Challenges and Approaches to Teaching CS1 in Prison

Emma Hogan
emhogan@ucsd.edu
UC San Diego

Ruoxuan Li
ruli@ucsd.edu
UC San Diego

Adalbert Gerald Soosai Raj
asoosairaj@ucsd.edu
UC San Diego

William G. Griswold
bgriswold@ucsd.edu
UC San Diego

Leo Porter
leporter@ucsd.edu
UC San Diego

ABSTRACT

Efforts to bring incarcerated and formerly incarcerated individuals into the field of computing stand to improve equitable access to both computing jobs, and consequently the benefits of our tools and innovations through the inclusion of more diverse perspectives. This report describes the design and execution of a college level introductory computing course conducted with 26 students currently incarcerated at a prison in the United States in Fall 2022. We discuss the ways that the prison environment and the student body differ from traditional college computing classes, and how this impacted the design and execution of the course. We found that despite significant environmental barriers to learning to program, such as not having access to a code interpreter, there were unique affordances of the student population, including maturity and community, that could be leveraged in the course design and policies. We conclude with many lessons learned for the purpose of improving future offerings of computing courses in prisons.

CCS CONCEPTS

• **Social and professional topics** → **Computing Education**; **CS1**; **Adult education**.

KEYWORDS

CS1; Prison Education; Adult Learners

ACM Reference Format:
Emma Hogan, Ruoxuan Li, Adalbert Gerald Soosai Raj, William G. Griswold, and Leo Porter. 2024. Challenges and Approaches to Teaching CS1 in Prison. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630802

1 INTRODUCTION

The United States has the largest prison population in the world and a justice system that disproportionately targets those in poverty and people of color [28]. Additionally, while over 96% of incarcerated people are eventually released, over 50% return to prison within three years and 83% within nine years [5].

Education is the most effective known means of lowering recidivism rates [16]. In particular, participation in higher education programs in prison has been shown to reduce the likelihood of returning to prison by over 50% [16]. A 2016 survey of incarcerated adults found that only one in five was currently pursuing any formal degree or credential. Yet, 79% of those not currently enrolled reported interest in doing so [15, 33]. Postsecondary opportunities are even more rare [18, 28], and only a handful offer college-level *computing* courses [28]. Soon, however, more computer science educators may have the opportunity to teach courses in prisons. Postsecondary opportunities are increasing as recent changes in US policy have restored eligibility for Pell Grants to incarcerated people. Approximately 760,000 incarcerated people are newly eligible for federal funding to pursue college in prison [21].

With this emerging opportunity arises the concern that there is scant recorded experience on how best to approach the unique challenges of teaching computer science in a prison environment. Given the constraints of the prison setting, a key question is what resources and capabilities are available to the instructor and incarcerated students to help overcome those challenges.

In this report, experiences teaching an introductory programming course in an adult prison are described. With just a little guidance on adapting an introductory college computing course to the prison context [6], a course was designed with a few broad considerations in mind, and then adaptations made along the way through listening and responding to the concerns and suggestions of students. In addition to the generally unpredictable nature of working in prisons, students did not have access to Python interpreters on which they could run their code. Outcomes of the course, including class performance on the final exam and students' perspectives on the need for collaboration, are reported. Finally, we share many lessons learned which will serve to inform future offerings of computing courses in prisons.

2 RELATED WORK

2.1 Teaching and Learning in Prisons

There are well-documented nuances of teaching [11, 30] and learning [20, 28] at the college-level in prison environments. Teachers in prison have knowledge and skills gaps in adapting instructional styles to adults in prison [30] and are required to be more flexible [11]. Documented student struggles include limited access to resources, inadequate preparation in pre-requisite classes, limited choice of and available funding to pursue courses, and lack of places to study [28]. Meyer's work on identifying factors affecting student outcomes in college in prison programs found that high school credential type (having a diploma instead of a an equivalency credential) was significantly related to the most positive

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03.
https://doi.org/10.1145/3626252.3630802

student outcomes, institutional climate (students' relationship with institutional staff) was associated with educational aspirations, and both instructor and peer support were associated with positive achievement motivation [27].

Conway applied andragogy theory [23] to teaching in prison, noting the importance of adapting teaching practices to the context of teaching and learning there, where "students assume responsibility for their educations within settings that seek to constrain and limit their freedom to pursue those goals" [10]. Adult learners are more intrinsically motivated to learn, which instructors can leverage by creating a student-centered learning environment both active and engaged. In addition, adult learners typically want to understand the value of learning something before expending the effort to do so. Differences amongst incarcerated students – such as time remaining in their sentences – also impact what value they hope to gain from a course, emphasizing the importance of understanding students' goals and encouraging exploration. Lastly, adult learners typically perceive themselves as being "responsible for their own lives and decisions," which leads to a "deep psychological need to be seen by others and treated by others as being capable of self-direction" [10]. As incarceration denies self-direction, it is not surprising that it can cause a number of mental health issues. College programs present a unique opportunity to serve as a space where students can take responsibility for their own success while experiencing a level of autonomy [10] and intellectual freedom [11].

2.2 Teaching Computing in Prisons

In his 1990 experience report, Aman described the computer science major offered in prison through Wilmington College in the 1970s through the early 1990s, and the "problems and opportunities" of teaching CS in prison [6]. A full sequence of courses for the major was offered in the prison, where approximately 20% of the students majored or minored in CS. Curriculum and GPA requirements were the same for the prison program as on the main campus. The college maintained a computer lab in the prison where students were able to transcribe and run their code. Students at the prison nominally had access to the lab for three hours weekly, with further limitations created by the volatile environment. Aman noted that "compared with the open lab on the [main] campus, this amount of computer access is trivial." He emphasized this as one of the major issues for CS education in a prison setting: "students learn by necessity the importance of careful planning [of] programming projects. They simply cannot afford the luxury of composing, testing, and debugging at the keyboard." Beyond computing access, he found that the two major differences between teaching CS courses in the prison versus the main campus were students' pre-college preparation and maturity. While many students in the prison had gaps in academic preparation, their greater age and vastly different life experiences showed up in the classroom as wisdom and maturity, often generating questions that "probe to the core of a subject" [6].

Over three decades later, computing courses, as well as STEM courses in general, are scarcely available in higher education prison programs [3, 11, 19, 28]. A 2022 investigation into the experiences of incarcerated college students found that technology available to them is generally severely limited or unavailable, as well as outdated and frequently malfunctioning [9]. However, momentum is growing to expand these opportunities [3, 19, 36], and the technologies that make them possible [4, 20]. Organizations such as The Last Mile and Unlocked Labs currently offer training programs for coding skills in prisons, heavily oriented towards landing students post-release jobs in computing [1, 2]. These programs have made great strides in equipping individual prisons with computer labs and other technical infrastructure in order to conduct these programs. Our work additionally seeks to discover strategies to adapt computer science education in the great majority of facilities where this is not the case.

3 COURSE DESIGN AND ASSESSMENT METHODOLOGY

3.1 Design Experiment Methodology

As there were many unknowns about teaching CS in the prison, the authors were limited in our ability to optimize the course design for the prison environment from the start. Having never learned CS without an interpreter or been incarcerated ourselves, we also had limited insight into the struggles students would face. As a way to gather information about students' experience in the course, we incorporated weekly reflection assignments into the course, which counted for a small number of points. Reflections included a mix of multiple choice/Likert-scale questions and open-ended questions that changed every week (often pertaining to pressing issues, such as how the collaboration policy should be modified).

To help overcome these blind spots, we adopted the *design experiment* methodology [7]. In researching communities of learning in inner-city classrooms, Brown saw a need for this new research methodology "to engineer innovative educational environments and simultaneously conduct experimental studies of those innovations" [7]. She argued that using more complex methodologies, instead of avoiding multiplying confounds for the purpose of research, was a necessary trade-off to capture the richness of the classroom environment. Brown's methodology was adopted in Wolfman's work on classroom technology, echoing that the best innovations arise from users' natural patterns of behavior, and allowed feedback from instructors and students to shape the design of his tool [37].

3.2 Analysis Methods

After the course concluded, we returned to the above-mentioned reflections for a more thorough analysis. A third-party removed identifying information from all the submissions, which were then transcribed. The first two authors performed open coding on the responses pertaining to collaboration policies and the format of code examples. As a student response often contained more than one idea, we applied all labels relevant to the response. In this process, we first coded all of the responses independently, compiling independent initial code books. We then met to compare our codes, and agreed on one combined code book before moving onto a second round of coding. In the second round, we again coded all of the responses independently, using our combined code book. For both the collaboration and code example responses, we were able to reach an acceptable inter-rater reliability (IRR=0.837 and IRR=0.849, respectively) after two rounds of coding.

To assess student learning, we included the ten "benchmarking" questions detailed in Simon et al. [35] on the final exam. We report

the average scores of our students on these questions covering core topics from the course, which included code *tracing*, *explaining*, and *writing*, three types of common questions in CS1 [25].

4 COURSE LOGISTICS

Aspects of prison environments themselves limit higher education programs in prison [11], as discussed in Section 2.1. Although the course was conducted at a maximum-security male prison, it was conducted on a yard where the students had relatively more privileges, including newly debuted individual laptops for educational use and relatively more freedom to meet with one another outside of class. The demographics of these students had the nuances of the particular facility and the yard they were on. Most of the students were middle-age or older adults who had been incarcerated for a decade or more, and had little or no experience with modern computers going into the course.

Both instructors and students lose a great amount of control in a prison environment: instructors have less control over the successful running of the course, and students have less control over their ability to succeed. We note two specific occurrences as examples: 1) Nearly half of the students were prohibited from attending the first lecture, as one of the two housing units where students came from was under quarantine for Covid. As attending the lecture or watching a recording were not an option, we adjusted the course schedule to catch students up the following week. 2) Two weeks before the final exam, all student laptops were confiscated for security reasons, leaving them without access to resources available through Canvas. While we printed and delivered some of these resources to them as soon as we could, there was still a delay in their ability to access important study material.

4.1 Access to Resources

Students were given laptops immediately before the course began. Student reactions to the new technology were mixed: some excited, and many overwhelmed at first. The laptops have access to a modified version of the Canvas LMS, closely monitored by the prison, and Microsoft Office. In the months prior, it was unclear whether the prison would allow Python to be installed on the computers, which was eventually denied without explanation. In this first iteration of the course, we did not rely on Canvas for any part of the course for several reasons: student laptops regularly malfunctioned, many students were still learning the basics of using the laptops, and there was a constant threat of them being confiscated to handle security issues. All of these were related to the laptops being introduced just as we were starting the course.

4.2 Lecture, Office Hours, and Exams

Lectures occurred once per week for 2.5 hours, and there were no lab hours. While neither the students nor the instructor were able to access the internet from inside the prison, the classroom was equipped with a projector and large screen, and the instructor was able to project from a personal laptop.

Space and staff availability is a significant inhibitor to programs in prisons, so we were fortunate to have 2.5 hours per week available for office hours. It was not unusual for nearly all 26 students to come for the entire duration. On reflection assignments (Section 3.1), students reported that the most helpful activity during office hours was a long example on the board. Taking the students' advice, we began using office hours exclusively for this purpose. In presenting the example, we would take suggestions from the class to write the program on the board line by line. Then, we would visualize the running of the program by drawing a memory diagram and tracing its execution. Finally, the instructor would switch to live coding the same program with edits suggested by the students.

The course had two exams, a midterm and a final. These exams took up lecture time, as no additional time was allotted to hold either exam. Consequently, there was a strict time limit on the exam. Exams in the prison were conducted on paper, as they are in our CS1 course on campus as well. Students were allowed a sheet of notes, but exams were otherwise closed-book. Both exams included a variety of question types, including multiple choice, short answer, explain in plain English, tracing, code writing, and a stack diagram.

5 COURSE DESIGN AND ADAPTATIONS

The course is a CS1 course using Python that taught variables, loops, conditionals, functions, lists, and strings. The course fulfilled the computing requirement toward the undergraduate degree in sociology, the only bachelor's degree offered at this prison so far. All 26 students in the bachelor's program were enrolled in the course. Best practices from the computing literature [8] were included in the teaching of the course, including Peer Instruction [12, 31], Live Coding [34], and other forms of active learning such as in-class worksheet exercises [17].

As described in Section 3.1, our approach to designing the course was modeled off of Brown's "design experiment" methodology [7], relying heavily on students' feedback and articulation of their learning experiences to guide course design decisions. We were also committed to making changes to adapt as soon as we became aware of an issue or potential solution in the interest in student success. This methodology facilitated more rapid discovery of best practices (and mistakes to avoid) for teaching CS1 in prisons, as presented in this paper, and enabled us to ground course design decisions in the experience of the students.

5.1 Adaptations to No Code Interpreter

With the decision by the prison to not allow Python to be installed on the laptops approximately three months before the start of the course, we recognized an opportunity to explore the potential benefits of alternative teaching and learning methods. A challenge in standard classrooms is that students have access to interpreters which allow for a hack-until-it-works mentality, whereas recent work emphasizes the importance of teaching conceptual understanding early in a programming course [13, 14]. This restriction by the prison became an opportunity to stress conceptual understanding. We decided to structure the course assignments to split the grade percentage that would normally be just programming assignments between code-based programming assignments and problem sets that tested conceptual understanding using paper-based activities such as tracing, short answer, and fill-in-the-blank questions. We also recognized the importance of the in-person instruction time, as live coding [34] in lecture was the only chance students

had to experience code being run. Below, we describe other major adaptations given the lack of a code interpreter.

5.1.1 Handwriting Code. Since students could not run code on their laptops, as well as their unpredictable availability, we decided to have students hand-write code for programming assignments. Later in the course, we made it an option to submit typed code (in a Word document) on Canvas, but only a handful of students chose this option and many were more comfortable with hand-writing. In order for students to see the output of their code (and adjust based on errors or unexpected output), the instructor typed and ran their handwritten code, printed the output, and returned this to the students. The instructor picked up and dropped off printed output with comments 2-3 times per week total.

5.1.2 Grading. We transcribed the students' code exactly as it was written, including all errors. Students were held to the same standard for their code as the CS1 students on our campus and only received full points for fully working code. However, we decided it was necessary to give partial credit, given the limited ability for students to debug their code. To this end, we created detailed rubrics for each programming assignment. Additionally, we accumulated a running list of common errors (described in section 5.1.3). In grading, we counted the total number of errors and took off a certain number of points per error that was a much smaller deduction than missing a key element of the program defined in the rubric items.

5.1.3 Common Errors Resource. A problem that quickly arose with the system of dropping off printed output was the extremely limited amount of information students could gather from this output, only showing the first error to occur. The students needed a way to get more information. One option was to provide detailed written feedback on all errors in the code, not just the one showing in the output. However, this was infeasible due to the sole instructor of the course being the only grader. Our solution was to create a running list, which we called "Common Errors" .[1] Each time an error occurred in a code submission, we added examples of this specific error to the list, along with an explanation of the error and how to fix it. By the end of the course, this list had 47 errors.

We believe the Common Errors resource was ultimately a success. Along with students' feedback on reflections and course evaluations that this was a helpful resource, it served multiple purposes in the course. First, it was a more feasible way to provide necessary additional feedback for students to debug more than one error per submission. Instead of providing written feedback for each error, we could point to that numbered description in the assignment. Second, while we do not have confirmation that it had this effect, we intended for this to lessen the emotional toll on students [22] of seeing errors in their code by showing that they were not alone.

5.2 Course Policies

We began the course with essentially the same course policies we had in our CS1 course on campus. However, we quickly came to realize (with the help of the students) that some of these policies did not work in the prison. The two major areas of change in course policy were homework re-submissions and collaboration.

[1]The Common Errors resource can be found here: https://perma.cc/CXH3-MBKG

	Type	Primary Q. Topics	Difficulty	Avg
Q1	Trace	Compound conditionals	easy	92.0%
Q2	Trace	Variable assignment	easy	40.0%
Q3	Write	Swap variables	medium	59.1%
Q4	Trace	Conditionals	easy	84.0%
Q5	Trace	Complex conditionals	easy	52.0%
Q6	Trace	Loop, conditional, and lists	medium	24.0%
Q7	Explain	Loop and conditional	easy	24.0%
Q8	Explain	Loop	hard	16.7%
Q9	Write	Loop and lists	medium	27.9%
Q10	Write	Loop and lists	medium	42.9%

Table 1: Benchmarking questions from Simon et al. [35] and the average scores from the students in the class.

5.2.1 Mastery Learning. Mastery learning in CS1 is both theoretically supported [29], and necessary in the prison where students have limited attempts to run code before submitting. In our CS1 courses on campus, students are given a single deadline for their programming assignments. Given our students' lack of access to a Python interpreter, we adapted this policy and allowed students unlimited re-submissions of their programming assignments. We had initial concerns that this might cause students to procrastinate, but we found that our students were eager for any feedback they could receive and consistently turned in assignments on time and kept submitting until they were correct. This was an overall positive adjustment we will keep for future offerings.

We used the midterm exam as another opportunity to incorporate mastery learning into our course. Students were given the opportunity to submit corrections on their Midterm after the fact, and receive credit back. Partial credit was awarded similar to the programming assignments (Section 5.1.2).

5.2.2 Collaboration Policy. After beginning the course with a strict no-collaboration policy on programming assignments, students raised valid concerns in class and on reflection assignments (Section 6.2.1). In response, we decided to try an open policy allowing collaboration on all assignments.

6 COURSE OUTCOMES

6.1 Student Performance

Results of the benchmarking questions [35] are shown in Table 1, including the type of each question, topics involved in the question, question difficulty[2], and results from the most recent class. Some trends emerged in these findings: First, students did better on questions that stressed a single concept rather than multiple concepts. This is a common challenge in computer science assessments, as questions must often combine multiple concepts [32]. Second, students did poorly with code explaining questions. These questions ask students to take a block of code and explain its purpose. This is a common practice employed by software developers but is challenging for novices [25]. Third, the students performed reasonably well on code writing questions despite having no access to a Python interpreter all term.

[2]Question difficulty is determined based on the average student performance reported in Simon et al. [35], where if students in that study earned greater than 80% we denote this as an "easy" question, 60-80% as "medium", and less than 60% as "hard".

6.2 Student Reflections

6.2.1 Collaboration Policies. By the third week of the course, it became apparent that a change would have to be made to the collaboration policies for the students to succeed. On the next reflection assignment we asked students what they would make the collaboration policy if they were teaching the course, and why.

The majority of responses said that they would allow increased collaboration (N=10): making all work collaborative (N=2), more of the work collaborative (N=5), or stating that collaboration was necessary for them to succeed (N=3). While some student responses acknowledged that some individual work was still important, no response requested keeping the collaboration policy as is.

A major argument for allowing more collaboration was the lack of access to outside resources, tutors, or the instructor outside of class time (N=4). Several students added that it was particularly difficult to grasp the concepts in an environment with limited technology (N=5), with one student writing: "The material is new to all of us, the concept is alien to most." Other students talked about the usefulness of running their code by classmates to "debug" in the absence of a code interpreter (N=3), writing: "Because we do not have access to the Python App ... we cannot troubleshoot our code, however as a group we can help each other to troubleshoot." Similarly, students wrote about going to each other as a means of getting "unstuck" (N=2). Multiple students talked about the culture amongst the students of helping each other (N=4), saying "when a few people get it, they help others," and some students characterized this as "pooling" or "concatenating" their knowledge (N=2). Many students said that collaboration promoted better learning overall (N=5), comparing their understanding of the concepts (N=2) and resolving misconceptions (N=1). Others mentioned the benefits of hearing things explained a different way by classmates (N=2), and one student noted in particular how students with learning disabilities may struggle to pick up concepts during lecture that could be resolved through collaboration with peers. Finally, students wrote that collaboration with classmates guided new understanding (N=2), and gave students more equal opportunities to succeed (N=1).

6.2.2 Student Preferences for Presentation of Code Examples. In the fifth week of the course, we asked students which format they preferred for presenting code examples during lecture and office hours. Initially, we had intended for students to compare live coding with static coding examples as we typically define them in CS education literature. However, many students interpreted this more generally about which format (including writing code on the white board, pre-written code examples in slides, and projecting live coding from the instructor's computer) they preferred to see code examples. These results are reported in Table 2, showing that the most popular responses were writing code on the white board, or a combination of both the white board and live coding.

Students who preferred code examples to be written on the white board gave reasons including a) the white board was helpful for breaking down code (N=2), b) the students were used to the white board (N=1), c) it was hard to read code on the screen (N=1), d) they liked seeing code written out step-by-step on the white board (N=3), e) the white board was used in previous math courses to show proofs (N=1), and f) seeing code written on the white board was easier to follow (N=1) and visualize (N=3).

7 DISCUSSION

7.1 Lessons Learned

Leverage the Strong Student Community. In addition to the adult students being mature and self-sufficient, there was a deep sense of collective effort to succeed —often born out of situations of collective struggle— that shaped a thriving learning community. On reflection assignments, students often wrote in terms of "we" as opposed to "I". Students formed study groups in the limited space and time available to them outside of class (access to a room where they could meet was limited to 6-8 hours per week), where they described having struggling students write their code on the white board, and they would together draw a memory diagram and trace through the code to help identify the errors. Higher performing students would walk to the other side of the classroom during Peer Instruction activities to sit with students who were struggling.

Make Students a Resource for Each Other. Sometimes there were only a few students who picked up a specific topic well during lecture. In the absence of outside resources, students relied on each other to fill in the gaps. It seemed that no student was left behind, and that overall there were many positive relationships formed between lower- and higher-performing students. However, the collaboration was difficult to control, given this dependency on others for information outside of the limited lecture and office hour time. This was likely a sizable factor in the majority of students struggling to write code independently on the exams.

Code Resubmission Policies are Critical. One of the most significant hurdles for the students was the absence of a code interpreter to test their programs before submitting. In addition to making it extremely difficult to have programs work on the first few tries, students were also robbed of the critical learning that happens through debugging. Our approach was to offer unlimited resubmissions on programming assignments, and encourage students to take advantage of this by "debugging" their code based on the output and instructor feedback on the original submission.

Mix Live Coding with Long Examples on the Board. Live coding during lecture had a unique significance in the prison classroom, as this was the only opportunity for students to actually see code run in real time. Still, students had a strong desire for code being written on the board as well. What worked best was to write out a long example program first on the board, then trace its execution in a memory diagram, and finally project its execution in a code interpreter (as described in Section 4.2).

Use Relevant Examples. For this first offering, we used programming assignments similar to those in the CS1 at our university. Some students commented that they would like the assignments to be more relevant to their life experiences. Andragogy advises that adult learners want to understand the value of learning something before putting in the effort to do so (Section 2.1), and encourages the incorporation of a greater variety of prior life experiences compared to younger learners. Existing frameworks such as culturally relevant pedagogy both acknowledge and leverage students' diverse backgrounds in the classroom [24], and have been effective amongst K-12 students from underrepresented groups in CS [26]. In the future, we plan to incorporate these theories into our practice by creating examples for lectures and assignments that are culturally relevant to adult students in prison.

Preferred Method of Presenting Code Examples				
Code	N	%	Properties	Examples
white board	7	46.7%	1. Prefer writing code "live" on the white board	"Writing on the white board is easier because what we are used to is the white board, and breaking down code point by point allows me to grasp each part and put it together"
pre-written	4	26.7%	1. Prefers pre-written code examples (on slides)	"I prefer the pre-written code examples. It seems that once we switch to "live" it starts to get confusing. This may be because we are not using the computer ourselves."
live coding	2	13.3%	1. Prefers live coding (projected on screen)	"Writing code 'live' works better for me...it kinda slows everying down a bit such that I'm better able to digest the material"
combination	8	53.3%	1. Prefers a combination of white board and live coding	"I like both methods. Going through prewritten code gives me the opportunity to really focus on the code. Writing live code gives me the opportunity to see the execution"
no preference	4	26.7%	1. Both methods are equally effective 2. No preference, or left blank	"Both were instrumental and effective in enhancing learning."

Table 2: Qualitative coding results for open-ended reflection question about preferred method for presenting code examples.

Make Use of Students' High Engagement in Lecture. Peer instruction worked particularly well during lectures. Students were not only cooperative in gathering with their groups to discuss, but they would also often self-organize such that high-performing students moved to sit with lower-performing students. Students were overall fully engaged in productive discussion about differing opinions. After the fact, some students wrote on their course evaluations that they found this part of the lecture particularly helpful.

Students are Skilled Independent Learners. Similar to the impact of students' maturity discussed by Aman [6], many students in the prison had much experience with self-teaching during their incarceration. For example, correspondence courses are a common way for incarcerated students to pursue higher education, and require an immense amount of discipline and independent learning. Given the the lack of any reliable communication with the students outside of class, and having no tutors/TAs, we have seen the importance of providing students additional resources for learning outside of class. This includes textbook readings or other written resources, as well as uploading video tutorials when possible (given students' varying comfort with and access to the laptops).

Create Opportunities for Self-Expression. In addition to allowing programming assignment resubmissions, we also offered students the opportunity to submit any other code written for practice or for fun. Similar to the programming assignments, we would transcribe the code, print the output, and deliver this back to them. Many students, both higher-performing and lower-performing, took advantage of this opportunity throughout the quarter to submit programs that were often personal in nature and creatively expressive. In future iterations of the course, we plan to explore how we can encourage this creative expression more.

7.2 Limitations and Future Work

There are many possible contributors to our results, particularly student performance on the exam. While these findings provide us with areas of improvement for our next offering, the benchmarking questions might have been easier for students in Simon et al. [35] than for our students in this course, perhaps due to limitations of this first offering of the course. We observed that exams seemed to be particularly emotional and stressful for students in the prison. In fact, multiple students would leave entire sections incomplete, later citing this as a result of mental and emotional reactions to the exam setting itself on weekly reflections. While many students who are not in prison also experience exam stress, we believe that this deserves special consideration in the prison setting. Students in prison are more likely to have failure and trauma as part of their past educational experiences, as well as exposed to a highly-stressful environment in the prison. In the future, we plan to consider ways we can reduce the stress of exams in the course, as well as including more comprehensive ways of assessing students' performance.

Prison environments are idiosyncratic by nature, and ours was no exception. Available resources, rules and restrictions differ across correctional facilities, and will also change over the course of time as educational opportunities expand. That being said, many prisons share some or all of the restrictions described here. In the future, we plan to study programming courses in different facilities, as well.

8 CONCLUSION

The expansion and improvement of computing opportunities in prison college programs has the potential to benefit not only incarcerated students, but also the advancement of the computing field and society as a whole. In this report, we described our experience teaching CS1 in a college in prison program. Obstacles included students not having access to code interpreters or outside resources. However, students in the prison were skilled independent learners, and found ways to simulate the debugging process in study groups outside of class. In addition, we discovered several strategies to adapt to the limitations of the environment —with the guidance of students' feedback throughout the course— such as combining live coding with writing code examples on the white board, and annotating code submissions using a list of common errors to make the debugging process more feasible with fewer opportunities to run code. Finally, we share our takeaways from this experience that will be used to improve future offerings of the course in prison.

9 ACKNOWLEDGMENTS

This work is supported by the Ford Foundation Fellowship and NSF Award #2315909. We thank Dr. Jim Aman for his valuable feedback, and for continuing to graciously share his unique wisdom with us.

REFERENCES

[1] [n.d.]. The Last Mile – Paving The Road To Success. https://thelastmile.org/

[2] [n.d.]. Unlocked Labs. https://www.unlockedlabs.org/

[3] 2020. STEM Opportunities in Prison Settings (STEM-OPS). https://stemforall2020.videohall.com/presentations/1801

[4] 2022. *Building the Technology Ecosystem for Correctional Education: Brief and Discussion Guide.* Brief. Office of Career, Technical, and Adult Education, U.S. Department of Education. https://lincs.ed.gov/sites/default/files/tech-ecosystem-correctional-ed.pdf

[5] Mariel Alper, Matthew R Durose, and Joshua Markman. 2018. *2018 Update on Prisoner Recidivism: A 9-year Follow-up Period (2005-2014).* US Department of Justice, Office of Justice Programs, Bureau of Justice. https://bjs.ojp.gov/content/pub/pdf/18upr9yfup0514.pdf

[6] James R Aman. 1990. Computer science in correctional education. *ACM SIGCSE Bulletin* 22, 1 (1990), 147–151.

[7] Ann L Brown. 1992. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The journal of the learning sciences* 2, 2 (1992), 141–178.

[8] Neil CC Brown and Greg Wilson. 2018. Ten quick tips for teaching programming. *PLoS computational biology* 14, 4 (2018), e1006023.

[9] Erin L. Castro, Caisa E. Royer, Stephanie Gaskill, and Estefanie Aguilar-Padilla. 2022. *"It's Useless, to Put it Politely": Experiences with Technology Among Incarcerated Students Receiving Second Chance Pell at Four Institutions.* Brief 9. Collaborative for Higher Education Research and Policy, The University of Utah. https://cherp.utah.edu/projects/pell_is_not_enough.php#publications-slide

[10] Patrick Filipe Conway. 2022. Andragogy in prison: Higher education in prison and the tenets of adult education. *Adult Education Quarterly* 72, 4 (2022), 361–379.

[11] Trevor Craft, Nicholas Gonzalez, Kevin Kelleher, Miki Rose, and Ofu Takor. 2019. *A Second Chance: College-in-Prison Programs in New York State.* Technical Report. Nelson A. https://eric.ed.gov/?id=ED605777 Publication Title: Nelson A. Rockefeller Institute of Government ERIC Number: ED605777.

[12] Catherine H. Crouch and Eric Mazur. 2001. Peer instruction: Ten years of experience and results. *American Journal of Physics* 69 (2001).

[13] Quintin Cutts, Matthew Barr, Mireilla Bikanga Ada, Peter Donaldson, Steve Draper, Jack Parkinson, Jeremy Singer, and Lovisa Sundin. 2019. Experience Report: Thinkathon - Countering an 'I Got It Working' Mentality with Pencil-and-Paper Exercises. *ACM Inroads* (2019).

[14] Quintin Cutts and Maria Kallia. 2023. Introducing Modelling and Code Comprehension from the First Days of an Introductory Programming Class. In *Proceedings of 7th Conference on Computing Education Practice* (Durham, United Kingdom) *(CEP '23).* Association for Computing Machinery, New York, NY, USA, 21–24. https://doi.org/10.1145/3573260.3573266

[15] Lois M. Davis. 2019. Higher Education Programs in Prison: What We Know Now and What We Should Focus on Going Forward. (2019).

[16] Lois M. Davis, Robert Bozick, Jennifer L. Steele, Jessica Saunders, and Jeremy N. V. Miles. 2013. *Evaluating the Effectiveness of Correctional Education: A Meta-Analysis of Programs That Provide Education to Incarcerated Adults.* Rand Corporation.

[17] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary P Wenderoth. 2014. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences of the United States of America* (2014).

[18] Stephanie Gaskill, Mary R. Gould, Ved Price, Erin L. Castro, and Amy Lerman. 2023. *The Landscape of Higher Education in Prison, 2020-2021.* Technical Report. Alliance for Higher Education in Prison. http://higheredinprison.org

[19] Jo Hardin, Karl Haushalter, and Darryl Yong. 2020. Turning STEM Education Inside-Out: Teaching and Learning Inside Prisons. (2020).

[20] Susan Hopkins and Helen Farley. 2014. A Prisoners' Island: Teaching Australian Incarcerated Students in the Digital Age. (2014). https://doi.org/10.25771/4021 Publisher: Universtity of Bergen Library.

[21] The White House. 2023. FACT SHEET: Biden-Harris Administration Takes Action During Second Chance Month to Strengthen Public Safety, Improve Rehabilitation in Jails and Prisons, and Support Successful Reentry. https://www.whitehouse.gov/briefing-room/statements-releases/2023/04/28/fact-sheet-biden-harris-administration-takes-action-during-second-chance-month-to-strengthen-public-safety-improve-rehabilitation-in-jails-and-prisons-and-support-successful-reentry/

[22] Paivi Kinnunen and Beth Simon. 2010. Experiencing programming assignments in CS1: the emotional toll. In *Proceedings of the Sixth international workshop on Computing education research.* 77–86.

[23] Malcolm S Knowles, Elwood F Holton III, and Richard A Swanson. 2014. *The adult learner: The definitive classic in adult education and human resource development.* Routledge.

[24] Gloria Ladson-Billings. 2021. *Culturally Relevant Pedagogy: Asking a Different Question.* Teachers College Press.

[25] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between Reading, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the Fourth International Workshop on Computing Education Research.* ACM.

[26] Tia C Madkins, Alexis Martin, Jean Ryoo, Kimberly A Scott, Joanna Goode, Allison Scott, and Frieda McAlear. 2019. Culturally Relevant Computer Science Pedagogy: From Theory to Practice. In *2019 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT).* IEEE, 1–4.

[27] Stephen J. Meyer. 2011. Factors Affecting Student Success in Postsecondary Academic Correctional Education Programs. *Journal of Correctional Education* 62, 2 (2011), 132–164. https://www.jstor.org/stable/23282667 Publisher: Correctional Education Association.

[28] Stephen J. Meyer, Linda Fredericks, Cindy M. Borden, and Penny L. Richardson. 2010. Implementing Postsecondary Academic Programs in State Prisons: Challenges and Opportunities. *Journal of Correctional Education* 61, 2 (2010), 148–184. https://www.jstor.org/stable/23282637 Publisher: Correctional Education Association.

[29] Claudia Ott, Brendan McCane, and Nick Meek. 2021. Mastery Learning in CS1 - An Invitation to Procrastinate?: Reflecting on Six Years of Mastery Learning *(ITiCSE '21).* 18–24.

[30] Nicole Patrie. 2017. Learning to be a Prison Educator. (2017). https://doi.org/10.25771/5528 Publisher: Universtity of Bergen Library.

[31] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A Multi-institutional Study of Peer Instruction in Introductory Computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.*

[32] Leo Porter and Daniel Zingaro. 2014. Importance of Early Performance in CS1: Two Conflicting Assessment Stories. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education.* Association for Computing Machinery.

[33] Bobby D Rampey, Shelley Keiper, Leyla Mohadjer, Tom Krenzke, Jianzhu Li, Nina Thornton, and Jacquie Hogan. 2016. Highlights from the US PIACC Survey of Incarcerated Adults: Their Skills, Work Experience, Education, and Training–Program for the International Assessment of Adult Competencies: 2014. NCES 2016-040. *National Center for Education Statistics* (2016).

[34] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1.* ACM.

[35] Simon, Judy Sheard, Daryl D'Souza, Peter Klemperer, Leo Porter, Juha Sorva, Martijn Stegeman, and Daniel Zingaro. 2016. Benchmarking Introductory Programming Exams: Some Preliminary Results. In *Proceedings of the 2016 ACM Conference on International Computing Education Research.* ACM.

[36] Julie E. Speer and Zain Clapacs. 2022. Creation of a Novel Biomedical Engineering Research Course for Incarcerated Students. *Biomedical Engineering Education* 2, 2 (Sept. 2022), 157–165. https://doi.org/10.1007/s43683-022-00071-6

[37] Steven A Wolfman. 2004. *Understanding and promoting interaction in the classroom through computer-mediated communication in the classroom presenter system.* Ph.D. Dissertation. Citeseer.