstraction.

Students disagreed on the value and excitement about running computational experiments. Some clearly like that aspects, while others would have liked to be done after the programs ran. It is probably valuable to have a course which is a straight programming course available as a computing requirement option.

4. EVALUATION

The first offering of the course had 15 students enrolled initially and 13 completed the course. 10 of the 13 students were physics majors, the other three were chemistry majors. About a third of the students double majored in Math. The class had only one female students. 27% of the students had no programming experience, 40% had done some programming on their own, and 33% had taken a programming class. No student had taken a college level programming class.

The objective of the course is providing a foundation of programming and computational principles that students can and will apply to scientific inquiry. Our goal was not to turn science majors into CS majors. Actually, if this would have been a goal, the cooperation and collaboration from the other science departments would not have existed. While almost all students enrolled to fulfill the computing requirement, our goal was to get students interested in actively using computation in their major and to consider a taking a second course.

During the semester, the course did explain what material would be covered in what CS course and we tried to give students a sense on what they would learn in other CS courses. There was some discussion on what course sequence would lead to a minor in CS (which would fulfill the required multidisciplinary requirement for science majors). We encouraged students to realize the benefit of writing simple programs using MatPlotlib or VPython in other projects and classes. One of out goals is to provide students with the tools to write small Python programs for visualization and data analysis purposes. In follow up studies we plan to to track how many students end up taking another CS course.

Students taking the course completed an entry and an exit survey (id's allowed us to link survey responses). The presurvey was completed by 15 students and the post-survey by 13 students. Looking at the background of the students, 27% had no programming experience, 40% had done some programming on their own, and 33% had taken a high school level class. 10 of the 13 students who completed the course were physics majors, the other three were chemistry majors. About a third of the students were Math double majors, but they viewed Math as their secondary major.

Two interesting questions to compare responses for are: "How would you rate your current interest in:"
Q1: Taking another computer science course
Q2: Pursuing a career that requires programming skills

Table 1: Results for two entry and exit questions

	Entry			Exit		
	Mean	Median	Std. Dev.	Mean	Median	Std. Dev.
Taking another CS course	1.62	1	0.92	2.46	3	1.45
Pursuing a career that requires programming skills	1.69	1	1.14	3	2.85	1.7

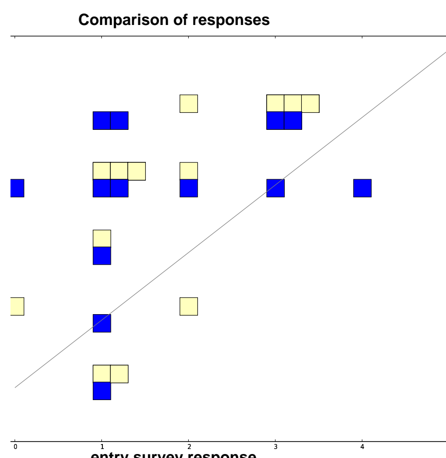

Figure 1: Comparison of entry and exit responses.

There was a choice of five answers: not interested (0), somewhat uninterested (1), undecided (2), somewhat interested (3), and very interested (4). Table 1 shows statistical results and Figure 1 shows a graphical comparison of entry and exist responses. About two thirds of the responses indicate an increase in interest, and no decrease was by more than one score. the exit responses indicate that 60% of the students plan to take another CS course and 40% plan to minor in CS (which would fulfil the multidisciplinary requirement of the a science curriculum).

We point out that in our introductory courses for majors (Purdue students declare a major as freshmen), we see a decrease in the interest in computer science and only 30% of CS freshmen complete a B.S. in computer science. The data reported by the Emerging Scholar Program carried out by a number of institutions also shows a decrease in interest, while showing that students in the program receive better grades [?]. The inability of introductory CS courses to maintain student's interest is viewed as one reason for the decreased enrollment in computer science.

From the feedback we have from students in the computational thinking class, it seems the problem driven format of the course, the ability to quickly be able to write programs meaningful to them, and the use of visualization tools played a crucial role in the overall increased interest. Future instances of the course will assess this further and will track students with respect to additional computing courses taken.

5. CONCLUSIONS

We believe that the interaction with science faculty is a critical element in designing an effective course in computational thinking for science majors. Based on our experience, Python is an excellent vehicle. It is used by many scientific disciplines, it allows us to teach modern concepts of programming, and it can be used interactively, giving students immediate feedback and giving them a convenient way to experiment with different constructs. Such concepts include recursion (which was considered challenging by the students) and object-oriented design (which they found natural). Moreover, visualization is an important element in computing and brings many quantitative scientific facts to life. VPython was a good vehicle because it focuses on a few simple basics in visualization and is learned quickly by doing.

6. ACKNOWLEDGMENTS

We would like to thank Sagar Mittal and John Valko for their valuable help on course and project development. We thank Olga Vitek and Sabre Kais for productive discussions in the areas of bioinformnatics and computational chemistry.

7. REFERENCES

[1] 2020 - Future of Computing. *Nature*, 440, March 2006.

[2] College of Science, new Science Undergraduate Curriculum, Purdue University. http://www.science.purdue.edu/core/requirements2.asp, 2007.

[3] VPYTHON: 3d programming for ordinary mortals. http://www.vpython.org/, 2007.

[4] Lectures and course material for introduction to computational thinking, purdue university. http://secant.cs.purdue.edu/, 2008.

[5] R. W. Chabay and B. Sherwood. *Matter and Interactions, Volume 1: Modern Mechanics.* John Wiley and Sons, Hoboken, NJ, 2002.

[6] R. W. Chabay and B. Sherwood. *Matter and Interactions, Volume 2: Modern Mechanics.* John Wiley and Sons, Hoboken, NJ, 2002.

[7] T. J. Cortina. An introduction to computer science for non-majors using principles of computation. In I. Russell, S. M. Haller, J. D. Dougherty, and S. H. Rodger, editors, *SIGCSE*, pages 218–222. ACM, 2007.

[8] Z. Dodds, R. Libeskind-Hadas, C. Alvarado, and G. Kuenning. Evaluating a breadth-first cs 1 for scientists. In J. D. Dougherty, S. H. Rodger, S. Fitzgerald, and M. Guzdial, editors, *SIGCSE*, pages 266–270. ACM, 2008.

[9] P. B. Henderson, T. J. Cortina, and J. M. Wing. Computational thinking. In I. Russell, S. M. Haller, J. D. Dougherty, and S. H. Rodger, editors, *SIGCSE*, pages 195–196. ACM, 2007.

[10] W. von Rüden. LHC computing needs, large scale computing in high energy physics, astrophysics, accelerator physics, nuclear physics and biophysics, fermi labs, 2002. Also available as http://conferences.fnal.gov/lccws/papers/tues/LHC_Tues_WvR.

[11] G. Wilson, C. Alvarado, J. Campbell, R. Landau, and R. Sedgewick. Cs-1 for scientists. *SIGCSE Bull.*, 40(1):36–37, 2008.

[12] J. M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, 2006.

[13] M. Zhang, E. Lundak, C.-C. Lin, T. Gegg-Harrison, and J. M. Francioni. Interdisciplinary application tracks in an undergraduate computer science curriculum. In *SIGCSE*, pages 425–429, 2007.