Exploratory Learning in Engineering Programming

Campbell R. Bego
Department of Engineering Fundamentals
J. B. Speed School of Engineering
University of Louisville
campbell.bego@louisville.edu

Angela K. Thompson
Department of Engineering Fundamentals
J. B. Speed School of Engineering
University of Louisville
angela.thompson@louisville.edu

Cenetria Crockett
Department of Anthropology
College of Arts & Sciences
University of Louisville
cenetria.crockett@louisville.edu

Raymond J. Chastain
Department of Information Systems, Analytics & Operations
College of Business
University of Louisville
raymond.chastain@louisville.edu

Jeff L. Hieb
Department of Engineering Fundamentals
J. B. Speed School of Engineering
University of Louisville
jeffrey.hieb@louisville.edu

Linda Fuselier
Department of Biology
College of Arts & Sciences
University of Louisville
linda.fuselier@louisville.edu

Ryan Patrick
Department of Psychological and Brain Sciences
College of Arts & Sciences
University of Louisville
ryan.patrick@louisville.edu

Marci S. DeCaro
Department of Psychological and Brain Sciences
College of Arts & Sciences
University of Louisville
marci.decaro@louisville.edu

***Abstract*—This WIP paper presents new research on exploratory learning, an educational technique that reverses the order of standard lecture-based instruction techniques. In exploratory learning, students are presented with a novel activity first, followed by instruction. Exploratory learning has been observed to benefit student learning in foundational math and science courses such as calculus, physics, and statistics; however, it has yet to be applied to engineering topics such as programming. In two studies, we tested the effectiveness of exploratory learning in the programming unit of a first-year undergraduate engineering course. We designed a new activity to help students learn about different python error types, ensuring that it would be suitable for exploration. Then we implemented two different orders (the traditional *instruct-first* versus exploratory learning's *explore-first*) across the six sections of the course. In Study 1 (*N*=406), we did not detect a difference between the instruct-first and explore-first conditions. In Study 2 (*N*=411), we added more scaffolding to the activity. Students who received the traditional order of instruction followed by the activity scored significantly higher on the assessment. These findings contradict the exploratory learning benefits typically shown, shedding light on potential boundary conditions to this effect.**

Keywords—exploratory learning, engineering education, programming.

I. Introduction

Active learning techniques in foundational science, technology, engineering, and mathematics (STEM) courses have gained considerable research attention in recent years. Many studies have suggested that active learning implementation in classroom settings can improve student engagement and learning, which can be useful in difficult introductory STEM courses [1]. One useful active learning technique is *exploratory learning*, in which students explore a new topic with an activity prior to instruction. Even if students are not able to complete the activity, their efforts can be considered "productive failure" [2], because engagement in the activity prior to instruction often results in increased student conceptual knowledge of the topic [2]–[5]. Exploratory learning benefits have been observed in undergraduate physics, statistics, psychology, precalculus, biology, and chemistry courses [6]–[11].

However, positive results are not always found (e.g., [12]–[14]). Researchers have proposed that there may be boundary conditions such as cognitive load that may limit the effectiveness of exploratory learning. In addition, exploratory learning research is still nascent; many topics, potential moderators and boundary conditions have not yet been studied. For example, exploratory learning has not been studied in introductory programming. More research is needed to help determine when and why exploratory learning will be beneficial, and when providing instruction first will lead to higher learning outcomes.

This paper presents two studies that tested exploratory learning as students learned about python error messages in a first-year engineering course at the University of Louisville. In both studies, half of the students received the activity prior to instruction, while the other half received the content instruction followed by the same activity. In the first study, the activity was unstructured. The second study used a more scaffolded activity.

II. Literature Review

In exploratory learning, students are instructed to engage in an activity that they have not seen before. Exploration activities are typically challenging, and students are often unsuccessful. After the activity is attempted, instructors teach students how to appropriately complete the activity and other similar problems. This order of instruction is opposite of traditional lecture-based learning, where students receive instructions on the topic first, and then complete the activity afterwards.

During exploration, students naturally activate their prior knowledge while searching for a known solution [4]–[6], [15]. They then recognize that there are gaps in their current knowledge, which prompts them to become more attentive during the instructional phase with the goal of fulfilling the knowledge gaps [4], [16]. In addition, students' work on the activity allows them to discern important problem features either during the activity or during the following instruction.

Many studies have indicated that incorporating exploratory learning in the classroom can increase students' *conceptual understanding* [5], which is an understanding of the relationships between features (as opposed to stepwise solution procedures, i.e., procedural knowledge [17]). Conceptual understanding is a higher level of knowledge that can also lead to better transfer, or application of knowledge to a new context or situation [18], [19]. This type of benefit could have many positive effects for students within STEM programs, as more advanced courses build on introductory prerequisites.

A majority of the exploratory learning studies have been conducted in grades K-12 [5], however, studies in higher education have started to gain popularity. In recent years, several researchers have implemented exploratory learning in undergraduate introductory STEM courses [6]–[10]. One such study was conducted by Weaver and colleagues [8], where undergraduate physics students learned about electric potential. Students assigned to explore-first conditions outperformed the students in the instruct-first conditions. The same outcomes were observed in another exploratory learning study where a vector activity was implemented in an online format [9]. In this engineering mathematics course, students in the explore-first condition again outperformed the student in the instruct-first condition.

There are several other studies that found that students in an explore-first condition scored better than students in a traditional lecture-first condition, suggesting that the implementation of exploratory learning as a teaching technique can frequently be more beneficial to student learning than the instruct-first method of instruction. But there are not many studies that test exploration learning in other topic areas. In this current study, we seek to explore the exploration learning technique in a different subject area within STEM: programming.

A. Activity Design

Activities used in exploratory learning research have taken various forms. Some activities include *contrasting cases*, or images or diagrams that vary only by critical problem features [20], [21]. For example, when teaching about the concept of density, Schwartz and colleagues [21] presented an activity with 2D images of busses and riders. The contrasting cases varied by size of bus (volume) and number of riders (mass) to try to encourage students to understand those features and the relationship between them. Students were told to estimate the "crowdedness" of the busses, to determine which bus company was most crowded, and to develop a "crowdedness index" for estimating the crowdedness of bus lines in general.

Other activities were built around *rich datasets*, or datasets with many different points. For example, Kapur [16] presented data for two basketball players and the number of points they scored in 20 different games. Students were asked to determine which player was more "consistent," and to develop multiple strategies to estimate consistency mathematically. In rich datasets, the critical problem features are hidden and not obvious to participants.

These activity designs vary by the amount of scaffolding, or support provided for students during exploration. Instructors often use scaffolding to simplify challenging tasks [22]. A significant attribute of scaffolding is that it is temporary. As student learning develops and evolves, the scaffolding would need to change to match student needs at the time. This idea comes from Vygotsky's Zone of Proximal Development theory, that sought to explain the learning development of students [23]. This theory considers student learning along proximal (i.e., potential) levels of development and not based on current knowledge levels [24]. Vygotsky argued that students were influenced by their social and cultural environments, and these influences are manifested through social interactions between their peers and instructors, which aids in the development of cognition [24], [25]. The zone of proximal development theory asserts that students learn best when working through novice to challenging problems through collaborations amongst their peers and instructor (expert) assistance [24].

Some studies have compared the effectiveness of different activity types on student learning from exploration, but results have been inconclusive. In one case, Bego and colleagues [6] found that the scaffolding through contrasting cases appeared to improve learning. In the other case, contrasting cases appeared to decrease performance in procedural knowledge [12]. More research is needed that varies activity type to learn best strategies for learning through exploration.

B. The Current Studies

In the current studies, we incorporated exploratory learning methods while teaching about python error messages in a first-year engineering course at the University of Louisville's J. B. Speed School of Engineering. These studies further the research in exploratory learning in two ways, first by incorporating it into a programming topic, and second to compare results across two versions of an activity. The research questions were as follows:

RQ1. Does exploratory learning help engineering students learn an introductory programming concept?

RQ2. Does scaffolding the exploration activity help students learn from exploration?

III. Methodology

A. Participants

Participants were first-year engineering undergraduate students at the University of Louisville enrolled in Engineering Methods, Tools, and Practice I in Fall 2021 (Study 1, $N = 406$) and Fall 2022 (Study 2, $N = 411$). Participants were included in the study if they attended class on the day of the experiment and completed all phases of the experiment. Typically, student background in computer science varies widely in this course. Self-report of prior knowledge was surveyed during this study, but the survey data was not available for this WIP paper, so this factor is not described here.

Table 1
Experimental condition by course section

Fall 2021 (Study 1)				Fall 2022 (Study 2)			
1	Explore-first (Instr. 1)	4	Instruct-first (Instr. 2)	1	Instruct-first (Instr. 2)	4	Explore-first (Instr. 2)
2	Instruct-first (Instr. 1)	5	Explore-first (Instr. 2)	2	Explore-first (Instr. 2)	5	Instruct-first (Instr. 2)
3	Explore-first (Instr. 1)	6	Instruct-first (Instr. 2)	3	Explore-first (Instr. 3)	6	Instruct-first (Instr. 3)

B. Procedures

Students were divided into 2 conditions by course section: the experimental condition (explore-first), and the control condition (instruct-first; see Table 1). In both studies, there were six course sections and two instructors; one instructor (Instructor 2) was common in both years (also listed in Table 1).

Students in the *instruct-first condition* received instruction on interpreting error messages in Python programs, followed by an activity, as practice of what they just learned. Students in the *explore-first condition* received the same materials in reversed order. First, they completed the activity, as a novel exploration activity. Then, they received the instruction. After the instruction and activity, students in both conditions completed a short survey, followed by an assessment to evaluate their learning. The purpose of the survey was to assess student attitudes about the activity; however, the current paper is focused on learning outcomes, and will not report the survey results.

Students were allowed to work with peers on the exploration activity but were instructed to work alone (silently) on the assessment. Students received participation credit for attempting the assessment, regardless of their performance, and were told to do their best. All activities (instruction, exploration activity, survey, assessment) were completed within one 50-minute class period. Following the experiment, the assessments were graded to determine how well students had learned the topic.

C. Materials

The class session focused on three Python language-based errors: Name Errors, Syntax Errors, and Type Errors.

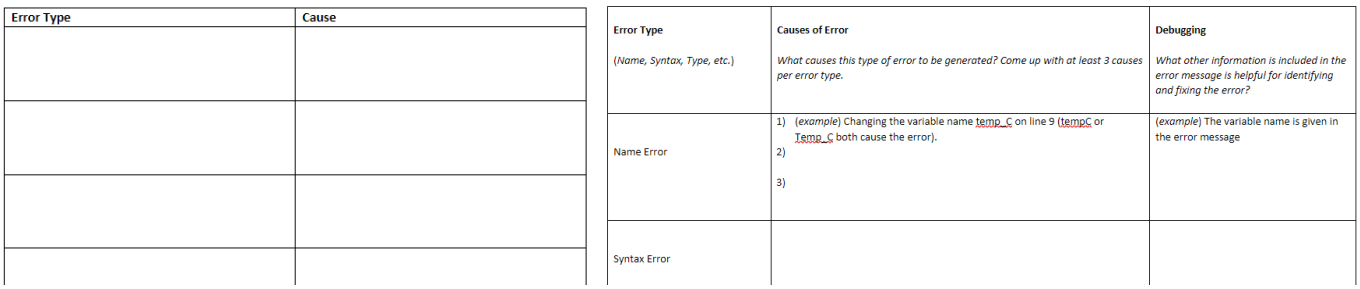
Instruction. The three most common error types were described and illustrated with code examples that would generate these error messages. The composition of the error message was discussed, including hints about the type and location of the error in the code. For example, Name errors can occur when variables are misspelled or when one forgets to add quotation marks around a string. Syntax errors frequently result from incorrect punctuation. Type errors indicate an invalid datatype. Corrections to the code (to fix the errors) were also discussed as appropriate. The same lecture was provided in both years/studies and will be made available upon request.

Activity. Students were provided a complete, working program that asked the user to enter a temperature in degrees Fahrenheit, converted the temperature to degrees Celsius, and then displayed a message about the temperature (e.g. "Freezing" or "Hot"). Students were instructed to "break" the code (by adding, changing, or deleting text) to generate different error messages. Students recorded their findings on a team worksheet identifying the error types and causes. Students worked on the activity in groups of 2-3 at tables of 4-5 (2 worksheets were given to each table). In Study 1, the worksheet was simple and open-ended (see Figure 1). In Study 2, the worksheet was modified to be scaffolded, such that students were told which error types they were looking for (Name, Syntax, Type) and were asked to consider how to use the information provided in the error message to debug the code (see Figure 1).

Assessment. The assessment included 12 questions. In Study 1, all questions were multiple choice. Given sample code or scenarios, six questions asked students to select which error type would be generated. In three additional questions, students were provided with code as well as the generated error message and were asked to select the error cause (e.g., missing parenthesis). Finally, the last 3 questions tested students' conceptual understanding, considering plausible explanations for a given error message, and understanding when and why python generates an error message. The items were designed to assess both procedural and conceptual knowledge, but preliminary analyses showed no differences based on item type, so they were combined into a total score.

In Study 2, four of the questions from Study 1 were modified to be open response instead of multiple choice, asking students to explain *why* an error message was generated or what was incorrect in the code to generate the error. Responses were coded, with 1 point given only to complete and correct responses (no partial credit), and an average total percentage was computed for each student.

Figure 1
Activity Worksheets in Study 1 (left) and Study 2 (right).



Error Type	Cause

Error Type (Name, Syntax, Type, etc.)	Causes of Error *What causes this type of error to be generated? Come up with at least 3 causes per error type.*	Debugging *What other information is included in the error message is helpful for identifying and fixing the error?*
Name Error	1) (*example*) Changing the variable name temp_C on line 9 (tempC or Temp_C both cause the error). 2) 3)	(*example*) The variable name is given in the error message
Syntax Error		

Note. The worksheet for Study 2 was more scaffolded, and provided students with clearer goals.

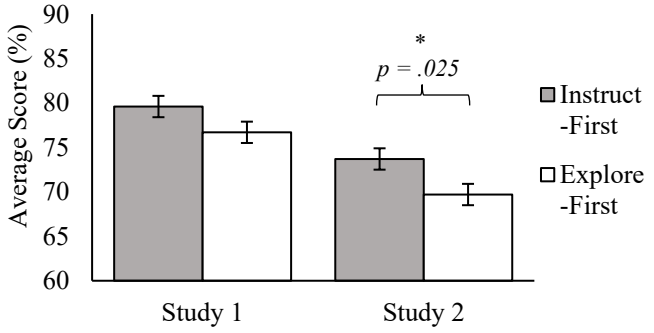
D. Analysis

We assessed learning in each study by conducting a one-way analysis of covariance (ANCOVA) on overall assessment score (percent correct) with an independent factor of *order* (explore-first, instruct-first) and a covariate of course performance (end of semester weighted average). This covariate was included to account for individual performance differences across sections.

IV. Results

In Study 1, the covariate of course performance was significant, $F(1, 405) = 42.78$, $p < .001$, but there was no effect of order, $F(1, 405) = 2.74$, $p = .099$. The average performance of students in the instruct-first condition ($M = 79.6\%$, $SE = 1.2\%$) did not significantly differ from the average performance of students the explore-first condition ($M = 76.7\%$, $SE = 1.2\%$). In Study 2, the covariate of course performance was again significant, $F(1, 410) = 69.98$, $p < .001$, and there was an effect of order, $F(1, 410) = 5.04$, $p = .025$, $\eta_p^2 = .012$. The average performance of students in the instruct-first condition ($M = 73.7\%$, $SE = 1.2\%$) was higher than the average performance of students the explore-first condition ($M = 69.7\%$, $SE = 1.2\%$). All results are illustrated in Figure 2.

Figure 2
Assessment results from Study 1 and Study 2 by condition.



V. Discussion

In the current set of experiments, exploratory learning—engaging in a novel activity followed by instruction—did not benefit student learning above a traditional instruct-first order. In fact, in Study 2, students who experienced the traditional order scored higher on the immediate assessment than students who explored the activity first. This was the first study to implement exploratory learning in introductory programming, which is an important topic for first-year engineering students. These results are therefore important findings for both programming instructors and exploratory learning researchers.

These results contradict the majority of exploratory learning studies [5]. However, they are consistent with some other studies showing null results or instruct-first benefits (e.g., [12]–[14]). One possible reason we did not find the explore-first benefit is that for this topic (error types in python), it may be necessary to give students an idea of the framework prior to letting them work on an activity. In Study 2, students were provided the names of the error types they were looking for, but no context for what those error types were. Once students have a basic understanding of the error types, it is likely that they can better relate the changes in the code to each error. Without this framework, perhaps, they do not see the relationships between the error messages and what they have changed in the code.

Another possibility is that the exploration activity was too complex. By including multiple concepts to learn, the activity could have been too taxing and increased students' cognitive load [6], [7]. This load could have prevented students from deeply exploring the problem features, or could have caused students to give up working on the activity. We measured cognitive load following the experiment, but have not yet analyzed those results.

Alternatively, it is possible that students received the benefits of "exploration" even when the activity followed instruction. By trying new methods to "break" the code after instruction, students were able to go beyond mere practice after the lesson. We have records of student work during the activity that may reveal different levels of engagement for students in different conditions. More analysis is needed on the activity participation and output that can be done in future work. For example, we could look at how many different error type examples were discovered in each group.

Finally, the benefits of exploration are typically found on students' conceptual understanding, rather than learning basic facts or procedures. It is possible that the target topic was not as conceptual as we anticipated. Alternatively, our assessment may not have accurately captured conceptual knowledge. In the second year, we changed the assessment to be more open-ended in an attempt to better measure conceptual understanding. This change also made it more difficult for students to answer the questions correctly and thus reduced overall scores. In addition, it appears to have widened the learning gap between students in the instruct-first and explore first conditions. Therefore, we believe we assessed knowledge of the topic accurately.

A. Limitations

This is only one topic in one course, using one set of materials, and should not be interpreted as a rejection of exploratory learning. However, it is important to measure student learning and identify where and when educational strategies are effective. More work is needed to understand when, how, and for whom exploratory learning works best.

VI. Conclusions & Future directions

Exploratory learning did not benefit student learning in a programming topic in a first-year engineering course. In fact, students in the instruct-first order outperformed students in an explore-first order. Continued research with new educational techniques is necessary. Future work includes analyses of additionally collected data in this experiment, further experimentation in other programming topics, and further application of exploratory learning to other engineering topics.

VII. Acknowledgements

This work has been supported by the NSF Division for Undergraduate Education under grant number 2012342. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the University of Louisville.

VIII. References

[1] S. Freeman *et al.*, "Active learning increases student performance in science, engineering, and mathematics," in *Proceedings of the National Academy of Sciences*, 2014, pp. 8410–8415. doi: 10.1073/pnas.1319030111.

[2] M. Kapur, "Productive failure," *Cogn Instr*, vol. 26, pp. 379–424, 2008, doi: 10.1080/07370000802212669.

[3] D. L. Schwartz and J. D. Bransford, "A time for telling," *Cogn Instr*, vol. 16, no. 4, pp. 367–398, 1998, doi: 10.1207/s1532690xci1604.

[4] M. S. DeCaro and B. Rittle-Johnson, "Exploring mathematics problems prepares children to learn from instruction," *J Exp Child Psychol*, vol. 113, pp. 552–568, 2012, doi: 10.1016/j.jecp.2012.06.009.

[5] K. Loibl, I. Roll, and N. Rummel, "Towards a theory of when and how problem solving followed by instruction supports learning," *Educ Psychol Rev*, vol. 29, pp. 693–715, 2017, doi: 10.1007/s10648-016-9379-x.

[6] C. R. Bego, R. J. Chastain, and M. S. DeCaro, "Designing novel activities before instruction: Use of contrasting cases and a rich dataset," *British Journal of Educational Psychology*, 2022, doi: 10.1111/bjep.12555.

[7] P. M. Newman and M. S. DeCaro, "Learning by exploring: How much guidance is optimal?," *Learn Instr*, vol. 62, 2019, doi: 10.1016/j.learninstruc.2019.05.005.

[8] J. P. Weaver, R. J. Chastain, D. A. DeCaro, and M. S. DeCaro, "Reverse the routine: Problem solving before instruction improves conceptual knowledge in undergraduate physics," *Contemp Educ Psychol*, vol. 52, pp. 36–47, 2018, doi: 10.1016/j.cedpsych.2017.12.003.

[9] J. L. Hieb, M. S. DeCaro, and R. Chastain, "Work in progress: Exploring before instruction using an online GeoGebra activity in introductory engineering calculus," in *ASEE Annual Conference and Exposition, Conference Proceedings*, 2021.

[10] M. S. DeCaro *et al.*, "Exploring an online simulation before lecture improves undergraduate chemistry learning," *International Society of the Learning Sciences*, 2022.

[11] S. G. Chowrira, K. M. Smith, P. J. Dubois, and I. Roll, "DIY productive failure: boosting performance in a large undergraduate biology course," *NPJ Sci Learn*, vol. 4, no. 1, 2019, doi: 10.1038/s41539-019-0040-6.

[12] K. Loibl, M. Tillema, N. Rummel, and T. van Gog, "The effect of contrasting cases during problem solving prior to and after instruction," *Instr Sci*, vol. 2020, pp. 1–22, 2020, doi: 10.1007/s11251-020-09504-7.

[13] E. R. Fyfe, M. S. DeCaro, and B. Rittle-Johnson, "An alternative time for telling: When conceptual instruction prior to problem solving improves mathematical knowledge," *British Journal of Educational Psychology*, vol. 84, no. 3, pp. 502–519, 2014, doi: 10.1111/bjep.12035.

[14] G. Ashman, S. Kalyuga, and J. Sweller, "Problem-solving or explicit instruction: Which should go first when element interactivity is high?," *Educ Psychol Rev*, vol. 32, no. 1, pp. 229–247, 2020, doi: 10.1007/s10648-019-09500-5.

[15] M. Kapur, "Productive failure in learning the concept of variance," *Instr Sci*, vol. 40, pp. 651–672, 2012, doi: 10.1007/s11251-012-9209-6.

[16] M. Kapur, "Productive failure in learning math," *Cogn Sci*, vol. 38, pp. 1008–1022, 2014, doi: 10.1111/cogs.12107.

[17] D. Jonassen, "Reconciling a human cognitive architecture," in *Constructivist Instruction: Success or Failure?*, S. Tobias and T. M. Duffy, Eds., New York, NY: Routledge, 2009, pp. 13–33.

[18] D. Klahr, "'To Every Thing There is a Season, and a Time to Every Purpose Under the Heavens' What about Direct Instruction?," in *Constructivist Instruction: Success or Failure*, S. Tobias and T. M. Duffy, Eds., New York, NY: Routledge, 2009, pp. 291–310.

[19] C. C. Chase and D. Klahr, "Invention versus direct instruction: For some content, it's a tie," *J Sci Educ Technol*, vol. 26, no. 6, pp. 582–596, 2017, doi: 10.1007/s10956-017-9700-6.

[20] N. Hallinen, D. Chin, K. Blair, and D. Schwartz, "Using contrasting cases for generation and instruction," in *Proceedings of the International Society of the Learning Sciences*, 2014, pp. 1185–1188.

[21] D. L. Schwartz, C. C. Chase, M. A. Oppezzo, and D. B. Chin, "Practicing versus inventing with contrasting cases: The effects of telling first on learning and transfer.," *J Educ Psychol*, vol. 103, pp. 759–775, 2011, doi: 10.1037/a0025140.

[22] J. van de Pol, M. Volman, and J. Beishuizen, "Scaffolding in teacher-student interaction: A decade of research," *Educational Psychology Review*, vol. 22, no. 3. 2010. doi: 10.1007/s10648-010-9127-6.

[23] L. S. Vygotsky, "Interaction between learning and development," in *Mind in Society*, 2019. doi: 10.2307/j.ctvjf9vz4.11.

[24] K. Shabani, M. Khatib, and S. Ebadi, "Vygotsky's Zone of Proximal Development: Instructional implications and teachers' professional development," *English Language Teaching*, vol. 3, no. 4, 2010, doi: 10.5539/elt.v3n4p237.

[25] R. M. Silalahi, "Understanding Vygotsky's Zone of Proximal Development for learning," *Polyglot: Jurnal Ilmiah*, vol. 15, no. 2, 2019, doi: 10.19166/pji.v15i2.1544.