Rethinking CS0 with JavaScript

David Reed
Department of Mathematics and Computer Science
Creighton University
Omaha, NE 68178
DaveReed@creighton.edu

Abstract

Traditional approaches to CS0 have emphasized either breadth, through an overview of computer science, or depth, through intensive programming. This paper describes an alternative teaching method that strikes a balance between these two approaches through the use of JavaScript and the World Wide Web. By taking advantage of JavaScript's simplicity and natural Web-based interfaces, the CS0 course described here is able to maintain a strong emphasis on programming and problem-solving, integrate programming skills with Web technology, and still provide reasonable breadth on general computer science topics. This balance between depth and breadth makes the course attractive to both non-majors and majors alike, providing a broad perspective of the field as well as a foundation for continuing studies in computer science.

1 Motivation

The debate over the definition of CS0 has generally revolved around the issue of breadth vs. depth. The breadth-first approach (as described in [3]) stresses a general understanding of computer science as a field of study. By surveying a wide range of topics such as computer organization, graphics, networking, and technology in society, students experience the breadth of the field and develop a perspective to later understand and appreciate the role of technology in their lives. This approach, in various forms, has been adopted at many schools (e.g., [2] [9] [12]).

SIGCSE 2001 2/01 Charlotte, NC, USA

A potential weakness of the breadth-first approach is that it can be too superficial, presenting a broad perspective to students who lack the context and experience to fully comprehend it. The discipline of programming not only develops problem-solving skills, but also is central to many areas of computer science and thus important to appreciating their significance. (See [13] for compelling arguments.) Consequently, many schools offer a CS0 course that stresses programming and problem solving in order to expose students to computer science along deeper (albeit narrower) avenues, e.g., Web design [11], spreadsheets [7] [8], or graphics [6].

Ideally, a CS0 course should contain elements of both breadth and depth in order to provide a broad perspective as well as an experienced-based context for appreciating the significance of the field. In practice, however, this balance is difficult to achieve. Attempting to convey the full breadth of computer science may allow for only a cursory coverage of programming. Conversely, developing competency as a programmer and problem-solver requires extensive hands-on experience (especially when learning a complex language such as C++ or Java), which leaves little time for other topics. The adoption of a simpler programming language such as Scheme [4] [16] or ISETL [14] can decrease the difficulty associated with learning to program, but may also introduce new challenges. Students are less motivated by languages they see as non-commercial, and they may also have difficulty applying their programming skills to more traditional languages if they choose to continue in the computer science curriculum.

2 A Balanced Approach to CS0

This paper describes a CS0 course that utilizes JavaScript and the Web as tools for balancing depth and breadth. While the use of the Web as a central theme in CS0 is not new (see [1] [5], [10]), this course is unique in that it maintains a strong emphasis on programming and problem-solving, integrates programming skills with Web technology, and still provides reasonable breadth on general computer science topics. In addition, the course

emphasizes the development of empirical skills through laboratories and interdisciplinary applications.

The balance between depth and breadth is made possible by the choice of JavaScript as the programming language. JavaScript's simplicity, natural interfaces, and seamless integration into Web pages make it possible for novices to develop interesting and engaging programs quickly. Time previously devoted to complex language features or the idiosyncrasies of development environments can instead be spent studying computer science topics and resources on the Web. Thus, only about 50% of class time is devoted to programming concepts. An additional 15% of class time is devoted to HTML elements and interface design, enabling students to apply their programming skills to the development of attractive, easy-to-use Web applications. The remaining 35% of class time is then devoted to exploring other topics in computer science.

From a CS0 perspective, the choice of JavaScript has the additional advantage of being relevant and useful to the wide range of CS0 students. The universal availability of a JavaScript programming environment through (free!) Web browsers provides the opportunity for everyone to apply programming skills learned in CS0 to everyday problems. Syntactic similarities between JavaScript, C++, and Java also make the transition into these other languages simpler for students who continue in the computer science curriculum. Of course, there are some features of JavaScript that are not ideal: Web browsers are notoriously buggy and features such as typeless variables can lead to obscure errors. As in any course, designing class and laboratory time so that students work under close supervision can help to address any confusion over language issues.

2.1 Programming Tutorials

Beginning programmers master concepts at various speeds and utilize a variety of learning styles. As such, the programming component of our CS0 course emphasizes self-paced, interactive learning over traditional lectures. Students are introduced to new programming concepts through a series of online tutorials (see Table 1). Each tutorial contains explanatory text, examples, and exercises for applying new concepts and techniques. In early tutorials, these exercises mainly involve tracing and explaining the behavior of code, modifying existing programs, and writing small code segments to meet well defined specifications. As the students' programming skills develop, however, the exercises become more ambitious and open-ended, culminating in the design and implementation of complex programs and their interfaces (such as an interactive slot machine or an encryption/decryption page).

1.	HTML & Web Pages
2.	Dynamic Pages via JavaScript
3.	Data Types & Expressions
4.	Abstraction & Libraries
5.	Abstraction & Functions
6.	Event-driven Programming
7.	Conditional Execution
8.	Dynamic Images
9.	Conditional Repetition
10.	Repetition & Strings

Table 1: Programming Tutorials

With the use of tutorials, students are able to work through the material at their own pace. Two or three class periods are devoted to each tutorial, so there is ample opportunity for the students to obtain assistance from the instructor or teaching assistants if needed. Novice programmers, who might otherwise be intimidated in front of the entire class, are more likely to seek individual help. And by having the instructor directly observe their behavior and address problems as they arise, the help they receive is personalized to match their needs.

The ability to create professional-quality programs can be extremely motivating to CS0 students who may be initially skeptical of programming. As such, HTML interface elements such as buttons, text boxes, text areas, and dynamic images, are introduced as soon as applications allow for their use. For example, once functions have been introduced in Tutorial 5, event-driven applications involving text boxes and buttons become feasible. In Tutorial 6, students write a MadLib program that has text boxes for inputting words (e.g., a color, a place), and a text area for output. When the user clicks on a button, a function is called to display a story with the input words substituted for blanks. Similarly, once if-statements have been introduced in Tutorial 7, it is possible to program an interactive craps game with buttons and dynamic, clickable dice images (see Figure 1). By introducing interface elements in conjunction with related programming concepts, students are able to apply new programming skills to attractive and engaging programs.

2.2 Closed Laboratories

An important component of the class is the use of weekly closed laboratories to emphasize the application of programming skills to solving problems. Interdisciplinary applications are frequently chosen to demonstrate the relevance of computing to other fields of study. For example, one lab assignment involves the design and implementation of strategies for the Prisoner's Dilemma, a scenario with applications in economics and evolutionary

biology. Students design and implement strategies in the form of JavaScript functions, and then match their strategies against one another in an interactive Web page (see Figure 2). A popular activity is then to have a class-wide championship, where all of the student strategies compete in a round-robin tournament.

In recognition of the fact that empirical skills are playing an increasingly important role in computing and in society [15], many of the lab assignments emphasize programming as a tool for experimentation. Students are introduced to a problem or simulation model and are asked to form hypotheses, design and conduct experiments, and analyze their results. For example, in the first lab, students utilize a random letter-sequence generator, provided in a Web page, to estimate the number of 4-letter words in the English language (see Figure 3). By generating random letter-sequences and counting the number of real words, the students are led through the process of constructing and justifying a reasonable estimate of the number of 4-letter words.

As is the case with programming skills, empirical skills are built incrementally over the course of the semester. Early labs, such as the 4-letter word example above, provide structured frameworks for experimentation. In later labs, students may be given a hypothesis and be asked to design an experiment to verify or refute that hypothesis. Eventually they must form their own hypotheses and design their own experiments to provide supporting arguments. For example, in the fourth lab, students simulate two-dimensional random walks in a Web-based turtle graphics environment. They are given the theoretical result relating expected distance to number of steps (distance D requires $\sqrt{D}$ steps), and are asked to verify this experimentally. In the 8th lab, students revisit the concept of a random walk, only now along one dimension. They must consider two variations of one-dimensional walks, one in which the walker can move freely in both directions and another in which a wall impedes movement in one direction. They must then hypothesize how these variations will behave relative to one another and develop programs to test their hypotheses (see Figure 4).

2.3 CS Breadth

The breadth component of the CS0 course is focused on topics that help students to understand computer technology and its impact on society. Class periods are scheduled throughout the semester for researching and discussing topics such as the structure of the Internet, the history of computers, and ethical issues in computing (see Table 2). Students are assigned readings and questions to research, and must email responses to the questions before their class period. This ensures that the students come to class prepared, and also provides feedback to the instructor as he or she prepares to lead the discussion.

1. Computer Basics
2. Internet and the Web
3. History of Computing
4. How Computers Work
   a. von Neumann architecture
   b. ALU and data path
   c. memory and assembly code
   d. binary repr. and machine code
   e. logic & digital circuits
5. Human-Computer Interaction
6. Computer Science Sub-fields
7. Computers and Society

Table 2: Breadth Topics

Many of the breadth topics involve experimentation using online applications. A colleague at Dickinson College, Grant Braught, has developed a collection of resources for exploring the internal workings of a computer. Over the course of the semester, students experiment with data representation, circuit design, data flow and the ALU, and program translation using interactive applications in a Web browser.

3 Integrating CS0 into the Curriculum

A common weakness of traditional CS0 courses is that they often do not integrate well with the rest of the curriculum. For the CS0 student who does choose to continue with CS1, a breadth-first course does little to prepare them for the intensive programming required. The use of specialized languages in a depth-first course can also lead to problems, as students may have difficulty applying programming skills to a new language paradigm.

The CS0 course described in this paper was developed as part of an integrated computer science curriculum at Dickinson College. The three course introductory sequence was designed to be cohesive, with smooth transitions between courses. The CS1 course is taught using Java, so the basic syntax of the language is similar to JavaScript (although the same could be said of C++). In addition, common themes such as experimentation, analysis, and explanation are stressed. The CS2 course continues these themes, teaching data structures and algorithm analysis using Java.

CS0 and CS1 are both considered entry points into the computer science curriculum at Dickinson. While the majority of students begin in CS0, those with prior programming experience and an interest in computer science are advised to enroll in CS1. Both courses are

taught each semester, so it is also possible for students to change levels within the first few weeks of the semester if desired. With respect to the general requirements of the college, the completion of any two consecutive courses, either CS0-CS1 or CS1-CS2, suffices to fulfill the laboratory science requirement.

In the fall of 2001, a version of this course will be introduced to the computer science curriculum at Creighton University. The course will be offered each semester and similarly serve both non-majors and potential majors (who then continue with CS1 in C++).

4 Outcomes

The JavaScript-based CS0 course described here has been taught at Dickinson College since the fall of 1998. Student reaction has been very positive, with many comments suggesting that the balance between breadth and depth has provided a more rewarding and engaging experience for non-majors and potential majors alike. Enrollments in the course have increased steadily, forcing the addition of extra sections in the second year (from 4 to 5). In the third year, pre-registration almost filled the available sections, and we were forced to cut upperclassmen in order to provide spots for incoming freshmen.

Non-majors have found programming in JavaScript to be less intimidating than in past versions of the course, and see a greater relevance between programming and their chosen disciplines. Students who would probably never have written another program had they learned Java or C++ continue to write JavaScript programs for integration into Web pages. In addition, the breadth component of the course provides a broad perspective of what computer science is about, and how computers will relate to their lives.

For the potential major enrolled in CS0, the mixture of breadth and depth provides a more balanced picture of computer science, leaving them more informed when selecting a major. For those who do continue in computer science, this course serves to level the playing field with experienced programmers entering CS1. In addition, the JavaScript programmer is not only familiar with basic programming concepts and practices, but also has an immediate familiarity with Java due to the similarities between the two languages.

An additional advantage of a balanced CS0 course is that it does not require an early decision on the part of the student. A student who is undecided about his or her major may enroll in this course, knowing it will be worthwhile whether they choose to continue in computer science or not. Our experience has also shown that many students who are not initially interested in computer science will become attracted to the field after being exposed to programming. As such, a balanced CS0 course that is attractive to non-majors can be an effective means of recruiting computer science majors.

Materials for this course can be found online at `http://www.creighton.edu/~davereed/cs0`.

References

[1] Boroni, Christopher, Frances Goosey, Michael Grinder and Rockford Ross (1998). "A Paradigm Shift! The Internet, the Web, Browsers, Java and the Future of Computer Science Education." *SIGCSE Bulletin* **30**(1): 145-152.

[2] Bagert, Donald, William Marcy and Ben Calloni (1995). "A Successful Five-year Experiment with a Breadth-first Introductory Course." *SIGCSE Bulletin* **27**(1): 116-120.

[3] Denning, Peter J. et. al (1988). *Computing as a Discipline.* Final Report of the ACM Task Force on the Core of Computer Science, Association for Computing Machinery.

[4] Findler, Robert Bruce, Cormac Flanagan, Matthew Flatt, Shriram Krishnamurthi and Matthias Felleisen (1997). "Dr. Scheme: A Pedagogic Programming Environment for Scheme." In *Proceedings of PLILP '97 (Programming Languages: Implementations, Logics, and Programs).* LNCS 1292, Springer.

[5] Gurwitz, Chaya (1998). "The Internet as a Motivating Theme in Math/Computer Science Core Courses for Non-Majors." *SIGCSE Bulletin* **30**(1): 68-72.

[6] House, Donald and David Levine (1994). "The Art and Science of Computer Graphics: A Very Depth-first Approach to the Non-majors Course." *SIGCSE Bulletin* **26**(1): 334-338.

[7] Herrmann, Nira and Jeffrey Popyack (1994). "An Integrated, Software-based Approach to Teaching Introductory Computer Programming." *SIGCSE Bulletin* **26**(1): 92-96.

[8] Kolesar, Mary and Vicki Allan (1995). "Teaching Computer Science Concepts and Problem Solving with a Spreadsheet." *SIGCSE Bulletin* **27**(1): 10-13.

[9] King, L.A. Smith and John Barr (1997). "Computer Science for the Artist." *SIGCSE Bulletin* **29**(1): 150-153.

[10] Lim, Billy B.L. (1998). "Teaching Web Development Technologies in CS/IS Curricula." *SIGCSE Bulletin* **30**(1): 107-111.

[11] Mercuri, Rebecca, Nira Herrmann and Jeffrey Popyack (1998). "Using HTML and JavaScript in Introductory Programming Courses." *SIGCSE Bulletin* **30**(1): 176-180.

[12] McFall, Ryan and Gordon Stegink (1997). "Introductory Computer Science for General Education: Laboratories, Textbooks, and the Internet." *SIGCSE Bulletin* **29**(1): 96-100.

[13] National Research Council Committee on Information Technology Literacy (1999). Being Fluent with Information Technology, National Academy Press, Washington, D.C.

[14] Reed, David (1997). *Introduction to Computing: An Interactive Approach Using ISETL*. Published internally at Dickinson College.

[15] Reed, David, Craig Miller and Grant Braught (2000). "Empirical Investigation throughout the CS Curriculum." *SIGCSE Bulletin* **32**(1): 202-206.

[16] Vandenberg, Scott and Michael Wollowski (2000). "Introducing Computer Science Using a Breadth-First Approach and Functional Programming." *SIGCSE Bulletin* **32**(1): 202-206.

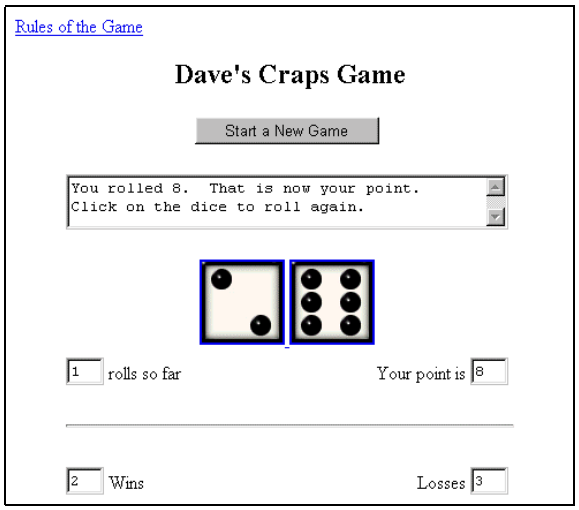

Figure 1

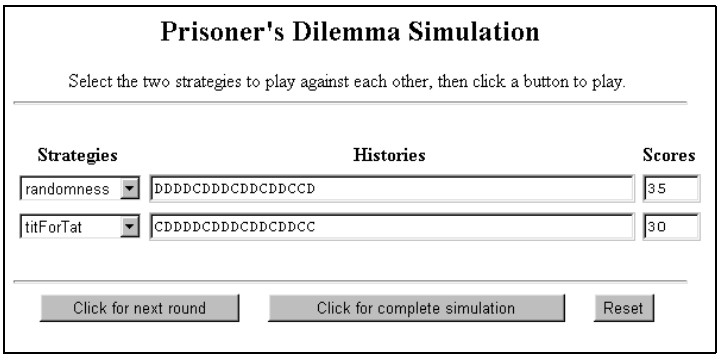

Figure 2

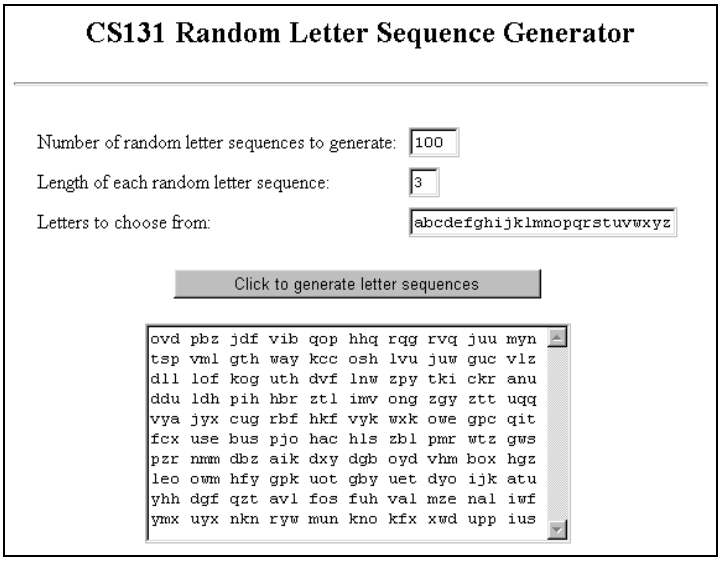

Figure 3

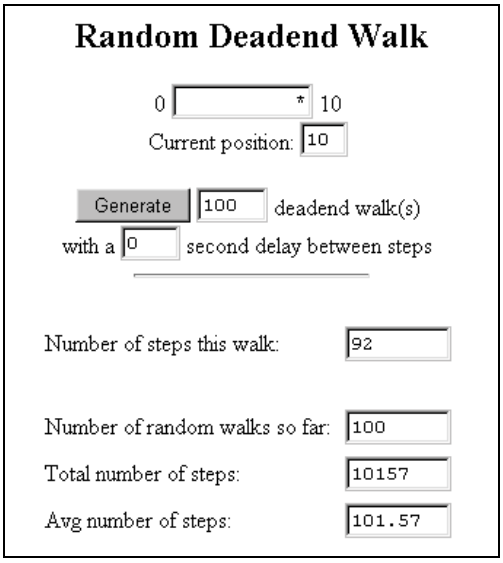

Figure 4