Student attitudes toward syntax exercises in CS1

Shelsey Sullivan
shelseyb@gmail.com
Utah State University
Logan, Utah

Hillary Swanson
hillary.swanson@usu.edu
Utah State University
Logan, Utah

John Edwards
john.edwards@usu.edu
Utah State University
Logan, Utah

ABSTRACT

Syntax is a barrier to success for many students in Introductory Computer Programming (CS1). A supplemental approach to standard CS1 curricula that has gained attention recently is a "syntax-scaffolded" or "syntax-first" approach where students practice necessary syntax before a lesson on problem solving or program design. In this study, we analyze student perceptions of a syntax-first teaching method. For the first five weeks of a university semester, we assigned students syntax exercises to complete before coming to class. At the end of the semester they were given a prompt asking them to write their thoughts about the exercises. A qualitative approach was used to investigate student responses and understand perceived value of the exercises. Our results show a strong positive reaction to a syntax-first approach as well as an awareness among students of the exercises' effect on their own learning.

CCS CONCEPTS

• **Social and professional topics → CS1**.

KEYWORDS

CS1, syntax, student attitude

ACM Reference Format:
Shelsey Sullivan, Hillary Swanson, and John Edwards. 2020-. Student attitudes toward syntax exercises in CS1. In *The 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 17–20, 2021, Toronto, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Introductory Computer Programming (CS1) is a first university course in computer programming. Finding the best way give students this introduction has been the object of study for a number of researchers [1]. Research has found that students struggle when learning to program especially when it comes to programming language syntax [3–5, 17, 26]. Not only must students learn how to think in new and foreign ways, but they must do so while navigating strange, sometimes seemingly arbitrary rules about the symbols and words they are allowed to use [32].

SIGCSE '21, March 17–20, 2021, Toronto, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In an effort to help students overcome the cognitive burden of learning programming language syntax, a number of methods have been suggested and investigated. Some of these methods include removing syntax from learning to program entirely, as is the case with block-based or visual programming languages such as Scratch [24] and Alice [8]. However, research has shown that although students perform better when originally learning to program using these languages and techniques, once the students are transitioned to a text-based programming language, the struggle of learning syntax is the same as for students that begin learning with one of those languages [24, 37].

Two recent studies have tested a different approach to teaching students syntax [12, 13]. Instead of removing syntax until logic has been mastered, students were taught syntax first. In order to accomplish this, a program called Phanon was created. Phanon guides students through sets of syntax exercises to give students experience with syntax in small, manageable pieces, before they are expected to complete more complex programming assignments. These exercises are designed to give students consistent practice typing the programming language syntax as suggested by the procedural memory literature [18]. The idea is that by completing the exercises before class, students are able to focus more on the problem-solving and logic side of programming during lectures. Indeed, these studies found that by requiring students to complete syntax exercises, their exam scores increased and attrition decreased [12, 13].

The purpose of this paper is to explore student perception of the usefulness of the syntax exercises. Using a qualitative coding system, students' responses to a survey about the syntax exercises were assigned themes and analyzed. This work pursues two research questions:

RQ1 *What are student attitudes toward a syntax-first curriculum?*
RQ2 *What are student perceptions of how a syntax-first curriculum affects academic outcomes?*

2 THEORY AND RELATED WORK

Syntax has been shown to be a major contributor to student failure and attrition in CS1 [11, 21, 32]. It has been suggested that the cognitive load associated with syntax is extraneous, or unnecessary to learning problem solving and computational thinking [26, 33]. In order to reduce student strain in learning programming-language syntax, researchers have investigated numerous interventions and tool sets. Approaches to solving the syntax problem fall into two main areas: simplifying or removing syntax altogether and promoting competency of syntax through practice.

2.1 Simplified and visual languages

Some have argued that if syntax is a barrier for beginning CS students, the barrier should be removed completely or at least

simplified while students learn computational thinking. This has led to the creation of new languages, such as Quorum, Alice, and Scratch.

Quorum is a simplified-syntax programming language created by Stefik and Siebert [32] and designed after studying what syntax and key words are most difficult for beginning programmers to understand. The simple, intuitive syntax in Quorum was found to help beginners.

In another study, Mannila et al. [27] found that students were able to transfer knowledge from a language with simpler syntax (Python), to a language with a more complicated syntax (Java). The students were able to recognize the matching concepts, such as `for` loops, despite the difference in syntax. In addition, the majority of syntax errors the students created were caused by new topics not discussed when learning Python, such as error handling, rather than by the differences in syntax between the languages. This shows evidence that beginning with a language with simpler syntax will not hinder the students' learning.

Alice [6] and Scratch [30] are visual programming languages. In these languages, syntax is removed completely and rather than typing out code to program, blocks are moved around and values are changed [8]. These languages allow students to learn quickly at the beginning, but frustration returns when the students begin learning a text-based programming language [37]. The value of visual programming languages over text-based languages is still discussed [24, 29, 36].

2.2 Practice

We classify the types of syntax practice into three main categories: *programming exercises*, *typing exercises*, and *syntax exercises*.

2.2.1 Programming Exercises. Programming exercises are designed to allow students to practice using logic to solve simple programming prompts. An example of this type of exercise might be asking students to count how many times the number "1" appears in a list or array. These exercises require the students to practice typing and using programming language syntax, as well as programming logic in order to solve a problem [11, 28, 34].

Programming exercises are commonly used on websites to teach the basics of programming [28, 34]. These exercises require students to write entire pieces of code, which range in difficulty depending on the problem. They require students to use logic and problem-solving skills – syntax is not specifically targeted as the primary learning objective. However, one website featuring programming exercises, CodeWrite, was used in a study by Denny et al. [11] to investigate student difficulty with programming syntax. Students participating in the study were required to create their own programming exercise prompt and correctly answer at least ten other students' exercises. Denny et al. found that even those students who performed well in the class struggled with the syntax necessary to complete the exercises [11]. No conclusion is drawn, however, on the effect of the exercises on student performance.

2.2.2 Typing Exercises. Another type of practice is "typing exercises." These exercises require no use of logic or knowledge of programming language syntax. Instead, students type lines of code character for character as given to them. In this way, students can become familiar with programming syntax through typing rather than simply reading code examples [14, 23].

Typing exercises, unlike programming exercises, specifically target syntax for practice. Two studies have been conducted to determine the usefulness of these types of exercises on student performance. The first of the studies, by Leinonen et al. [23], required students to complete typing exercises for the first two weeks of the course and then assessed their performance until the end of the course. No significant impact on student performance was found [23]. Another study by Gaweda et al. [14] had different results. They found that students who completed typing exercises had fewer build failures. However, the exercises in this study were voluntary, so students were self-selected. In both of these studies, students were given the code to type, so the student was not expected to remember syntax.

2.2.3 Syntax Exercises. We term the final type of practice "syntax exercises." Syntax exercises require students to recall syntax rather than simply type code given to them as in typing exercises. Yet syntax exercises are designed to eliminate, or at least minimize, cognitive problem solving, focusing rather on repetitive practice for skill development. The low cognitive load and use of repetition help students integrate typing syntax into procedural memory, removing extraneous cognitive load [26] during classroom learning and larger programming projects.

The pedagogy used in our study has been investigated in two prior works. Edwards et al. [13] conducted a study where multiple new teaching methods and resources were explored, including syntax exercises along with pair and in-class programming with teacher assistance. The study found that with these new teaching methods, students spent less time on assignments and found the projects less frustrating. However, no direct conclusions relating to the syntax exercises could be drawn due to confounding variables. The syntax exercises are also discussed more at length in Edwards et al. [12], who conducted a larger controlled study. They found that syntax exercises helped students achieve higher exam scores and reduced attrition, although no conclusive effect could be found on project scores.

2.3 Attitude

Edwards et al. [12] suggested in their study that syntax exercises could not only scaffold learning but could also positively impact motivation and attitude. In an earlier exploratory study, Edwards et al. [13] presented anecdotal evidence that the exercises positively impacted student attitudes toward programming. Both of these studies demonstrated improvement in academic outcomes but only suggested attitudinal improvement, so in this work we explore student attitudes toward syntax exercises. Attitude is an important component of student experience. For example, self-efficacy has been shown to be an effective predictor of college success [38] and it has been suggested that improving the self-efficacy of underrepresented groups, particularly women, would improve their representation in CS [25]. Improving computer experience and performance [7, 15] could increase the likelihood for underrepresented groups to continue in CS. While previous studies explore the performance aspects of syntax exercises, this study looks specifically at student experience.

3 METHODS

3.1 Study Design

We conducted our study in three sections of a university CS1 course in a single semester. All three sections were taught by the same instructor. The course was lecture-based and had weekly programming assignments. Syntax exercises, administered with the *Phanon* software package [12], were used for the first half of the semester. For the second half of the semester, syntax exercises were not available to students. Withholding syntax exercises the second half of the semester was deemed ethical by our IRB as the instructor of the course was unconvinced that they would be effective and was also concerned they would have been an unreasonable additional workload for the students.

3.1.1 Syntax Exercises. Students were assigned a session of about 30 syntax exercises to complete three times a week, before each lecture, until halfway through the semester. Syntax exercises, in the context of this study, were small exercises designed to introduce the syntax of small programming structures, such as `for` loops. The exercises were designed to involve only lower-order cognition and be completed very quickly, about 10-15 seconds each for a total of about 10 minutes for a session. At the beginning of a session, the syntax exercises were extremely simple. In the example of a `for` loop, students were shown a `for` loop with the keyword `for` missing. They were then prompted to insert the word `for` before the `i`. As the session progressed, the exercises increased in difficulty until students were writing entire loops (or whatever construct was being practiced). We used the same exercise sets as were used in the study by [12].

The exercises taught the topics being discussed in the lectures during the week they were assigned. These topics included printing, variables, arithmetic operators, strings, loops, math functions, Boolean expressions, and conditionals. After the exercises were no longer in use, the topics functions, lists, classes, comprehensions, searching, sorting, and multidimensional lists were taught. Students were not given syntax exercises on these topics.

3.2 Survey

At the end of the semester, the students were given a survey to fill out concerning their experience in the course. Points were given for participating in the survey. The first question in the survey was:

> What are your thoughts on Phanon?
> Ideas to discuss:
> Did it help you learn? Did it not make a difference?
> We used it for approximately ½ of the semester. Would more or less be better?
> What could make it better?
> What were things you would change about it?

The students were free to respond with whatever thoughts they had concerning the syntax exercises. 208 students responded to the survey.

3.3 Codebook Creation

Before creating the codebook, the responses were cleaned. A member of the research team read through each response and removed references to areas of the curriculum not related to the syntax exercises. There was some confusion on the part of the students and many discussed the *Phanon* software used to administer the syntax exercises and projects rather than the exercises themselves. These references were removed, leaving 138 responses containing comments on the actual syntax exercises.

Coding proceeded using the method of Saldana [31]. First, the research team met to create a list of codes, or themes, contained in the survey responses. The researchers began by reading the first ten responses and creating a list of relevant codes contained in the responses. Then they split the remaining responses and each took a section to read and find themes in.

After reading their portions of the responses, the researchers discussed themes they had discovered. The result of the discussion was a list of codes in five main categories: "Myself as a Learner," "Requests for Change," "Dislikes," "Likes," and "Overall".

The codebook contained the code name, description, and at least two example responses. While creating the codebook, it was discovered that some of the original codes were too sparse (only mentioned in one or two of the responses). These were combined with other codes. In the end, the codebook contained 27 individual codes. See Table 1.

3.4 Coding Process

To code the student responses, two graduate students were recruited as coders. These coders met with one of the researchers who had created the codebook to receive training on how the coding should be accomplished.

First, the researcher read through each code's description and example with the coders. The codes were discussed and questions were addressed. The researcher and coders then read through six responses together and coded them, ensuring that an agreement was reached on each code.

Next, ten responses were given to each of the coders to code individually. Once this had been completed, the coders' codes were compared using Cohen's kappas [2] and pooled kappas [9] to determine inter-rater reliability. The pooled kappa was 0.609, which is considered moderate agreement [22]. For the individual codes' Cohen's kappas, the lowest was -0.111 and the highest was 1.0, or complete agreement. Two thirds of the responses were above 0.40, which is moderate agreement. For a complete list of the individual Cohen's kappas, see Table 1. Note that the missing values indicate that none of the responses contained the given code.

To improve inter-rater reliability, the codebook creator and coders met a subsequent time and discussed where the coded responses differed. Final codes for the mismatched responses were decided on, and the themes were once again discussed to ensure understanding.

After the discussion, the coders were given ten new responses to code separately. This round of coding led to an improved pooled kappa of 0.643, which is considered substantial agreement. Four of the codes had kappa values of 0.0, but the rest were above 0.40 (85.2%), which is considered moderate agreement [22]. Again, see Table 1 for each of the Cohen's kappas values.

Each coder took half of the remaining responses to code.

Code	Kappa (Rnd 1)	Kappa (Rnd 2)
Novice Programmer	1.0	-
Some Experience Programming	1.0	1.0
Add exercises to second half	0.8	0.814
Same amount of exercises	-0.111	0.744
Fewer exercises	-0.111	0.0
Increase Level of Challenge	-	0.7843
Buggy	1.0	-
Tedious/Too Repetitive	0.524	0.744
Annoying	0.615	0.0
Easy/Simple (positive)	0.0	0.621
Stress-free/Safe	1.0	-
Concepts in small pieces	0.615	-
Application/concrete examples	0.0	0.621
Helpful repetition	1.0	0.744
Helped develop “feel” (mental)	0.348	0.607
Helped muscle memory (physical)	0.737	0.0
Helped learn syntax/mechanics	1.0	1.0
Helped learn code outcomes	-	-
Entry point/on-ramp	0.412	-
Prepped for projects	0.0	0.621
Prepped for lectures	0.615	0.0
Enjoyment	0.737	-
Confidence	1.0	-
Helpful	0.615	0.421
Not Helpful	-	1.0
Liked	0.4	0.421
Disliked	0.0	1.0

Table 1: Kappa values from each round of coding. Pooled kappa for round 1 was 0.609 and for round 2 was 0.643.

Code	Responses	Percentages %
Novice Programmer	12	9
Some Experience Programming	5	4
Add exercises to second half	63	46
Same amount of exercises	18	13
Fewer exercises	10	7
Increase Level of Challenge	12	9
Buggy	13	9
Tedious/Too Repetitive	23	17
Annoying	5	4
Easy/Simple (positive)	16	12
Stress-free/Safe	5	4
Concepts in small pieces	3	2
Application/concrete examples	22	16
Helpful repetition	23	17
Helped develop "feel" for the code (mental)	56	41
Helped with fluency/muscle memory (physical)	8	6
Helped learn syntax/mechanics	18	13
Helped learn code outcomes	3	2
Entry point/on-ramp	17	12
Prepped for projects	15	11
Prepped for lectures	16	12
Enjoyment	11	8
Confidence	4	3
Helpful	119	86
Not Helpful	7	5
Liked	116	84
Disliked	7	5

Table 2: Counts and percentages of responses containing the shown codes.

3.5 Analysis

Once all responses had been coded, the codes were used to analyze the responses. Counts and percentages of how often each code occurred were found. Using these values, we can determine some of the general attitudes of the students toward the syntax exercises.

4 RESULTS

4.1 General attitudes

Our first research question is: *What are student attitudes toward a syntax-first curriculum?* Student responses to the survey were strongly positive toward syntax practice: the majority of the responses contained positive comments about the utility of the syntax exercises or enjoyment of the exercises (88.4%, 122), although a small number of responses (5.8%, 8) contained comments about the unhelpful nature of or negative feelings toward the exercises. Table 2 shows the number of responses containing comments regarding each code. In the following sections we discuss some key aspects of the students' responses.

84.1% (116) of the students reported that they liked the exercises while 5.8% (8) students didn't like them. Students generally understood the purpose of the exercises: fifty-nine responses (42.8%) contained mention of a specific characteristic of the exercises. In particular, students mentioned that they appreciated that the exercises were generally simple to complete (11.6%, 16) and that the exercises allowed them a stress-free zone to experiment with programming (3.6%, 5). One student commented, "The exercises ... really helped me to understand coding without stressing about it." Some students also mentioned that the exercises taught the programming concepts in small pieces (2.2%, 3), applied the concepts or gave concrete examples (15.9%, 22), and provided helpful repetition for learning to program (16.7%, 23).

4.2 Perceptions of utility

Our second research question is: *What are student perceptions of how a syntax-first curriculum affects academic outcomes?* Of the 138 relevant responses from students participating in the study, 86.2% (119) mentioned that the syntax exercises were helpful, and 62.3% (86) mentioned specific ways that the exercises helped them in the course. Some of the ways students mentioned the exercises were helpful included developing a mental "feel" for the code (40.6%, 56), building muscle memory (5.8%, 8), learning syntax or mechanics (13.0%, 18), learning code outcomes (2.2%, 3), being a good entry

point for beginners (12.3%, 17), and helping prepare for lectures or projects (18.1%, 25).

Possibly most interesting from these responses are those that mention the exercises helped them develop a "feel" for the code and those that mention they helped them learn code outcomes. These were not intended benefits of the exercises, so it is interesting that students identified them. Some of the responses included statements that the exercises "helped the student to understand some basic concepts of programming" or "helped to solidify new concepts." One of the students mentioned that the exercises helped "seeing the results of entered code." Perhaps it was helpful for the students to see the results of the small snippets of code bit by bit and repeated, as opposed to a few times in class and possibly never again, unless specifically explored by the student.

Sixteen (11.6%) of the students mentioned that the exercises helped them prepare for the lectures. In particular, one student's comments contained the following:

> "A lot of times in class the demo code could get complex and if I had missed or didn't quite understand the simpler concept being taught, I would be completely lost on the more complex application. Phanon covered the concepts taught in class before they were taught. So I could get the basic idea from Phanon, then come to class and expand upon that knowledge."

This demonstrates that the Phanon exercises were able to accomplish one of the goals set for them: to help the students better follow along with lectures, since they can focus on what the professor is teaching instead of getting lost in what he/she is typing. Jenkins discusses the importance of this: "there is surely little point in lecturing students on syntax when they have no idea of where and how to apply it" [17]. By introducing the syntax beforehand and demonstrating what it does, students can come to class and focus on problem solving and where to include the introduced programming constructs. This is especially interesting because it is was one of the goals of the syntax exercises: to reduce the cognitive burden felt by students during lectures and allow the instructor to focus on the problem-solving side of programming without worrying about overwhelming students.

Finally, 15 (10.9%) students mentioned that the exercises increased either their confidence or enjoyment of programming. One student mentioned that the exercises "got [her/him] excited to code." Another response included the following: "I felt confident and understanding of the material until Phanon stopped being used." Studies have found that beginning students find programming "boring and difficult" [17]. An increase in enjoyment of programming might help to explain the decrease in attrition seen in Edwards et al. [12]. In addition, confidence is a key factor in student success in programming courses, and especially in underrepresented groups [16].

4.3 Negative Attitudes

Not all of the students spoke highly of the syntax exercises in their responses. The responses included eight (5.8%) that mentioned disliking the exercises and 5.1% (7) that did not find them helpful. There were three major reasons why the students did not enjoy the exercises: they were buggy, too repetitive, or annoying.

Phanon was known to have some usability issues, such as being picky about white space. Some students mentioned the glitches, stating that "sometimes Phanon was a little frustrating with the way it checked to see if the answers were correct." A smoother IDE for the students might be able to reduce frustration with programming even further.

Twenty-three (16.7%) of the students mentioned that the exercises were too tedious or repetitive. Many of the students suggested decreasing the amount of repetition in the exercises. However, 13% of the students that mentioned that the exercises were repetitive also admitted that they felt the repetition was beneficial to their learning. For example, one student stated that the exercises "felt very repetitive (which honestly helped get the concept down) but some more variation would be nice."

An interesting point to mention about the students' responses is that of the 8 responses that contained negative sentiments toward the exercises, half also mentioned a way in which the exercises were helpful. This shows that even though the students may have found the exercises inconvenient, they could also see the value in the exercises.

Another complaint of the students was in the difficulty of the exercises. This was mentioned in 12 (8.7%) of the responses. Some of the students would have preferred less repetitive exercises that challenged them more. For example, one student wrote, "I saw Phanon more as a tutorial and I wanted a more exciting challenge." Although an understandable request, the syntax exercises were designed to be simple and repetitive, complementing the "more exciting challenge" of the weekly projects.

4.4 Effect of Prior Experience

Prior experience in programming affected how students viewed the exercises, sometimes negatively. Of the five students who reported having prior programming experience, two reported disliking the syntax exercises.

Two of the students with prior experience mentioned that although they had not found the exercises particularly helpful for themselves, they understood that beginners would find them useful. One of the students said, "I have a pretty good amount of experience programming so the Phannon exercises were a little bit annoying... I could definitely see how it would help newer programmers become less afraid of syntax so they can focus more on problem solving."

This also sheds some light on an area of possible concern: the expertise reversal effect. The expertise reversal effect is when a teaching technique that may be beneficial to novices is not beneficial to those with experience, and may even be detrimental to the more experienced student's learning [19]. From our results we can see this is possibly an issue with the syntax exercises, as a large percentage of the students with prior programming experience reported finding the exercises unenjoyable. However, due to the small number of students with prior experience participating in the study (5), another study with a larger sample size would have to be conducted to draw firm conclusions.

4.5 Length of Use

One of the prompts the students were given related to the fact that syntax exercises were only used for half of the semester. The

responses were mixed, with some students believing the amount of time to be just right (13%, 18), some asking for more (45.7%, 63), and some asking for less (7.4%, 10). Some of the students' comments requesting more exercises noted the exercises would have been beneficial while learning more complex topics, particularly classes. One student's response included the following: "I would have really benefited from [the exercises] when we got into the classes." Another student said, "I would have loved to have phanon exercises for object oriented coding, lists, sorting, and so on." Other students felt that the exercises would not have been beneficial for the more complex subjects. For example, "I think it was good to stop using it half way through. by that time the sentex [*sic*] wasn't so much the issue as was the main concepts and I feel the phanon exercises mainly help with the sentex." Another stated, "I don't know how Phanon would help us as we started learning bigger concepts like functions and classes."

Another recommendation made by two students was to keep the amount of time using the exercises the same, but to reduce the number of exercises per session. The response of one of these students contained, "I think it should be used more probably even all the way through the semester... I think some of the assignments were kind of annoying and could be improved by better hints and less lengthy for some." This echoes other comments about the redundancy and repetition of the exercises. However, even the student whose comment is previously quoted mentioned that they felt the "easy repetitive practice" was an effective way to learn.

5 DISCUSSION

Overall, students seemed to appreciate the exercises and believe them to be beneficial. In particular, it seemed that students seemed to perceive the exercises as beneficial for those new to programming. This was evidenced by students mentioning either their lack of experience with programming and the consequent benefit of the exercises, or their prior experience and the lack of benefit.

In addition, it was interesting to note that many students perceived that the exercises helped them understand how code works and the relationships between code and outcomes. This was not necessarily a motivation of the exercises, but it is consistent with the suggestion by Edwards et al. [12] that they could inherently improve cognitive understanding of the code: "repetitive exercises, sometimes derided as drills, can nonetheless be a fountainhead of insight and discovery" [35]. The fact that fully 40% of respondents indicated that the exercises gave them a mental "feel" for the code suggests that not only are the exercises helping the students with a deeper understanding of the syntactic structures, but that there may also be a metacognitive element at play, i.e., students' awareness of their own learning may be heightened.

One factor that *was* intended as a motivator for syntax exercises was a scaffold of classroom learning. It is known that syntax is an "extraneous cognitive load" [33] when students are also struggling with learning to problem solve [20]. 12% of students responded that the exercises prepped them for lectures and 11% said they were helpful specifically for the weekly programming projects.

It may not be surprising that students were generally positive toward the syntax exercises, but what is surprising is the level of their enthusiasm. After all, one might not expect students to get excited about a learning tool that will require roughly 25 minutes of extra work each week. Yet only 5% of respondents said they didn't like the exercises. Of course, there were negative comments, yet students appeared to overwhelmingly approve of syntax exercises, especially after being given the context of not having them available in the second half of the semester.

5.1 Threats to validity

Points were awarded for participating in the survey and students may have felt pressured to respond positively even though they were informed that points were awarded for participation only. Furthermore, while the instructor of the course did not create *Phanon* or the syntax exercises, students may have believed him to be the creator and therefore may have responded more positively than they might have otherwise [10]. In addition, while the instructor of the course in our study was, at first, not completely confident that the syntax exercises were a good idea, he may have mentioned some of the motivations in class, priming the students to respond on specific topics such as scaffolding for classroom instruction and helpful repetition.

6 CONCLUSIONS

A shortcoming to the purely quantitative approach of Edwards et al. [12] to evaluating syntax exercises in CS1 was that, while some attitudes of students were inferred from the data, direct deductions could not be made. Specifically, attrition went down, so the authors suggested that student attitude and self-efficacy may have been improved, and some interpretations of the data suggested a decrease in plagiarism, suggesting similar emotive responses. In our work reported in this paper, we qualitatively analyzed survey responses relating to syntax exercises. The surprising magnitude of positive attitudes toward an intervention requiring extra work from students indicates strong potential for a practice-based, syntax-first CS1 pedagogy and curriculum.

Our work is also consistent with the theoretical underpinnings of syntax practice that suggest increased affect with more manageable cognitive load. Further, Edwards et al. [12] suggested that students who would normally fail or drop out of the course are particularly affected by the intervention. The fact that students reported improvement in confidence and enjoyment in programming supports the findings of lower attrition and plagiarism. Importantly, it may also indicate that the experience of students from underrepresented groups may be affected, broadening participation and increasing diversity. Of course, this is conjecture, as our study did not gather demographic data, and would be a good direction for future study. We also suggest a more precise definition of "syntax exercises." What exactly constitutes an effective syntax exercise is somewhat nebulous. Indeed, after completing our planned weeks of syntax exercises, our instructor attempted authoring additional exercises and found it challenging to get them just right. We suggest either a clear definition of exactly what syntax exercises are or, at least, a recipe or template for authoring the exercises.

REFERENCES

[1] Vicki L Almstrum, Orit Hazzan, Mark Guzdial, and Marian Petre. 2005. Challenges to computer science education research. *ACM SIGCSE Bulletin* 37, 1 (2005), 191–192.

[2] Mousumi Banerjee, Michelle Capozzoli, Laura McSweeney, and Debajyoti Sinha. 1999. Beyond kappa: A review of interrater agreement measures. *Canadian journal of statistics* 27, 1 (March 1999), 3–23.

[3] Theresa Beaubouef and John Mason. 2005. Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. *SIGCSE Bull.* 37 (2005), 103–106.

[4] Jens Bennedsen and Michael Caspersen. 2019. Failure Rates in Introductory Programming: 12 Years Later. *ACM Inroads* 10 (2019), 30–36.

[5] Jens Bennedsen and Michael E. Caspersen. 2007. Failure Rates in Introductory Programming. *SIGCSE Bull.* 10 (2007), 32–36.

[6] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, Vol. 15. Consortium for Computing Sciences in Colleges, 107–116.

[7] Shelley J Correll. 2004. Constraints into preferences: Gender, status, and emerging career aspirations. *American sociological review* 69, 1 (2004), 93–113.

[8] Tebring Daly. 2013. *Influence of Alice 3: Reducing the Hurdles to Success in a CS1 Programming Course.* Ph.D. Dissertation. University of North Texas, Denton, TX.

[9] Han De Vries, Marc N Elliott, David E Kanouse, and Stephanie S Teleki. 2008. Using pooled kappa to summarize interrater agreement across many items. *Field Methods* 20, 3 (Aug. 2008), 272–282.

[10] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. "Yours is Better!": Participant Response Bias in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1321–1330. https://doi.org/10.1145/2207676.2208589

[11] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. Understanding the syntax barrier for novices. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*. 208–212.

[12] John Edwards, Joseph Ditton, Dragan Trninic, Hillary Swanson, Shelsey Sullivan, and Chad Mano. 2020. Syntax exercises in CS1. In *Proceedings of the 16th Annual Conference on International Computing Education Research (ICER '20)*.

[13] John M Edwards, Erika K Fulton, Jonathan D Holmes, Joseph L Valentin, David V Beard, and Kevin R Parker. 2018. Separation of syntax and problem solving in Introductory Computer Programming. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

[14] Adam M. Gaweda, Collin F. Lynch, Nathan Seamon, Gabriel Silva de Oliveira, and Alay Deliwa. 2020. Typing Exercises as Interactive Worked Examples for Deliberate Practice in CS Courses. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*. 105—-113.

[15] Jun He and Lee A Freeman. 2019. Are men more technology-oriented than women? The role of gender on the development of general computer self-efficacy of college students. *Journal of Information Systems Education* 21, 2 (2019), 7.

[16] Lilly Irani. 2004. Understanding Gender and Confidence in CS Course Culture. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)*. Association for Computing Machinery, New York, NY, USA, 195–199. https://doi.org/10.1145/971300.971371

[17] Tony Jenkins. 2002. On the Difficulty of Learning to Program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. 53–58.

[18] Addie Johnson. 2012. Procedural Memory and Skill Acquisition. In *Handbook of Psychology, Experimental Psychology*. John Wiley & Sons, Incorporated, 495–520.

[19] Slava Kalyuga. 2009. The expertise reversal effect. In *Managing cognitive load in adaptive multimedia learning*. IGI Global, 58–80.

[20] Paul A Kirschner, John Sweller, and Richard E Clark. 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist* 41, 2 (2006), 75–86.

[21] Sarah K Kummerfeld and Judy Kay. 2002. The neglected battle fields of Syntax Errors. Australian Computer Society. In *Proceedings of the fifth Australasian conference on Computing education*. 105–111.

[22] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (March 1977), 159–174. http://www.jstor.org/stable/2529310

[23] Antti Leinonen, Henrik Nygren, Nea Pirttinen, Arto Hellas, and Juho Leinonen. 2019. Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 84—-90.

[24] Colleen M. Lewis. 2010. How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. 346—-350.

[25] Hilary M Lips and Linda Temple. 1990. Majoring in computer science: Causal models for women and men. *Research in Higher Education* 31, 1 (1990), 99–113.

[26] Raymond Lister. 2011. Programming, syntax and cognitive load (part 1). *ACM Inroads* 2, 2 (2011), 21–22.

[27] Linda Mannila, Mia Peltomäki, and Tapio Salakoski. 2006. What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education* 16, 3 (2006), 211–227. https://doi.org/10.1080/08993400600912384 arXiv:https://doi.org/10.1080/08993400600912384

[28] N. Parlante. 2018. CodingBat. http://codingbat.com

[29] Dale Parsons and Patricia Haden. 2007. Programming Osmosis: Knowledge Transfer from Imperative to Visual Programming Environments. In *20th Annual Conference of the National Advisory Committee on Computing Qualifications*. 209–215.

[30] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

[31] J. Saldana. 2015. *The Coding Manual for Qualitative Researchers*. SAGE Publications, Los Angeles.

[32] Andreas Stefik and Susanna Siebert. 2013. An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 19.

[33] John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. 1998. Cognitive architecture and instructional design. *Educational psychology review* 10, 3 (1998), 251–296.

[34] D. Thomas. 2018. CodeKata. http://codekata.com

[35] Dragan Trninic. 2018. Instruction, repetition, discovery: Restoring the historical educational role of practice. *Instructional Science* 46, 1 (2018), 133–153.

[36] David Weintrop. 2016. *Modality Matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms*. Ph.D. Dissertation. https://login.dist.lib.usu.edu/login?url=https://search.proquest.com/docview/1826352865?accountid=14761 Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2019-10-19.

[37] David Weintrop and Uri Wilensky. 2017. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 3.

[38] Anna Zajacova, Scott M Lynch, and Thomas J Espenshade. 2005. Self-efficacy, stress, and academic success in college. *Research in higher education* 46, 6 (2005), 677–706.