8-2020

An Analysis of Syntax Exercises on the Performance of CS1 Students

Shelsey B. Sullivan
Utah State University

AN ANALYSIS OF SYNTAX EXERCISES ON THE PERFORMANCE OF CS1 STUDENTS

by

Shelsey B. Sullivan

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

John Edwards, Ph.D. Major Professor	Vladimir Kulyukin, Ph.D. Committee Member
Hillary Swanson, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

An Analysis of Syntax Exercises on the
Performance of CS1 Students

by

Shelsey B. Sullivan, Master of Science
Utah State University, 2020

Major Professor: John Edwards, Ph.D.
Department: Computer Science

Learning to program can be a frustrating experience. Programming language syntax rules can be especially confusing to learn. Different approaches have been proposed to minimize the barrier of learning syntax for beginning Computer Science students, such as the introduction of visual or block-based languages. However, although these approaches minimize the mental struggle of learning syntax at the beginning, programming students will eventually have to learn programming language syntax.

At Utah State University a study was conducted to analyze the effect of syntax exercises on students' performance in CS1. The study took place over two semesters. The first semester was the control group; the second semester syntax exercises were introduced. These exercises were implemented in a program created for the study called Phanon. Phanon was also used to complete the weekly homework assignments. While completing the homework assignments, students' keystroke data was collected. This thesis describes the analysis of this data, along with other data collected during the study. This data included the students' exam and assignment scores, assignments, GPAs, and survey responses about the Phanon exercises.

The prediction was that syntax exercises improve student performance and reduce the frustration felt by many when attempting to program. The analysis included looking at the difference between the time and number of events taken by students to complete their assignments. To see if there was a difference between the effect of syntax exercises on high and low performing students, the students' GPAs were used to separate the students into two groups. The data was analyzed to see if the syntax exercises affected the groups differently. In addition, qualitative and quantitative approaches were used to analyze the students' affective states, as reported in survey responses about their feelings towards the Phanon exercises.

If the syntax exercises are found to be beneficial, they could be introduced into CS1 classes to help more students join the field of Computer Science. More students would continue to the next CS courses, including those in minority groups, bringing more diversity into the workplace.

(51 pages)

PUBLIC ABSTRACT

An Analysis of Syntax Exercises on the
Performance of CS1 Students
Shelsey B. Sullivan

Students in introductory programming classes (CS1) generally have a difficult time learning the rules of programming. Although the general concepts of programming are relatively easy to learn, it can be difficult to learn what exactly can be typed in what order, which is known as syntax. To attempt to help students overcome this barrier, a study was conducted that introduced exercises into a CS1 class which taught the programming syntax in simple steps. The results of this study were obtained by analyzing the keys the students pressed, the errors of their code, their midterm exam scores, and their responses to a short survey.

It was found that the syntax exercises did not reduce the amount of time spent on the assignments or the number of keys the students pressed. However, they did help build the students' confidence, as shown by an increased number of students that continued to attempt the assignments.

This could have a huge impact on the field of computer science, as a decrease in student frustration and an increase in student success will lead to more students remaining in computer science programs. More students in computer science programs will lead to a fuller, more diverse workforce.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer programming is known as a particularly difficult subject to teach and to learn [1]. Programming is a complicated skill to learn and, as Jenkins points out, is a hierarchy of skills. Not only must students understand how to think in a new way in order to solve problems, they must do so using words and symbols in a way that at first can seem meaningless and confusing [2].

One indicator that the ways of teaching introductory programming can be improved is in the courses' attrition rates. In 2007, Bennedsen and Casperson found that the rate of students failing introductory Computer Science (CS1) classes was 33% [3]. A repeat study in 2019 found the failure rate had dropped to 28%, which although better, could still be improved [4]. Many researchers have investigated ways to improve teaching programming.

Many reasons have been proposed to explain why so many students fail CS1 courses [5], and still more completely drop out of Computer Science programs [6]. One reason that stands out is the difficulty of learning syntax [1], [2]. Multiple solutions have been proposed to help reduce the cognitive burden of learning syntax in programming. Such solutions include creating visual or block-based programming languages, such as Alice and Scratch. These languages are discussed more in the Related Works section.

The purpose of this thesis is to analyze the results of a recent study done at Utah State University. The study investigated a different approach to solving issues of student frustration with learning programming syntax. Instead of taking the syntax out, as is done in visual programming languages, students were given exercises to complete before each class. These exercises allowed students to practice typing small amounts of code from the concepts currently being taught in their classes.

The prediction was that doing so would introduce them to and create muscle memory for the syntax [7]. In turn, the hope was that this would allow the students to finish their

assignments faster, with fewer syntax errors, and perform better on their midterm exams. In addition, it is predicted that the students who could be considered "high-performing," or who consistently achieve in classes, will improve less due to the syntax exercises than their "low-performing" peers. This is because high-performing students will find ways to succeed no matter what, but low-performing students may need more help.

CHAPTER 2

RELATED WORkS

To improve CS1 pass rates and increase the number of students who remain in Computer Science programs, much research has been conducted. Computer Science Education research has explored a variety of different ways to improve students' understanding of Computer Programming. Some of these areas include the use of exercises, simplified languages, and visual programming languages. Previous research relating to these areas are discussed in this section.

2.1 Exercises

Three types of exercises have been suggested previously in literature. First is what will be termed "programming exercises." These exercises include logic, asking students to solve various problems. For example, a programming exercise might ask students to write a function to find all prime numbers between two given numbers. In this way, the student practices both typing and using the programming language syntax, as well as using logic to solve problems. The second type of exercise is called "typing exercises." These exercises do not include logic. Instead, students are shown snippets of code, such as an if statement. Students then must type the code exactly, thereby actively learning about the syntax, as opposed to passively, such as by simply reading code snippets [8], [9]. Finally, as is done in this study, there are "syntax exercises." These exercises break down programming concepts into granular bits for the students to fill in, thereby introducing the syntax step by step. More details on these syntax exercises are provided in the study methodology section of this thesis.

2.1.1 Programming Exercises

Many websites use programming exercises to teach beginners the basics of programming

[10], [11]. These websites include, but are not limited to, CodingBat and CodeKata. Denny et al. [12] conducted a study using a website called CodeWrite that allowed students to create their own programming exercises. The students were then required to complete at least ten of their peers' exercises. Based on the students' performance, the authors were able to determine that all students, even those that achieved high scores in the class, wrote code with syntax errors about two thirds of the time. This interfered with the students' ability to receive feedback on their code's logic, showing that syntax is a major barrier that students must overcome [12].

2.1.2 Typing Exercises

Leinonen et al. [9] and Gaweda et al. [8] both utilized what they termed "typing exercises." The two studies differed in length and yielded contradictory results. Leinonen et al.'s study lasted two weeks at the beginning of the course. The results of this study showed that there were no major differences between the students who completed the typing exercises before completing programming exercises and those who did not complete typing exercises. It did not significantly affect the number of typing events the students had, nor the amount of time it took them to complete the programming exercises.

Gaweda et al.'s study [8], on the other hand, lasted the entire semester. In addition, students were given the opportunity to find and fix the errors in their code by being shown the line of code where the errors occurred only, rather than the actual character that was out of place, as was done in Leinonen et al.'s study [9]. The results of this study showed that students who completed typing exercises, meaning submitted the typing exercise without errors, had fewer syntax errors on their homework assignments and performed better in the class. However, the exercises were completely voluntary, so it is also possible the positive results were caused by higher-performing students completing the exercises.

2.1.3 Syntax Exercises

This study was a follow up to an exploratory study done previously [13]. In the previous study, syntax exercises were combined with pair programming and in-class work with teacher

help. From the study the authors found that the combination allowed the test group to spend less time on assignments, although they spent more time on the class overall. The test group also found the projects less frustrating and challenging. However, due to the number of variables, no firm conclusions could be made.

2.2 Simplified Languages

Another area that has been proposed to help students overcome difficulties with syntax is to create simplified languages. Stefik and Siebert [2] investigated types of syntax that are easiest for beginning programming students to understand. They found that intuitive syntax does help beginners. Using their findings about what syntax is easiest and most intuitive for beginners to understand, they were able to create a language called Quorum [2].

Mannila et al. [14] investigated the ability of students to transfer learning from simpler programming languages (Python) to programming languages with more complex syntax (Java). They found that the students were able to understand the basic concepts of programming and carry them over to the more complex languages. The students also felt that learning a language with simpler syntax had prepared them better for learning the language with the more difficult syntax. However, the majority of the students' syntax errors while programming using the more complex language originated from differences in the two languages, such as the use of brackets and semicolons.

2.3 Visual Languages

Another proposed solution to the issue of learning syntax is visual or block-based languages, such as Alice and Scratch. Visual languages remove syntax completely by allowing programmers to simply move code boxes around and change number values within the boxes to create programs [15]. These languages help students at the beginning, but the students will have to face real syntax eventually [16], [17]. In addition, the languages are used solely for learning purposes and are not used in the industry, which is the destination of almost every Computer Science major [1], and therefore not an ideal language to learn.

Finally, previous research has shown these languages do not improve student confidence

[18], and students are not easily able to transfer their knowledge learned from them to "real" programming languages [19]. Another study found that students had higher levels of interest and slightly better performance when learning to program using block-based languages, but students learning to program using a simplified text-based language felt more confident once they began learning Java [20], [21]. In the end, all students, regardless of the type of language with which they began learning, had roughly the same feelings towards programming [20], [21].

In conclusion, if visual programming languages are not going to help students learn basic programming concepts, they offer no real pedagogical benefit. Instead, researchers should focus on other areas to improve students' learning. Simplified programming languages have been shown to be able to teach students generalizable programming concepts. To continue to investigate the best practices to help students overcome the strain of learning syntax, this study explored the utility of syntax exercises.

CHAPTER 3

STUDY METHODOLOGY

The purpose of this study was to test how the addition of syntax exercises affected students' performance in a CS1 class. The analysis of this study includes both quantitative data, such as the students' keystrokes and grades, and qualitative data, obtained from student responses at the end of the test semester.

3.1 Study Overview

This study was conducted over the course of two semesters of CS1 classes, Spring and Fall 2019. The first semester was the control group, and the second semester was the test group. Participation was voluntary and the first semester 271 of 353 students opted to participate. The second semester 254 of 373 opted to participate. Both semesters were taught in person using Python. The first semester had three sections taught, one by professor A and the other two by professor B. The second semester had three sections, all taught by professor B.

For the first half of both semesters, the students used an online program called Phanon to complete their weekly programming projects. Phanon was created for the purpose of a previous study [13] and recorded the students' keystroke data. This data included what and when the students typed, as well as their run events with compilation information. The compilation information included whether the program had syntax errors.

After the first half of the semester, Phanon use was discontinued and students finished the semester using PyCharm, a regular editor. As such, the analysis presented in this thesis focuses on the data collected during the first half of the semester.

The keystroke data collected by Phanon, along with exam grades, was used to analyze the students' performance in the class. The researchers were able to use this data to calculate and compare the amount of time students spent on their assignments, as well as the number

of keystrokes and events that occurred.

3.2 Phanon Exercises

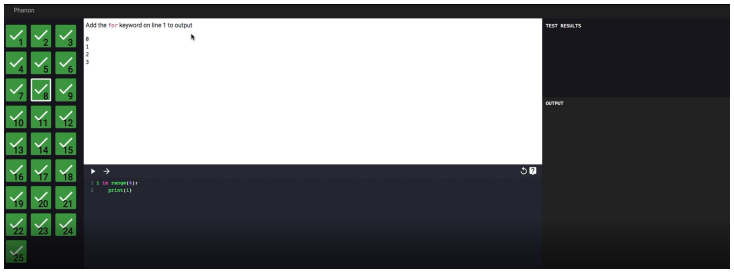
The second, or test, semester, syntax exercises were introduced. Once again, Phanon was used as the medium through which the students completed these exercises. These exercises are designed to be small, taking about 10-15 seconds each. Each session contains about 30 exercises and the students were assigned about three sessions per week, due before class. The sessions contain exercises pertaining to what would be taught the following class period.

Unlike previous research and programs involving programming exercises that require logic and critical thinking [12], [11], these exercises are designed to engage lower-order cognition. The syntax exercises show the students programming structures, such as for loops, with parts missing. The students are then given specific instructions on how to fill in the missing pieces of syntax. For example, a student might be shown a Python *for loop* with the keyword *for* missing and instructions to put it before the i on the first line, as shown in Fig. 3.1.

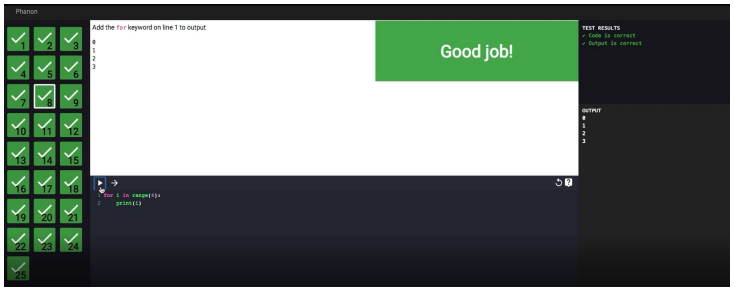
If the student makes a mistake in the exercise, the program tells them that something is wrong and shows them the console output. When the student correctly fixes the syntax, a message appears telling them "Good job," and the student is shown the next syntax exercise. The exercises become progressively more complicated as the student continues, going from filling in a single keyword or variable to writing entire lines of code.

The exercises allow the students to get used to writing common code snippets and put them into procedural memory, as well as allow them to build their confidence in writing code by giving them many successes. In addition, the hope was to build the students' muscle memory and mechanical skills through repeated practice and practice schedules [7].

Phanon is inspired by "The Virtuoso Pianist" by C. L. Hanon, a book of piano exercises designed to increase a pianist's skill. The name Phanon comes from "Programming Hanon."



(a) Screen and prompt.



(b) After a student has successfully completed a Phanon exercise.

Fig. 3.1: Example of the Phanon exercise screens.

3.3 Survey Responses

At the end of the second semester, the students were given a chance to give their feedback on the Phanon exercises. This provided an opportunity to do qualitative research, as well as quantitative, as suggested by Hazzan in order to better understand the effect on the students [22]. The students were given the following prompt, but were allowed to respond however they liked:

> "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 of the semester. Would more or less be better?
> What could make it better?
> What were things you would change about it?"

One thing to note is that some responses referred to the Phanon software, in addition to the Phanon exercises. References to the Phanon software were removed, as they were not applicable to the data analysis. A total of 208 participants responded to the survey, and 138 of those contained references to the Phanon exercises that could then be analyzed.

To analyze the students' responses, the guidelines outlined by Saldaña in *The Coding Manual for Qualitative Researchers* [23] were followed. First, the research team (myself, John Edwards, and Hillary Swanson) read through ten of the students' responses and came up with a list of possible themes or codes. Then, the remaining responses were split and read by the researchers individually. From those responses, more codes were determined, and then a final list of codes was created.

I created a codebook to explain each of the codes or themes for which the responses would be coded. The codebook contains descriptions of each code, along with at least two examples from the students' responses. During the process of creating the codebook, some codes were combined with others, mostly due to a small number of responses containing those codes. In the end, the codebook contained 27 codes. For the complete codebook, see Table A.1 in the Appendix.

Once the codebook was created, two students were recruited to help with the coding process. The coders and I met. First, the codebook was discussed. The description of each code/theme was read, along with the examples from the responses. Then, six responses were read through and labeled for each code together. When it was determined that the coders had a sufficient understanding of the codes, they were each given ten responses to code separately.

These ten responses were then checked for interrater reliability by finding the Cohen's kappa for each code [24], as well as the pooled kappa [25]. The pooled kappa was 0.6086, which is considered moderate agreement [26]. The individual Cohen's kappas can be seen in Table 3.1.

As can be seen in Table 3.1, some codes resulted in high agreement, such as "Novice Programmer" or "Buggy." Others resulted in very low kappas, such as "Same amount of exercises," or "Disliked." Even others resulted in "NaN." This was due to the sparsity of the data. The codes with "NaN" values were coded completely in agreement, but none of the ten responses coded contained those codes.

Due to the very low kappas, the codebook creator and coders met again to discuss

Code	Kappa (Round 1)	Kappa (Round 2)
Novice Programmer	1.0	NaN
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	NaN	0.7843
Buggy	1.0	NaN
Tedious/Too Repetitive	0.524	0.744
Annoying	0.615	0.0
Easy/Simple (positive)	0.0	0.621
Stress-free/Safe	1.0	NaN
Concepts in small pieces	0.615	NaN
Application/concrete examples	0.0	0.621
Helpful repetition	1.0	0.744
Helped develop “feel” for the code (mental)	0.348	0.607
Helped with fluency/muscle memory (physical)	0.737	0.0
Helped learn syntax/mechanics	1.0	1.0
Helped learn code outcomes	NaN	NaN
Entry point/on-ramp	0.412	NaN
Prepped for projects	0.0	0.621
Prepped for lectures	0.615	0.0
Enjoyment	0.737	NaN
Confidence	1.0	NaN
Helpful	0.615	0.421
Not Helpful	Nan	1.0
Liked	0.4	0.421
Disliked	0.0	1.0

Table 3.1: Kappa values from each round of coding.

where the coders had differed in their coding. Some of the disagreement was discovered to be due to small mistakes, others were due to misunderstanding the codebook. In addition, the small sample size very likely played a role in the low reliability.

The coders then coded ten more responses. The results of the second round were slightly better, with a pooled kappa of 0.643, which is substantial agreement. Unfortunately, there were still 4 codes with a value of 0.0, but the rest were above 0.42, or moderate agreement [26]. The individual kappa values can be seen in Table 3.1.

Despite some of the reliabilities being less than desirable, it was decided that a higher interrater reliability would probably not be reached. The rest of the responses were divided,

and each researcher coded their half of the remaining responses individually. These codes were then used to analyze the students' responses.

CHAPTER 4

RESULTS

This section documents the results of the study. It is comprised of two major sections: Class Performance and Phanon Responses. In the Class Performance section, the students' assignment keystroke data, assignment submissions, midterm exam scores, and attrition are analyzed. The Phanon Responses section covers the analysis of the students' responses and feelings towards the Phanon exercises.

One thing to note is the number of times p-values are reported. Over 20 p- and t-values are reported. Although used to get an idea of the difference between the two groups, the p-values should be considered cautiously, as more significance tests that are run, the less meaningful the p-values become.

4.1 Class Performance

Using the data collected by Phanon, analyses were done on the students' keystroke data. This includes the amount of time taken on the assignments, the number of keystrokes done by the students, and the number of syntax errors the students encountered while working on their assignments. This section will look at these analyses, as well as the students' midterm scores and their attrition.

This section is broken into two sections: a general analysis into the effect of syntax exercises on the students, and an analysis into the effect of syntax exercises on students who would normally struggle in a CS1 class. The determination of which students are high- and low-performing was based on previous math and CS grades. A full explanation is given in section 4.2.

4.1.1 General Analysis

In this section we analyze the keystroke data from projects 4 to 8. Project 4 was the

first to involve programming, and project 8 was the last project to be done using the online program, Phanon. Project 8 was completed about halfway through the semester.

First, an examination into the effect of syntax exercises on the amount of time taken on assignments was conducted. The test group took significantly less time than the control group on project 6.1 (t=20750, p=0.0236). The test group took significantly more time on both tasks in project 8 (task 0: t=11990, p=$4.227e^{-9}$, task 1: t=14460, p=$1.654e^{-6}$). A graph of the amount of time taken can be seen in Fig. 4.1.

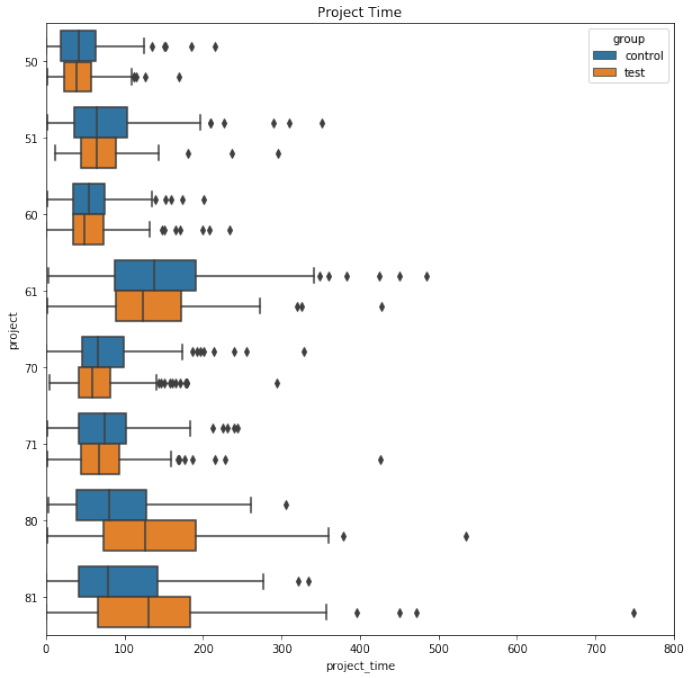


Fig. 4.1: Amount of time taken by the students per assignment, separated by groups.

Research has found that the amount of time a student takes on their assignments is positively correlated with the student's grades [27], so this increase in time on project 8 could be a positive effect. However, it was the hope of the researchers that the amount of time the students spent on their assignments would decrease, since the students would be more familiar with the syntax and therefore be able to program with greater ease. We hoped that the students would have to spend less time looking up syntax or trying to remember syntax, and as such spend less time on their assignments.

It is also important to note that the time the students spent in the Phanon IDE is not a complete indicator of the total amount of time the students spent on their assignments. It is possible the students spent time planning their assignments using a different medium, such as pencil and paper.

Students' keystroke data also show some interesting effects of syntax exercises. For all the projects, the test group had a higher number of events, which included paste, cut, deletion, and typing. The projects where the test group had a significantly higher number of events were 5.1 (t=4994, p=0.0248), 7.0 (t=16762, p=0.0218), 8.0 (t=11813, p=$1.559e^{-9}$), and 8.1 (t=14038, p=$2.564e^{-7}$). Much like the effect of time on student scores, it has been found that an increase in keystroke events correlates to higher assignment scores [27], showing that this is possibly a positive outcome of the syntax exercises. However, similar to the amount of time the students took on their keystrokes, it is the hope that the students would decrease in the necessary number of keystrokes overall, since they would hopefully require less experimentation to find the correct syntax.

One hypothesis as to why students in the test group took longer on their assignments and had more keystrokes was because of pasting. Phanon was notoriously finicky (see section 4.3 Phanon Survey Responses). Because of this, some students would type their assignments in a different Integrated Development Environment (IDE), such as PyCharm, and then paste them into Phanon to submit them. It was therefore hypothesized that the test group, by the addition of the Phanon exercises, became more accustomed to the online environment, and therefore used it for development more often than students in the control group.

An analysis of the number of paste events on project 8 task 1 shows that this may possibly be a factor. Although the difference between the average length of pastes was significant (t=16302, p=0.001), the difference between the total length of the pastes was not significant (t=18834, p=0.198). The number of pastes was also significant for p¡0.05 (t=17654, p=0.03). This means that the control group was pasting more characters per paste and were pasting slightly less often. The test group pasted more often, but with fewer

characters per paste.

By determining "pasters" to be the students who had at least four times as many characters pasted as they did keystrokes, it was determined the control group did, in fact, have more pasters. The percentage of pasters can be seen in Table 4.1 and a graph can be seen in Fig. 4.2.

Project	Control Pasting %	Test Pasting %
4	22.63	14.17
5	18.98	11.55
6	17.34	9.826
7	16.92	11.79
8	22.26	11.11

Table 4.1: Percentage of students from the control and test groups that completed their assignments in a separate IDE and pasted into Phanon.

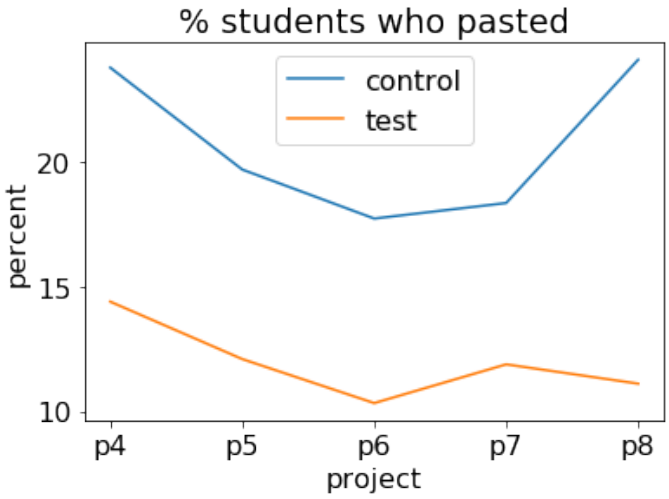


Fig. 4.2: Percentage of students who wrote their programs in a separate IDE and pasted into Phanon.

In addition to possibly explaining why students had more keystrokes and took longer, the reduction of pasting could indicate a reduction in plagiarism. Instead of copying and pasting code snippets and changing them to fit their assignments, this may show that students were more comfortable with the code syntax, and therefore typed pieces of code

themselves.

Another interesting comparison and possible reason for the increase in time taken by the test group was in the amount of comments each group wrote in their programs. For the eighth project, the test group had significantly more lines of comments (task 0: t=51540, p=0.0070; task 1: t=52190, p=0.0138) and characters in their comments (task 0: t=52970, p=0.0288; task 1: t=52850, p=0.0259) on both tasks.

The last hypothesis for why the test group did not complete their assignments faster or with fewer keystrokes is because of a decrease in attrition. Using project 4 as a baseline for the number of students in the class, the number of students who attempted project 8 in the control group dropped to 94.05%. The number of students who attempted project 8 in the test group only dropped to 97.2%. Using a proportions z-test, this was found to not be statistically significant (z=-1.74, p=0.0816), but it is interesting all the same. For a figure showing the percentage of students who attempted the assignments, see Fig. 4.3.

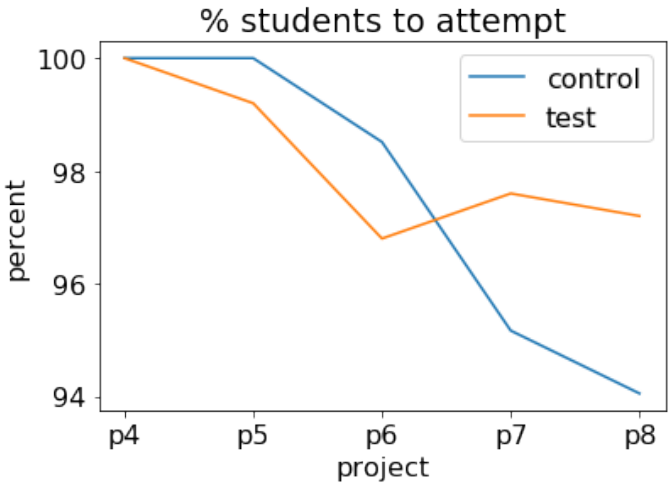
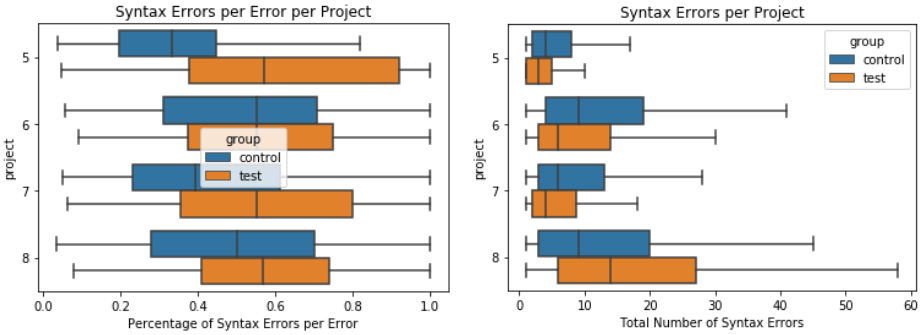


Fig. 4.3: Percentage of students who attempted each project.

Syntax errors were also analyzed. The test group ran their programs significantly more times per assignment than the control group (t=23864.5, p=0.0003), with means of 56.7 and 47.2 runs per assignment respectively, a positive outcome and an indication of higher likelihood of good grades on assignments [27]. Contrary to the predictions, the test group

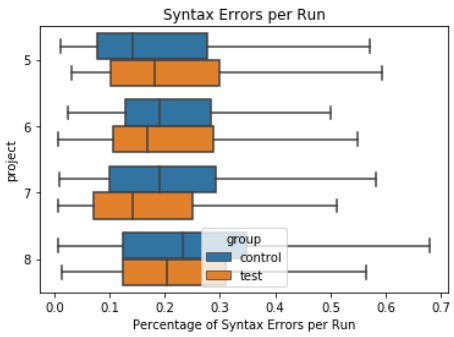
did not have a significantly lower percentage of syntax errors per run (t=28290, p=0.285) with a mean percentage of 21.5% and the control group with a mean percentage of 22.7%. In addition, the test group had a significantly higher percentage of syntax errors per error (t=17550, p=$1.891e^{-14}$). The control group had an average percentage of 45.3% and the test group had an average percentage of 56.4%.

Looking at the assignments individually, the test group had a significantly smaller percentage of syntax errors per run on project 7.0, and project 8, both tasks 0 and 1. They had a significantly higher percentage of syntax errors on project 5, task 1. However, the test group had a lower percentage of total errors per run (t=16674, p=$1.975e^{-16}$). The test group had an average of 37.8 and the control had 50.4%. For graphs, see Fig. 4.4.



(a) Number of syntax errors per error that the control and test groups had.

(b) Average number of syntax errors per project of the control and test groups.



(c) Percentage of syntax errors per run of the control and test groups.

Fig. 4.4: Student syntax errors.

One threat to validity that arises from the analysis of the students' errors is that the

errors were recorded differently for the control group than the test group. For the test group, the errors were recorded along with the keystrokes. While the control group was using the Phanon software to complete their assignments, the errors were not recorded. Instead, the research team had to go back and recreate the assignments at the time of the runs and run them to determine if they had errors. This may have caused some inconsistency and introduced errors into the analysis.

Next, the students' midterm exam scores were compared. The exam for both semesters contained the same questions, so the two groups' exam scores can be safely compared. The exam contained 50 questions and contained a mix of fill in the blank, multiple choice, and True/False questions.

The mean score of the control group was 71.9%, while the mean score of the test group was 77.7%. A t-test indicates that the test group performed significantly better than the control group (t=-6.0, p=$3.02e^{-9}$). A boxplot comparing the students' scores can be seen in Fig. 4.5.

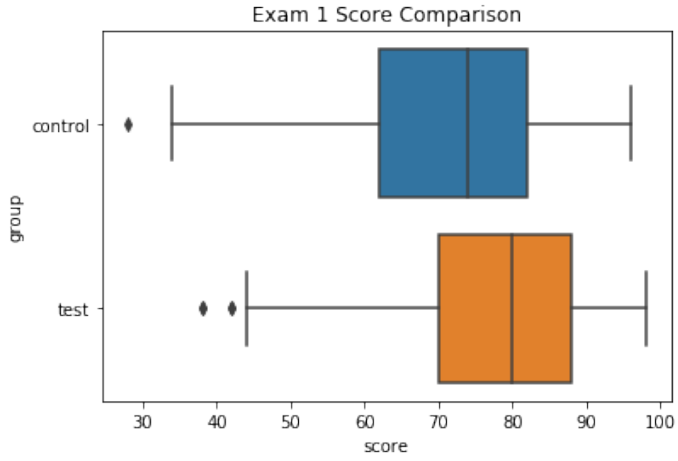


Fig. 4.5: Comparison of the control and test group Exam 1 scores.

In addition to comparing the students' overall score, the exam questions were divided into four categories: *Write One*, *Interpret One*, *Interpret More*, and *General Computer Science*. The single lines of code questions can be considered syntax questions, while the

Interpret More questions can be considered problem solving questions.

Write One refers to questions where students were asked to either write or fix a line of code or provide which character or function performs a certain function in the programming language the students learned, Python. For example, one of the *Write One* questions was: "To comment a single line, what character(s) are used?"

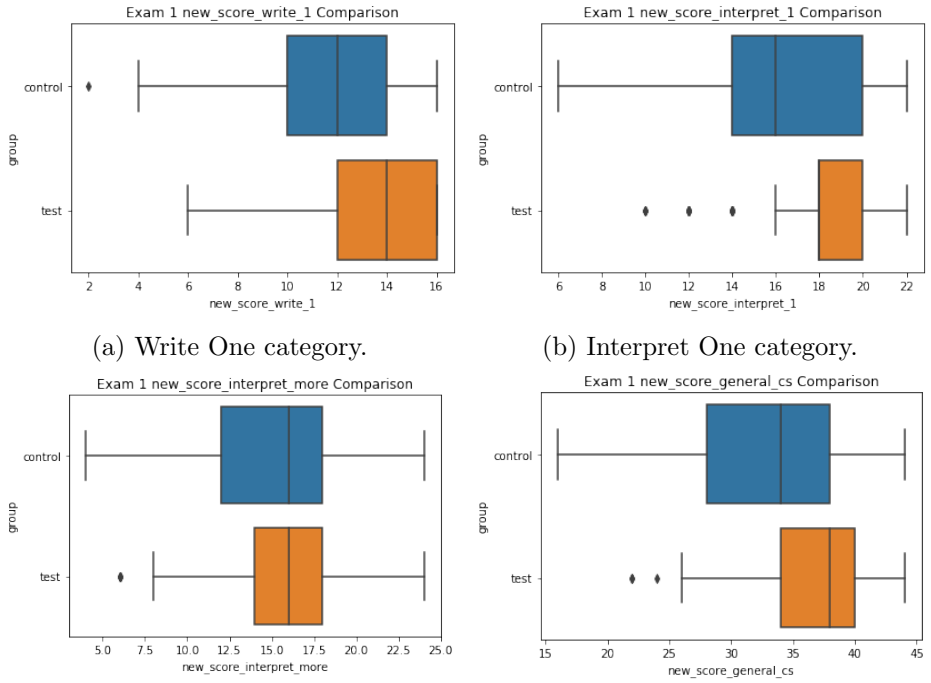
Interpret One and *Interpret More* refer to questions where students were asked if lines of code were correct, to state what results the code would have, or to identify certain types of programming constructs, such as string literals. *Interpret More* questions require students to interpret multiple lines of code in sequence. An example of an *Interpret One* question is "What will be the output/result of the following program? print(5+2)." One of the *Interpret More* questions is "What is the output of the following code? val1 = 3 val2 = 2 print("The sum is " + val1 + val2)."

Finally, *General Computer Science* questions are questions that do not have another label. These are questions that ask about languages, types of errors, conventions, software development processes, etc. A few examples are "According to standard naming conventions, which of the following would be a proper name for a constant value that designates the maximum value?" and "What type of error prevents a program from running at all?"

After placing all the questions into categories, there were 8 *Write One* questions, 11 *Interpret One* questions, 12 *Interpret More* questions, and 22 *General Computer Science* questions. The students' scores were recalculated for each of these categories and the control group was compared to the test group. The test group performed significantly better in all categories except for *Interpret More*. The t and p values and means can be seen in Table 4.2, and the boxplots can be seen in Fig. 4.6.

	T value	P value	Control Mean %	Test Mean %	Difference %
Write One	-7.61	$3.51e^{-13}$	70.69	86.19	15.50
Interpret One	-6.12	$2.82e^{-9}$	72.97	83.56	10.58
Interpret More	-1.20	0.230	63.68	65.90	2.217
General CS	-5.81	$1.55e^{-8}$	74.83	83.98	9.151

Table 4.2: Exam 1 scores for each category.



(a) Write One category.

(b) Interpret One category.

(c) Interpret More category.

(d) General Computer Science category.

Fig. 4.6: Box-and-whisker plots comparing the control and test groups for each exam 1 category.

It is interesting that the students improved on the *General CS* questions in addition to the *Write One* and *Interpret One* categories. It was expected that the syntax exercises would help the students improve on questions relating to syntax such as the *Write One* and *Interpret One* questions, but it is more difficult to see the relation between the exercises and the *General CS* questions. Although impossible to draw absolute conclusions from this study, some speculation is that the students could spend more time studying the non-syntax subjects in preparation for the exam because they already felt more confident to answer syntax questions. Another possible explanation is that the students were able to pay better attention during lectures because they could focus more on what the professor was saying rather than what he was typing.

4.2 Low-Performing Students

One of the predictions proposed was that the addition of syntax exercises would improve

the scores of low-performing students more than high-performing students. High-performing students will do well in their classes no matter what, but low-performing students may need more help to succeed in their classes. In addition, aptitude in mathematics has been shown to be linked to success in learning to program [28] [29], so it is predicted that students who have strong mathematical backgrounds will need less help to succeed in CS1 courses.

To ascertain the high- and low-performing students, the students' GPAs were recalculated to only include math and CS classes, excluding CS 1400. High-performing students were determined to be those whose math and CS GPAs had a mean higher than 3.5 (between a B+ or A- average) and a minimum math and CS GPA of 2.7, meaning that they had never received a grade lower than a B- in a math or CS class. Low-performing students were all other students.

4.2.1 Assignments

When comparing the low-performing students alone, no more improvement can be seen than by their high-performing counterparts. In fact, in some cases they even degraded more than the high-performing students. For example, when comparing the number of keystrokes, the low-performing students went from an average of 2160 keystrokes per assignment in the control group to an average of 2852 keystrokes in the test group, meaning an average of 7000 more keystrokes per assignment. On the other hand, the high-performing students went from an average of 2828 keystrokes in the control group to 3387 in the test group, an addition of only 5000 keystrokes.

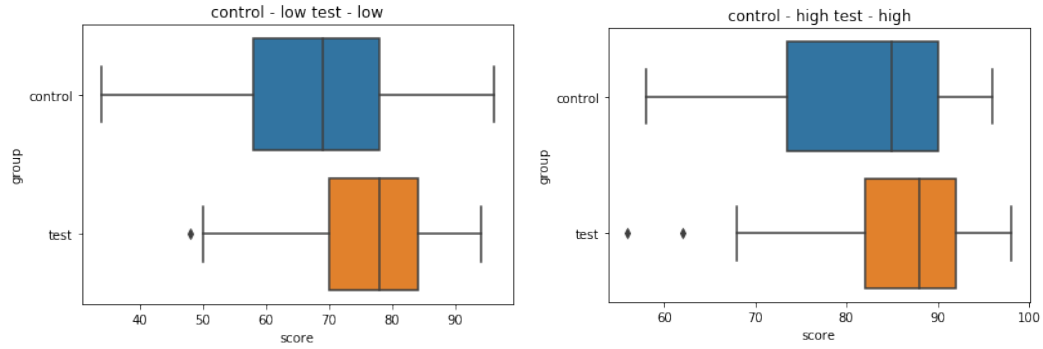
However, as stated in section 4.1, this could also have been caused by a decrease in attrition, or by a decrease in the number of students pasting their assignments. Also, the fact that the low-performing students increased in number of keystrokes is positive, as it shows that they were trying harder on their assignments [27].

4.2.2 Midterm Exam Scores

It was found that the low-performing students in the test group performed significantly better on the exam than the low-performing students in the control group (t=4.48,

p=1.38e^{-5}). The mean scores for the low-performing students in the control and test groups were 68% and 76.1%, respectively. Fig. 4.7a shows a boxplot comparing the students' performance.

The high-performing students in the test group also performed significantly better than the high-performing students in the control group (t=2.70, p=0.00794), but to a lesser degree, with mean scores 85.8% and 80.9%, respectively. A boxplot comparing the high-performing students' performance can be seen in Fig. 4.7b.



(a) Low-performing student scores.

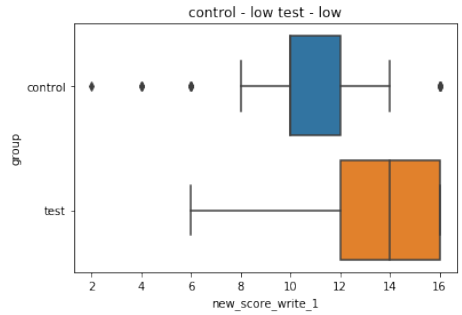
(b) High-performing student scores.

Fig. 4.7: Comparison of low- and high-performing student exam 1 scores.

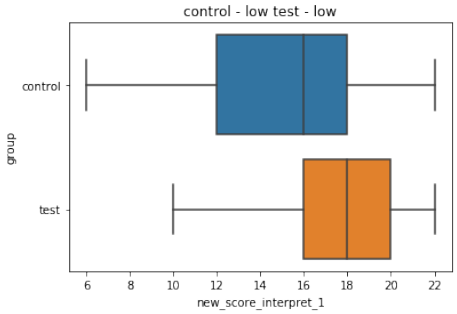
Like before, the students' scores on the questions in the categories *Write One*, *Interpret One*, *Interpret More*, and *General Computer Science* were compared. However, this time the scores were also compared on the bases of high- and low-performing students. The low-performing students in the test group performed significantly better than those in the control group for all categories except for *Interpret More*.

However, for the high-performing students, the test group only significantly out-performed the control group on the categories *Write One* and *General Computer Science*. For a complete list of means and t- and p-values, please see Table 4.3. Fig. 4.8 shows the boxplots comparing the students' exam 1 scores.

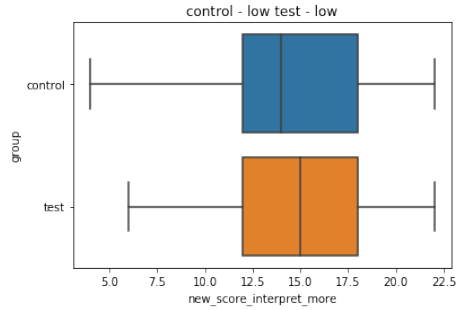
These results confirm our prediction that the addition of syntax exercises would help the low-performing students more than the high performing students, at least in terms of



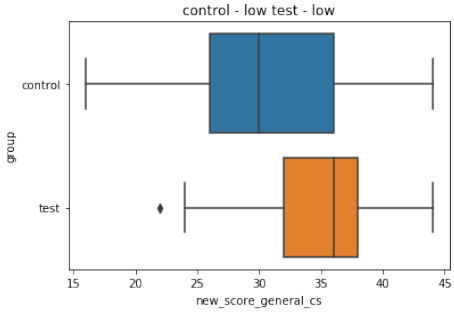
(a) Write One category, low-performing.



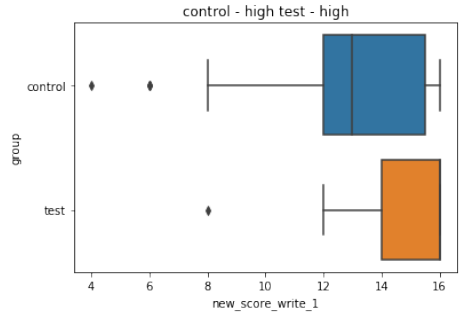
(b) Interpret One category, low-performing.



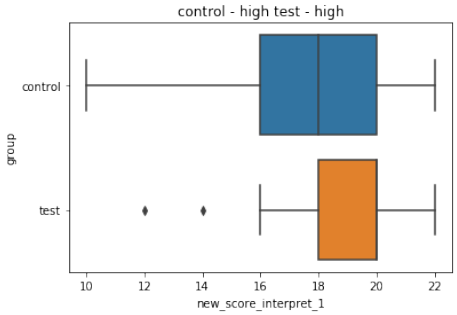
(c) Interpret More category, low-performing.



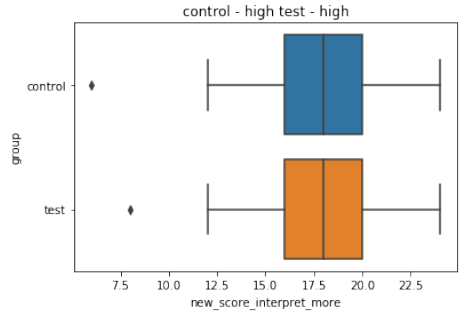
(d) General Computer Science category, low-performing.



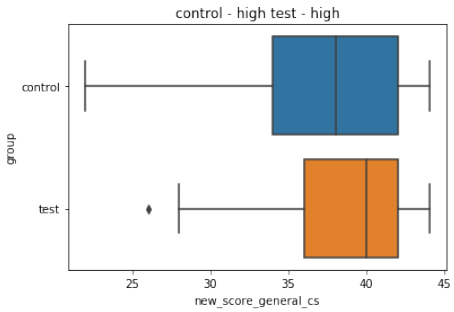
(e) Write One category, high-performing.



(f) Interpret One category, high-performing.



(g) Interpret More category, high-performing.



(h) General Computer Science category, high-performing.

Fig. 4.8: Plots for each category from exam 1 for both performance groups.

	Category	T value	P value	Control Mean %	Test Mean %	Difference %
Low	Write One	-4.78	$3.88e^{-6}$	67.39	81.09	13.70
	Interpret One	-4.15	$5.27e^{-5}$	69.47	79.95	10.49
	Interpret More	-0.482	0.630	60.05	61.22	1.16
	General CS	-4.92	$2.02e^{-6}$	69.61	79.90	10.28
High	Write One	-4.89	$3.49e^{-6}$	78.80	92.69	13.89
	Interpret One	-3.42	$8.76e^{-4}$	80.43	88.25	7.817
	Interpret More	-0.720	0.473	72.28	74.10	1.820
	General CS	-1.81	0.0729	84.58	88.46	3.877

Table 4.3: Exam 1 scores for each category and separated by performance.

exam scores. Overall, the low-performing students improved more than the high-performing students on the midterm exam. The low-performing students' mean score rose 8.1%, while the high-performing students' mean score only rose 4.9%. It is surprising to notice, however, that the high-performing students improved slightly more than their low-performing counterparts on the exam questions in the categories *Write One* and *Interpret More*.

4.3 Phanon Survey Responses

Next, an analysis of the survey responses regarding the Phanon exercises was conducted. In general, the attitude towards the Phanon exercises was largely positive, with 86.2% (119) mentioning that the exercises were helpful, and 84.1% (116) being generally positive, or saying that they liked the exercises. Meanwhile, 5.07% (7) found the exercises unhelpful and 5.07% (7) did not like the exercises. A graph showing the number of responses with each code can be found in Fig. 4.9 and the counts and percentages can be seen in Table 4.4.

The students were asked whether they felt the amount of time spent using the exercises was adequate, or whether they would have liked more or fewer exercises. Fewer exercises referred to either fewer exercises per session or fewer sessions. From those who responded to the survey, 45.7% (63) mentioned wanting more exercises. Some students even mentioned that they would have loved to have had the exercises for the more complex concepts, such as *classes*. Of the responses, 7.25% (10) asked for fewer exercises, and 13% (18) thought that they had used the exercises for just the right amount of time. Some of the responses

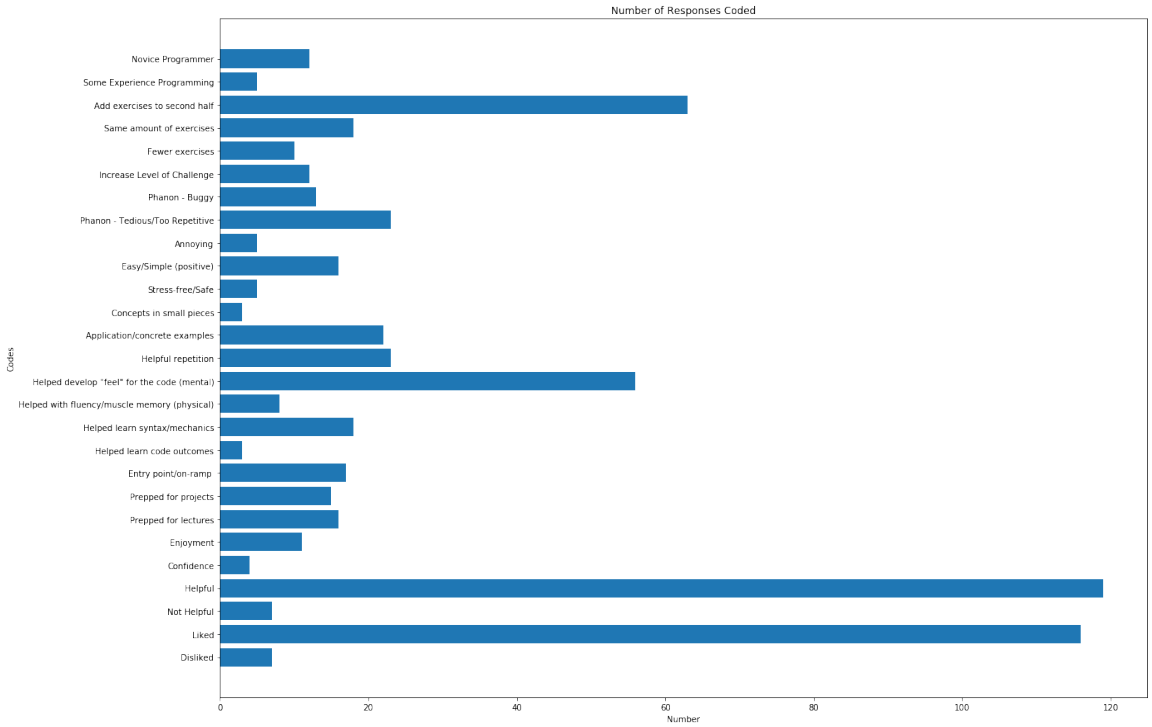


Fig. 4.9: Number of responses that contain each code.

overlapped, such as saying that they thought that the exercises had been given for the right amount of time for the semester, but that there could have been fewer exercises per session.

The responses were coded both for mentioning general helpfulness, as well as being helpful for developing a mental "feel" for the code, building muscle memory, learning syntax, learning code outcomes, and being a good entry point or good for beginners. As mentioned before, 86.2% mentioned that the exercises were helpful in general, but 58.7% (81) mentioned the exercises being helpful in one of the aforementioned ways.

In addition, 10.9% (15) stated that the exercises helped them complete their weekly homework assignments, and 11.6% (16) said that the exercises helped prepare them for lectures. One student even mentioned that the exercises "made the concepts discussed in class make more sense." Another student said 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

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 4.4: Counts and percentages of responses containing the shown codes.

come to class and expand upon that knowledge."

This is especially important because, as Jenkins points out, "there is surely little point in lecturing students on syntax when they have no idea of where and how to apply it" [1]. If students are able to first see the syntax and how it works in a program before getting taught the concepts, they will be able to focus more on the concepts and less on the confusing syntax being presented.

Although not an intended outcome of the exercises, it was interesting to note that three of the participants (2.17%) mentioned that the exercises helped them understand the

connection between what they typed and what the outcome of the code was. For example, one of the students mentioned that it helped "seeing the results of entered code."

Fifteen students mentioned that it had an impact on their affective state, specifically their enjoyment of coding (11, 7.97%) and their confidence in coding (4, 2.9%). The enjoyment of programming is significant, since so many students find programming "boring and difficult" [1]. In addition, confidence is important because it has been shown to have a positive impact on students' likelihood to continue in the major and take more challenging courses [30]. This is especially vital for those in minority groups, such as women [30].

However, not all the responses were positive. Of the responses, 5.07% (7) disliked the exercises and the same amount, 5.07%, found that they were not helpful. Also, 11.59% (16) found the exercises annoying, and 16.67% (23) thought they were too repetitive. However, of the eight students that disliked the exercises or thought they were not helpful, half also mentioned a way in which they were helpful.

It was common knowledge that the Phanon software had its bugs. This was represented by the fact that 9.42% of the responses (13) mentioned that Phanon was buggy or finicky. Finally, 8.7% (12) thought that the exercises were too easy or asked for more of a challenge.

One reason some students mentioned disliking the exercises was that they had prior programming experience. This applied to two of the students, which was 40% of those who did not like the exercises. It also comprised of 28.6% of the students who had had previous programming experience.

Overall, the responses were generally positive, and represented an understanding of the goal of the exercises. For example, 16.7% (23) mentioned that the exercises were tedious, but 13% (3) of those students also thought the repetition was beneficial. The response of one of those students included "they felt very repetitive, (which honestly helped get the concept down) but some more variation would be nice."

Another example is in the two students who mentioned they had previous programming experience and therefore it did not help them much. However, they felt that the exercises would be beneficial for beginners. One of those students said: "I could definitely see how

it would help newer programmers become less afraid of syntax so they can focus more on problem solving."

CHAPTER 5

CONCLUSION

This thesis discussed the results of a study that added syntax exercises to the curriculum of a CS1 class. The predictions made were that the syntax exercises would help the students perform better on their assignments and on their exams. Although this did not prove itself entirely true, the syntax exercises did have positive effects on the students.

The students in the test group took longer than the students in the control group to complete their assignments and used more keystrokes. A variety of factors may have played a role in this, such as the amount of copying and pasting done by the students. In addition, this surprising effect may have been caused by more students attempting the projects. This is paramount, since it shows the exercises increased the students' confidence.

An increase in confidence was also shown by the Phanon survey responses. Students were mainly appreciative of the exercises. A few mentioned that the exercises increased their confidence. One student even mentioned that he had taken the class previously and failed it, but that the Phanon exercises "really helped [him] out this time."

This study did have its limitations. One such limitation was that the students could copy and paste from another IDE, thereby skewing the keystroke and run data. Also, the total amount of time the students spent on their assignments could not be captured entirely, since there is no way of knowing if students wrote pseudo-code or pre-planning before beginning to code because self-reported metrics are unreliable [31]. Another limitation was with the Phanon survey responses. Because the students knew that the creator of Phanon would be viewing their responses, they may have responded with participant bias [32]. In addition, the students were given the survey at the end of the semester, at which point they had not used the Phanon program in a few months. As such, they might not have remembered their experience completely.

To help overcome these limitations, some students mentioned that it would have been

nice to have the Phanon exercises the entire semester, but to use PyCharm or another IDE to complete assignments, instead of the online Phanon program. If this were done and keystroke data were collected in the other IDE, it might mitigate some of the issues seen in this study.

Students learning how to program have a difficult time learning the syntax. Syntax errors are a major source of frustration, as any programmer, novice or professional, can attest to. Students need help in overcoming this challenge. If teachers can reduce the frustration felt by students and increase their confidence, more students will stay in Computer Science programs and continue on to fulfilling careers. This increase in confidence is especially important for those students in minority groups. Integrating syntax exercises into the curriculum is one way programming teachers can accomplish this.

REFERENCES

[1] T. Jenkins, "On the difficulty of learning to program," in *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, Aug. 2002, pp. 53–58.

[2] A. Stefik and S. Siebert, "An empirical investigation into programming language syntax," *ACM Trans. Comput. Educ.*, vol. 13, pp. 1–40, 2013.

[3] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *SIGCSE Bull.*, vol. 10, pp. 32–36, 2007.

[4] J. Bennedsen and M. Caspersen, "Failure rates in introductory programming: 12 years later," *ACM Inroads*, vol. 10, pp. 30–36, 2019.

[5] N. Hawi, "Casual attributions of success and failure made by undergraduate students in an introductory-level computer programming course," *Computers & Education*, vol. 54, pp. 1127–1136, 2009.

[6] T. Beaubouef and J. Mason, "Why the high attrition rate for computer science students: Some thoughts and observations," *SIGCSE Bull.*, vol. 37, pp. 103–106, 2005.

[7] A. Johnson, "Procedural memory and skill acquisition," in *Handbook of Psychology, Experimental Psychology.* John Wiley & Sons, Incorporated, 2012, pp. 495–520.

[8] A. M. Gaweda, C. F. Lynch, N. Seamon, G. Silva de Oliveira, and A. Deliwa, "Typing exercises as interactive worked examples for deliberate practice in cs courses," in *Proceedings of the Twenty-Second Australasian Computing Education Conference*, Feb. 2020, pp. 105—-113.

[9] A. Leinonen, H. Nygren, N. Pirttinen, A. Hellas, and J. Leinonen, "Exploring the applicability of simple syntax writing practice for learning programming," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, Feb. 2019, pp. 84—-90.

[10] N. Parlante. (2018) Codingbat. [Online]. Available: http://codingbat.com

[11] D. Thomas. (2018) Codekata. [Online]. Available: http://codekata.com

[12] P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "Understanding the syntax barrier for novices," in *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, Jun. 2011, p. 208–212.

[13] J. M. Edwards, E. K. Fulton, J. D. Holmes, J. L. Valentin, D. V. Beard, and K. R. Parker, "Separation of syntax and problem solving in introductory computer programming," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–5.

[14] L. Mannila, M. Peltomäki, and T. Salakoski, "What about a simple language? analyzing the difficulties in learning to program," *Computer Science Education*, vol. 16, no. 3, pp. 211–227, 2006. [Online]. Available: https://doi.org/10.1080/08993400600912384

[15] T. Daly, "Influence of alice 3: Reducing the hurdles to success in a cs1 programming course," Ph.D. dissertation, University of North Texas, Denton, TX, 2013.

[16] R. Lister, "Programming, syntax and cognitive load," *ACM Inroads*, vol. 2, no. 2, pp. 21–22, 2011.

[17] ——, "Programming, syntax and cognitive load (part 2)," *ACM Inroads*, vol. 2, pp. 21–22, 2011.

[18] C. M. Lewis, "How programming environment shapes perception, learning and goals: Logo vs. scratch," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, Mar. 2010, pp. 346—-350.

[19] D. Parsons and P. Haden, "Programming osmosis: Knowledge transfer from imperative to visual programming environments," in *20th Annual Conference of the National Advisory Committee on Computing Qualifications*, Jul. 2007, pp. 209–215.

[20] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, Oct. 2017. [Online]. Available: https://doi.org/10.1145/3089799

[21] D. Weintrop, "Modality matters: Understanding the effects of programming language representation in high school computer science classrooms," Ph.D. dissertation, 2016, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2019-10-19. [Online]. Available: https://login.dist.lib.usu.edu/login?url=https://search.proquest.com/docview/1826352865?accountid=14761

[22] V. L. Almstrum, O. Hazzan, M. Guzdial, and M. Petre, "Challenges to computer science education research," *ACM SIGCSE Bulletin*, vol. 37, no. 1, pp. 191–192, 2005.

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

[24] M. Banerjee, M. Capozzoli, L. McSweeney, and D. Sinha, "Beyond kappa: A review of interrater agreement measures," *Canadian journal of statistics*, vol. 27, no. 1, pp. 3–23, Mar. 1999.

[25] H. De Vries, M. N. Elliott, D. E. Kanouse, and S. S. Teleki, "Using pooled kappa to summarize interrater agreement across many items," *Field Methods*, vol. 20, no. 3, pp. 272–282, Aug. 2008.

[26] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, Mar. 1977. [Online]. Available: http://www.jstor.org/stable/2529310

[27] J. P. Munson, "Metrics for timely assessment of novice programmers," *J. Comput. Sci. Coll.*, vol. 32, no. 3, p. 136–148, Jan. 2017.

[28] P. Byrne and G. Lyons, "The effect of student attributes on success in programming," in *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 49–52. [Online]. Available: https://doi.org/10.1145/377435.377467

[29] N. Rountree, J. Rountree, A. Robins, and R. Hannah, "Interacting factors that predict success and failure in a cs1 course," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 101–104. [Online]. Available: https://doi.org/10.1145/1044550.1041669

[30] L. Irani, "Understanding gender and confidence in cs course culture," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 195–199. [Online]. Available: https://doi.org/10.1145/971300.971371

[31] J. Leinonen, L. Leppänen, P. Ihantola, and A. Hellas, "Comparison of time metrics in programming," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ser. ICER '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 200–208. [Online]. Available: https://doi.org/10.1145/3105726.3106181

[32] N. Dell, V. Vaidyanathan, I. Medhi, E. Cutrell, and W. Thies, ""yours is better!": Participant response bias in hci," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1321–1330. [Online]. Available: https://doi.org/10.1145/2207676.2208589

APPENDICES

APPENDIX A

Phanon Survey Response Codebook

Table A.1 is the compiled codebook used to code the Phanon survey responses.

Table A.1: Code names, numbers, hierarchies, descriptions, and examples from the students' responses

		Code #	Code Name	Description	Examples
Myself as a Learner		I.A.	Novice Programmer	Responses include a mention of being new to programming, or this being their first programming class.	"greatly help everyone, especially those new to programming such as myself. (496)", "This class was my first introduction to programming. (786)"
		I.B.	Some Experience Programming	Responses include a mention of not being new to programming, or having some programming experience.	"but thats because I have already programmed in the past. (812)", "I have a pretty good amount of experience programming (891)"
Requests for change		II.A	Add exercises to second half	Responses include a request to add the exercises to the second half of the semester, or to include exercises for the more complex topics.	"I think those [Phanon exercises] should be used the entire semester. (917)", "I would suggest to keep it for the entire semester (1154)"
		II.B	Same amount of exercises	Responses mention that they liked the point when the class stopped using Phanon, or that they used it for just the right amount of time.	"I think the amount of time we spent using it was perfect. (193)", " I thought there was the perfect amount of the semester that was focused on phanon (816)"

		II.C	Fewer exercises	Responses include a request to have fewer exercises per session or reduce the number of sessions.	"I think they could be a little shorter and more to the point indiviudally (1031)", "I think that if it was used less it would be a lot better(277)"
		II.D	Increase Level of Challenge	Responses include a request to increase the exercises' challenge or states that the exercises were too easy.	"I saw Phanon more as a tutorial and I wanted a more exciting challenge (726)", "Most of the logic in the Phanon excercises were very basic and too easy to be productive (846)"
Dislikes	Characteristics of Phanon	III.A.1.	Phanon - Buggy	Responses mention that the Phanon program is buggy, glitchy, or complain that small, meaningless differences would cause the program to count their answers as wrong.	"it is also glitchy (1146)", "While sometimes phanon was a little frustrating with the way it checked to see if the answers were correct... (738)"
		III.A.2.	Phanon - Tedious/Too Repetitive	Responses specifically mention being too tedious or too long. Responses specifically mention being too repetitive, or repetitive in a bad way.	"It was likely extra tediousbecause I already had a good muscle memory for what it was asking (761)", "Some of them could maybe be a little bit shorter. (891)", "Make things less repetetive. (938)", "I felt as though phanon was very repetitive and didnt instil much learning (1008)"

	Affective Experience	III.B.1.	Annoying	Responses specifically state that they found the Phanon exercises annoying or frustrating.	"the Phannon exercises were a little bit annoying (891)", "I think some of the assignments were kind of annoying (1070)"
Likes	Characteristics of Phanon	IV.A.1.	Easy/Simple (positive)	Responses mention that they enjoyed the fact that the Phanon exercises were easy.	"they were also super super easy so it wasn't a problem (891)", "I appreciated that the phanon exercisizes were easy and repetitive so that I could get use to the syntax (940)"
		IV.A.2.	Stress-free/Safe	Responses mention that the Phanon exercises allowed room for experimentation, that they felt like a safe space (including things such as not being punished for mistakes), or that it was a nice, stress-free environment to learn how to code.	"I wasn't docked points if I didn't get it right away (1140)", " The exercises from Phanon really helped me to understand coding without stressing about it, (1051)."
		IV.A.3.	Concepts in small pieces	Responses mention that the Phanon exercises broke concepts down into smaller pieces or taught in a step-by-step manner.	"it was really helpful for me to see the concepts step by step. (180)", "It broke the concepts down into simpler terms so they were easier to grasp. (847)"

	IV.A.4.	Application/concrete examples	Responses mention that the Phanon exercises allowed them to apply what they were learning or gave them concrete examples of how to use programming concepts. Responses mention that the Phanon exercises were good practice.	"Those helped apply what we learned in class. (769).", "Having the many different examples of how to make the code work and where errors typically are was tedious, but really helpful. (806)", "I also think that it helped to use the things we were learning about in class in a real situation instead of seeing it for the first time in our homework. (810)", "I felt like I could actually grasp the concepts and get a little bit of practice (890)", "it was a good way to practice through exersizes (903)"
	IV.A.5.	Helpful repetition	Responses mention that the Phanon exercises featured helpful repetition to help them learn.	"I liked phanon and it helped me learn the basics through repetition. (815)", "I felt that the repetition of the exercises helped programming to feel more natural and also helped my typing skills too. (866)"

	Outcomes of Using Phanon	IV.B.1.	Helped develop "feel" for the code (mental)	Responses mention that the Phanon exercises helped them get used to the code in a mental way, such as getting used to the way the code looks or helping them get a "feel" for the code. Responses mention that the exercises helped them to remember pieces of code. Responses mention that the exercises helped them learn the basics/concepts of programming. Responses mention that the exercises helped their understanding of programming and programming concepts. Responses mention that the exercises helped them read and/or understand code. Responses mention that the exercises helped cement or solidify the concepts of programming or what they were learning.	"get used to the way they were supposed to look. (1000)", "Phanon was very beneficial to help me remember (848)", "Sometimes it's hard to remember what all of the functions are. (971)", "The exercises from Phanon really helped me to understand coding without stressing about it, (1051)", "they did help me to quickly learn the basics of the python language. (851)", "I thought that phanon was very helpful in learning simple concepts about code (856)", "Phanon helped me be able to understand code more easily (876)", "They helped me to practice the syntax and become familiar with reading code (971)", "I thought it really helped to cement the core principles (1084)", "I felt that they helped to cement common practices of CS. (742)"

IV.B.2.	Helped with fluency/muscle memory (physical)	Responses mention that the Phanon exercises helped them in a physical way, such as with their muscle memory or with their typing of the syntax.	"I felt that the repetition of the exercises helped programming to feel more natural and also helped my typing skills too. (866)", "I liked Phanon because it gave you a little bit of muscle memory on syntax (1154)"
IV.B.3.	Helped learn syntax/mechanics	Responses mention that the exercises helped them learn syntax or the mechanics of programming.	"It helpd me get a feel of the syntax (957)", "I thought it was useful in learning proper syntax. (968)"
IV.B.4.	Helped learn code outcomes	Responses mention that Phanon helped them understand the relationship of what outcomes certain pieces of code have.	"I thought that phanon was very helpful in ... seeing the results of entered code. (856)", "It helped me understand what the code was doing (768)"
IV.B.5.	Entry point/on-ramp	Responses mention that the Phanon exercises helped "ease them in" to programming or served as an on-ramp or entry point into programming, or that the exercises were good for beginners (whether that applied to them or to others).	"I think that it helped a lot with the beginning of the learning curve to computer science (825)", "the Phanon exercises were a good way to ease students into coding.(842)"
IV.B.6.a.	Prepped for projects	Reponses mention that the Phanon exercises helped them when they were working on their homework projects.	"Phanon helped me with the assignments for the beginning of the semester. (881)", "which in turn helped me with the big projects.(744)"

		IV.B.6.b	Prepped for lectures	Responses mention that doing the Phanon exercises helped them prepare for lecture/class.	"It was also very preparatory, teaching simple operations in a way where I was more prepared for the next class and assignment. (791)", "made the concepts discussed in class make more sense. (908)"
	Affective Experience	IV.C.1	Enjoyment	Responses mention that the exercises increased their enjoyment of programming or the class. Or, they mention that it increased their excitement to program.	"That being said, I did enjoy the exercises for the first couple of weeks. It got me excited to code. (865)", "I really enjoyed phanon! I thought it was very helpful and even fun. I found myself looking forward to the next exercise. (1145)"
		IV.C.2	Confidence	Responses mention that the exercises increased their confidence in programming or in the class.	"When we stopped using phanon, I noticed a signifigant shift in my confidence to accomplish the weekly homework assignments.(1032)", "I felt confident and understanding of the material until phanon stopped being used. (823)"
Overall		V.A.	Helpful	Responses mention that the exercises were helpful in general (the word "help" is specifically included).	"i thought it helped with getting concepts to stick. (1048)", "I feel like Phanon exercises were helpful (1057)"

		V.B.	Not Helpful	Responses mention that the exercises were not helpful in general.	"I think that Phanon didn't help me learn. (846)", "I didn't learn much from it becsuse I already knew some python. (759)", "I felt as though phanon was very repetitive and didnt instil much learning. (1008)", " but I don't think it made a huge differencein my understanding of the theory behind the material (323)"
		V.C.	Liked	Responses mention that they liked the exercises in general. Or, overall the response is positive.	"I liked the phanon exercises. (756)", "I really liked Phanon (768)"
		V.D.	Disliked	Responses mention that they didn't like the exercises in general, or complain about the exercises.	" Also, I felt the Phanon were exercises redundant and a waste of time after a little. (715)", "I have a pretty good amount of experience programming so the Phannon exercises were a little bit annoying (891)"