2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #39391

WIP: Skip the Lecture: A Decoding First Approach to Introductory Computing Education

David Zabner, Tufts University
Trevion S Henderson, Tufts University

Trevion Henderson is Assistant Professor of Mechanical Engineering at Tufts University. He earned his Ph.D. in Higher Education at the University of Michigan.

(WIP) Skip the Lecture: A Decoding First Approach to Introductory Computing Education

David Zabner
david.zabner@tufts.edu

Trevion Henderson
trevion.henderson@tufts.edu

Abstract

Existing research suggests introductory computer programming courses (i.e., CS1) constitute a significant barrier for students' entry into computer science and related disciplines. For example, extant literature suggests as high as 33% of students fail or drop out of introductory computer programming courses [1], precluding these students from pursuing computer science education in college. The purpose of this work in progress is twofold. First, we describe the development and implementation of a pedagogical approach to computing education that draws and expands on the Use-Modify-Create (UMC) framework [2, 3] to support students' learning in introductory computer programming courses. Second, drawing on unique qualitative data sources, such as student-produced lists of Python's rules, students' code annotations produced during the "use" phase of UMC, and ethnographic fieldnotes, we describe our pilot of a mixed-methods study examining students' sense-making, competency, and computing self-efficacy. Our preliminary examination of rules notebooks indicated that some students developed code tracing strategies and that most were able to discover many of the rules that underlie Python's runtime execution.

1 Introduction

Computing educators must balance several, at times competing, pedagogical and learning goals. For example introductory computer programming courses must foster students' learning about fundamental computing concepts, like iteration, variables, and recursion, as well as prepare students for professional practice by developing learning activities that resemble real-world computing work. Thus, it is important that instructors teaching introductory computer programming courses, which can serve as either deterrents or inspiration to continuing computing education, draw on frameworks that support these various goals.

Active learning, a broad category of learning activities that entails anything students participate in beyond passive, instructor-centered, lecture-based learning activities, [4] has long been studied as a method of improving student engagement, self-efficacy, and learning outcomes in STEM courses. This research is concerned with inquiry based learning (IBL), defined as a pedagogical approach where students "arrive at an understanding of the subject matter by engaging in self-directed investigations [5]."

Inquiry based learning pedagogies are common in STEM education and have been studied in many science and engineering education contexts. Existing research on IBL has found that these pedagogies improve student affect, self-efficacy, and academic performance [5–7]. Still, inquiry learning approaches remain understudied in computer science education, especially at the college level.

In this research study, we draw on IBL strategies in the process of teaching students new programming languages. To do so, we also draw on second language acquisition literature (SLA), which we contend shares many parallels with the learning of new programming languages. Specifically, we combine inquiry learning pedagogies with elements of Krashen's "Input Hypothesis" [8] to teach an introductory

programming course. In this work in progress paper, we focus on the theoretical framework guiding our teaching in an introductory computer science course, as well as share a preliminary examination of data from the pilot phase of a study of the course. Finally, we discuss preparations for the study phase of the research.

2 Theoretical Framework

2.1 Inquiry Based Learning

Inquiry-based learning (IBL) is a type of active learning in which students are encouraged to pose and investigate questions as the primary form of learning [5]. IBL draws inspiration from the scientific method and has been primarily studied as a pedagogical and curricular approach in science classes. The foundations of IBL pedagogy lies in encouraging students to learn about a scientific field by scientific means [5].

Research suggests IBL strategies offer students ownership of knowledge by allowing them to choose the phenomena they want to explore, as well as the questions they wish to answer [9]. While student agency is thought to be an important mechanism underlying the benefits of IBL strategies, offering students agency in the classroom can prove challenging since instructors often value control over the learning environment, and research suggests offering unassisted opportunities for inquiry may undermine learning goals [10, 11]. However, in IBL pedagogies, instructors tacitly shape the ways students ask questions by carefully constructing the environment for student learning activities, and through providing feedback to direct student learning.

Pedaste and colleagues [12] describe IBL as a 5-part cycle: Orientation, Conceptualization, Investigation, Conclusion, and Discussion. In the orientation phase the instructor introduces a general topic and students begin to observe some phenomena. In the conceptualization phase students begin to explore questions they have about the phenomenon and make predictions. In the investigation phase students undertake a large variety of activities in order to try to answer their questions and test hypotheses. In the conclusion phase students grapple with and celebrate the results of their investigations before finally sharing with their classmates in the discussion phase.

Research has found that the process of inquiry-based learning is rarely linear [12]. As a result, students participating in IBL learning activities are likely to be seen rapidly moving between the phases, and the length of each of the phases is likely to vary. IBL may involve multi-day experiments with formal data collection, or it may involve students less formally testing many small hypotheses over the course of a few minutes. Still, IBL contrasts to the primary form of instructor-led education in computer science - direct instruction - wherein instructors primarily lecture on the material they expect students to learn.

2.2 Krashen's Input Hypothesis

The input hypothesis, a hypothesis about second language acquisition we believe is applicable to programming language learning, states that progression in a learner's comprehension of a second (natural) language happens as a result of striving to understand input in that language that is just above the learner's current level. Importantly the input hypothesis argues that language input (e.g., reading, hearing) to the learner is the primary driver of learning and that output (e.g., writing, speaking) is an effect of that learning. Two of Krashen's claims, as explained by Lichtman and Vanpatten [13] are of particular interest to this research. The first is that mental representations of language are constructed by learners through implicit learning processes as they work to comprehend a new language. The second is that the most important data for language acquisition is in the "communicatively embedded comprehensible input that learners receive (p. 296)" and that comprehension of a language must come before production of that language.

We contend that these hypotheses are also applicable to the learning of programming languages in computer science education. These theoretical claims suggest that students might learn programming lan-

guages best by attempting to comprehend messages (i.e., programs) directed to them in those languages, as they do during the conceptualization and investigation phases of IBL. Thus, pedagogical practices that focus too heavily on learning the rules of formal languages (e.g., Python, C++, Java) might undermine students' comprehension and application of these languages in introductory computer programming courses.

The second claim suggests to us that the limiting factor in student acquisition of a programming language is the amount of code students read and understand. If the principal data for the acquisition of a programming language is found in comprehensible code, students' ability to apply a programming language might be predicated on the quantity of code they have seen and attempted to comprehend. Thus, we developed pedagogical strategies that position students to view and attempt to comprehend code prior to activities wherein they produce their own code—the Use-Modify-Create approach.

2.3 Use-Modify-Create as IBL for CS Education

Use-Modify-Create (UMC) is a CS-specific pedagogical approach introduced by Lee and colleagues in 2011[14] for use with K-12 students. They proposed a three-stage model for learning to code. In the first stage, students first *used* instructor designed code. In the second stage, students *modified* existing code. Finally, after gaining the skills to code their own projects, students responded to programming problems by *creating* their own code.

We argue that the UMC approach constitutes three complementary cycles of IBL in introductory CS education, where the subject of students' inquiry are the rules of programming languages, such as programming syntax and semantics. For example, just as IBL approaches begin with orientation activities, UMC begins from instructor-scaffolded learning activities that make use of pre-written code in structured tasks.

Other phases of IBL—the conceptualization, investigation, conclusion, and discussion phases—are manifested across the use-modify-create model in specific student activities (Table 1).

UMC	IBL	Student activity
Use	Orientation	Read and run pre-written code
	Conceptualization	Develop questions about what is happening in the program and why
	Investigation	Attempt to answer the questions generated in the conceptualization phase often by running and rerunning the code
	Conclusion	Document new coding rules
	Discussion	Discuss discovery with nearby students and instructors
Modify	Orientation	Read and run pre-written code
	Conceptualization	Develop hypotheses about how to achieve changes to code behavior
	Investigation	Delete, edit, copy, or move code to affect desired outcome
	Conclusion	Examine success or failure of code change and document new coding rules
	Discussion	Share solutions with nearby students and instructors
Create	Conceptualization	Understand or invent a goal for a new piece of code
	Investigation	Create code to accomplish the goal. Test, debug, and retest
	Conclusion	Examine success or failure of code in accomplishing the goal and update understanding of coding rules
	Discussion	Share solutions with nearby students, instructors, and the whole class

Table 1: Mapping of IBL and UMC

3 Course Design: Adapting UMC for the College Context

In this research, we have redesigned an introductory Python-based computer programming course for first-year engineering students. Because of this context we opted to develop a curriculum that builds from Python's basics towards data analysis using NumPy, Pandas, and Matplotlib. We organized the code students see in the course, as well as the amount of code students see each week, to align with the UMC and IBL framework guiding this research. In the first three weeks of the course, we asked students to focus the majority of their in-class time on using code in order to annotate and explain its behavior. Starting in week 2, we began to ask students to modify code, progressively increasing the portion of the days' activities that involved modifying code over the next 3 weeks. Starting in week 3, we asked students to implement simple functions, which entailed modifying existing code that we provided. The overlapping of the Use, Modify, and Create stages was chosen with the goal that starting in week 3 students would be using a new concept in code, modifying code that used a recently used concept, and creating code using concepts with which we hoped they were comfortable. Only in the course's final project did we ask students to begin coding from a blank page. Our curriculum design was also informed by the fact that programming language acquisition takes time. For this reason we aimed to introduce new coding concepts for annotation several weeks before asking students to create code that used those concepts.

4 Methods

4.1 Research Setting

The setting for this study is one section of an introductory computer programming course at a private university in the Northeast of the United States. Participants in the study were primarily first-year engineering students with little experience coding in Python. The course consisted of two types of content: (a) UMC content that positioned students to participate in IBL activities and (b) projects designed to catalyze students' sociotechnical thinking by integrating coding with broader social issues. The section studied had 30 students, of which 10 (6 female, 4 male) agreed to participate in our study.

4.2 Data Sources

We collected two forms of data over the course of the pilot phase of this research. First, the lead author documented observations of learning activities in field jottings. Second, we asked students to document their learning in Python "rules notebooks" and annotations on coding assignments. The rules notebooks are explained to students as a note-taking tool to distill their understanding of Python. For students they are meant to act as personal documentation of Python's syntax and semantics. In this research, rules notebooks were a window into students' conceptions of Python's syntax and semantics. We view the rules notebooks as reflective activity germane to the conclusion and discussion phases of IBL.

5 Data and Preliminary Analysis

Our analysis of the data from this course is ongoing, and we are collecting more data on a second iteration of the course. This work-in-progress paper reports on preliminary review of rules notebooks as well as written assignments and observations from the course. Preliminary analysis of students' rules notebooks indicated that, while students did not consistently utilize the notebooks past the first 5 weeks of the course, submissions pointed to the investigation strategies students used to understand Python syntax and semantics. For example, tracing tables, wherein students traced variables in their code, were a common fixture by which students developed an understanding of code functionality.

Students also documented their emerging understanding of Python syntax rules, keywords, and other concepts in their rules notebooks during the use and modify stages of UMC process, which we also documented in fieldnotes. For example, the quote below, taken from a student's rules notebook, illustrates

the types of conclusions students drew about a program's behavior on their first interaction with the language:

> * Some words are color coded. I realized that green text is indicative of text that is not involved in the code, I think the black text denoted a particular word that can be then referred to later in the code. Blue text words are 'action' words that carry with them a certain function such as print that allows a phrase to be written and input that indicates that the user must submit information into the code to be processed. Red text seems to be indicative of text that is to appear in the code directly such as print (What is your name?), where "print" will be in blue and "What is your name" will be in red.
>
> * Programs must be defined by a function.

Still, evidence in students' rules notebooks pointed to the ways their understanding grew and shifted during the course. For example, one student wrote "Every command or print out message must be in quotation marks" in the first week. In the second week the student updated the rule to read, "String data must always be in quotations." Although the word 'string' was something that came from discussions with teaching staff, the understanding that string literals are created with quotation marks was something the student discovered as part of the inquiry process. This combined with their success on coding assignments is clear evidence that students were learning to code and raises questions about how the UMC approach may shape students learning, particularly in comparison to more standard approaches.

Our analysis of in-class observations, as documented in fieldnotes, and student's coding assignments has so far focused on finding areas of student difficulty with the curriculum. Many of our findings have led to changes in activity design (i.e. clarification of instructions, changes in pacing, improved topic sequencing). We have also found evidence that our students were comfortable, by the end of the semester, working in unfamiliar programming languages and libraries.

6 Conclusions and Future Work

Existing research suggests that the degree to which course activities are scaffolded in IBL pedagogies is an important factor for supporting students' learning [11]. Based on our preliminary review of data from the pilot phase, we determined that further scaffolding was necessary to realize the promise of IBL in introductory programming courses. As a result, we developed further scaffolding for students' learning by adapting several learning and reflection activities in the course.

For example, during the orientation phase, we ask students to document questions, new rules, code examples, and other information they discover during the class activities. We also offer students the last 5 minutes of class to update their rules notebooks, and remind students throughout class sessions to document their thinking as a reflective activity. Moreover, whereas we reviewed rules notebooks only at the conclusion of the course during the pilot phase, we now conduct weekly reviews of rules notebooks to understand patterns of misunderstanding, using these and other observations to responsively develop later class activities.

In the conceptualization and investigation phases, during which we offer students previously generated code examples, learning activities now include guided questions pointing students to explore specific concepts (e.g., syntax, data structures, error messages) as well as to report on their understanding of those concepts in open ended responses.

Our future work will continue to study the results of applying this pedagogical strategy. We will collect more data, including surveying students to measure self-efficacy and other indicators of student affect and collect high quality audio and video recording in order to explore the benefits and drawbacks of this approach. Our goal is to more fully understand the utility of UMC approach in introductory computer science courses.

References

[1] J. Bennedsen and M. Caspersen, "Failure rates in introductory programming," *SIGCSE Bulletin*, vol. 39, pp. 32–36, Jun. 2007. DOI: 10.1145/1272848.1272879.

[2] I. Lee *et al.*, "Computational thinking for youth in practice," en, *ACM Inroads*, vol. 2, no. 1, pp. 32–37, Feb. 2011, ISSN: 2153-2184, 2153-2192. DOI: 10.1145/1929887.1929902. [Online]. Available: https://dl.acm.org/doi/10.1145/1929887.1929902.

[3] N. Lytle *et al.*, "Use, Modify, Create: Comparing Computational Thinking Lesson Progressions for Stem Classes," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19, New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 395–401, ISBN: 978-1-4503-6895-7. DOI: 10.1145/3304221.3319786. [Online]. Available: https://doi.org/10.1145/3304221.3319786 (visited on 10/04/2021).

[4] R. M. Felder and R. Brent, *Teaching and Learning STEM: A Practical Guide*, en. John Wiley & Sons, Mar. 2016, Google-Books-ID: 1Qh0CgAAQBAJ, ISBN: 978-1-118-92581-2.

[5] A. W. Lazonder, "Inquiry Learning," in *Handbook of Research on Educational Communications and Technology*, J. M. Spector, M. D. Merrill, J. Elen, and M. J. Bishop, Eds., New York, NY: Springer, 2014, pp. 453–464, ISBN: 978-1-4614-3185-5. DOI: 10.1007/978-1-4614-3185-5_36. [Online]. Available: https://doi.org/10.1007/978-1-4614-3185-5_36.

[6] J.-M. G. Rodriguez, K. H. Hunter, L. J. Scharlott, and N. M. Becker, "A Review of Research on Process Oriented Guided Inquiry Learning: Implications for Research and Practice," *Journal of Chemical Education*, vol. 97, no. 10, pp. 3506–3520, Oct. 2020, ISSN: 0021-9584. DOI: 10.1021/acs.jchemed.0c00355. [Online]. Available: https://doi.org/10.1021/acs.jchemed.0c00355.

[7] V. R. Vishnumolakala, S. S. Qureshi, D. F. Treagust, M. Mocerino, D. C. Southam, and J. Ojeil, "Longitudinal impact of process-oriented guided inquiry learning on the attitudes, self-efficacy and experiences of pre-medical chemistry students," *QScience Connect*, vol. 2018, no. 1, p. 1, Aug. 2018, ISSN: 2223-506X. DOI: 10.5339/connect.2018.1. [Online]. Available: https://www.qscience.com/content/journals/10.5339/connect.2018.1 (visited on 02/27/2023).

[8] S. D. Krashen, *Second language acquisition and second language learning* (Language teaching methodology series). Oxford: Pergamon Pr, 1985, ISBN: 978-0-08-025338-1.

[9] C. D. Clayton and G. Ardito, "Teaching for Ownership in the Middle School Science Classroom: Towards Practical Inquiry in an Age of Accountability," *Middle Grades Research Journal*, vol. 4, no. 4, pp. 53–79, 2009, ISSN: 1937-0814. [Online]. Available: https://www.proquest.com/eric/docview/61816490/B1DD9F08E30F48EFPQ/2.

[10] A. R. Cavagnetto, B. Hand, and J. Premo, "Supporting student agency in science," *Theory Into Practice*, vol. 59, no. 2, pp. 128–138, Apr. 2020, ISSN: 0040-5841. DOI: 10.1080/00405841.2019.1702392. [Online]. Available: https://doi.org/10.1080/00405841.2019.1702392.

[11] Á. Suárez, M. Specht, F. Prinsen, M. Kalz, and S. Ternier, "A review of the types of mobile activities in mobile inquiry-based learning," *Computers & Education*, vol. 118, pp. 38–55, Mar. 2018, ISSN: 0360-1315. DOI: 10.1016/j.compedu.2017.11.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131517302397.

[12] M. Pedaste *et al.*, "Phases of inquiry-based learning: Definitions and the inquiry cycle," *Educational Research Review*, vol. 14, pp. 47–61, Feb. 2015, ISSN: 1747-938X. DOI: 10.1016/j.edurev.2015.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1747938X15000068.

[13] K. Lichtman and B. VanPatten, "Was Krashen right? Forty years later," *Foreign Language Annals*, vol. 54, no. 2, pp. 283–305, 2021, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/flan.12552, ISSN: 1944-9720. DOI: 10.1111/flan.12552. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/flan.12552.

[14] M. J. Lee *et al.*, "Principles of a debugging-first puzzle game for computing education," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, ISSN: 1943-6106, Jul. 2014, pp. 57–64. DOI: 10.1109/VLHCC.2014.6883023.