The CS1 Python Bakery: A Modern "Batteries Included" Open-Source Curriculum With All The Fixings

Austin Cory Bart
acbart@udel.edu
University of Delaware
Newark, DE, USA

Megan Englert*
megan.englert@colorado.edu
University of Colorado Boulder
Boulder, CO, USA

John Aromando
jaro@udel.edu
University of Delaware
Newark, DE, USA

Hye Rin Lee
hyerin@udel.edu
University of Delaware
Newark, DE, USA

Teomara Rutherford
teomara@udel.edu
University of Delaware
Newark, DE, USA

ABSTRACT

Despite rising enrollment, CS Education struggles with training adequate educators, leading to increased teaching loads. Open-source teaching materials alleviate this by streamlining course preparation. Yet, there is a scarcity of free, open curricula that offer a contemporary coding experience while covering CS fundamentals. To address this gap, we introduce the CS1 Python Bakery curriculum with a "Batteries Included" approach, aiming to furnish instructors with comprehensive teaching resources. This curriculum refines an earlier open-source CS1 with detailed lesson plans, slides, rubrics, reference answers, student answers, and more. We present the learning content in a cross-platform, autograded textbook format and embrace modern Python features such as Dataclasses and static types. We deployed the curriculum in multiple university CS1 courses and collected data on the tradeoffs of our approach. This paper offers a thorough self-assessment based on student learning outcomes, code snapshot analyses, and reflection via the TEC Rubric for curriculum evaluation. Although we improved teacher accessibility, the change in student learning outcomes was unexpectedly minimal. Recognizing room for advancement, we conclude with recommendations for our next iteration to emphasize Equity, Community, and Identity.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; **Model curricula**; **CS1**.

KEYWORDS

Python; Instructional Design; Open Curriculum; Bakery

*Englert was at University of Delaware at the time of the work; is now at the University of Colorado Boulder.

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07 https://doi.org/10.1145/3649217.3653630

ACM Reference Format:
Austin Cory Bart, Megan Englert, John Aromando, Hye Rin Lee, and Teomara Rutherford. 2024. The CS1 Python Bakery: A Modern "Batteries Included" Open-Source Curriculum With All The Fixings. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653630

1 INTRODUCTION

For the past decade, enrollments have increased in Computer Science but the number of Computer Science educators has remained relatively stagnant [15]. This inconsistency strains teachers as they are forced to teach more courses for more students, often with relatively little preparation or training. In the secondary education space, teachers report the urgent need for updated curricular resources [16]. Open Source curricular materials are one source of these resources, but a curriculum must be more than an archive of slides or a textbook – instructors need a wide range of materials. We have previously created a curriculum that attempts to alleviate this problem, which has seen some success and adoption [2]. This paper describes our next iteration of that curriculum, making a number of updates particularly for the teacher experience. The key contributions of this paper are as follows:

(1) The description of the **Bakery** curriculum and its tooling,
(2) The analysis of data from offerings of the Bakery curriculum,
(3) Rigorous self-reflection on future work needed.

We begin with the theoretical grounding of our approach and identify some related curriculum. Next, we describe the latest iteration of our materials, focusing on updates. Then, we evaluate the new version against the old version for various metrics to gauge the changes' impact. Finally, we suggest the direction of our next iteration, in hopes of eventually yielding more improvements.

2 THEORIES AND RELATED WORK

Our approach is based on Design-Based Research (DBR), well-known in the field of education. Those new to DBR may refer to [4]. To summarize, DBR has several crucial principles:

(1) Development follows a repetitive cycle of design, intervention, data collection, and analysis.
(2) Interventions must be viewed within their original context.
(3) Documentation and sufficient context should accompany every development stage for reproducibility and replication.

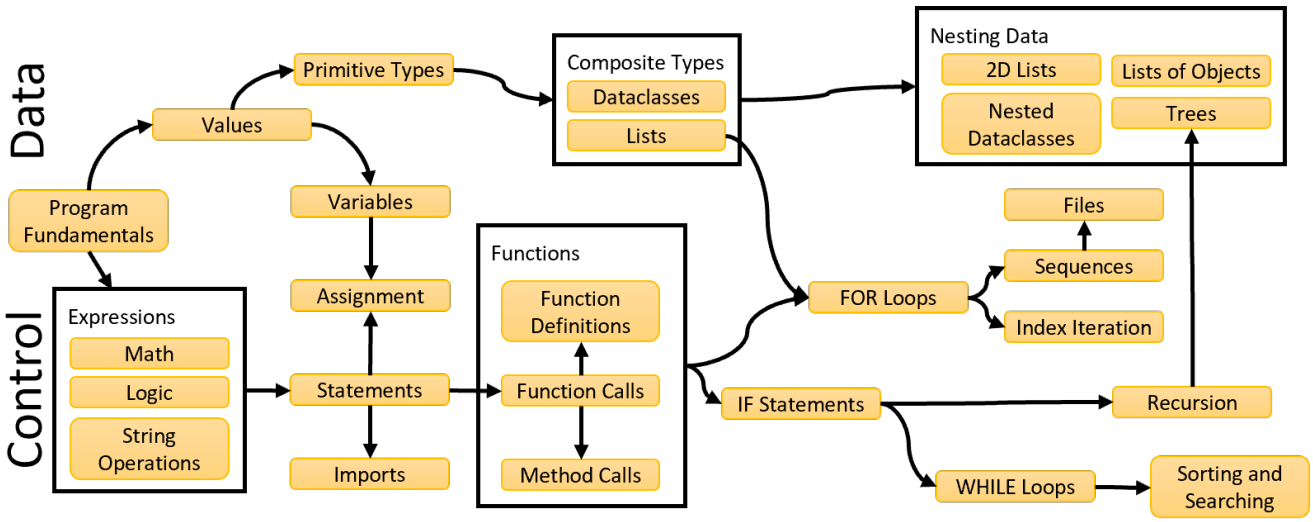

Figure 1: Curriculum Topics Layout

(4) The creation of learning experiences and the development of learning theories are intertwined.
(5) Outcomes from an intervention should guide the subsequent cycle and be shared with a wider audience.

Inherent to the DBR process is a "messy authenticity," which restricts the generalizability of findings. Because of this, any conclusions are context-dependent and should not be assumed to be universally applicable. DBR can lead to the development of new learning theories; this is not our ambition within the current paper. Instead, we apply DBR as a way to iteratively improve our curriculum for larger and larger audiences. Nelson and Ko (2018) have argued that Computing Education research should focus on applying principles like those in DBR [13] to avoid the tension between the goals of theoretical work and design work.

In addition to DBR, our work draws upon a number of important modern educational theories including Active Learning, Self-efficacy, and Mastery Learning. Active Learning suggests that students learn best when they are actively engaged in the learning process, rather than being passive recipients of information. Integrating active learning strategies, such as hands-on coding exercises, structured group problem-solving (e.g., POGIL [12]), and peer instruction [20], can foster a deeper understanding of programming concepts and enhance the overall learning experience [17].

The theory of Self-Efficacy posits that individuals' beliefs about their capabilities to perform specific tasks influence their motivation, effort, and persistence. Fostering self-efficacy is crucial, as students with a stronger belief in their coding and problem-solving abilities are more likely to engage deeply, persist through challenges, and achieve higher levels of mastery in computer science concepts [14]. Mastery learning is an educational philosophy where students are expected to achieve a high level of understanding in a specific topic before moving on to the next, ensuring that foundational concepts are truly grasped. Mastery Learning seems to be more effective for more novice coders, probably has a small (but positive) effect on learning, and definitely seems to help with student motivation [8]. In our curriculum, students get many attempts at material until mastery is demonstrated. Many other theories influenced the design of our course, in small and large ways; this is merely a sample of some of the biggest influences.

2.1 Other Curriculum

When we began the original version of this project, there were few alternatives for an open-source CS1 Python curriculum (see [2]). Since then, a number of paid and free options have appeared on the scene as solutions for teaching introductory computing in Python (CodeHS, Code.org). However, relatively few strive to be an open-source collection of instructor materials, and not just a collection of slides or readings from a textbook. One of the most obvious alternatives is the CMU CS Academy[19], which is available for multiple audiences and grade levels. Reviewing their materials, we believe that we do have some contextual differences in our approaches. They are focused on K-12 education, whereas our curriculum is a more oriented towards first-year college and university students (although both curricula seem to be appropriate for the other audience). They also seem to have a stronger emphasis on student-facing materials, whereas we are a little more concerned with providing what instructors need. Finally, their focus is on "creativity and problem-solving," whereas our approach has students generally completing most problems the same way. Still, they have done tremendous amounts of work to provide professional development, detailed alignment with state standards, and many other resources that instructors need. We share their belief in combining modalities, given that we do both provide opportunities for graphic manipulation, interactive animations, and data processing. Both curricula rely on autograding technology and streamlined web-based delivery of material. There are many micro-differences in our approach, such as the order of topics or the emphasis of certain concepts. We believe that this healthy diversity provides more options and will eventually lead to a stronger understanding of the

effectiveness of various models. We limit our review of alternative curricula, given the dynamic nature of the space.

3 THE CURRICULUM

Our curriculum is the next generation version of the PythonSneks curriculum [2], with similar goals and principles. Briefly summarizing, the original goals were: 1) materials should be documented and made externally available; 2) modern topics should be taught with modern approaches; 3) the course must scale while fitting within the university model. The major innovation of this version of the curriculum (justifying the change in name and branding) can be summarized as an additional guiding principle: 4) the course should provide all the ingredients an instructor needs to quickly pick up and teach each day. Additional changes have been made to the curriculum, including further modernization of the topics, improvements to the pedagogy, and upgrades to the technical workflow.

3.1 Components

Our central site[1] hosts links to the following components:

- **Canvas Student Course Template**: 15 weeks of content, broken up into digestible pieces, in a Canvas template.
- **Canvas Staff Course**: A parallel Canvas course for training, managing, and guiding the course staff with training materials and daily complete lesson plans.
- **Digital Textbook** 8 chapters of interactive textbook material including transcribed videos, runnable code examples, low-stakes quizzes, and autograded programming problems
- **Bakery Google Drive** 12 weeks of lecture and lab content, including instructor slides, worksheets, reference answers, grading rubrics, and examples of common student mistakes.
- **Poll Everywhere Folder** Dozens of in-class lecture questions to warm up students and keep their attention.

A static mirror of these resources is available as a repository utilizing plain-text file formats[2], to avoid locking the materials into specific platforms, as described in Edmison et al [6]. However, adopters can also just clone our two Canvas courses–one for staff (where teaching assistants or other managed staff are enrolled as students) and one for students. The staff course provides all the instructions necessary for setting up our curriculum and running all of its activities each week. The course also contains a wealth of training materials delivered through autograded quizzes to help onboard new teaching assistants, synchronize them on course policies, and give them some actual teacher training. These training materials are fully described elsewhere in [7]. Ideally, instructors can simply check the lesson plan the day before and find everything they need to run the activity.

The student course provides all of the students' assignments organized into weekly modules. First and foremost, this course delivers the 56 individual lessons, which are composed of readings, autograded comprehension quizzes, and 159 coding problems. Figure 1 gives a birds-eye view of all the topics in the course, while 2 shows the breakdown of the textbook material into parts. Each weekly module corresponds to a chapter of the textbook, with each chapter broken into two equally-sized parts–students are expected

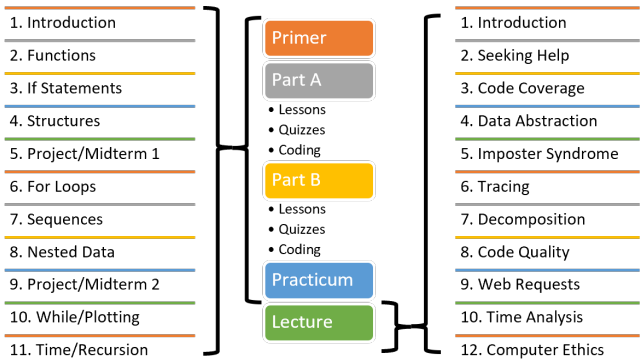

Figure 2: Weekly Textbook Layout

to complete the first part in the first half of the week, and the rest in the second half of the week. All of these assignments are embedded in Canvas through LTI via BlockPy [3]. We have also worked with Microsoft to release a free version of the curriculum on Visual Studio Code for Education [3], a new web-based coding environment. The course delivers the weekly lecture and practicum (lab) assignments, which are described in section 3.2.

During two of the weeks, and in the final third of the course, students are assigned larger-scale coding projects. Our latest iteration of these projects has their basic specification fully autograded, but are intended to also be graded with a rubric by a human for code quality. The final project is significantly larger than the other two projects and is spread out over multiple weeks, punctuated regularly by milestones to ensure sufficient progress along the way.

3.2 Classroom Activities

Our curriculum was designed around the belief that the time spent in the classroom is the most valuable time teachers have, and should therefore be spent developing a learning community and instilling higher-order values to improve student motivation and belonging [5, 9]. Figure 2 shows the weekly breakdown of topics covered in the Lecture and Practicum. We distinguish between lecture (classtime run by the instructor) and practicum (classtime that may need to be run by teaching assistants, usually in a lab) to provide flexibility for different course structures. The practicum activities typically review the textbook material for that week, serving as an opportunity to reinforce that material. Meanwhile, the lecture activities are related but distinct from the textbook. For example, module 3 of the textbook covers `if` statements, whereas the week 3 lecture is about code coverage and the power of unit testing.

The usual structure of lecture is as follows:

(1) Warm-up questions via clickers or Poll Everywhere.
(2) 20- to 30-minute presentation on topic with more interactive polling questions.
(3) 20- to 30-minute group work activity where students complete a Google Doc-based worksheet or a collaborative coding problem.
(4) 10- to 20-minute walkthrough of the answers to the worksheets, as time allows.

[1] https://python-bakery.github.io/
[2] https://github.com/python-bakery/bakery-curriculum-public
[3] https://vscodeedu.com/

Many of the activities are about developing students' abilities to work in groups, sense of community, to dissect ethics, to prevent imposter syndrome, and otherwise develop their so-called "softer" skills (which we suspect are often more important than their coding skills, in the long run [11]). There are also a lot of high-level ideas that students may not fully understand at this point in their career: good variable naming, the power of writing good tests, how to decompose complicated functions. Although these are concepts that might only take root once students have more significant coding experience, we feel there is value in exposing them now to prepare them later.

3.3 Teacher Guide

A major feature of our revision is the upgrade of our teacher guide to a more holistic resource. In the initial version of the curriculum, the guide more like a basic ingredient list, mostly a collection of half-baked lectures and under-cooked activities. As we taught with colleagues and reused the curriculum ourselves across semesters, we better understood the pain points that potential adopters faced. We aimed to make the new version a complete cookbook that enumerates exactly what an instructor does to teach their class.

Each week of the course has dedicated lesson plans detailing the lecture activity, practicum activity, and the textbook lessons. Each lesson plan begins with a summary, background explanation/justification of the content knowledge, learning objectives, expected student prior knowledge, and explanation of the formative assessment of the activity (describing how you will know that students have learned). The lecture and practicum have links to the forkable slides, the worksheets, the reference solution, grading instructions, grading rubrics, and example student answers. For the textbook lessons (and a few of the lectures) that have coding assignments, links are provided to the reference solution and example student answers. In both cases, suggestions are made about how to help students overcome common misconceptions and "gotchas" in the curriculum. A thorough walk-through is provided for each step of the class activities with time estimates and suggestions for how to restructure the activity for different weekly schedules (e.g., courses that meet three times a week for 55 minutes vs. two times a week for 80 minutes). Warm-up questions for Poll Everywhere (or similar mass response systems) are provided for all lectures to review previous material or to introduce the day's activities.

3.4 Textbook Authoring

Our textbook was written as Markdown files with a regular structure: headers followed by a content block and text. This allows us to export the entire curriculum in a friendly web-based format and also as slides (where each slide is titled by a header and shows the content block). Those slides, in turn, can be synchronized to prerecorded voice-over files and exported as captioned, multimedia video files. To streamline our workflow, we developed a script to generate the slides and videos[4] and a web-based tool to record our narration[5]. The advantage of this approach is that alternative voices can be created for the curriculum and minor edits can be made without needing to re-record entire videos. Currently we provide a masculine and a feminine voice, with plans to eventually record more. The text of the book was proofread by a K-12 teacher and a professional copy-editor, both for content and for typos.

[4]https://github.com/python-bakery/bake-mark
[5]https://github.com/python-bakery/bake-dubs

	E (Engineers)		R (CS Majors)	
	%	#	%	#
In Engineering Major	92%	305	1%	1
In Non-CS/Engi. Major	8%	27	28%	40
Identifies as Woman	22%	74	24%	34
White Student	78%	260	57%	81
Black Student	8%	26	9%	13
Asian Student	10%	34	31%	44
Formal Prior Experience	37%	122	64%	91
Informal Prior Experience	13%	44	17%	24
No Prior Experience	50%	166	19%	27
Total Students	100%	333	100%	142

Table 1: Demographics of the Fall 2022 Course Offerings

	E1	E2	E3	R
Number of Students	145	139	51	144
Percentage of Students	30%	29%	11%	30%
Gender of Instructor	Woman	Man	Man	Man
Senior Instructor	No	No	Yes	Yes
Taught Curriculum Before	No	No	No	Yes
Weekly Schedule	TR	MWF	MWF	TR

Table 2: Details across Course and Sections for Fall 2022

4 COURSE OFFERING DATA

The second contribution of this paper is an evaluation of our materials from actual usage data. We previously analyzed our curriculum in [1]; this paper replicates some of those metrics to compare the original version with the new version. The curriculum was deployed at the University of Delaware in Fall 2022 as part of a 15-week semester in two different courses. The first course ("R") was a CS1 intended for Computer Science majors; the second course ("E") was a CS1 intended for Engineers. The demographics of students are in Table 1. The Engineering course was much larger and taught by three different instructors, as shown in 2. Exploring outcomes across the Engineering instructors, we did not find statistically significant differences, so that course is analyzed as an aggregate. However, students in the R course performed statistically significantly better than the E course, so the data was kept separate between courses. All data was collected under an IRB-approved protocol. Students were required to complete all activities but chose whether their data could be analyzed. The consent response rate was 75% for the E course and 81% for the R course.

4.1 Basic Course Outcomes

Figure 3 shows the distribution of exams scores and the final project score between the two courses. Compared to the 2019 course offering, final project scores were much higher, especially for the R section; the final project that semester had suffered from a number

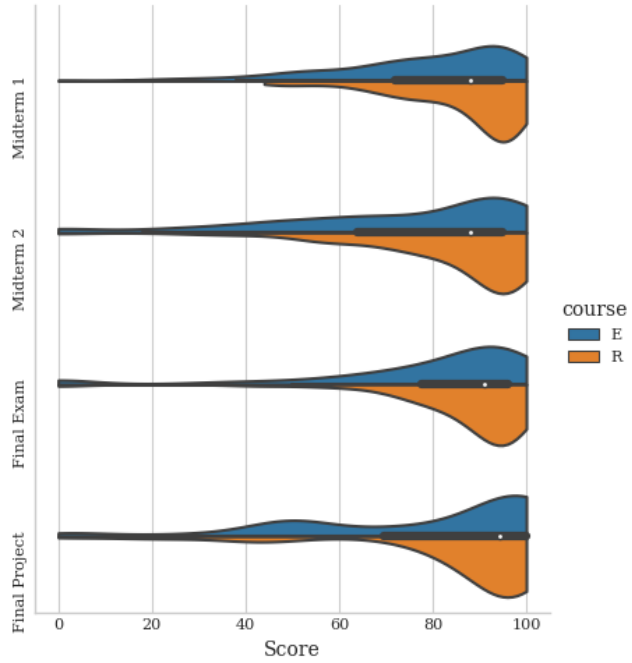

Figure 3: Distribution of Exam and Project Scores between Courses

	Failed		Passed	
Minimum Hours Per Week	**#**	**%**	**#**	**%**
<1 hours	53	11.3%	126	26.9%
1-2 hours	23	4.9%	161	34.4%
<3 hours	7	1.5%	62	13.2%
3+ hours	2	0.4%	34	7.3%
Total	85	18.1%	383	81.8%

Table 3: Final Exam Outcomes vs. Minimum Time Spent Coding Each Week

of pedagogical mistakes (described in [1]), so the large improvement is less impressive with context. The bimodal shape of the final project distribution for the E course is explainable by an unfortunate cheating incident that involved almost a quarter of the students (students were blatantly sharing code for the entire final project); we responded in subsequent semesters by strengthening anti-cheating measures, which mitigated the issue.

Table 3 compares the pass/fail rates of the final exam with the average minimum time spent each week by students on the coding problems (as calculated from the log data). The rate for the final exam was 81.8%, similar to the overall pass rate for the classes (79.1% overall, 91.5% for the R course, and 74.8% for the E course). This is a mixed result compared to the previous iteration of the course, which had an 85.5% pass rate overall. For students who put in at least one hour a week, the pass rate was 88.9%. However, the minimum amount of time spent did not have a statistically significant relationship with the number who earned an A.

4.2 Function Subskill Analysis

In the analysis of our previous iteration, we focused on a particular set of seven subskills related to functions (because it would be intractable to publish results on the more than one hundred learning objectives). We analyzed the abstract syntax trees of student code submissions on the exams' coding problems to detect the absence or presence of specific misconceptions. Table 4 shows the result of this analysis (higher values are better). In the end, most students were able to correctly define a valid header but most failed to *attempt* to provide *any* header types. Most also correctly avoided overwriting parameters, using the `input` function instead of parameters, or `printing` instead of `returning`. However, only half of students tried using unit tests on the final exam, and only a tiny fraction ever attempted to decompose their functions. For most subskills, the data are not significantly different from the previous iteration of the course. A notable exception is the use of decomposition, which unfortunately decreased even further–virtually no students demonstrated functional decomposition on the final exam (although the ones that did were twice as likely to complete those problems). Although there were some statistically significant differences between course iterations on the second midterm, the actual differences were relatively small in most cases. For example, the percentage of students who could define a syntactically correct header increased by about 8%, while the number of students who overwrote parameters decreased by about 6%. By and large, the function subskills score were the same as the original version of the course. Given that we expected the improvements to the curriculum materials to lead to more dramatic improvements in student learning, this is a disappointing outcome and requires further intervention. Despite improvements in the curriculum targetting these specific functional subskills, we still see students failing to apply decomposition, testing, and types!

4.3 Evaluation with the TEC Rubric

Given most of our innovations were to improve the teacher experience, changes in student outcomes are not a definitive evaluation. To further evaluate our materials, we used the TEC Rubric on both the new Bakery materials and the PythonSneks materials [21]. The TEC (Teacher Accessibility, Equity, and Content) Rubric was designed by Weintrop et al to evaluate curriculum like ours, and has proven to be a useful tool for checking over our materials. As shown in Table 5, Teacher Accessibility improved from 1 to 2, likely owing to the superior lesson plans and additional material about students' common misconceptions. Content improved from 2 to 3, reflecting the improvements in aligning the content with standards and the learning objectives with assessment rubrics. However, the Equity category remained shockingly low. These results were not entirely surprising, given that we did not use TEC in our design, so it may be a less-than-perfect evaluation of our particular changes. Additional work is needed to understand differences in instructor experiences between versions (e.g., interviews, targeted surveys).

5 DISCUSSION OF RESULTS

By and large, we are not unhappy with the raw numbers from the original version of our curriculum or our redesign–most students seem to succeed overall, especially if they invest the time. We seem

	Pretest		1st Exam			2nd Exam			3rd Exam		
Subskill	**E-F22**	**R-F22**	**R-F19**	**E-F22**	**R-F22**	**R-F19**	**E-F22**	**R-F22**	**R-F19**	**E-F22**	**R-F22**
Defined Header Correctly	78.9%	75.7%	83.5%	85.5%	91.5%	84.5%	92.1%	93.6%	91.3%	88.8%	91.4%
Provided Header Types	85.%	88.6%	40.8%	59.5%	63.4%	45.6%	60.9%	53.6%	37.9%	38.1%	37.1%
Did Not Overwrite Parameters	96.9%	97.1%	88.3%	88.8%	90.8%	98.1%	91.2%	98.6%	99.%	94.2%	96.4%
Used Return, Not Print	86.5%	82.1%	80.6%	88.5%	83.8%	89.3%	87.4%	83.6%	91.3%	93.6%	93.6%
Used Parameters, Not Input	74.3%	76.4%	96.1%	98.8%	99.3%	100%	100%	100%	99.%	99.7%	100%
Wrote Unit Tests	0%	0%	88.3%	87.9%	93.7%	79.6%	78.5%	76.4%	67.%	60.3%	56.4%
Decomposed Function	0%	0.7%	1.0%	0%	0.7%	17.5%	6.9%	12.9%	19.4%	2.2%	6.4%
Total Students	327	140	103	331	142	103	317	140	103	312	140

Table 4: Functional Subskill Analysis across Exams and Course Offerings

Category	**Bakery**	**Old Curriculum**
Teacher Accessibility	**2**	**1**
Teacher Support	Adequate	Inadequate
Supplemental Materials	Adequate	Adequate
Equity	**0**	**0**
Culture (Community-Level)	Inadequate	Inadequate
Identity (Individual-level)	Inadequate	Inadequate
Exceptionalities	Adequate	Adequate
Content	**3**	**2**
Computer Science Content	Extensive	Adequate
ID - Pedagogical Practices	Extensive	Extensive
ID - Content	Extensive	Extensive
Theme	Extensive	Extensive
Assessment	Extensive	Adequate

Table 5: Self-Evaluation with the TEC Rubric

to have found a largely successful model for Computer Science majors in particular. However, we were disappointed that we did not see improvement in the functional subskills, which we had invested considerable curricular time to covering. The data suggest we did not particularly improve from previous semesters. Why did our changes not significantly improve the learning outcomes? We offer a few hypotheses.

An important, uncontrollable factor is that the 2022 data were collected after the global COVID pandemic ravaged the world's educational and societal structures, which may have led to students being less prepared than the original 2019 cohort. If we had conducted pretests back in the original iteration of the course, we would have a better sense of students' entering skill level. However, the lack of statistically significant differences between cohorts on the first midterm exam suggests that the students were unlikely to be that different in terms of ability; that does leave the possibility of other skills (e.g., self-regulation) being affected.

And indeed, even ignoring COVID, natural fluctuations in the entering class could have had a greater impact than we expected. Further differences between instructors, sections' time of day, and other unknown factors could have impacted our ability to get a comparable sample. There are always more controls possible that would help spotlight an effect; but as we expected a large effect, these other factors seem unlikely to have completely removed an effect that was actually there.

Therefore, an obvious possibility is that our approach is simply limited and needs to be reassessed. Better pedagogical techniques, more targeted interventions, and novel ideas might lead to the improvements we were hoping for. In particular, the TEC rubric suggests that we are failing to engage students in an important dimension related to their cultural, community, and personal identity, important for both learning and motivation [10].

We should not ignore the possibility, though, that we are simply hitting an upper limit on what is feasible in a CS1 course. The Rainfall problem (a correlate to our final exam problem) has defeated Computer Science Educators for decades [18]; it is unlikely that it would be solved just because we switched from dictionaries to dataclasses. Still we remain hopeful that we simply haven't found the right approach. We aim to try some more radical approaches, and are eager to see whether following the suggestions of the TEC rubric might lead to a meaningful, large improvement.

We do not feel that our inability to match all the rubric criteria is a condemnation of our work, but instead helpful guidance for our next iteration. As an open-source, transparent effort, the project invites criticism. The coarse rubric grading does over-simplify the situation; our self-evaluation was that our materials do support some individual-level Identity aspects (particularly in terms of providing an authentic and meaningful context for students, and providing opportunities for students to contribute their knowledge and perspective). However, the rubric helped us identify that, regardless of their skills with functions, we must focus more on providing opportunities for students to connect their cultural backgrounds and various communities (home, neighborhood, etc.) to the content. We believe that there is ample opportunity within the structure that we have created to provide those kinds of opportunities.

6 CONCLUSION

This paper has presented the evolution of our previous CS1 Python curriculum and described some evaluation of its course offerings. We saw substantial gains in teacher accessibility and sustained high learning outcomes. However, we saw little to no improvement in certain subskills of particular interest related to functions. In light of our findings, we propose directions for our next iteration to focus on integrating more content about equity, community, cultural, and identity. Regardless, we remain committed to the goal of providing modern, easy-to-use open source curricular materials for CS1 Python.

REFERENCES

[1] Austin Cory Bart, Teomara Rutherford, and James Skripchuk. 2020. Evaluating an Instrumented Python CS1 Course.. In *CSEDM at EDM*.

[2] Austin Cory Bart, Allie Sarver, Michael Friend, and Larry Cox II. 2019. PythonSneks: an open-source, instructionally-designed introductory curriculum with action-design research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 307–313.

[3] Austin Cory Bart, Javier Tibau, Eli Tilevich, Clifford A Shaffer, and Dennis Kafura. 2017. Blockpy: An open access data-science environment for introductory programmers. *Computer* 50, 5 (2017), 18–26.

[4] Design-Based Research Collective. 2003. Design-based research: An emerging paradigm for educational inquiry. *Educational researcher* 32, 1 (2003), 5–8.

[5] Jacquelynne S Eccles and Allan Wigfield. 2020. From expectancy-value theory to situated expectancy-value theory: A developmental, social cognitive, and sociocultural perspective on motivation. *Contemporary educational psychology* 61 (2020), 101859.

[6] Bob Edmison, Austin Cory Bart, and Stephen H Edwards. 2021. A Proposed Workflow For Version-Controlled Assignment Management. *Seventh SPLICE Workshop at SIGCSE 2021* (2021).

[7] Megan Englert, Lecia Barker, and Austin Cory Bart. 2024. Hiring, Training, and Managing Undergraduate Teaching Assistants for Large CS1 Classes. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*. 1636–1637.

[8] James Garner, Paul Denny, and Andrew Luxton-Reilly. 2019. Mastery learning in computer science education. In *Proceedings of the Twenty-First Australasian Computing Education Conference*. 37–46.

[9] Carol Goodenow. 1993. Classroom belonging among early adolescent students: Relationships to motivation and achievement. *The Journal of early adolescence* 13, 1 (1993), 21–43.

[10] 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.

[11] Gerardo Matturro, Florencia Raschetti, and Carina Fontán. 2019. A Systematic Mapping Study on Soft Skills in Software Engineering. *J. Univers. Comput. Sci.* 25, 1 (2019), 16–41.

[12] Chris Mayfield, Sukanya Kannan Moudgalya, Aman Yadav, Clif Kussmaul, and Helen H Hu. 2022. POGIL in CS1: Evidence for Student Learning and Belonging. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*. 439–445.

[13] Greg L Nelson and Amy J Ko. 2018. On use of theory in computing education research. In *Proceedings of the 2018 ACM conference on international computing education research*. 31–39.

[14] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. 2004. Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 171–175.

[15] Eric Roberts, Tracy Camp, David Culler, Charles Isbell, and Jodi Tims. 2018. Rising cs enrollments: Meeting the challenges. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 539–540.

[16] Olgun Sadik and Anne Todd Ottenbreit-Leftwich. 2023. Understanding US secondary computer science teachers' challenges and needs. *Computer Science Education* (2023), 1–33.

[17] Kate Sanders, Jonas Boustedt, Anna Eckerdal, Robert McCartney, and Carol Zander. 2017. Folk pedagogy: Nobody doesn't like active learning. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 145–154.

[18] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do we know how difficult the rainfall problem is?. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. 87–96.

[19] Mark Stehlik, Erin Cawley, and David Kosbie. 2020. Cmu cs academy: A browser-based, text-based introduction to programming through graphics and animations in python. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1420–1420.

[20] Cynthia Taylor, Jaime Spacco, David P Bunde, Andrew Petersen, Soohyun Nam Liao, and Leo Porter. 2018. A multi-institution exploration of peer instruction in practice. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. 308–313.

[21] David Weintrop, Merijke Coenraad, Jen Palmer, and Diana Franklin. 2019. The Teacher Accessibility, Equity, and Content (TEC) Rubric for Evaluating Computing Curricula. *ACM Transactions on Computing Education (TOCE)* 20, 1 (2019), 1–30.