EXPANDING CS1: APPLICATIONS ACROSS THE LIBERAL ARTS*

Bridget Baird and Christine Chung
Department of Computer Science
Connecticut College
New London, CT 06320
860 439-2008
bbbai@conncoll.edu

ABSTRACT

This paper will describe how applications in a variety of disciplines can enhance the teaching of the CS1 course. Examples will be given from a range of disciplines, including mathematics, economics, linguistics, history, biology, art and music. The applications will be linked to the computer science concepts being discussed. Such an approach broadens the appeal of the introductory course and also teaches students valuable problem solving skills.

INTRODUCTION

There have been a variety of approaches for teaching introductory computer science that have been used to entice students into the discipline. Some of these include the use of animation [10], film [7], or web design [2]. Other approaches have integrated the natural sciences into the computer science curriculum [1, 5]. At the same time that computer science is broadening its reach into other disciplines, there has also been an increased emphasis on problem solving techniques in the CS1 course. Some of these techniques have been quite extensive and complex [4, 9]. Others stem from the research by Margolis and Fisher [8] that shows that women are more likely to major in computer science if they can see the tangible applications and ways in which computer science can be used. Problem-solving activities also seem to increase the self-confidence of women taking CS1 [6]. Since the last Taulbee survey [3] shows that 11.8% of undergraduate degree recipients were women and nearly two-thirds of the undergraduate majors were white and non-Hispanic, it is imperative that efforts continue to be made to broaden the

---

appeal of CS1 to underrepresented groups. Thus, using problem solving across a wide range of disciplines is a natural approach.

At smaller colleges the CS1 course has multiple audiences: serving as a base for those desiring to major or minor in CS but also serving as an introduction to the discipline for students who might only take one or two courses in the subject. Thus one of the first goals of the course is to produce a challenging but accessible and interesting course for all audiences. Another goal for this course, obviously, is to introduce basic programming concepts in computer science. Discussions at the college also focus on two additional goals. The first is to emphasize the importance of computer science in solving a variety of interesting, practical problems; students should become active problem solvers themselves. And then the final (implicit) goal is to interest a broader group of students in the joys of computer science; in the best case they are enticed to become majors and active practitioners and in any case all students finish the course with a greater appreciation for the discipline and a confidence in their abilities to apply these concepts to their own fields.

The language that is used to teach the introductory course is Python. This language seems particularly well suited to meeting the goals of this course: it is relatively easy to teach and learn, it does not have a lot of jargon, the programming environment is free and thus promotes continued use beyond the end of the course, and there is a wide range of modules that provide accessibility to a host of applications.

CONCEPTS AND APPLICATIONS

As the basic concepts of computer science are taught and as students develop proficiency with the language, an effort is made to show these concepts in use, linking them to practical applications. The choice of applications reflects teaching in a liberal arts college, where students from all disciplines take the introductory course. The institution has a long history of supporting interdisciplinary collaborations and includes a number of centers, including one in arts and technology (which requires this CS course). Thus the students not only come from a variety of backgrounds but also have a certain expectation about combining different fields in their studies. Table 1 shows the broad links among concepts and applications that are used in this course.

Application	Programming Concepts
Linguistics, History, Political Science, English	String/file processing, conditionals, loops, arrays (or Python lists/data collections), searching and sorting algorithms
Art and Music (image and sound processing, graphics, animation)	Nested loops, modules, recursion, event-driven programming, functions and parameters
Biology, Genetics	Object-oriented programming (including inheritance), string/file processing
Mathematics, Games, Simulation	Pseudorandom numbers, software engineering, object-oriented programming, algorithms
Statistics, Finance, Economics	File processing, graphics and visualization, web-crawling, algorithms

Table 1: programming concepts and the applications used to illustrate them

String Processing for Text Analysis

One of the strengths of Python is that it allows for early accessibility to file and string processing. These concepts and an early introduction to conditionals and loops mean that within the first few weeks of class, students can write intriguing and interesting applications involving large texts.

One cross-disciplinary application is analyzing texts of famous literary works (which can easily be obtained online from websites like www.gutenberg.org). The students can study literary styles of different authors (average sentence length, average word length, commonly used words or phrases), or they can alternatively perform word processing tasks on the texts (find/replace, spell check, etc.). These same techniques lend themselves to analyses of historical documents such as diaries, census information and historical texts.

Another compelling application is the problem of analyzing presidential or campaign speeches. For example, students can be given the task of comparing the 2008 convention speeches of Senators Barack Obama and John McCain. They can be asked to compare the number of times each candidate mentioned specific words, drawing conclusions about their priorities or campaign styles. They could also easily modify the speech to have all instances of a certain word replaced by another, creating a new file with an interestingly altered version of the speech. This particular application happens during the third week of class, giving students an early idea of the extensive and interesting programming concepts at their disposal.

Later in the course, when students have learned some basic graphics, and have been taught arrays/lists, searching and sorting, they can apply these skills to writing a program to generate word clouds. A word cloud is a graphical representation of the words from a body of text wherein more frequent words from the text are displayed in larger fonts, providing an interesting visualization of the given text. Such visualizations have recently been popularized by a number of internet websites. This engaging and multi-faceted task requires students to exercise skills such as processing text from a file, counting word frequencies, sorting the words in order of frequency (using parallel arrays or Python dictionaries), searching for and removing any “stop words” (less important words that should not be included in the word cloud such as “the” and “is”), and displaying them in a graphical window in a pleasing and randomized fashion so that more frequent words appear larger and closer to the center.

Functions and Recursion for Graphics and Animation

A nice way to illustrate the power and flexibility of parameterized functions is by using graphics. A basic beginning exercise might be to ask the students to draw a simple shape (like a square or circle) whose color is specified as a parameter. Other properties of the shape can then of course be parameterized to teach students how to create functions with multiple parameters. The use of a graphics package to create custom shapes using student-defined functions transitions nicely into the introduction of boolean functions, which can be used to compute various geometric properties of the shapes being drawn.

Once students understand basic graphics (which also gives them an early taste of object-oriented programming) and loops, it is instructive to have them create graphical animations. Even simple animations are useful for teaching a number of lessons: they can again be used to help students practice writing modular and parameterized functions (e.g., speed or direction of the animation can be parameters), they can be used to show creative uses for loop counters (at this point it is still early in the course and students are still not quite comfortable with the idea of using the built-in definite-loop counter variable within the body of the loop), but it also gives students a sense of the processing speed of the computer. While they may feel many other tasks could be done manually (if slower), animation exercises can concretely affirm to them that there are tasks where the role of the computer is truly indispensible.

More complicated graphics introduce visualization of data. By using graphics as well as functions with parameters, students are asked to take actual data sets from economics (such as census information, employment statistics, etc.) and decide how best to visually impart the data. They often choose data which has particular meaning to their own interests. Although they produce fairly complicated and interesting graphics, this is still a topic that occurs quite early in the course, in the first month.

Students start out in the course writing console applications, and in that setting it is naturally harder for them to relate to the context of the program's flow (as most students have only used applications with graphical user interfaces). Thus, giving them an early chance to create a graphical user interface with clickable buttons creates a more familiar user-setting that helps to motivate indefinite loops and event-driven programming techniques. Because of their extensive experience as users of GUIs, students can fairly easily grasp the concept of a user's button-click driving the control of program flow.

Recursion is often a challenging concept for students. An engaging way to illustrate the power of recursion and allow students to really experience it is to have students create drawings of fractals. Using a simple graphics package, students can create a straightforward program that draws straight lines and keeps track of changes in direction. Then, after some practice with simpler recursive images, the students can be given the task of creating more complex, well-known and beautiful fractals, such as the Koch snowflake and Sierpinski Triangle. They can also be given the opportunity to create/devise their own fractal drawings.

Nested Loops and Modules for Image and Sound Processing

Image processing is an appealing and convenient way to not only incorporate art into an introductory course, but also to demonstrate nested looping to students. Students are able to get a concrete grasp for the purpose and structure of nested loops when they can physically see each of the pixels of a two-dimensional image being processed. Interesting applications for the students include: changing all pixels of a certain color in an image; analyzing paintings (was Picasso's Blue Period really "blue"); changing the hue, brightness, or saturation of images; and doing some basic pattern recognition of images by finding outlines of objects.

Sound processing of wav and MIDI files presents more opportunities to illustrate the power of computer programs to manipulate information. It also demystifies the

structure of wav files, showing them to be collections of bytes that can be controlled and changed. Some of the programs for this application include generating simple songs and tunes, doing simple analyses of wav files, and altering wav files (for example, changing the volume).

As students look at the quantity of data necessary for pixel information (and then extrapolate to video) and also look at the size of wav files, this presents an opportunity to talk a bit about compression methods, how important they are, and how crucial good algorithms become. The processing of images and sounds also presents an opportunity to show the power and convenience of external, already-written modules. Students not only learn how to interface with these modules (reinforcing the versatility of functions and parameters, the importance of modular programming, as well as object-oriented programming concepts like encapsulation), but also realize the potential of tapping into a well-established but evolving body of knowledge.

Pseudorandom Numbers for Games and Simulation

When programming a game or simulating game play, there is a natural need for random number generation. With this motivation, incorporating pseudorandom number concepts into the curriculum becomes natural.

The famous Monty Hall problem is a great application for demonstrating the power of simulation using random numbers. In it, the game show host, Monty Hall, asks the game show contestant to choose one of three doors. Only one of the doors has a prize behind it. After the contestant chooses a door, Monty opens one of the doors *not* chosen by the contestant that also does not have the prize behind it. He then gives the contestant the chance to change his/her mind about which door s/he chose. It is well known and easily shown (if counter-intuitive) that the contestant is better off switching his/her choice of door. The students can create a program to simulate this famous game show and via simulation compute the likelihood of winning when the contestant chooses to stick with his/her original choice compared to when s/he switches. It is a nice way to affirm a counter-intuitive theoretical result for introductory-level students.

There are many other games and simulations that illustrate uses of probability, functions and parameters and user interaction. Some of the simulations used in this course have included Black Jack, Monte Carlo simulations for estimating pi, baseball simulations to analyze the occurrence of “streaks” and “batting slumps,” and tennis simulations. As students develop reasonably complicated games, there is also an opportunity to discuss software development, top-down and bottom-up design, and the importance of planning in order to write a cohesive program. Students are required to incorporate these concepts in their large, final project, which is on a topic of their own choosing.

Object-Oriented Programming for Genetics and Biology

Object-oriented techniques enter the CS1 course in many places: the structure of Python itself, string processing and file manipulation, introduction of graphics, game simulations, etc. Another place where this concept is introduced to the students is in a

lovely application in genetics. Genomes are composed of DNA molecules which contain sequences of millions of bases (A, G, C, T). These molecules contain chromosomes, which in turn contain genes. It turns out that genes can be identified by using straightforward string processing techniques (they are subsequences with certain easily identifiable starting and stopping codons). This topic presents an opportunity to talk about classes, inheritance and methods for those classes. Data for this application are taken directly from the National Center for Biotechnology Information (NCBI) website. Students are able to download strands of DNA and then pick off the genes. They can then compare genes in different organisms, looking for identical ones or ones that are similar. Once again they are able to see both the computing power that is necessary for such applications but also the possibilities for asking and answering interesting questions about this discipline.

Other areas in science where object-oriented techniques and external data have been used include environmental research information obtained from colleagues. Often this information is contained in spreadsheets. In most institutions one can find colleagues who have data sets that can be mined for interesting conclusions.

Web Crawling for Statistics and Finance

Students bring great familiarity with the web into this class and it is important to show them how they can write computer programs to tap into this resource. The applications in this section range from the straightforward (taking data from a single source) to web crawling (where the program moves from one web site to another). The simpler applications illustrate the use of current data that is easily found online (such as stock prices, or data on recent earthquakes from the US Geological Survey) to produce informative statistics. The more complex programs involve opening a web site and then using information from that web site in interesting ways, including moving to web sites contained in its links. These more complicated interactions lead to a discussion of the complexities of recursion (every computer has limits that can be quickly reached), the importance of searching and sorting algorithms, and the importance of algorithms in general. It is a good place to talk about search algorithms, such as Google's algorithm for searching the web, and what those mean for computer science.

Sources for Applications

Some of the applications, such as some of the fractal drawings and several of the simulations involving random numbers, are taken from the text by Zelle [11] that is used in the course. Many others come from conversations with colleagues. The authors are happy to share their ideas, experiences and course materials (including labs, homework and class activities).

CONCLUSION

This cross-disciplinary CS1 curriculum has been created and shaped over several semesters. It has been well-received and seems to have attracted an increased number of majors to the computer science department. The fact that the number of female computer

science majors is greater in our department than in neighboring colleges of comparable size and quality might also be attributed in part to our introductory curriculum.

The CS1 course is constantly evolving as different kinds of applications are introduced. In all cases an effort is made to show "real" problems and not artificially-produced ones. In the future there will be applications that incorporate some of Google's applications and other kinds of mashups. Students are very familiar with using many kinds of Internet applications and it is beneficial for them to see how these services can be brought to bear under their control in a computer program. Another area being examined is computational chemistry, where one of the central pieces of software that allows chemists to study and manipulate visual representations of molecules is written in Python. Efforts are being made to see if an interface is possible so that students can write Python modules that plug into the software and manipulate the data. The philosophy of this CS1 course is that it should illustrate computer science as a dynamic, relevant discipline with beautiful structures and functional techniques.

It is important that in the setting of a smaller liberal arts college, computer science is incorporated as much as possible into the liberal arts curriculum, rather than introduced in isolation from the many other rich areas of study that the students expect to be exposed to. While students should have the chance to appreciate computer science for its own sake, it is also vital that students come away from an introductory course with an understanding that computer science permeates all areas of study and is a fundamental part of a liberal arts curriculum.

REFERENCES

[1] Adams, J. B., Computational science as a twenty-first century discipline in the liberal arts. *J. Comput. Small Coll.* 23 (5), 15-23, 2008.

[2] [2] Baird, B., Web design: interface to the liberal arts, *J. Comput. Small Coll.* 21 (6), 14-19, 2006.

[3] CRA Taulbee Survey, Computing Degree and Enrollment Trends 2007-2008, www.cra.org/info/taulbee, retrieved November 16, 2009.

[4] Faulkner, K., Palmer, E., Developing authentic problem solving skills in introductory computing classes, *Proceedings of the 40th SIGCSE technical symposium on Computer science education*, 4-8, 2009.

[5] Ivanov, L., The N-body problem throughout the computer science curriculum, *J. Comput. Small Coll.* 22 (6), 43-52, 2007.

[6] Kumar, A., The effect of using problem-solving software tutors on the self-confidence of female students, *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, 523-527, 2008.

[7] Lim, D., Lights..camera..computer science: using films to introduce computer science to non-majors. *J. Comput. Small Coll*. 23 (5), 58-64, 2008.

[8] Margolis, J., Fisher, A., Unlocking the Clubhouse, Women in Computing, Boston, MA: MIT press, 2003.

[9] Rao, T. M., Mitra, S., Canosa, R., Marshall, S., Bullinger, T., Problem stereotypes and solution frameworks: a design-first approach for the introductory computer science sequence, *J. Comput. Small Coll.* 22 (6) 56-64, 2007.

[10] Stiller, E., Teaching programming using bricolage, *J. Comput. Small Coll*. 24 (6), 35-42, 2009.

[11] Zelle, J., *Python Programming: an Introduction to Computer Science*, Wilsonville, OR: Franklin, Beedle & Associates, Inc., 2004.