Fall 9-11-2020

Teaching Introductory Programming Online: Lessons Learned

Xiaoyun He
Auburn University at Montgomery, xhe@aum.edu

TEACHING INTRODUCTORY PROGRAMMING ONLINE: LESSONS LEARNED

Xiaoyun He
Auburn University at Montgomery
xhe@aum.edu

ABSTRACT

Programming is considered a fundamental skill for Information Systems students. Yet, it is generally regarded as hard for students to learn and challenging for instructors to teach. Fully online programming courses can make it even more challenging than the face-to-face version. In this paper, we share our experience of taking a multi-faceted approach in teaching an introductory programming course online. We discuss pedagogical considerations in our approach that incorporates best practices with experimentation to be suitable for our student body while achieving desired learning outcomes.

KEYWORDS

Introductory Programming, Pedagogical, Online, Lessons Learned.

INTRODUCTION

Employment of computer and information technology careers is projected to grow 12 percent from 2018 to 2028, much faster than the average for all occupations (US Bureau of Labor Statistics, 2020). The demand for computing and programming skills is expected to continue to increase for the foreseeable future. In the discipline of Information Systems, technical aspects of software development including programming are considered some of the foundational material that we will expect our students to master by the time of graduation (Topi, 2019).

Yet, learning to program can be a difficult task, to the point where the phrases "high failure rate" and "programming course" are almost synonymous (Bennedsen & Caspersen, 2007). Over the past decades, many theories have been put forward as to why learning to program is a difficult task. Much efforts have also been put into developing and applying various teaching approaches that facilitate students' learning (Vihavainen, Airaksinen, & Watson, 2014). However, pass rates were not found to have significantly differed over time, or based upon the language taught in the course (Watson & Li, 2014). A recent systematic review on the curriculum, pedagogy, and languages for teaching introductory programming (Luxton-Reilly et al., 2018) concludes that "Despite many years of study and debate, some significant issues in teaching programming remain unresolved and probably unresolvable." It appears that introducing students to programming is still one of computing education's grand challenges and that we as a community have a challenge in developing effective learning environments and instructional methods that are suitable for our specific student body.

Meanwhile, both the number of online courses and the number of students enrolling in online courses have been increasing. Even though more students are choosing distance education, the literature shows that attrition rates are higher in online courses than in face-to-face courses (Angelino, Williams, & Natvig, 2007). Yet, few studies have focused on teaching introductory programming course online. Instead, the literature has primarily focused on the traditional face-to-face delivery mode, and a few have examined the effects of flipping the traditional lectures (Breimer, Fryling, & Yoder, 2016; Sharp, 2016; Sharma et al., 2020).

In this paper, we share our experience in taking a multi-faceted approach in teaching an introductory Java programming course online. We intend to meet the standards and curriculum recommendations of teaching programming fundamentals while creating an online learning environment in which students can develop competence and confidence as budding programmers. We discuss pedagogical considerations in our approach that incorporates best practices with experimentation to meet the needs of our specific student body while achieving desired learning outcomes.

In particular, the introductory programming course is offered at a public comprehensive university in the South of the United States. Undergraduate students majoring in information systems (IS) are required to complete an introductory programming course with the option to choose either Java or C++. Students must pass this course with a letter grade

of C or higher. It is deemed to be one of the core knowledge areas that the students should possess to thrive in the information age. Our objective for this course is to introduce students to programming as a problem-solving process so that they can understand the fundamental components in coding and write programs for basic problems.

RELATED LITERATURE

Over the past two decades, many studies have devoted to teaching introductory programming related issues and practices. These studies have a variety of particular focus, such as student misconceptions, teaching approaches, program comprehension, automated feedback for exercises, competency enhancing games, student anxiety, program visualization, etc. (see Luxton-Reilly et al. (2018) for a recent review).

The advances in computer technologies has made it possible to deliver courses to remote (off-campus) sites via audio and/or video (live or prerecorded), including both synchronous (i.e., simultaneous) and asynchronous (i.e., not simultaneous) instruction. Chenoweth, Corral, & Scott (2016) conducted a study to evaluate two content delivery options for teaching a programming language to determine whether an asynchronous format can achieve the same learning efficacy as a traditional lecture (face-to-face) format. Their results suggest that an asynchronous tutorial can achieve the same learning outcomes as a traditional lecture format by using automated feedback for convergence (using PPT slides with button choices).

Breimer, Fryling, & Yoder (2016) examined traditional, semi-flipped and fully flipped classroom models by comparing three sections of an Introduction to Programming (Java) course. Their data and observations collected suggest that incorporating in-class activities improves student satisfaction but a semi-flipped classroom, including in-class activities, some outside-class lecture videos, and some in-class lectures, may generally provide the best overall experience for the students. However, while students may be more satisfied and get more programming practice in a flipped paradigm, overall student performance did not appear to be greatly impacted. Sharp (2016) conducted a survey of what students have to say about the flipped classroom. The results indicated that overall, the participants viewed the flipped C# classroom positively, although some participants still preferred the traditional classroom environment.

Dutton, Dutton, and Perry (2002) did a study to compare student outcomes in an online version and a face-to-face version of an introductory course in the computer language C++. They found that for those who completed the course, students in the online version did significantly better than students in the lecture version of the class, which is true even when differences in effort and maturity were adjusted. However, the online class had a higher dropout rate, though the difference is not statistically significant when the differences in effort and maturity level were controlled.

He and Yen (2014) aimed to assess the predictive relationships between distance course delivery method (face-to-face, satellite broadcasting, and live video-streaming) and students' perceived learning performance and satisfaction in IT software programming courses taught by the same instructor. Their results suggest that the choice of delivery method was related to students' satisfaction and programming skill enhancement. However, they did not find a relationship between the delivery method and the students' perceived learning performance. Specifically, the participants in the face-to-face delivery method group were more likely to feel satisfied with the delivery method than the students using the other two delivery methods (i.e., satellite broadcasting and live video streaming).

A MULTI-FACETED APPROACH AND LESSONS LEARNED

In this section, we share the main components of our approach and lessons learned. We intend to incorporate best practices into our approach with experimentation to meet the needs of our student body while achieving desired learning outcomes.

Establishing Clear and Consistent Expectations at the Outset

On the first day of the semester, we made it clear about the learning expectations through both emails and the announcement in the Learning Management System - the Blackboard. The expectations were also clearly listed in the syllabus. These include learning objectives and assignments as well as their role as students is to do their best work at all times. The message sent to the students was that work in a programing course could be challenging at times, but each student would be capable of finding success in learning about programming if they put in the effort even in online delivery mode.

It has been suggested that the difficulties faced by novices may be a consequence of unrealistic expectations rather than intrinsic subject complexity (Luxton-Reilly, 2016). Better outcomes are likely to arise from focusing less on student deficits and more on actions that we can take to improve teaching practice. We made sure that the learning expectations are realistic with respect to our student body. We created weekly modules in the Blackboard,

including weekly learning objectives that are aligned with the weekly activities and assessment. The students can work on the assignments any time prior to the end of a particular week, so it provides some degree of flexibility, especially in the situation where many of our students work either part-time or full-time. We supported them consistently with our own communication and actions throughout the term.

We believe there are benefits of doing so. First, when a course is delivered online, the students may feel anxious and fear that their grades would suffer. Because they would not be able to have much interactions with the instructors and their peers, or they could not get the immediate help if they run into problems when working on the programming assignments. The anxiety has a negative impact on student performance (Slavin, 2003), as well as student engagement in any given activity (Tharayil, et al., 2018). Studies suggest that faculty efforts to clarify expectations do result in a change in student attitudes regarding the acceptability of certain behaviors (Aasheim, et al. 2012). By setting clear learning expectations in the beginning, we were able to foster an environment where students are less anxious about their grades and recognize the benefit of engaging in active learning for their academic success. We not only addressed these questions right away, also reiterated that the focus of each student should be on learning, not simply receiving a good grade.

Second, like all of us, students are more likely to succeed when they understand what is expected of them. Maintaining the consistency in expectations throughout the semester can help in this regard. Students who know that the same deliverables such as code, quizzes, and hands-on exercises will be expected on a regular basis are less likely to become frustrated than students in courses where such demands seem to be ad-hoc (Zhang et al., 2020).

Short Video Lectures: One Clip for One Specific Topic

We created short videos by using Kaltura Capture. The videos captured the desktop of the instructor showing students the coding statements being crafted one by one in the NetBeans IDE with the instructor narrating to explain the underlying core concepts as well as demonstrating how they work in the examples of concrete programs. The videos range from approximately 9 minutes to 18 minutes in length. The videos were then uploaded into the Blackboard for students to view at their convenience.

Each video focuses on only one specific topic or concept, instead of covering an entire chapter. For example, in order to cover the chapter of Looping Structures, we created four videos. Each of them is focused on one of the following four specific looping statements: counter-controlled while loop, event-controlled while loop, for loop, and do…while loop.

It turned out that these short videos worked well for our students. We received a number of positive comments from the students soon after we had started creating and placing them in the Blackboard. The helpfulness of short videos was confirmed by the following students' responses: "The short videos are very useful in helping me understand the key programming concepts", "Since these videos are short with one concept at a time, I do not feel overwhelmed by the amount of the new content.", "I can learn at my own pace, and review the videos again and again.", and "The short videos are great, I feel like that I am learning as much as I would in the F2F classroom". In addition, many of the students mentioned that the program examples used in the videos were helpful for them complete the programming assignments that require to apply the similar techniques or concepts.

The evidence that our recorded videos were well received by the student appears to be consistent with the prior studies. For example, Sharp (2016) noted several strengths of employing demonstration videos in programming courses, such as the ability to utilize the videos at students' own convenience and pace, the videos provide good introductions to the assigned topics, and served as a convenient reference while working on assignments. Bergmann and Sams (2012) point out that while some instructors may use generic lecture videos to flip their classroom, it is better for instructors to create their own. Creating the lecture videos is very time consuming but they can certainly be reused (Breimer, Fryling, & Yoder, 2016).

The short length of the videos can be a positive contributing factor. Academic studies in effective teaching have long held the belief that short attention span is one of the characteristics of modern students. McKeachie & Svinicki (2013) has maintained that attention typically increases from the beginning of the lecture to 10-15 minutes into the lecture and decreases after that point. For example, the well-known TED talks are required to follow 18 min rule, which is based on the notion that 18 min is long enough to have a "serious" presentation but short enough to hold a person's attention.

Moreover, showing students concrete and practical coding examples in the videos appears to be beneficial. As noted in Zhang et al. (2020), showing introductory students well-written code as an exemplar can have many benefits. For

instance, if the instructor emphasizes the modularity of the code, students may be able to recognize the syntax for defining the scope of each module. The style of the code can make a good first impression that can be emulated. The proper use of methods from the main program of a carefully chosen example can make the overall purpose of the program readable, even to novice programmers.

Using the Same Problem with Different Techniques

Instead of giving a new problem for every assignment, we designed the majority of our assignments around the same problem. In the beginning, the coding assignment is pretty simple and only requires a small chunk of code to practice basic skills. It is similar to what we did in the pre-recorded video. As we gradually introduce new concepts and techniques over time, the new assignment is built upon the previous one but requires students to apply the new material being taught. For example, once we introduce the if … else statements, the students are required to use them in the next assignment that is an extended version of the previous one. Then, we introduce the looping structure, which is required to be used outside the if … else statement in the previous assignment that they have worked on. Eventually, the students were able to complete the working version of a large project that incorporates the core coding concepts and techniques we have introduced in the course.

We believe that using the same problem with different techniques can accomplish several pedagogical goals. First of all, it allows students to focus on the coding techniques rather than having first to understand new program requirements (Newby & Nguyen, 2010). Second, starting with something simple and small can help the students to build more confidence in programming over time. If they can successfully build a small chunk of code, compile it, debug it, and test it, then they have a working component. This gives them a starting point for the next layer of the solution. The students seem to need time to experiment and become comfortable with a topic area before feeling capable of engaging in more difficult tasks.

Frequent Communication and Video Help Sessions

Frequent communication can keep everyone in the loop. We kept students informed about what was expected for the coming weeks and important dates via the announcements in the Blackboard and emails. We also made sure that the students truly understand what's going on. We let the students know that we were with them and that we were all in this together, which helps alleviate the stress some of the students might be experiencing. Sharma et al. (2020) has noted that, even though it might be obvious, frequent communication has particular applicability for undergraduate students enrolled in an online programming course.

We offered both online office hours and optional one-on-one help sessions through either Blackboard Collaborate Ultra or Zoom. In particular, the students that utilized the help sessions considered them very helpful, as reflected by the following comments: "I do not think that I can figure out what is wrong with my program without your video help session."; "Without the one-on-one help sessions, I would have given up doing the coding assignments."; "I was losing confidence in my ability to identify and correct the errors in my code. Your help session really helped me to see the big picture, and I learned more about debugging a program.".

Offering extra help to students can have greater impact on students' learning than otherwise, especially in online programming courses. Prior studies suggest that students can lose interest if they have trouble running a program and experience a lack of opportunities to get help (Sharma et al., 2020). When students are learning to program, they may easily get frustrated especially when their program is not working as expected. The sheer frustration can be overwhelming. When we help them to manage some of the frustrations of learning to program, we send out a signal to our students that we care about them and their success in our course. We have also found that when students know we care about them, they seem to be more determined to do well in our class. They know that we are there to walk them through the difficult parts while showing them how to deal with the challenges of learning to program (Zhang et al., 2020).

EFFECTIVENESS

To assess the effectiveness of our approach, we looked at both student performance on final exam and semester-end course evaluations. Final exam includes both multiple-choice and short answer questions. The short answer questions require writing the code snippets to meet the requirements specified in the question. We compared student performance in four semesters, two of them were taught online, and another two were face-to-face. Note that, for each semsester, the questions given in the exam are different in wording but used to test the same programming concept or technique. Overall, the students in the online classes performed as well as those in the face-to-face version.

Based on the semester-end course evaluations given by the students in all those four semesters, the online classes received as high marks as the face-to-face ones. Some of the students in the online classes provided positive comments about the usefulness of the short videos as well as the flexibility of doing the work throughout the week, while some appreciated the instructor's extra help via the videos.

CONCLUSION

Teaching programming is a complex process. When the introductory programming course was offered in fully online mode, our goal was to design and deliver it to meet the needs of our undergraduate IS student body while achieving desired learning outcomes. To this end, we have taken a multi-faceted approach that incorporates best practices with experimentation. We have received positive comments from our students. The preliminary results suggest that student performance and perception of programming are similar to those in face-to-face version. We hope that our experience and the lessons shared in this paper will be helpful to our peer instructors.

REFERENCES

1. Aasheim, C. L., Rutner, P. S., Li, L., & Williams, S. R. (2012). Plagiarism and programming: A survey of student attitudes. *Journal of information systems education*, *23*(3), 297-313.
2. Angelino, L. M., Williams, F. K., & Natvig, D. (2007). Strategies to engage online students and reduce attrition rates. *Journal of Educators Online*, 4(2), n2.
3. Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *AcM SIGcSE Bulletin*, *39*(2), 32-36.
4. Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2), 30-36.
5. Bergmann, J., & Sams, A. (2012). *Flip your classroom: Reach every student in every class every day*. International society for technology in education.
6. Breimer, E., Fryling, M., & Yoder, R. (2016). Full flip, half flip and no flip: Evaluation of flipping an introductory programming course. *Information Systems Education Journal*, *14*(5), 4-16.
7. Chenoweth, T., Corral, K., & Scott, K. (2016). Automated Feedback as a Convergence Tools. *Journal of Information Systems Education*, 27(1), 7-15.
8. Dattero, R., Quan, J. J., & Galup, S. D. (2003). Estimating the value of Java and C++ skills. *Communications of the Association for Information Systems*, *11*(1), 17.
9. Dutton, J., Dutton, M., & Perry, J. (2001). Do online students perform as well as lecture students?. *Journal of Engineering education*, *90*(1), 131-136.
10. H, W., & Yen, C. J. (2014). The role of delivery methods on the perceived learning performance and satisfaction of IT students in software programming courses. Journal of Information Systems Education, 25(1), 23.
11. Lahtinen, E., Ala-Mutka, K. & Jarvinen, H. (2005). A Study of the Difficulties of Novice Programmers. *ACM SIGCSE Bulletin*, 37(3), 14-18.
12. Lishinski, A. (2016, August). Cognitive, Affective, and Dispositional Components of Learning Programming. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 261-262).
13. Luxton-Reilly, A. (2016, July). Learning to program is easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (pp. 284-289).
14. Luxton-Reilly, A., Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., ... & Szabo, C. (2018, July). Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 55-106).
15. McKeachie, W., & Svinicki, M. (2013). *McKeachie's teaching tips*. Cengage Learning.
16. Newby, M., & Nguyen, T. H. (2010). Using the same problem with different techniques in programming assignments: An empirical study of its effectiveness. *Journal of Information Systems Education*, 21(4), 375-382.
17. Paolini, A. (2015). Enhancing Teaching Effectiveness and Student Learning Outcomes. *Journal of Effective Teaching*, *15*(1), 20-33.
18. Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: What works? *Communications of the ACM*, 58(8), 34-36.

19. Reisetter, M., LaPointe, L., & Korcuska, J. (2007). The impact of altered realties: Implications of online delivery for learners' interactions, expectations, and learning skills. *International Journal on E-learning*, *6*(1), 55-80.
20. Slavin, R. E. (2003). Educational psychology: Theory and practice. 7/E, Pearson.
21. Sharma, M., Biros, D., Ayyalasomayajula, S., & Dalal, N. (2020). Teaching Tip: Teaching Programming to the Post-Millennial Generation: Pedagogic Considerations for an IS Course. *Journal of Information Systems Education*, 31(2), 96-105.
22. Sharp, J. H. (2016). The flipped C# programming classroom: What students had to say. In *Proceedings of the EDSIG Conference ISSN* (Vol. 2473, p. 3857).
23. Sharp, J. H., & Schultz, L. A. (2013). An exploratory study of the use of video as an instructional tool in an introductory C# programming course. *Information Systems Education Journal*, *11*(6), 33-39.
24. Tharayil, S., Borrego, M., Prince, M., Nguyen, K. A., Shekhar, P., Finelli, C. J., & Waters, C. (2018). Strategies to mitigate student resistance to active learning. *International Journal of STEM Education*, 5(1), 7.
25. Topi, H. (2019). Reflections on the current state and future of information systems education. *Journal of Information Systems Education*, 30(1), 1-9.
26. US Bureau of Labor Statistics (2020). *Computer and information technology occupations*. Retrieved July 3, 2020 from https://www.bls.gov/ooh/computer-and-information-technology/home.htm.
27. Vihavainen, A., Airaksinen, J., & Watson, C. (2014, July). A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the tenth annual conference on International computing education research* (pp. 19-26).
28. Watson, C., & Li, F. W. (2014, June). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39-44).

Zhang, X., Terwilliger, M. G., & Jenkins, J. T. (2020). Teaching Introductory Programming from A to Z: Twenty-Six Tips from the Trenches. *Journal of Information Systems Education*, *31*(2), 106-118.