3-1-2019

Punch Cards to Python: A Case Study of a CS0 Core Course

Thomas Babbitt  
*United States Military Academy*, thomas.babbitt@westpoint.edu

Charles Schooler  
*United States Military Academy*, charles.schooler@westpoint.edu

Kyle King  
*United States Military Academy*, kyle.king@westpoint.edu

Follow this and additional works at: [https://digitalcommons.usmalibrary.org/usma_research_papers](https://digitalcommons.usmalibrary.org/usma_research_papers)  
Part of the Curriculum and Instruction Commons, Other Computer Sciences Commons, Programming Languages and Compilers Commons, and the Science and Mathematics Education Commons

Recommended Citation


This Conference Proceeding is brought to you for free and open access by USMA Digital Commons. It has been accepted for inclusion in West Point Research Papers by an authorized administrator of USMA Digital Commons. For more information, please contact nicholas.oliynyk@usma.edu.
We review the relevant pedagogical research for CS0 and present
We demonstrate similarities between our course and the Advance
ACM Reference Format:
There is an immense interest in teaching computer science con-
Thomas Babbitt, Charles Schooler, and Kyle King. 2019. Punch Cards to
ABSTRACT
There is an immense interest in teaching computer science con-
SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA.
This paper is authored by an employee(s) of the United States Government and is in
the public domain. Non-exclusive copying or redistribution is allowed, provided that
the article citation is given and the authors and agency are clearly identified as its
source.
This paper discusses the history of our CS0 course, reviews the
challenges of teaching computer science concepts and programming tools, computational thinking
ACM Reference Format:
Thomas Babbitt, Charles Schooler, and Kyle King. 2019. Punch Cards to
Python: A Case Study of a CS0 Core Course. In Proceedings of the 50th ACM
Technical Symposium on Computer Science Education (SIGCSE ’19), February
27-March 2, 2019, Minneapolis, MN, USA. ACM, New York, NY, USA, 7 pages.
https://doi.org/10.1145/3287324.3287491

1 INTRODUCTION
Interest in computing education for everyone has grown rapidly in
the last few years. In 2016, the White House launched the Computer
Science for All initiative [38], a large-scale effort to support pre-K
through 12th graders using local and state programs. The CS for All
initiative is maintained today through the National Science Found-
ation (NSF) and already has awarded over 200 grants [8]. In 2017,
the first Advanced Placement Computer Science Principles (AP
CSP) exams were administered. The AP CSP is a multidisciplinary
computing course with seven focus areas: Creativity, Abstraction,
Data and Information, Algorithms, Programming, The Internet, and
Global Impact [6].

Today, most colleges offer an introduction to Computer Science
course (CS0) geared toward students outside the computing disci-
pline. There is little consensus on the role of computer programming
in a CS0 course, with some courses focusing exclusively on pro-
gramming but others focusing on programming concepts instead
of programming itself [33]. For the last 50 years, every student
at West Point has been required to learn computer programming.
Since 1989, it has been as part of the school’s compulsory CS0
course. We believe that programming occupies a central role in
computing literacy and thus about a third of our CS0 focuses on
programming instruction.

This paper discusses the history of our CS0 course, reviews the
challenges in teaching computer programming to novices, explores
the current best-practices in programming instruction, analyzes the
last decade of student feedback for our institution’s CS0 course, and
enumerates the most valuable lessons learned in the most recent
curricula redesign.

We assert that CS0 should be a requirement for all college stu-
dents. While programming instruction is inherently challenging,
we maintain that a CS0 education is attainable for everyone.

2 BACKGROUND
This section explores the challenges of teaching computer science
concepts, specifically programming, to novice programmers and
explores the current thinking and best practices for teaching a
CS0/CS1 course.

2.1 History of CS0 at Our Institution
West Point’s CS0 course has changed a number of times over the
past 30 years (see Figure 1). Prior to 1989, a required course taught
a number of different programming languages including FORTRAN
with punch cards. Starting in 1989, with the restructuring of depart-
ments, a designated CS0/CS1 course was implemented and required
for all students. The course focused exclusively on programming
and problem solving and was similar to an introduction to Com-
puter Science (CS1-level) course taught at most universities. The
course was originally based on Turbo Pascal, which was popular
at the time. The documentation on why the language was chosen
is lost; however, it is worth noting that the AP Computer Science
A/AB at that time also used Turbo Pascal.

In 1996, the programming language changed to Ada 95 in order
to align with other CS courses taught at our institution. In 2001,
the course switched to Java.
In 2007, the course transitioned to Raptor with Java [9]. While the functionality is different from MIT’s Scratch [33], many of the concepts are similar. It uses flow charts to solve a problem and generates much of the Java code for the student. The focus was on the concepts and problem-solving process which reduced the emphasis on programming.

In 2012, a pilot course using Jython [15] ran and was implemented the next year. This course used a multimedia approach [12, 14, 25]. It increased the emphasis on programming concepts, but maintained the rigor in networks, security, hardware, and ethics.

The most recent redesign, piloted in 2018, transitioned students to a Javascript (Skulpt) Python 3 implementation. The course material for the programming section borrow heavily from the Think Like a Computer Scientist text on Runestone Academy [26]. Section 3 provides details on the most resent course redesign.

### 2.2 AP CS Principles Overview

The AP Computer Science Principles (AP CSP) course covers six computational thinking practices: connecting computing, creating computational artifacts, abstracting, analyzing problems and artifacts, communicating, and collaborating. There are seven “Big Ideas”: creativity, abstraction, data and information, algorithms, programming, the internet, and global impact [6, 20, 28]. Each “Big Idea,” has a number of enduring understandings, learning objectives, and essential knowledge. In Section 3.1.1, we discuss how our course overlaps with AP CSP.

### 2.3 CS0 vs CS1 Expectations

The expectations of a CS0 course are different from those of a CS1 [10]. The AP CSP course is described as an introductory computing course [6], which we would consider a CS0. CS1 courses go into more depth on subjects that are germane to only those students who will pursue a Computer Science degree. Some examples of this include: introduction to data structures, object-oriented programming, and inheritance [7, 36].

Since programming is a “Big Idea” in AP CSP it should be in a CS0 course. Additionally, there should be an expectation of networking, cyber, and hardware lessons in a CS0 versus a CS1 course. We include all of these concepts in our redesigned course.

### 2.4 Relevant Pedagogical Research

Before embarking on the latest redesign of our CS0 course, we conducted a literature review. Based on the review, Problem Based Learning (PBL) [17, 37] and Worked-Out Examples [21, 30] were the best options.

The Jython Environment for Student (JES) [12, 15] version uses a PBL approach to teach programming. Students are given instruction and then directed to apply their knowledge to challenges, typically homework problems. Course scaffolding includes detailed discussion on picture objects with properties such as height, width, pixel coordinate, and color values. Students are taught basic programming constructs such as conditionals and iteration. Then students were expected to apply the programming constructs such as sequence, iteration, and selection to effect changes to a picture.

From the literature and the collective experience of the course designers, we decided to move to a worked-out examples approach in our new course in which we dedicate a large portion of the class to “walking” students through example problems. Examples give students a place to start and provide the recurring aspects of a task. Worked-out examples help novices reach early successes without cognitive overload, i.e. excessive stress [21].

Once the novice programmer has acquired a baseline set of recurring aspect knowledge and some rudimentary problem solving skills, instruction can transition from product-based to process-based worked-out examples. Product-based worked-out examples consist of only the recurring knowledge such as how to assign a variable. Process-based worked-out examples add in the how and the why, such as the realization that a variable is a location in memory with a particular value. This improves the novice programmer’s problem solving skills and helps transfer these skills to high order problems and other domains.

Fading is the key to making novice programmers self-sufficient. Forward fading consists of removing the first part of the problem and letting the novice programmer solve the problem by providing the solution. Backwards fading provides the beginning of the problem and the novice programmer completes the problem [41].

There are also extraneous difficulties in learning to program, such as an Integrated Development Environment (IDE) or compiler/interpreter. These programs can be unintuitive to the novice and may contribute to the cognitive overload that novices experience in learning to program [5].

### 2.5 General Purpose Languages

Hiding the details of program execution makes beginner languages more accessible to novices. Beginner languages like MIT’s Scratch [33] feature drag-and-drop interfaces for programming constructs like for loops and function declarations. There is research that suggests that using a beginner language is appropriate for a CS0 course [13]; however, some experiences suggest that beginner languages are limited in their utility for solving real-world problems [18].

### 2.6 Historical CS1 Failure Rates

The general consensus is that introductory programming courses are difficult [16, 39, 40]. In fact, CS1 courses have both the highest
enrollment rates and the highest dropout rates at many schools [23, 24, 35]. Study after study indicates that students in introductory programming courses will generally fall into one of two groups: a highly successful group and a failing group [31, 32, 34]. Generally there are very few students in between. The most common grade in a CS1 course is an ‘F’ followed by an “A” (Figure 2) [34].

The distribution of IQ across the general population is, by definition, a normal distribution. We asked ourselves, if student performance in an introductory programming course is a function of intelligence, why doesn’t student performance exhibit the same distribution as IQ? Our research shows that student grades are not a function of cognitive capacity, development, or style; attitude or motivation; demographic factors, such as previous exposure to math or computers; or a number of aptitudes [34, 40].

Prior work argues that the highly integrated nature of the computer programming splits learners into groups that either succeed or fail [34, 40]. Computers have a multitude of components that work simultaneously to perform a single task. Programs similarly have a multitude of concepts that work simultaneously to generate the instructions for the completion of that task. The concepts must be learned serially, but no individual concept makes sense in isolation.

Since programming concepts are so integrated, a failure to understand one single concept makes it increasingly difficult to understand the next. Conversely, each idea that a student successfully grasps provides more context for reasoning about additional concepts. These students find programming concepts to be increasingly complementary and intuitive and have a greater chance of success [34, 40]. The result is one group of successful students and a second group of comprehensively confused students. This manifests in a bi-modal distribution of final grades.

Over the past seven years, the percentage of students who pass our CS0 course has been approximately 99%. Most years there are over 1,100 students taking the course. Unlike the statistics in the previous paragraphs, the AP CSP course has a passing rate of 72.7% for 2018 [4]. Passing is considered three or higher on the AP exam. The percentage to earn the lowest grade of one in 2018 is 7.6%. Because we focus on many computing topics, a student that struggles in programming does not automatically fail our course.

3 COURSE DESCRIPTION

Our redesigned CS0 course is divided into three modules: Hardware, Software, and Cyber. Table 1 enumerates the topics in each module. Each module is composed of roughly ten 75-minute lessons. The redesigned course starts with hardware then transitions to software with lessons covering operating systems, algorithms, and high level programming languages prior to programming. The course culminates with discussions on networks, data, ethics, cyber security, and cyber law & warfare.

This redesign commits much more time to hardware and cyber concepts than did previous course iterations. Over the past few years, the software module had slowly grown to give instructors more time on the topic that students struggle with the most i.e., programming. However, our direct experience from last year’s pilot argues that spending more time on hardware actually does more to help students comprehend programming concepts than do additional software lessons. We believe that ordering the hardware lessons before the software lessons, and expanding the time spent on hardware, together increase student understanding of computer programming. We intend to collect the data necessary to explore that hypothesis.

Each lesson is structured so that it begins with a short quiz on the assigned reading to check knowledge and frame the discussion [3, 22]. We discuss the lesson topic for about 20–30 minutes. Every lesson concludes with a 15-20 minute in-class activity/worked out examples (Section 2.4) when students apply or explore the lesson concepts through some interactive activity. Importantly, interactivity helps to solidify concepts and make them less abstract. For example, during the robotics lesson, we watch a robot refine its stored map as it navigates obstacles in the classroom. During our web lesson, students manipulate a chat room using HTML, Javascript, and cookies.

3.1 Key Decisions

There were a number of key decisions made in redesigning the course: to keep it a CS0 instead of a CS1 course, to use a high level language instead of a beginner language, and to ensure a large overlap with the AP CS Principles course.

3.1.1 Use AP CS Principles as a model. Our course has a large overlap with the AP CSP course. We cover approximately 84% of the learning outcomes. The lesson topics to CS Principle “Big Ideas” crosswalk are found in Table 1. See Section 2.2 for an overview of AP CSP.

The major area where our course differs from the AP course is a reduced emphasis on collaborative work (a AP CSP computational thinking practice). While there are opportunities to collaborate, graded events are almost exclusively individual. Second, we also reduced emphasis in the Data and Information “Big Idea” in favor of other topics.

There are two concepts we think are important albeit slightly outside of the scope of the AP CSP. The first is hardware; we spend quite a bit of time discussing the hardware in a computer, then use an Arduino and robots to help solidify the concept. The second important concept is Cyber Law and warfare.

3.1.2 Follow CS0 Expectations. Trying to model our CS0 course after the typical CS1 courses is not feasible. As a compulsory course for mostly non-Computer Science majors, the design of a CS0 course is, and ought to be, different from a CS1 course. Our course trades a rigorous programming education for more breadth across IT topics [1]. However, we believe that their exposure in our CS0
course sets a solid foundation for students that are interested in doing more in that domain.

3.1.3 Use Updated and Online Textbooks. We use three textbooks for our new course. The first is Understanding the Digital World [19] which covers hardware, the internet, privacy, and security. The second is How to Think Like a Computer Scientist online book (thinkcspy) on the Runestone Academy website [26]. The thinkcspy 'book' embeds course material, videos, and knowledge checks alongside executable programming challenges. This approach is well documented in [2, 27]. The third is an arduino uno that we use for labs.

The course textbook becomes stale over time (see Section 4.1). Defenses against this are readily available. Using a newer book for the non-programming portion facilitates up-to-date instruction on many of the non-programming AP CSP "Big Ideas." An on-line interactive book for programming can be modified each semester to help keep the course interactive and fresh.

3.1.4 Use of a General Purpose Language. We chose to use a general purpose language, specifically Python. Python is a popular language with a minimal syntax that is readily used to solve a variety of real-world problems. This real-world utility is limited by our choice to use a JavaScript-based Python implementation. We use JavaScript (Skulpt) so that we can integrate and assess students directly on the Runestone Academy website [26]. The thinkcspy 'book' embeds course material, videos, and knowledge checks alongside executable programming challenges. This approach is well documented in [2, 27]. The third is an arduino uno that we use for labs.

The course textbook becomes stale over time (see Section 4.1). Defenses against this are readily available. Using a newer book for the non-programming portion facilitates up-to-date instruction on many of the non-programming AP CSP "Big Ideas." An on-line interactive book for programming can be modified each semester to help keep the course interactive and fresh.

Ideally, we want the simplicity of a beginner language, such as a simple web-based editor, with the power and utility of a general-purpose language, so that students can use their knowledge more directly to solve real-world problems.

3.1.5 Ensure Interactive Feedback. Students struggle to understand how to combine the many components of a language into a meaningful program. To reduce the amount of students’ time spent on syntactic errors, we’ve configured our editor to detect and provide feedback on most syntactic errors. Programming challenges in the new course are accompanied by a battery of unit tests. This gives immediate and detailed feedback to the student each time the program is executed and reduces cognitive load [29]. For instance, if a student must define a function that adds two numbers together, the unit tests will verify that they’ve defined a function, the function is named correctly, the function takes two arguments, and the function leverages the addition operator and a return statement, and that the function returns the correct value given two numbers.

We can achieve a high-level of interactivity for programming questions through Runestone Academy’s unit tests. If unit tests have been defined for a question, then students receive immediate feedback on issues with their code. A robust set of tests for code will edge students toward solving a problem on their own.

This immediate and detailed feedback has been crucial in the early programming lessons in helping students diagnose their programs’ errors. We’ve found that these unit tests greatly reduce the time students spend randomly trying new approaches in the hope that something will work.

3.2 Transitioning to Web-Based Instruction

For the pilot course offering, we explored various web-based programming platforms. Computer programming is increasingly moving to the web. Platforms like Microsoft’s Azure Notebooks, Binder, Trinket, and CodeEnvy provide programming environments and development tools through a web browser. These programming tools have the benefit of being accessible from any Internet-connected device.

An online book eliminates the need for students to install and learn a programming tool, like JES [15]. Every student is already intimately familiar with how to use a web browser, and a web-based environment is much less intimidating for a novice programmer.

3.2.1 Use an Interactive Web-Based Programming Environment (Gadget). Our Gadget (Figure 3) incorporates the look and feel of trinket.io’s Trinket with Runestone’s Python 3/Skulpt code evaluation engine. Gadgets are hosted directly within Blackboard and, like Runestone, allow students to write and execute code within the

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
<th>Cyber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Lesson Topics</td>
<td>AP CSP &quot;Big Ideas&quot;</td>
<td>Our Lesson Topics</td>
</tr>
<tr>
<td>Computers</td>
<td>Abstraction</td>
<td>OS and File Systems</td>
</tr>
<tr>
<td>Encoding data</td>
<td>Abstraction</td>
<td>Algorithms</td>
</tr>
<tr>
<td>CPU</td>
<td>Abstraction</td>
<td>Languages</td>
</tr>
<tr>
<td>Analog to Digital Conversion</td>
<td>Abstraction</td>
<td>Variables and Expressions</td>
</tr>
<tr>
<td>Sensor Lab: Arduino</td>
<td>Creativity</td>
<td>Turtle Graphics &amp; Modules</td>
</tr>
<tr>
<td>LANs</td>
<td>Internet</td>
<td>Functions; Selection; Iteration</td>
</tr>
<tr>
<td>Robots &amp; Drones</td>
<td>Abstraction; Global Impact</td>
<td>Software Development; Debugging</td>
</tr>
</tbody>
</table>

Table 1: Our Lesson Topics Crosswalk the AP Computer Science "Big Ideas"
We hope to see an LMS with a native code execution ability in the future or another more seamless plugin framework for integrating tools in the Blackboard ecosystem, such as the Lockdown Browser.

We created a robust unit-testing framework that makes it easy for instructors to incorporate a wide variety of tests in their programming questions. Gadgets also support sending code to the Python Tutor website for visual debugging [11].

3.2.2 Addressing Limitations of Web-based Programming. Our initial pilot suffered from a severe lack of integration. The hardware and cyber modules were hosted and accessed through Blackboard but the software module was hosted and accessed through Runestone. On top of requiring a second account for students, student scores had to be regularly transferred between the two systems.

In the end, we ported the relevant lessons from How to Think Like a Computer Scientist to Blackboard and created our own inline code execution tool (see above). Integrating with Blackboard significantly reduced the overhead of teaching programming across our nearly 30 offerings of the course. Unfortunately, the integration is not native and gadgets are slow to load and don’t play well with other tools in the Blackboard ecosystem, such as the Lockdown Browser. We hope to see an LMS with a native code execution ability in the future or another more seamless plugin framework for integrating a code execution tool into a LMS.

4 STUDENT FEEDBACK ASSESSMENT

Figure 4 has three plots that show the results from the last decade of end-of-course surveys. The number of respondents per year is listed at the bottom (n). Each value statement on the end-of-course survey uses a five-value "Likert" rating scale. The scores on the rating scale, starting at one and ending at five, are: Strongly Disagree, Disagree, Neutral, Agree, and Strongly Agree. The higher the value the more a given student agrees with a particular statement. The vertical line at AY13 represents the major revision of the course from Raptor/Java to Jython depicted in Figure 1.

These results helped inform the decision to modify the course. The statement topics fall into three general categories describing: (1) textbook contribution (Figure 4a); (2) motivation (Figure 4b); and (3) confidence in being able to use programming or technology to solve future problems (Figure 4c).

The end-of-course survey value-statements are:

A6 My motivation to learn and to continue learning has increased because of this course.

C6 The textbook and readings made a major positive contribution to how much I learned.

D1 I learned to think about the implications of technology as a result of this course.

D4 If you have to learn a new piece of software that requires you to program it, how confident are you that you can use the software to solve your problem?

D5 How confident are you that you can learn and use a new piece of information technology on your own?

Our CS0 course was redesigned twice in the last decade, with pilots running in academic years 2012 and 2018. They have significantly smaller sample numbers, n = 107 and n = 16 respectively and results are annotated with pilot.

4.1 Textbook Utility

The statement with the largest downward trend for both the Raptor/Java and Jython version of the course is the one based on the perceived utility of the textbook (C6). Figure 4a illustrates the trend. Students initially enjoy a new textbook but after a year or two that level of satisfaction drastically reduces.

We cannot pinpoint the exact cause for the changes; however, there are a number of reasons students’ satisfaction with the utility of a textbook could wane over time. Some possible causes includes the following: concern over the textbook cost, or the fact that all students must take this core course, so over time the key focus areas become known and free study guides replace the textbook. Additionally, slides and in-class examples are actively incorporated and the textbook is emphasized less by the instructors [3, 22]. In a CS0 course, like ours, a textbook falls behind current technologies quickly. Due to the small sample size, the significant increase in perceived utility for the online interactive textbook for the pilot in 2018 is not considered definitive.

4.2 Motivation

Proving relevance and engendering effort for a programming course, for a History or Chinese-language major who sees minimal value beyond graduation, is challenging. Table 4b shows the results for question A6. Enthusiasm for a new version of the course initially increases and then begins to drop. From 2008-2012 the values remain relatively consistent with a slight drop in 2012. In 2013, there is a significant drop, which corresponds to a transition year. The JES focused course shows a similar trend starting in 2013. Motivation initially increases for a few years and then shows a downward trend.

The first year is likely lower because students’ older peers, who took the previous version of the course, cannot assist in study. It increases as the student body becomes more familiar with the content and then declines once the content becomes too well known. We speculate it is due to extensive familiarity with the course (as when numerous examples are available for students to cut and paste), staleness of tools, or predictability of instruction itself.
4.3 Problem Solving Using Programming or Technology

There are three survey statements related to problem-solving using programming and technology (Figure 4c). Response levels for D1 remained consistently high over the past decade and focused on awareness of technology. Today’s companies must take into account the moral and ethical issues, as well as second and third order effects of using technology. They must not be timid in learning or using new technology in their day-to-day lives. Most students have grown up in a ubiquitous technology world that likely contributes to this consistency.

Items D4 and D5 relate to students’ confidence in their skills in programming or in using technology to solve a problem. Both of those statements have shown a downward trend over the past five years. Reasons may include: relevance of the material, presentation methods, scaffolding, textbook, and tools used. Many of these problems have been discussed herein previously.

5 LESSONS LEARNED

This section summarizes the most significant lessons learned through our redesign. These lessons are discussed in details in the other sections of this paper, but we enumerate them here so that it might be a handy summary for other schools that are also looking to implement or redesign their CS0 courses:

Include programming instruction in a CS0 course. There is no consensus on the role of programming in a CS0 course. We believe that programming is a central component of computer literacy.

Include computer hardware instruction. Our initial pilot results strongly indicate that students with a firmer grasp of computer hardware concepts will better absorb programming concepts.

Keep the textbook fresh. Our student assessments over the last 10 years indicate that the value of a CS0 textbook degrades over time.

Use a general purpose language. Use Python or JavaScript, instead of a beginner language. While hiding the details of program execution makes beginner languages more accessible to novices, exposure to more challenging details may be necessary for the student to advance beyond trivial program designs [39].

Make the course web-based. If using an LMS, it is best to integrate programming. This reduces the context switch between the course material and code execution. Many coding platforms exist online so this approach doesn’t deprive students of a realistic development environment.

Ensure there is interactive feedback. Immediate and detailed feedback is crucial, especially in early programming lessons, to help students diagnose programming errors.

Use the AP CS Principles Course as a guide. Every institution is different and its courses takes a unique approach; however, the AP CSP is a standard measure of what a CS0 course should look like. A good amount of overlap is desirable.

6 CONCLUSION

Our institution has been teaching Computer Science concepts and programming to every student for over 50 years. As computing becomes increasingly relevant to other disciplines, the need for every student to have a basic understanding of computers and computer programming will only increase. While the expectations for students in a CS0 course are lower than those in a CS1 course, we believe that CS0 is a valuable and necessary component of every college student’s education. Decades of student assessment and feedback support the conclusion that a college-level CS0 course should be compulsory and grades indicate it is attainable for all.

ACKNOWLEDGMENTS

Special thanks to Dr. Suzanne Matthews for her assistance in shaping and reviewing this paper. Her efforts significantly improved the final version. The views expressed in this article are those of the authors and do not reflect the official policy or position of the Department of the Army, Department of Defense or the U.S. Government.
REFERENCES


