5-2021

An In-Depth Look at Learning Computer Language Syntax in a High-Repetition Practice Environment

Stephanie Gonzales
Utah State University

AN IN-DEPTH LOOK AT LEARNING COMPUTER LANGUAGE SYNTAX IN A HIGH-REPETITION PRACTICE ENVIRONMENT

by

Stephanie Gonzales

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	Hillary Swanson, Ph.D. Committee Member
Vicki Allan, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

An in-depth look at learning computer language syntax in a high-repetition practice environment

by

Stephanie Gonzales, Master of Science

Utah State University, 2020

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

Computer science education experiments and research has determined repetition to be critical in early learning for computer programming [1]. An instructional tool called Phanon has been adopted into the curricula for introductory programming CS1 courses at multiple universities and is designed to help students automate their code production. As students overcome the difficulties of mastering syntax, their cognitive resources can be utilized for more demanding thought processes like problem-solving. Observing students while they complete syntax exercises may provide meaningful insight into their process while engaged in the procedural practice.

At Utah State University, a qualitative think-aloud study was conducted to explore the presence of basic patterns that provide insight into students' learning process when completing syntax exercises. Students in a CS1 course participated in three think-aloud sessions with the researcher while completing syntax exercises through Phanon. The topics of observed lessons were conditionals, for-loops, and nested for-loops. The sessions were facilitated and recorded through Zoom over one week. The recorded sessions were transcribed

and coded by the researcher into actions and events. This thesis comprises the design and setup of the study, data collection, and data analysis.

Throughout the study's analysis, patterns of repetition and discovery were observed and interpreted. Intrinsic and extraneous load were evaluated through a temporal decomposition approach, and evidence of intrinsic load was reflected throughout several participants. The amount of effort a student extends to the syntax exercises appears not to reflect their course performance. Several students demonstrated a significant improvement in writing syntactically correct code as they progressed through the lesson. They showed that learning the syntax can get intricate but allows the student to overcome the syntactic hump that comes with learning to program. Students generally had a positive attitude toward the exercises supporting the claim that Phanon provides a closed, stress-free environment conducive for mastery of syntax.

(83 pages)

PUBLIC ABSTRACT

An in-depth look at learning computer language syntax in a high-repetition practice environment

Stephanie Gonzales

Students in an introductory computer science course generally have difficulty producing code that follows the arrangement rules known as syntax. Phanon was created to help students practice writing correct code that follows the rules of syntax. Previous research suggests this tool has helped students improve their exam scores and strengthen effectiveness in the course. A study was conducted to observe students while they complete the syntax exercises to find meaningful patterns in the steps the students take to complete an exercise.

Evidence to support high intrinsic load was found throughout the study, which is a measure of difficulty learning a subject. The syntax exercise design's ineffectiveness, known as the extraneous cognitive load, was minimal throughout the study. It was also found that even if students seem to take longer completing the syntax exercises, it does not reflect a decrease in their performance for the class. This supports a theory that syntax is a separate process from problem-solving and mastering it can help students focus their cognitive process on problem-solving.

Finding ordinary moments of comprehension or struggle can provide insight into how improvements can be made in Phanon and computer science teaching methods. The effectiveness of Phanon can be applied to students with a variety of programming experience.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer science education research focuses on giving the student more opportunities to succeed in a computer science major. The high amount of attrition and the low contentment for novice programmers is a concern for most collegiate computer science departments [2,3]. Although computer science may be considered a complicated subject, most researchers agree that subject difficulty is not why failure rates are high. It is the way that the lecturer teaches the subject to novice programmers that causes frustration and confusion in the students. Luxton-Rielly argues that the inability to teach students is why the field lacks diversity and has an increased risk of plagiarism [4].

Syntax in computer science is a topic that causes new programmers a lot of frustration. The intrinsic cognitive load, the effort associated with the subject, is high in syntax. As a result, students spend a lot of time dealing with syntax problems rather than developing their problem-solving skills. In some introductory computer science courses, the lecturers ignore this issue and expect students to learn syntax while problem-solving. This method causes the extraneous cognitive load, the effort it takes to understand the material, and how the lecturer presented it.

A tool required for introductory computer programming course CS1 at Utah State University and other universities is called Phanon. The inspiration for Phanon came from a popular piano exercise book "The virtuoso Pianist" by Charles-Louis Hanon. Hanon designed the training book to train aspiring pianists to handle everyday difficulties with technical skills, increasing their speed, precision, and agility. The name Phanon is representative of the inspiration from Hanon and translates to "Programming Hanon" [5]. Like Hanon's exercise

book, Phanon provides exercises designed to increase the beginning programmer's ability to write precise, syntax error-free code.

Phanon provides sets of syntax exercises for essential introductory coding topics. In the lessons, the exercises increase slightly in difficulty as the student completes them. Phanon's creator designed the questions to be quick and straightforward so that the participant can complete many questions quickly, engaging procedural memory more than cognition. This strategy intends to help the participant automate their production of code with repetition and practice. Lessons in Phanon typically contain around thirty questions and are designed to take less than fifteen minutes to complete. The exercises are designed to teach and reinforce skills in writing syntactically correct code. The design complements other pedagogy that teaches structured problem-solving.

Through several studies, the tool has shown promising results in the decrease of attrition and the students overall affective state [5]. There have also been positive correlations made with an increase in project scores and a decrease in plagiarism [1,6]. Phanon has proven to be a valuable tool that helps students learn to produce code through short repetitive exercises. The developers created syntax exercises as a way to help students become familiar with writing code. As the student completes the code snippets from the exercises, they become better prepared for their studies. This preparation allows them to approach their weekly programming projects with a problem-solving focus rather than a syntactical focus.

This thesis aims to gain insight into the cognitive processes that a student engages in when completing Phanon syntax exercises. Data was collected by the researcher and analyzed from a think-aloud study conducted over the 2020 fall semester. Eleven students from a CS1 course at Utah State University completed three think-aloud sessions with the researcher. In each session, the researcher instructed the participants to think out loud as they used Phanon to complete a lesson selected by the researcher. The sessions were recorded through Zoom and coded by the researcher according to the actions of the participant.

Through a qualitative analysis of the study in this thesis, the researcher addresses the following questions.

- What patterns in student interactions with syntax exercises are observable?
- How do these behaviors affect student performance in learning computer syntax?

Think-aloud studies have been used in many technical subjects because they allow the researcher to witness the participant's moment of comprehension when they solve a problem [7]. Finding patterns or ordinary moments that lead up to understanding can improve computer science education methods. Researchers can analyze topics or circumstances that generally confuse students for similarities in disconnect. Observing the students as they face challenges that Phanon provides has provided insight into thought and discovery. It has also shown a light on what makes certain coding constructs difficult for some students.

CHAPTER 2

RELATED WORK

High attrition rates in computer science majors is a constant motivating factor in research regarding learning strategies. Computer science education researchers have developed several tools to help students grow their skills through programming, typing, and syntax exercises. Computer science education research has suggested that one way to prepare students for quality problem solving is by automating their syntactically-correct code production. Phanon is a syntax exercise tool used in several studies that yielded an improved overall affective state and decreased attrition [5, 6, 8].

2.1 Theories of Practice and Representations

2.1.1 Repetition and Discovery

Trninic et al. suggest that repetition and practice are essential for learning through explorative practice. The authors discuss that engaging in repetitive practice requires a consciousness of kinesthetic sensations [9]. They also mention that even though direct instruction and discovery are thought to be conflicting approaches, but can be interwoven techniques. The authors explain that procedural moves promote a substantial amount of insight and perception.

2.1.2 Mental Representations of Computer Programs

Pennington et al. discuss mental representations of a computer program's structure and the relationships involving that structure. The authors break down the components that make up a program and relate it to the necessary knowledge for program comprehension. One of the subjects that they say is vital for a developer to have is text structure knowledge.

Text structure knowledge means that the individual understands the different structures for control and flows in a program. These will include the ideas of sequence, iteration, and conditionals [10]. An individual's understanding of text structure is essential because it provides a mental representation and organization to some concepts.

The second type of knowledge that Pennington et al. describe as crucial is plan knowledge. Structures like methods and functions can represent this type of knowledge. Fundamentally it is the understanding of how components of a program fit together to accomplish a goal. The plans that the authors refer to can be higher-level plans and lower level plans. Higher-level plans usually consist of many functions (lower-level plans) that accomplish one goal. An example of this may include sorting algorithms or a file reader that parses the document.

2.1.3 Debugging Study

Katz et al. [11] conducted a study of the debugging process and methods used when students are problem-solving. Students with a variety of programming experiences were used in the study to provide quality results. Some students had little knowledge of coding, and others had taken a Pascal class previously to the study. The researchers completed several experiments that involved students in debugging LISP programs. Katz et al. describes the very general process and stages of troubleshooting, or in this case, debugging, and mentions that researchers do not fully understand how those stages interact with each other.

The authors make some interesting conclusions about how students interact with the coding problems they face. One of the conclusions that they came to from the first experiment was that different coding experience levels correlate with the student's approach to solving the problem. They found that after a student gives a correct response, it is improbable to follow with an error. Another conclusion is that the students usually approached debugging another person's code differently than they would for their own. They seemed to have more successful sessions when the students debugged their code rather than peers.

The authors also observed from the third experiment that students would decide on the code line that they believe the error was located. If they were incorrect, they were likely to get stuck and not solve the problem [11]. Katz et al. conclude that their observations and ideas about debugging are relatable to other problem-solving situations.

2.1.4 Think-Aloud Studies: Metacognitive Awareness

Prather et al. [12] highlights the process a student goes through when learning to program and explains a think-aloud study that allowed them to evaluate moments of difficulty when a student is engaging in debugging behaviors. In the article, the authors mention the idea of metacognitive awareness. Metacognitive awareness can describe the state of a student who understands the problem-solving process and can articulate the steps they plan to take while solving a problem.

In the study, the researchers used an automated assessment tool to facilitate the sessions. They explain the learning benefits and disadvantages of using an automated assessment tool for a think-aloud study. The authors claim that automated assessment tools need to have enhanced compiler error messages in order for the tools to improve a student's metacognitive awareness. They also state that automated assessment tools should guide the student through the stages of metacognitive awareness. Prather et al. say a tool can help a student by providing helpful error messages and gradually increasing difficulty.

Prather et al. [13] conducted another study about the metacognitive difficulties students face when using automatic assessment tools. The researchers focused on how understanding how a problem can improve a student's probability of success. This paper's authors collected code submissions from students in a think-aloud study from a control group and an experimental group. The students in the experimental group had additional metacognitive help through a quiz before starting the given problem. The experimental group had less time to solve the given problem and performed better than those in the control group.

The authors also saw an increase in the number of students who completed the problem in the experimental group. The researchers described the students as having a higher metacognitive awareness when discussing their problem with the researcher after they completed the correct code. Even though the experimental group generally performed better, there were still some students in that group that struggled to solve the problem.

2.2 Syntax Studies

Several studies have looked into different types of syntax practice pedagogies and their effect on learning outcomes.

2.2.1 TYPOS Syntax Exercise Study

Gaweda et al. discuss a study that focuses on the importance of syntax exercises and their relation to the growth and development of a student's lower-level coding skills. The authors claim that there is a difference between higher level and lower level skill development and that it is an easy task to improve lower-level skills. They state that by implementing syntax typing exercises, a student can develop lower-level skills through practice. By mastering lower-level skills, they will be able to put more cognitive effort into higher-level skills growth [14].

In the study that the researchers conducted, the participants used a TYPOS tool for their syntax exercises. The program provided around five exercises a week that correlated with the class learning material for that week. The creators of TYPOS prevented copying and pasting by providing the code example as an image. The student was required to retype the code as shown. Once the student submitted their solution, the database would compare their typed solution with a solution stored in the database and highlight any typos or errors.

The study yielded fascinating results in the student's graded projects. The researchers found that students who used TYPOS consistently made fewer mistakes on their graded

assignments than those who did not use TYPOS. GitHub integration workflows analyzed the student's submitted code. When the students pushed new commits, a workflow in the repository would analyze for compilation errors.

2.2.2 Embedded Syntax Tool Study

Leinonen et al. theorize that syntax exercises play an essential role in students' journey of learning to program. The authors suggest that syntax is not an easy subject for many students. Languages in programming can differ in syntax rules, and novice students often have trouble discerning compilation errors associated with syntax [15]. In the study that the researchers conducted, they developed a tool embedded in the online learning material for the study. The embedded tool helped students with syntax problems in their code by highlighting issues as they arise through coding. For example, a misplaced semicolon can be extremely hard to find. However, the syntax exercise tool would highlight the semicolon with red, signaling that its placement is incorrect.

Half of the students that participated in the study used the tool for the first two weeks with in-class projects. The second half of the students that participated did not use the tool for their projects. Leinonen et al. explain that there were flaws in the research that may have prevented them from getting valuable data. One aspect that could skew the data was that not all of the students who had access to the tool used it. The conclusion to this study was that the authors do not have sufficient evidence to prove that the syntax tool helped them reduce their time on the problem or real events.

2.2.3 Phanon Study

Edwards et al. discuss a study that focuses on the benefits of syntax exercises through a tool called Phanon. The authors present a syntax first oriented approach for teaching proper coding techniques to novice programmers. One significant benefit of Phanon that

the authors discuss is the ease of integration into the CS1 course in which they conducted the study [5].

The researchers conducted the study over two semesters in CS1 courses and separated the participants into a test and a control group. Participants in the test group were introduced to syntax exercises three weeks into the course. The test group participants completed syntax exercises three times per week that were correlated with the course material. The authors describe many positive results, including decreased attrition, improved exam scores, and a decrease in plagiarism.

A follow-up study analyzed the effect of syntax exercises on the affective state. Sullivan et al. [8] discuss the results from a survey received from students about the syntax exercises in Phanon. The majority of responses were positive and indicated that they felt like Phanon helped them in their studies. In the survey results, 86% of students said that Phanon was helpful, 84% said they liked the exercises, and 8% found enjoyment throughout the exercises. There were some negative responses concerning annoyance or not liking the exercises. 5% of the students in Sullivan's survey found the exercises to be annoying, and 5% of the students said they did not like the exercises.

CHAPTER 3

METHODOLOGY

This study aimed to find reoccurring themes of exploratory practice while students complete Phanon syntax exercises. This study's analysis includes qualitative data gathered from video recordings, audio recordings, and participants' surveys. The researcher analyzed the results with a grounded approach [16].

3.1 Overview

The researcher conducted study IRB #11325 in the fall 2020 semester with an introductory computer science course. In the course, students were taught the fundamentals of programming in Python. Along with weekly programming assignments, the students had course requirements to complete syntax exercises through a Phanon. The researcher invited students to participate in three think-aloud sessions while they completed select syntax exercises. The students that completed the three selected think-aloud sessions received a gift card for their participation. Thirteen students consented to participate in the study and completed the first syntax lesson with the researcher. Two of the students opted to discontinue the study and did not complete the second and third think-aloud sessions. Eleven students in total completed all three think-aloud sessions with the researcher.

The researcher conducted the think-aloud sessions through Zoom, an online videotelephony software program. In the sessions, the researcher instructed the participants to share their screen and declare their thoughts as they completed the chosen set of exercises. If the participant had long moments of silence, the researcher asked them to reveal their thoughts. Zoom automatically recorded each of the sessions using Zoom recording features for audio and video. The data collected from the study includes video of the screen share, video

of the participant, audio, transcripts, and survey data about the student's programming experience and academic information before the introductory course.

3.2 Phanon Syntax Exercises

The environment that Phanon emulates an IDE (Integrated Development Environment) by providing comprehensive components for testing excerpts of code. Phanon groups lessons by coding topics such as string manipulation, Boolean expressions, conditionals, and loops. Phanon breaks down some of the more complex topics like for-loops into several lessons for simplicity and scope. Figure 3.1 shows Phanon's (GUI) Graphical User Interface while a student is completing an exercise from a lesson on for-loops. Figure 3.2 shows Phanon's GUI after a student has completed exercise eleven. Phanon's GUI has several essential elements. The first element lists the exercise numbers for the current lesson. Completed exercises are marked green with a check-mark icon, the current exercise is marked blue with a bookmark icon, and future exercises are locked and marked with a white lock icon. Users can reference previous exercises by clicking on the number, but they cannot look ahead at future exercises.

The exercise instructions element is the white top center box in the GUI. The exercise instructions give the user precise information about what to modify or create in the code editor element, which is the box directly below the exercise instructions element. The code editor looks and behaves similarly to an IDE by showing numbered lines and a code completion aid built into the environment. If a student declares a syntactically correct Python for-loop and then presses enter to move to the next line, the editor will automatically add the proper indentation, similar to other popular IDEs. Directly above the code editor are buttons that run the code, move to the next question, revert the exercise, and reveal the answer.

Phanon marks the button to run the code as a play icon, similar to a start button on popular IDEs. After the user pushes the button to run the code, the output and test results

elements will respond according to the code editor's input's correctness. Phanon assesses their ability to write correct code and evaluates the correctness as taught by the instructor. It adheres to the instructor's coding standards. If a user enters syntactically incorrect code into the code editor and then runs the code using the play button, the output element displays a Python error message. Suppose a user enters syntactically correct code into the code editor that is not the correct solution. In that case, the output box will display the program's output, and the test results element will display an error message about hidden tests that have failed. Suppose a user enters syntactically correct code that is the correct solution. In that case, the test results element displays that the tests have passed, the output element displays the output of the code, and Phanon displays "good job" in the top right corner of the instructions element.

Phanon looks and behaves similarly to an IDE, but several features make it more helpful to the novice coder. A student who is actively learning while using the program has more options available to them when they get stuck. One feature that Phanon implements is the ability to revert the exercise. If a user tests their code several times, it may be challenging to remember the code's original format before modifications. In that situation, the user may push the mulligan button to revert the code to the original given format. To keep students from abusing the mulligan button, Phanon only allows a user to use that button every thirty minutes. The creator designed Phanon exercises to be quick, but the user may occasionally take longer than expected to complete the exercise. Phanon contains a button that reveals the user's answer to keep the exercises from becoming unnecessarily burdensome and stressful.

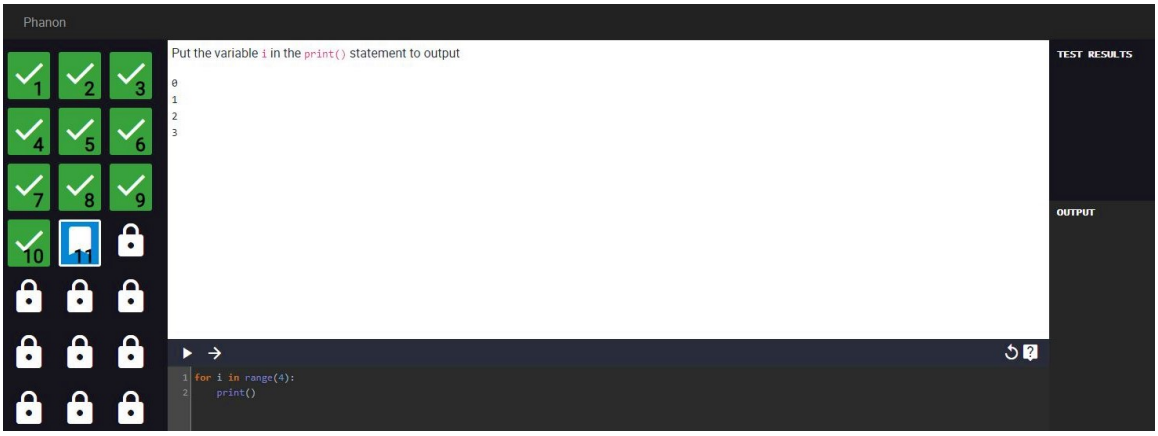

Fig. 3.1: Phanon Graphical User Interface

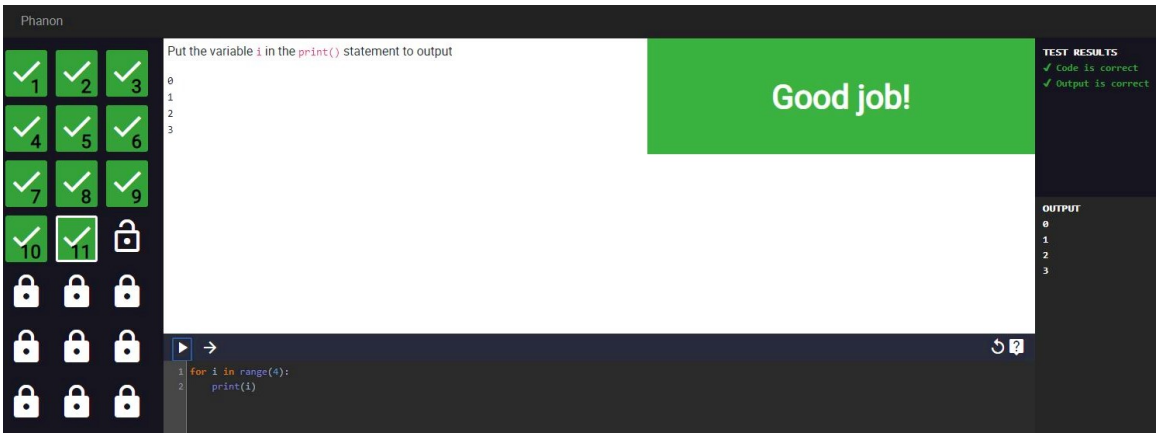

Fig. 3.2: Phanon Graphical User Interface after Successful Response to an Exercise

Phanon contains three different types of exercises within each lesson. The first type of exercise a user will encounter is a modification exercise. In a modification exercise, Phanon instructs the user to change the code that Phanon presents in the code editor. The modification may be concerning a variable or the body of the code. Another type of exercise is a debugging exercise. In the debugging problems, Phanon will instruct the user to fix a bug that is strategically located in the code editor's code. A bug can be a variety of errors or flaws in the given code. Some common purposeful bugs are a variable that the user has not declared, missing colon in the code, or incorrect indentation. The last type of exercise

that Phanon uses is a creation exercise. In a creation exercise, Phanon instructs the user to write the code themselves. These types of exercises usually precede modification exercises that have exposed the user to the code they are creating. Several of these exercises are prevalent in the lessons because they allow the user to practice creating the code several times with different variables and arguments. For example, one exercise might ask the user to create a for-loop that outputs zero through four values. The preceding exercise will then ask the user to create a for-loop that outputs the values one through eight.

3.3 Pilot

The researcher conducted the pilot study in the 2020 summer semester and the conclusive study in the 2020 fall semester. The same professor taught both semester's courses. The researcher sent out invitations to participate in the study to all students (approximately 70 people). From the 70 students that received invitations to participate, five students accepted the invitation and completed the think-aloud sessions. The student's low interest in the study's participation can be attributed to the lack of compensation. Even though the introductory course students still had to complete the syntax exercises regardless of the study, there was no appeal for letting the researcher observe the exercises.

The researcher intended the pilot study to be conclusive. However, due to time constraints and the students' lack of interest, it was determined that the researcher could improve methods with a 2020 Fall semester study. A method that improved with the conclusive study compared to the pilot was the selection of exercises. The course used for the pilot was a shortened seven-week semester course, and once the IRB approved the protocol, most of the cognitively engaging lessons had already been due. The lessons that the researcher observed for the pilot were functions I, Operator Overloading, and Lists II. Students have already experienced some of the more powerful fundamentals for computer science topics at this stage in the course.

3.4 Participant Selection

Although the students' interest was low in the pilot, it became a valuable learning experience and provided keys to success in the fall semester study. It was evident that compensation for participation was a necessary method to attract students. The compensation also provided a way for the researcher and PI to select students based on criteria that may influence learning. The researcher focused their selection of participants on various gender, programming experience, and academic background. The professor that taught the introductory computer science course was the same for the 2020 fall semester. The first section was taught as a seven-week shortened class and contained approximately 74 students. The second section was taught as a fourteen-week normal duration class and contained approximately 94 students. Within the first week of the course, the professor announced the online course management system, canvas, on both course pages. The professor posted the announcement as follows:

> "This semester, some students will have the opportunity to participate in an educational study being conducted by Dr. John Edwards. Students will receive emails inviting them to participate in the study. This study is approved by the instructor of the course and by the university IRB (#11325). The instructor of the course will not know who was invited to participate or who is participating at any stage of the study. To participate in the study you must be at least 18 years of age, have a personal laptop or desktop computer with a webcam, and have internet access in a private setting. Participation in the study will take approximately 30 minutes. Participants who complete the study participation requirements will be compensated with a $60.00 gift card. For information on the study, you may contact the Principal Investigator (PI) Dr. John Edwards or Stephanie Gonzales."

Following the professor's announcement on the course pages, the researcher commented on the announcement with a survey link. In the comment, the researcher instructed the students to complete the survey if they were interested in the study. The survey asked for contact information, gender identity, academic information, and programming experience. Academic information and gender identity questions were optional for the participant to answer with one of the answer options as 'prefer not to answer'. The number of survey

responses between both courses was reasonably even. So the researcher and PI determined that the selection pool for participants should be limited to the shorter seven-week course students.

Once the list of interested participants was limited to the shorter course students, the researcher and PI met to analyze survey results. The researcher preferred an equal number of female and male students and various programming experience and academic competence. Initially, fourteen of the applicants, six female, and six male, that expressed interest through the survey were chosen and contacted by the researcher. The researcher instructed the chosen participants to complete a consent form and schedule the think-aloud sessions. Of the initial fourteen individuals that the researcher contacted to participate, two female participants opted not to pursue the study, and another female never responded or signed the consent form. In place of the female participants that dropped the study, the researcher sent more emails to each female left in the survey responses. Three of the female participants that the researcher invited in the second round signed the consent forms and scheduled the think-aloud sessions.

Table 3.1 represents the participants in the think-aloud study. The researcher has changed the names to preserve confidentiality. Each individual on the list participated in all three of the think-aloud sessions except for Jennifer and Kristy. Jennifer and Kristy both dropped the course shortly after they participated in the first session. From the 18 students that were invited to participate in the study, 11 remained for all three sessions.

Participant List					
Name	GPA	ACT	Programming Experience	Used Python	Gender
Jennifer	High	Low	Low	No	F
Sarah	High	High	None	No	F
Scott	Medium	High	None	No	M
Kyle	High	High	High	Yes	M
Jack	High	High	None	No	M
Kristy	Low	High	High	No	F
Sam	Medium	Medium	High	No	M
Trevor	Medium	Low	None	No	M
Hailey	High	N/A	None	No	F
Emily	Low	High	None	No	F
Brooke	Low	Medium	None	No	F
Matt	Low	Medium	High	Yes	M
Blake	Medium	Low	None	No	M

Table 3.1: 2020 Fall Think Aloud Study Participants

3.5 Exercise Selection

Each week, the introductory computer science course had several lessons due: a collection of exercises. Because the lessons selected by the PI and researcher were all due within the same week, the researcher conducted all of the think-aloud sessions in one week. The researcher and PI carefully selected the topics based on the content and fundamentals that the lessons explored. The lessons that were selected were Conditionals I, For-Loops I, and For-Loops II.

3.5.1 Conditionals I - Lesson 3.2.1

Conditionals are a fundamental subject in computer science that deals with the control and flow of a program. The structure of a conditional consists of a condition statement that evaluates to true or false. The program behaves according to the evaluation of the condition statement. This principle is one of the most common constructs in any programming language and essential to development. Observing conditional exercises can show perspective in the students' perception of Boolean statements, flow and control, and scope.

3.5.2 For-Loops I - Lesson 3.5.2

For-loops are another critical topic that deals with the control and flow of a program. The for-loop executes code for a specific number of iterations. Like conditionals, for-loops are frequently implemented in many computer programs and are fundamental concepts to understand. For-loops are also common structures that are prone to elusive flaws and incorrect implementation from developers. Observing for-loop exercises may give perspective to a student's perception of boundary conditions and sequences.

3.5.3 Nested For-Loops II - Lesson 3.5.4

Nested for-loops are another form of meaningful control for a program. The nested for-loop structure consists of a for-loop nested inside of another for-loop. This type of control is useful for working with multiple dimension containers or permutations. Even more than the standard for-loop, students are prone to making mistakes when writing nested for loops. The cyclomatic complexity of a nested for-loop often confuses new programmers when implementing and reading. Observing the nested for-loop exercises may reveal necessary procedures that students engage in as they debug these complex structures.

3.6 Zoom

The researcher conducted the study over a semester that fell during the COVID-19 pandemic. Due to the pandemic, the course instructor taught the introductory computer science

course exclusively online. The researcher conducted the think-aloud sessions over Zoom for the student and researchers' safety and compliance with the IRB protocol. Zoom is a video and communications software program that allows individuals to video conference over an internet connection. The participants scheduled three sessions in twenty-minute time slots with the researcher at a time that was convenient for them. This allowed the participants to control their environment, and the researcher encouraged them to pick a quiet place with an internet connection. Zoom's waiting room feature was applied to each session to ensure confidentiality. Once the participant had clicked on the Zoom link, they were directed to a wait screen while the researcher admitted them to the session. The students could share their screen with the researcher and complete the syntax exercises while expressing their thoughts. Zoom generated video and audio files of the recordings that the researcher later coded.

3.7 Coding

After the study, the researcher analyzed thirty-five videos and thirty-five audio files with a grounded approach [16]. The researcher transcribed audio files with Otter AI, a speech to text language application. After Otter AI analyzed the transcripts, they were then audited for correctness by the researcher. The researcher used a qualitative data software called Nvivo to analyze the video recordings. The researcher first recorded the start and end times of each exercise. Once the start and end times were recorded, the researcher assigned the exercises to parent nodes. The parent nodes each had children nodes that represented the moves recorded.

The researcher used a codebook within each child node represented in table 3.2 to analyze the video and transcript files. The videos were divided into time-spans based on the action of the participant and then assigned a code. The researcher divided the language from the transcripts into the exercise that they corresponded with. Due to the large amount of data gathered from the study, the researcher and PI agreed on a limit of eleven exercises to code.

The exercises they selected were: four exercises from the conditionals lesson, four exercises from the for-loop lesson, and three exercises from the nested for-loop lesson. The exercises that were analyzed were a variety of modification, bug fix, and creation problem types. Other factors, such as duration and the participants' general response, were also considered when selecting the coding exercises.

The codes used in each exercise were divided into moves in table 3.3 to show the coding behaviors that the student engaged in during the exercises. The moves were categorized similarly to a theory-building analysis [17]. Building moves are behaviors that involve creating code. There is only one testing move for this study, and it is running the code. Debugging moves consist of modifying code, reverting the exercise, and pressing the mulligan button. Sense-making moves are actions in which the student senses the environment, such as reading instructions, hint, or code. Similar to sense-making moves, response sense-making moves are behaviors in which the student makes sense of the environment after an event or response from the environment. These moves usually follow a failed run of code and include looking at the output of test results. Drawing on knowledge resources is an action that involves using resources outside of the current exercise. These moves are referencing a previous question or using the web browser to find a solution.

Code Book	
Code	Description
Create Code	The user has created code
Hint	The user has referenced the hint in the instructions
Indentation - Auto	The user has hit the enter button and automatic indentation was applied
Indentation - Remove	The user has removed indentation
Indentation - Space	The user has added indentation through spaces
Indentation - Tab	The user has added indentation through tabs
Instructions	The user has referenced the instructions
Modify Code	The user has modified the code
Mulligan	The user has clicked on the button that reveals the answer
Output	The user has referenced the output
Read Code	The user has read the code
Reference Previous	The user has referenced a previous exercise
Researcher Question	The user has asked the researcher a question or the researcher has asked the user a question
Revert Exercise	The user has clicked on the button that reverts the exercise
Run Code	The user has clicked on the play button that runs the code
Test Failed	The user has submitted code that failed
Test Results	The user has referenced the test results

Table 3.2: Think Aloud Study Code Book

Coding Moves	
Move	Related Codes
Building	Create Code, Indentation - Auto, Indentation - Remove, Indentation - Space, Indentation - Tab
Testing	Run Code
Debugging	Modify Code, Revert Exercise, Mulligan
Sense Making	Hint, Instructions, Read Code
Response Sense Making	Output, Test Results
Drawing on Knowledge	Reference Previous
Resources	

Table 3.3: Coding Moves

3.7.1 Conditionals I - Exercise 6

Exercise six shown in figure 3.3 is a modification problem that asks the student to change the code on line two so that the code prints Hello if a is equal to seven. The instructions specify only to change line two. The researcher and PI selected problem six for coding because of the participants' general response to this question. This problem is the first and only time that Phanon represents the lesson's conditional statement by a Boolean keyword rather than an expression. Students who have little programming and Python experience may not be familiar with how the Boolean keyword affects a conditional statement.

Fig. 3.3: Lesson 3.2.1 Exercise 6

3.7.2 Conditionals I - Exercise 13

Exercise thirteen shown in figure 3.4 is a bug fix problem that asks the student to fix the bugs on line two. The instructions clarify that two bugs exist online two. The researcher and PI selected problem thirteen for coding because of the type of exercise and the fact that the instructions specify to fix two bugs. The exercises are designed to be quick and progressive, which can cause students to assume intended behavior without reading the instructions. This problem may be a good indicator to evaluate the significance of the instructions.

Fig. 3.4: Lesson 3.2.1 Exercise 13

3.7.3 Conditionals I - Exercise 16

Exercise sixteen shown in figure 3.5 is a problem that asks the student to write an if statement that prints Hello if the variable value is less than four. This exercise is the first create problem of the lesson in which the operator is meant to be the less than operator rather than the double equal sign operator. The transition should be easy for the student and provide a way to gain experience with different control statements.

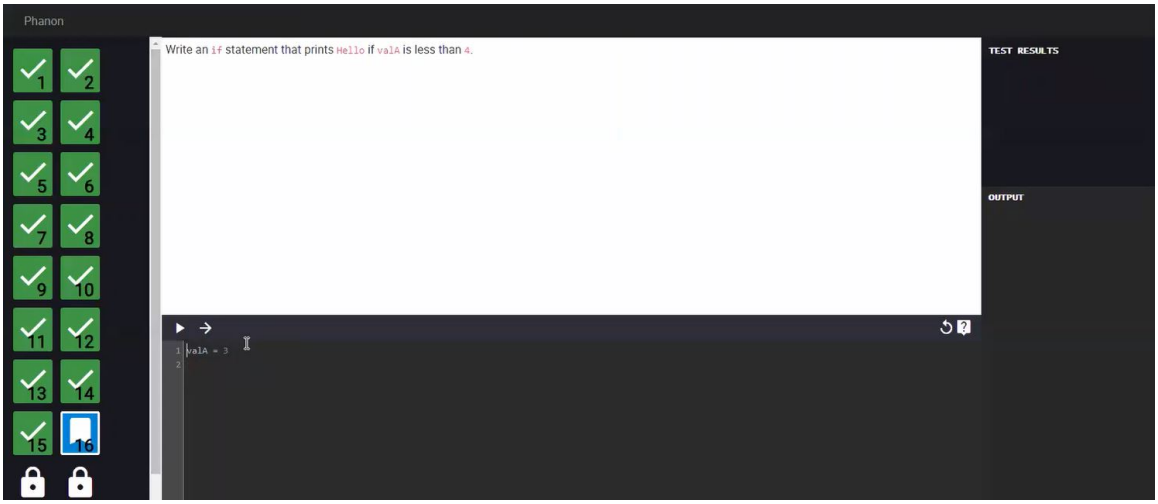

Fig. 3.5: Lesson 3.2.1 Exercise 16

3.7.4 Conditionals I - Exercise 28

Exercise twenty-eight shown in figure 3.6 is a create problem that asks the student to write a program that outputs Hello, Goodbye, and Done. Each word is meant to be on a different line, and the instructions specify that Hello and Goodbye should be printed only if a is equal to three. Done should be printed no matter what. This problem is the first creation problem that involves some understanding of the scope of the conditional.

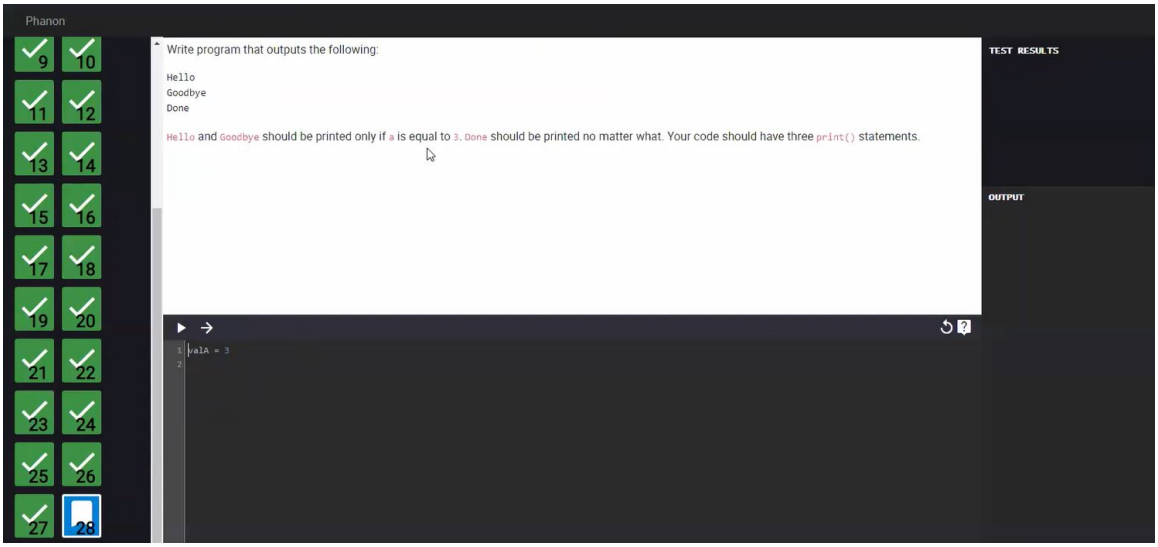

Fig. 3.6: Lesson 3.2.1 Exercise 28

3.7.5 For-loops I - Exercise 1

Exercise one shown in figure 3.7 is a modification problem that asks the student to run the code first and then change it so that it outputs the numbers zero through three. For some students this may be their first exposure to for-loops.

Fig. 3.7: Lesson 3.5.2 Exercise 1

3.7.6 For-loops I - Exercise 12

Exercise twelve shown in figure 3.8 is a creation problem that asks the student to add the print call to the for-loop statement. Two hints are given in this exercise that remind the student to indent the print statement and reference a previous exercise if they get stuck. The indentation part of this problem may give incite to the students perspective on the scope of a for-loop.

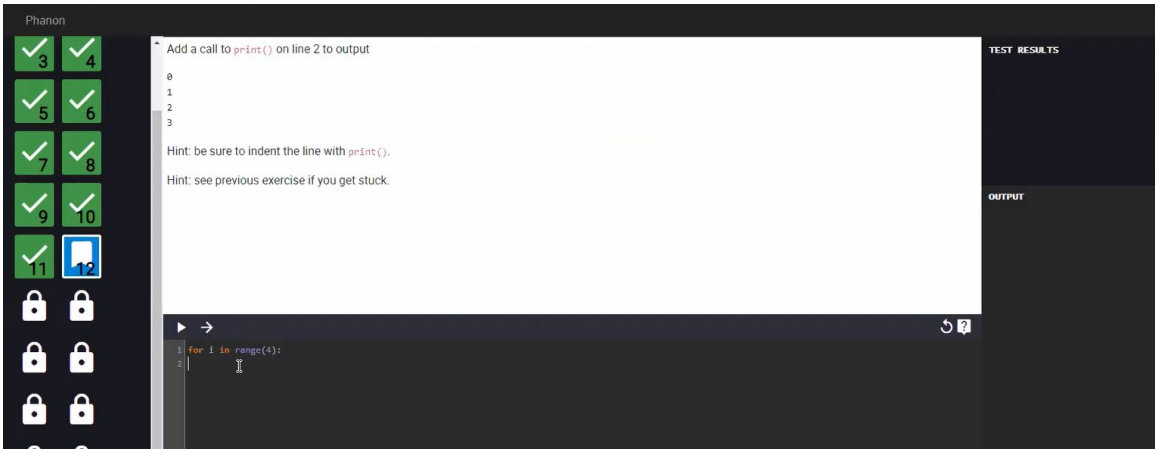

Fig. 3.8: Lesson 3.5.2 Exercise 12

3.7.7 For-loops I - Exercise 20

Exercise twenty shown in figure 3.9 is a creation problem that asks the student to write a for-loop that outputs the numbers zero through seven. The instructions specify that for this problem the student should only give one argument. The problem is preceded by several creation problems in the lesson and should demonstrate the students ability to relate their practice to a similar problem.

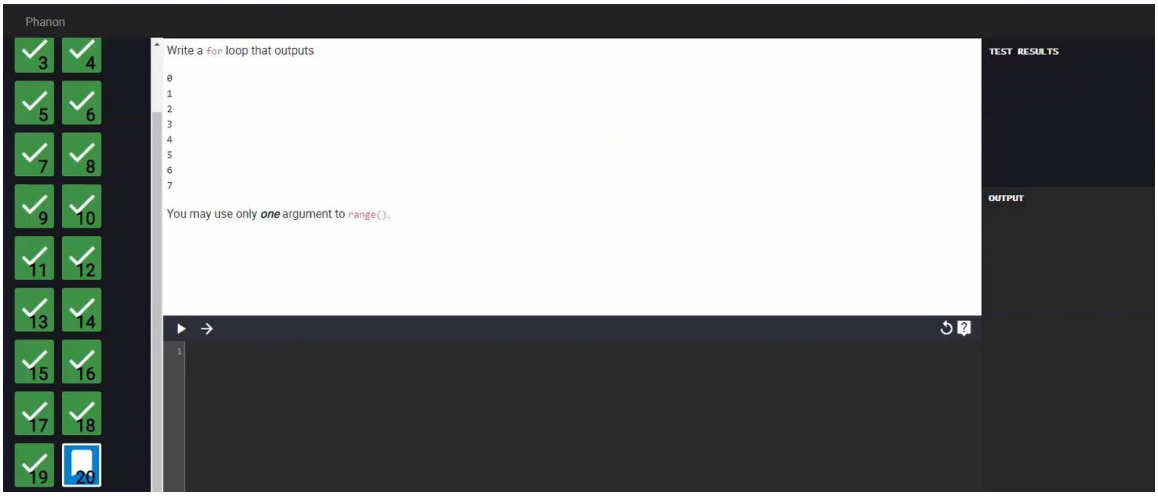

Fig. 3.9: Lesson 3.5.2 Exercise 20

3.7.8 For-loops I - Exercise 21

Exercise twenty-one shown in figure 3.10 is a creation problem that asks the student to write a for-loop that outputs the numbers two through four. The problem is preceded by several creation problems in the lesson and should demonstrate the students ability to relate their practice to a similar problem.

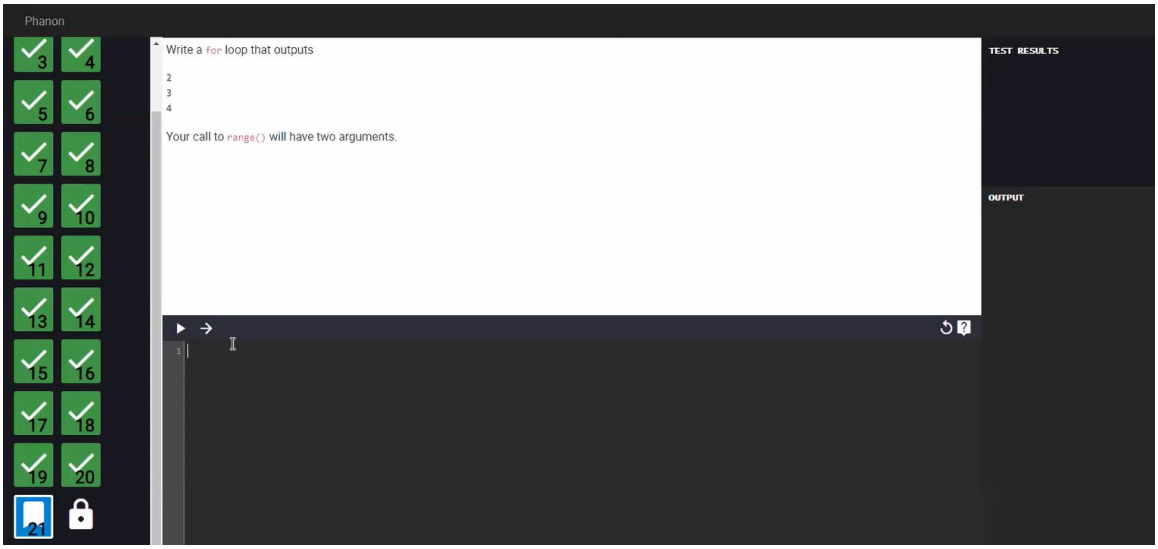

Fig. 3.10: Lesson 3.5.2 Exercise 21

3.7.9 For-loops III - Exercise 8

Exercise eight shown in figure 3.11 is a creation problem that asks the student to write the inner for-loop for a nested for-loop. This exercise is the first of the lesson that requires the student to write the print statement as well as the for-loop statement. The exercise may be a good indicator of the students understanding of scope for the outer for-loop.

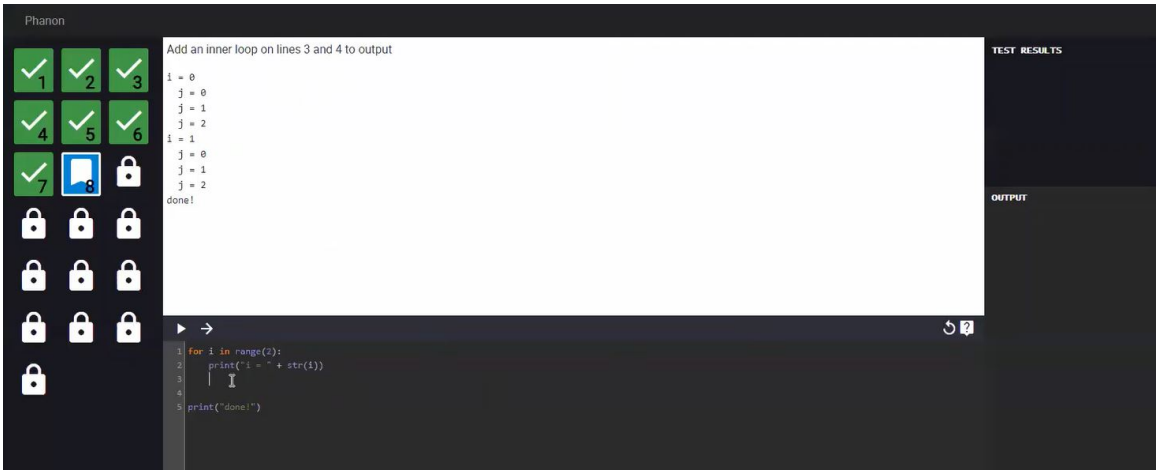

Fig. 3.11: Lesson 3.5.4 Exercise 8

3.7.10 For-loops III - Exercise 10

Exercise ten shown in figure 3.12 is a creation problem that asks the student to write the inner and outer for-loops from scratch. The exercise is the first creation problem that asks the student to write both loops without any starter code in the code editor except for the done print statement. This could indicate the students' understanding of scope for both loops and demonstrate their progression of creating nested for-loops.

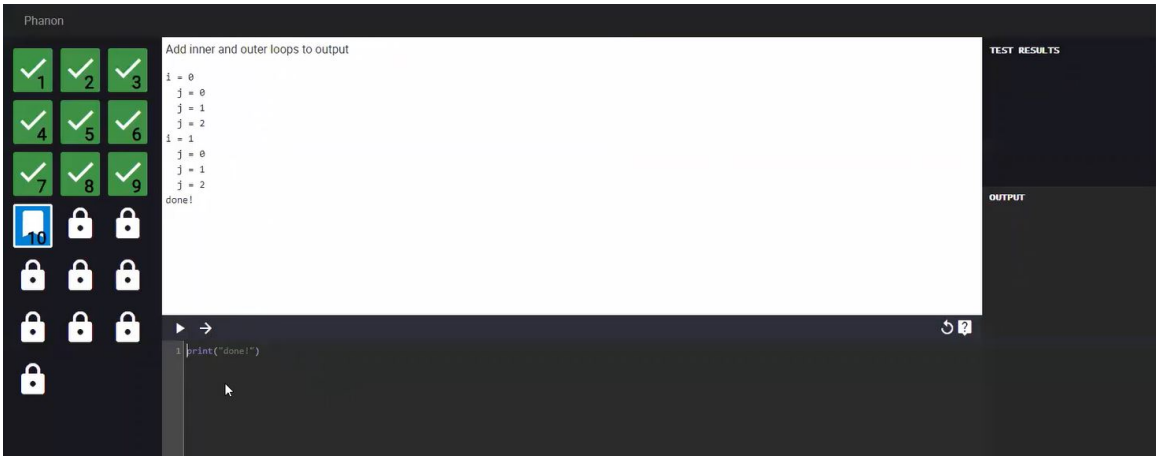

Fig. 3.12: Lesson 3.5.4 Exercise 10

3.7.11 For-loops III - Exercise 15

Exercise fifteen shown in figure 3.13 is a bug fix problem that asks the student to fix the nested for-loop in the code editor so that it outputs the representation given. The exercise may demonstrate the student's ability to recognize scope flaws in nested for-loops.

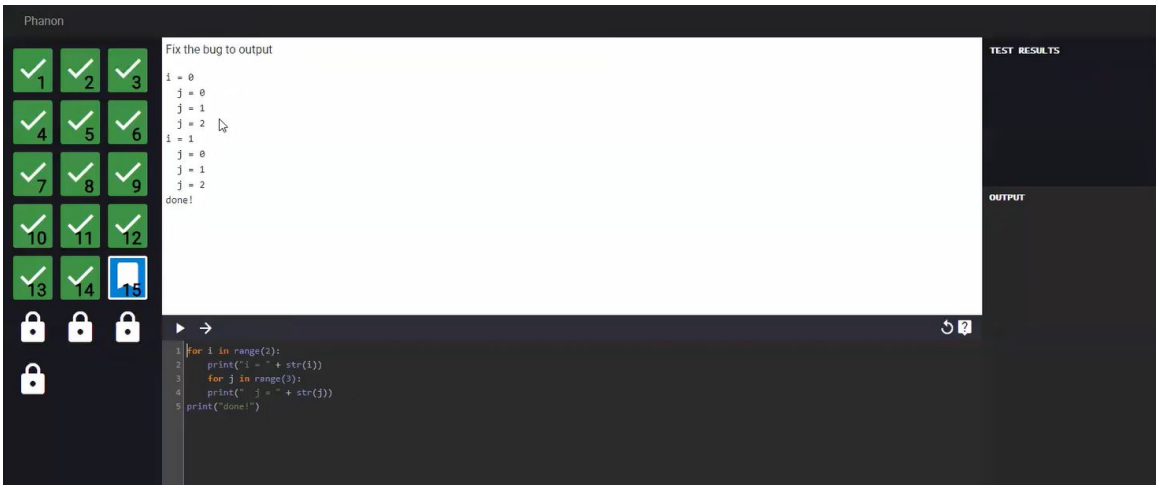

Fig. 3.13: Lesson 3.5.4 Exercise 15

CHAPTER 4

RESULTS

This section contains the results of the study. There are two significant types of analysis discussed in this section. One is a qualitative analysis of select exercises. The other type is a quantitative representation of the participant's performance in the course for those that agreed to disclose the information. Generally, the syntax exercises took little time and limited building and debugging moves. The exercises are designed to be quick and simple solutions, so exercises with an average duration longer than a minute represented vital debugging exercises. The minimum, maximum, and average duration of the coded exercises is represented in table 4.1. Some of those exercises were great resources for insight into the students' thought process as they engaged in theory coding moves.

Exercise Duration				
Exercise	Min	Max	Average	Standard Deviation
3.2.1 Exercise 6	0:00:25	0:06:01	0:02:13	0:01:44
3.2.1 Exercise 13	0:00:01	0:00:22	0:00:13	0:00:05
3.2.1 Exercise 16	0:00:21	0:01:19	0:00:32	0:00:14
3.2.1 Exercise 28	0:00:42	0:03:07	0:01:10	0:00:40
3.5.2 Exercise 1	0:00:01	0:00:37	0:00:17	0:00:10
3.5.2 Exercise 12	0:00:10	0:00:24	0:00:16	0:00:04
3.5.2 Exercise 20	0:00:14	0:00:23	0:00:18	0:00:02
3.5.2 Exercise 21	0:00:16	0:00:35	0:00:22	0:00:05
3.5.4 Exercise 8	0:00:34	0:02:24	0:01:04	0:00:33
3.5.4 Exercise 10	0:00:57	0:01:23	0:01:09	0:00:08
3.5.4 Exercise 15	0:00:08	0:00:46	0:00:18	0:00:11

Table 4.1: Exercise Duration Statistics

Standard Deviation for MIN=0:00:17, Standard Deviation for MAX=0:01:39, Standard Deviation for AVERAGE=0:00:36

4.1 Lesson 3.2.1 - Exercise 6

Exercise six is a simple debug exercise that requires the modification of less than one line of code. Before the participants had access to exercise six, they had to complete exercises one through five in lesson 3.2.1. In the lessons, the preceding exercises are meant to prepare the user for their work. This exercise is the first that places a Boolean keyword in the conditional, rather than an expression. The instructions for the user read "Change line 2 so that the code prints Hello if a is equal to 7. Change only line 2, and make sure it prints Hello only if a is equal to 7." The following code sits inside the code editor at the start of the exercise.

```
valA = 7
if False:
    print("Hello")
```

Despite the simplicity of the problem, many of the participants spent a significant amount of time on this question. Two participants elected to use the button that reveals the answer for that exercise after grappling with it for some time. Not all participants struggled with this exercise, which makes it an exciting choice for qualitative analysis.

3.2.1 Exercise 6 Duration		
Name	Duration (h:mm:ss)	Codes
Jennifer	0:05:18	44
Sarah	0:01:02	18
Scott	0:01:18	23
Kyle	0:00:33	5
Jack	0:06:01	87
Kristy	0:01:10	10
Sam	0:01:30	4
Trevor	0:02:03	24
Hailey	0:00:49	7
Emily	0:01:51	16
Brooke	0:03:33	33
Matt	0:00:25	3
Blake	0:03:20	48

Table 4.2: 3.2.1 Exercise 6 Performance Summary

$Mean \pm$Standard Deviation$for Duration$ =0:02:13±0:01:44, Mean±Standard Deviation for Codes = 24.8±22.9

Below are temporal decompositions for three of the participants as they debugged this exercise. For this study high performing students are students with a high and grade, A- or better, in the course. Jack's experience with the exercise is an example of a high performing student who engaged in many debugging moves. Sarah was chosen for decomposition because she is also another high performing student who engaged in many debugging moves. Her duration for completing the exercise was less than Jack's but longer than others. For perspective, Matt was selected as a low performing student who did not engage in very many debugging moves. His duration for completing the exercise was the lowest out of all of the participants.

4.1.1 Jack's Moves

Step 1: Sense Making

Jack quietly hovers over the instructions

Step 2: Debugging the Code

Jack then begins modifying the code. Jack replaces the False: on line two with True: and adds valA == 7.

> **Jack:** "So I have to change that to true. And then I make sure if valA is equal to seven."

```
1 valA = 7
2 if True: valA == 7
3     print("Hello")
```

Jack then runs the code and the test fails. The output message says "SyntaxError: bad input on line 3". Jack leans closer to the screen and examines the output.

> **Jack:** "I messed that up."

Step 3: Debugging the code

Jack's mouse drifts over the code in the code editor. He pulls back from the screen.

Jack: "Hmm."

After reading the code Jack removes the colon after the True keyword on line two.

```
1 valA = 7
2 if True valA == 7
3     print("Hello")
```

Jack runs the code again and the output remains the same "SyntaxError: bad input on line 3". He then removes one of the equal signs on line two.

```
1 valA = 7
2 if True valA = 7
3     print("Hello")
```

Jack runs the code again. The output message changes to "SyntaxError: bad input on line 2".

Step 4: Drawing on Knowledge Resources

Jack leans in close toward the screen and puts his fist up to his mouth.

Jack: "I'm just going to go back and look here to make sure"

He selects the previous exercise, number five, as a reference. Jack then selected another previous exercise, number three for a few seconds before returning to number five. Jack returns to exercise six and removes 'True' from line two.

```
valA = 7
if valA = 7
    print("Hello")
```

Jack runs the code again and the test fails. The output message remains the same as before "SyntaxError: bad input on line 2". He leans in towards the screen and squints slightly. Jack then selects a previous exercise, problem four and studies it.

Step 5: Drawing on Knowledge Resources

Jack leans in close toward the screen and puts his fist up to his mouth.

Jack: "I'm just going back and checking to see what I'm doing wrong on this one right here"

He then selects another previous exercise, problem three. Jack selects another previous exercise, problem two, but then quickly returns to exercise six.

Step 6: Debugging again

Jack: "We're going to restart this and see."

Jack clicks on the button to reset the exercise and the code goes back to the original form. He adds 'valA == 7' after the False and colon on line two.

Jack: "See if that does anything"

```
valA = 7
if False: val == 7
    print("Hello")
```

Jack: "That'll be right now"

Jack runs the code and the test fails. He leans in close to the screen briefly before changing 'val' to 'valA'.

```
valA = 7
if False: valA == 7
    print("Hello")
```

Jack runs the code again and the test fails. He leans in close and puts his hand up to his chin.

Jack: "No?"

Step 7: Sense Making

Jack's mouse drifts up toward the instructions and he mutters quietly as he reads them.

Jack: "change line two so.."

Step 8: Debugging

Jack removes the 'False' from like two and replaces it with 'True' then removes 'valA == 7' from line two.

```
valA = 7
if True:
    print("Hello")
```

Jack runs the code and the test fails. This time the output displays Hello but a hidden test has failed. Jack examines the output briefly. After 'True:' he starts to write 'valA' but pauses and then removes it. He then adds '== 7'.

```
valA = 7
if True: == 7
    print("Hello")
```

Jack runs the code again and the test still fails. He gives a slight shrug and shakes his head. He then removes '==7'.

Step 9: Drawing on Knowledge Resources

Jack references a previous exercise, problem number five, and exhales putting his hand on his mouth. He selects question four and then jumps immediately to question three. Then he clicks back to question five and pulls away from the screen.

Jack: "okay.."

Step 10: Debugging

Jack adds 'valA == 7' to line two in-between True and the colon.

```
valA = 7
if True valA == 7:
    print("Hello")
```

He runs the code and the test fails. His mouth sets into a hard line and he leans closer to the screen. He then removes 'valA == 7' and types 'valA' after the colon.

```
valA = 7
if True: valA
    print("Hello")
```

Jack runs the code and the test fails. He removes 'valA'.

Step 11: Drawing on Knowledge Resources

Jack clicks on exercise four and studies the code.

Jack: "More tricky than I thought it was gonna be."

Step 12: Sense Making

Jack returns to exercise size and follows the code with his mouse. He runs the code and the output prints 'Hello' but the test fails. He leans in close to the screen.

Jack: "Hidden test failed."

Jack's mouse moves over to the code in the editor and he puts his hand up to his mouth as he studies the code. On line one he adds another equal sign to the expression but immediately removes it. His mouse continues to follow the code in the editor and hovers over line two after the conditional statement. He types 'valA == 7'.

```
1 valA = 7
2 if True: valA == 7
3     print("Hello")
```

Jack runs the code and the test fails. He furls his brow and removes 'valA == 7' on line two.

Step 13: Drawing on Knowledge Resources

Jack clicks on question five, and then on question one briefly. He returns to question five and hovers over the code in the editor. He returns to question six and types 'valA == 7:' to the end of line two.

```
valA = 7
if True valA == 7:
    print("Hello")
```

Jack runs the code and the test fails. The output displays “SyntaxError: bad input on line 2”

Step 14: Debugging

Jack leans in closely to the screen and exhales as he removes ‘valA == 7:’. He mumbles and his eyes drift up towared the instructions. He removes the word ‘True’ and replaces it with ‘valA == 7 True’.

```
valA = 7
if valA == 7 True
    print("Hello")
```

He runs the code and the test fails. The output displays the same message “SyntaxError: bad input on line 2”. His lips tighten and he removes all of line two and replaces it with ‘True: valA == 7’.

```
valA = 7
True: valA == 7
    print("Hello")
```

Jack runs the code and it fails. The output message has changed to “SyntaxError: bad input on line 3”. He removes ‘valA == 7’ and replaces it with ‘a == 7’.

```
valA = 7
True: a == 7
    print("Hello")
```

Jack runs the code and the test fails. The output message stays the same. He removes 'a == 7' and lets out a sigh then replaces it with ' == 7'.

```
valA = 7
True: == 7
    print("Hello")
```

He runs the code and the test fails. The output message changes to "SyntaxError: bad input on line 2". As Jack considers the code he starts to remove the '7' and then puts it back.

Jack: "Hmm."

Step 15: Drawing on Knowledge Resources

Jack rests his mouse on line two and then clicks on exercise five. He pulls back quickly from the screen and returns to exercise six.

Jack: "So.."

Jack replaces 'True:' with 'valA' and then adds a colon after the '7'.

```
valA = 7
if valA == 7:
    print("Hello")
```

Jack runs the code and it passes. He shrugs while laughing and shakes his head.

Jack: "There we go."

Researcher: "Is having that false there the tricky part?"

Jack: "Yeah, well normally like if it has something new it has something to do with like, what we're doing at that time, so I figured that it was like that's another way you can command it to do that."

4.1.2 Sarah's Moves

Step 1: Sense Making

Sarah follows the instructions with her mouse as she silently mouths the words. She then follows the code with her mouse.

Sarah: "So now they want me to write it myself"

Step 2: Drawing on Knowledge Resources

Sarah quickly goes back to exercise number five and looks at the code in the editor.

Sarah: "See back here it said value.."

Step 3: Sense Making

Sarah returns to question six and rests her cursor on line two.

Sarah: "see it says false"

Her eyes drift up toward the instructions.

Sarah: "so then change line two so that... equal to seven"

Step 4: Debugging

Sarah clicks on line two after 'False: ' and types 'valA'.

Sarah: "Um.. so I know I want value A to only be seven"

Sarah types '++' and then immediately removes it. Her mouse follows the instructions and rests on 'Hello only if valA is equal to 7'. She adds ' == 7' after valA and then removes 'False'.

Sarah: "I don't think the false is supposed to be here"

```
1 valA = 7
2 if : valA == 7
3     print("Hello")
```

Sarah runs the code and the test fails.

Sarah: "Maybe it was, I'll put it back"

She types 'False' between the if and the colon on line two.

```
1 valA = 7
2 if False: valA == 7
3     print("Hello")
```

Sarah runs the code again and the test fails. The output displays "SyntaxError: bad input on line 2". her mouse hovers over the code.

Sarah: "Hmm."

Step 5: Drawing on Knowledge Resources

Sarah clicks on question five and follows the code with her mouse. Then she follows the instructions with her mouse.

Step 6: Debugging

Sarah moves back to exercise six.

> **Sarah:** "Oh okay yeah. So the false isn't supposed to be there but the semi colon needs to be on this side"

She removes 'False:' and then places the colon at the end of line two.

> **Sarah:** "or the colon I mean."

```
1 valA = 7
2 if valA == 7:
3     print("Hello")
```

Sarah runs the code and the test passes.

4.1.3 Matt's Moves

Step 1: Sense Making

Matt's mouse hovers over the instructions for a few seconds. His mouse drifts down to the code and highlights the word 'False'.

Step 2: Debugging

> **Matt:** "Change line two so that it is equal, or, so that it prints that it is equal to seven"

Matt removes the word 'False' from the code and replaces it with 'valA == 7'.

```
valA = 7
if valA == 7:
    print("Hello")
```

Matt: "Only line two, okay, so valA equal to seven"

Matt runs the code and the test passes.

4.2 Lesson 3.5.4 - Exercise 8

Exercise eight is a creation problem that asks the student to write the inner for-loop for a nested for-loop. This exercise is the first of the lesson that requires the student to write the print statement and the for-loop statement. The following code sits inside the code editor at the start of the exercise.

```
for i in range(2):
    print("i = " + str(i))


print("done!")
```

The participants that completed this exercise averaged one minute and four seconds for duration. The average number of codes recorded for this exercise is 12.45. Brooke took the longest to complete this exercise at two minutes and nineteen seconds, with twenty-six codes recorded. Jack took the shortest amount of time to complete the exercise at thirty-four seconds and seven codes recorded.

3.5.4 Exercise 8 Duration		
Name	Duration (h:mm:ss)	Codes
Sarah	0:00:49	5
Scott	0:01:42	33
Kyle	0:00:43	9
Jack	0:00:34	7
Sam	0:00:43	5
Trevor	0:01:27	16
Hailey	0:00:41	5
Emily	0:01:05	12
Brooke	0:02:19	26
Matt	0:00:39	4
Blake	0:00:54	15

Table 4.3: 3.5.4 Exercise 8 Performance Summary

$Mean \pm \text{Standard} Deviation for Duration$ =0:01:03±0:00:31, Mean±Standard Deviation for Codes =12.45±9.1

Represented in figure 4.1 compares the timeline and moves that three participants engaged in as they completed exercise eight—each of the participants engaged in each type of move at least twice. Multiple debugging and testing moves in their timelines reflect that they were learning through trial and error. Sensemaking moves and response sensemaking moves in the timeline reflect their attempts to understand the exercise's environment. Drawing on knowledge resources shows that they used resources outside of the code environment to gain understanding.

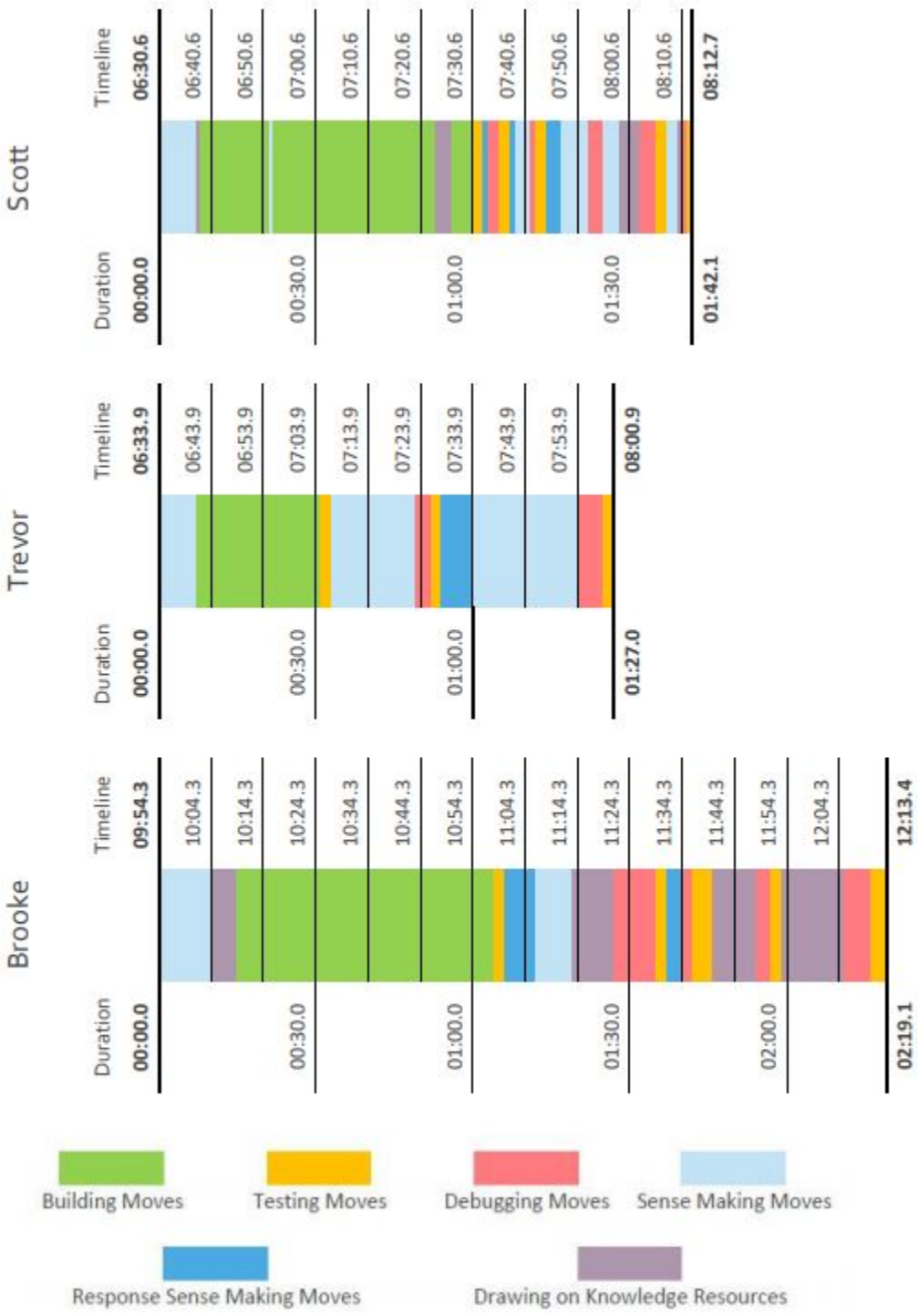

Fig. 4.1: Exercise Eight Moves Timeline

4.3 Lesson 3.5.4 - Problem 10

Exercise ten is a creation problem that asks the student to write the entire nested for-loop. This exercise is the first in the lesson that requires the student to write a nested for-loop without an inner or outer loop present in the code editor. The following code sits inside the code editor at the start of the exercise.

```
1 print("done!")
```

The participants that completed this exercise averaged one minute and nine seconds for duration. The average number of codes recorded for this exercise is 10.63. Emily took the longest to complete this exercise at one minute and twenty-three seconds with twelve codes recorded. Jack took the shortest amount of time to complete the exercise at fifty-seven seconds and nine codes recorded.

3.5.4 Exercise 10 Duration		
Name	Duration (h:mm:ss)	Codes
Sarah	0:01:06	9
Scott	0:01:05	9
Kyle	0:01:07	13
Jack	0:00:57	9
Sam	0:01:15	13
Trevor	0:01:00	9
Hailey	0:01:02	8
Emily	0:01:23	12
Brooke	0:01:14	14
Matt	0:01:02	10
Blake	0:01:18	11

Table 4.4: 3.5.4 Exercise 10 Performance Summary

Mean±Standard Deviation for Duration = 0:01:08±0:00:08, Mean±Standard Deviation for Codes = 10.7±/1.9

Represented in figure 4.2 is a comparison of the timeline and moves that three participants engaged in as they completed exercise ten. The absence of debugging moves in Scott and Trevor's timeline means that they submitted the correct code on the first attempt. Brooke engaged in debugging moves but not as frequently as she did in exercise eight.

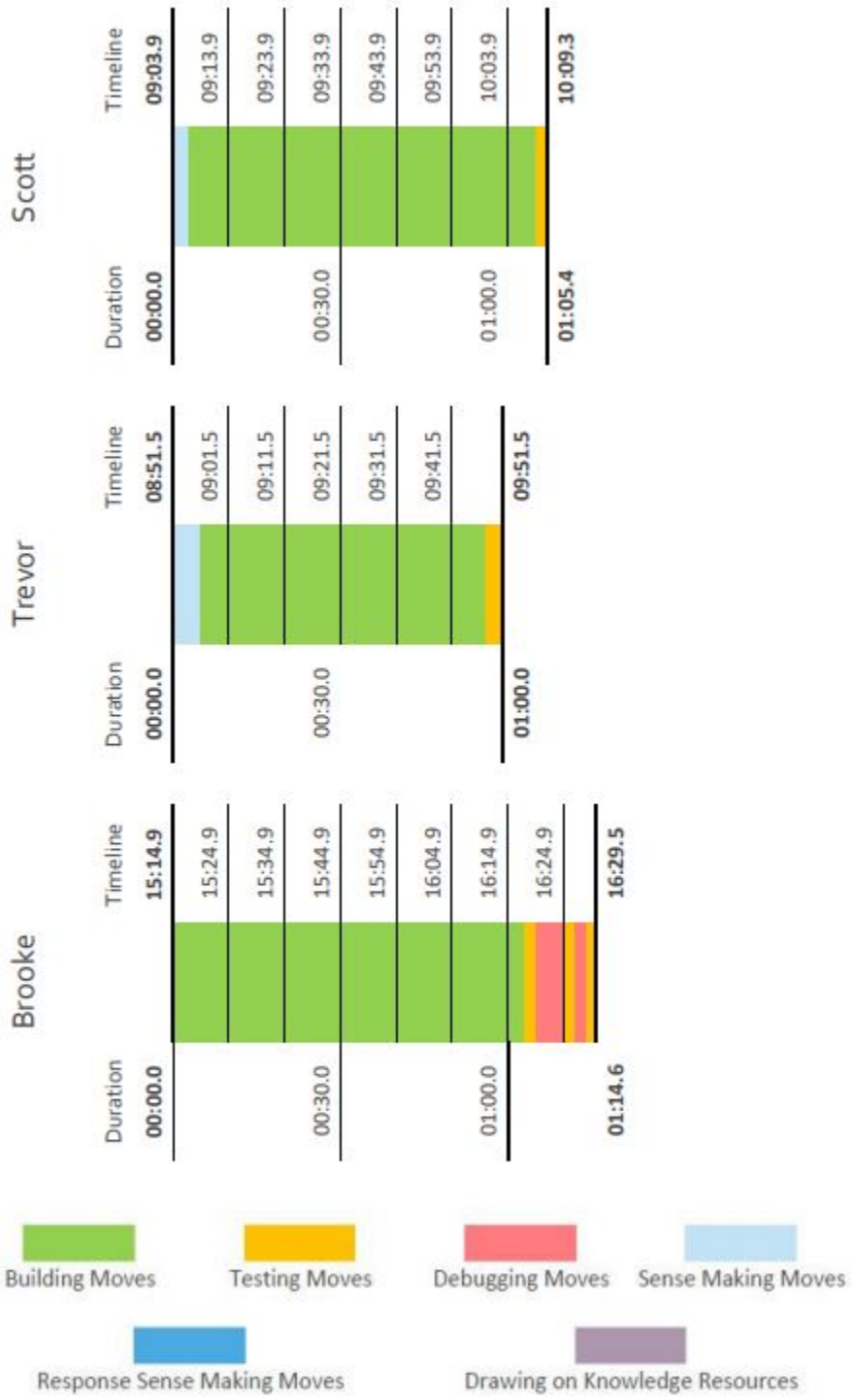

Fig. 4.2: Exercise Ten Moves

4.4 Participant Course Performance

Table 4.5 shows the grades of the participants who completed the study and gave permission to present their grades. A variety of performance in the course is reflected in the scores.

Participant Grades								
Name	Exercise Duration	Codes	Assignments	Exercises	Quizzes	Exams	Total	Letter
Sarah	29:10.9	154	100%	100%	98.79%	88.15%	96.32%	A
Scott	37:55.5	179	73%	100%	81.82%	71.11%	74.85%	C
Jack	28:53.6	215	100%	100%	100%	85.19%	95.56%	A
Sam	35:22.4	142	84.21%	100%	100%	78.52%	84.87%	B
Trevor	39:52.4	167	40.56%	100%	93.94%	74.81%	59.15%	D
Hailey	34:49.9	133	96.83%	100%	97.58%	78.52%	91.57%	A-
Emily	38:39.3	158	100%	100%	100%	84.44%	95.33%	A
Brooke	47:19.0	208	57%	94%	95.15%	65.19%	65.06%	D+
Matt	24:50.3	122	80%	96.97%	79.39%	41.48%	69.23%	C-

Table 4.5: 2020 Fall Think Aloud Study Participant Grades

Pearson Correlations between grade percentage and combined time is (r=-0.41, p=0.26), Pearson Correlations between grade percentage and total codes is (r=-0.07, p=0.85).

In figure 4.3 the blue plot represents the correlation between the grade percentage of the participants to the combined time it took the participants to complete all three lessons. The orange plot represents the correlation between grade percentage and total codes.

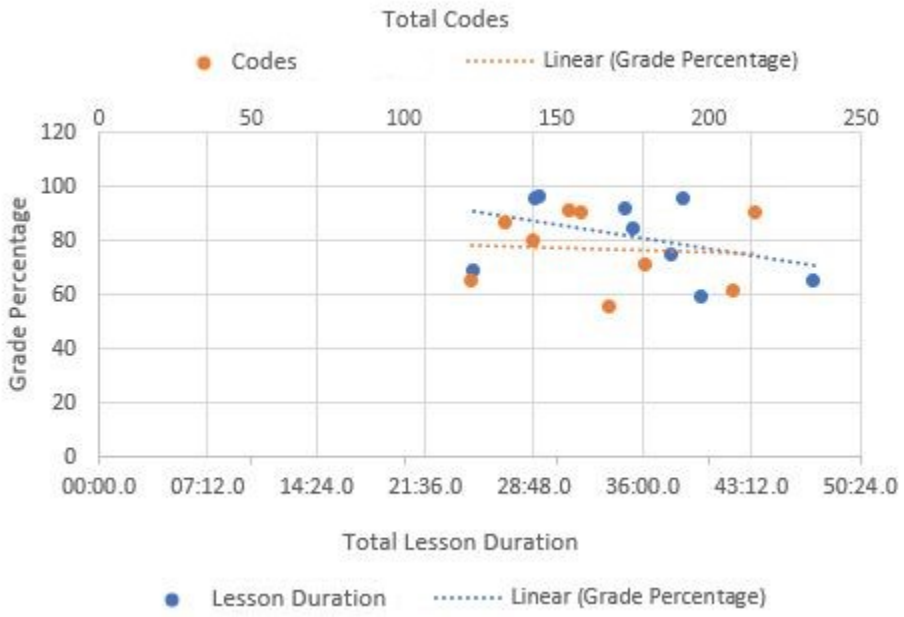

Fig. 4.3: Final Grades (Y Axis) Compared to Duration (Bottom X Axis) and Total Codes (Top X Axis), Pearson correlation between assignment score and combined time is (r=-0.57, p=0.11), Pearson correlation between assignment score and total codes is (r=-0.21, p=0.58).

In figure 4.4 the blue plot represents the correlation between the assignment score percentage of the participants to the combined time it took the participants to complete all three lessons. The orange plot represents the correlation between the assignment score percentage and the total codes.

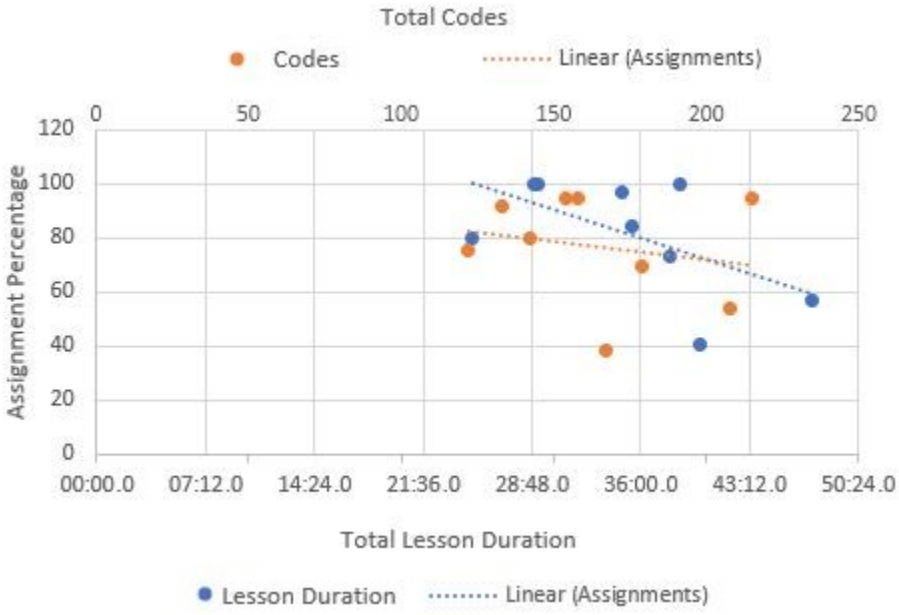

Fig. 4.4: Assignment Scores (Y Axis) Compared to Duration (Bottom X Axis) and Total Codes (Top X Axis), Pearson correlation of exam score to duration is (r=0.13, p=0.72), Pearson correlation of exam score and total codes is (r=0.29, p=0.44).

In figure 4.5 the blue plot represents the correlation between the exam score percentage of the participants to the combined time it took the participants to complete all three lessons. The orange plot represents the correlation between exam score percentage and the total codes.

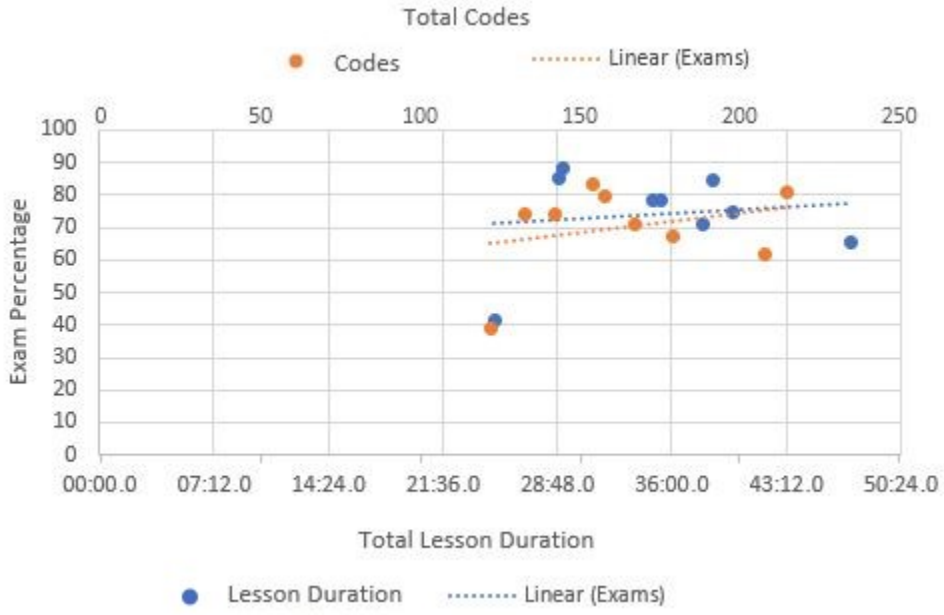

Fig. 4.5: Exam Scores (Y Axis) Compared to Duration (Bottom X Axis) and Total Codes (Top X Axis)

CHAPTER 5

DISCUSSION

Several patterns of student interaction with Phanon have emerged throughout the study. In this section, the interpretation of those patterns and an examination of the results are discussed. The exercises' effectiveness, in the context of the intrinsic and extraneous load, is also investigated in this section.

5.1 Duration and Codes

Duration, i.e., the amount of time it took students to complete the syntax exercises, was used as a measure of effort in this study, along with the number of codes. Each code represented an action that the user took while completing the exercises, so the higher the number of codes, the more effort they put into the exercise. Duration alone is not a valid indicator of effort because of the different types of problems that Phanon includes. A code creation problem in the nested for-loop lesson is meant to take a significantly longer time than a simple modification problem from the conditionals lesson. The minimum, maximum, and average duration for each exercise were used to determine the exercise's complication and was also a good indicator of whether the students generally engaged in debugging behaviors.

Exercise six in table 4.1 across participants shows a minimum of 0:25, a maximum of 6:01, and an average of 2:13. The average is significantly higher than the minimum, which indicates that the solution can be quick and straightforward. However, some of the participants took significantly more time to complete the exercise than others. In contrast, to the last exercise in the table, lesson 3.5.4 exercise fifteen had a minimum duration of 0:08, a maximum of 0:46, and an average of 0:18 seconds. From the representation of that problem,

an assumption can be made that this problem was generally not difficult for participants. If there were any debugging moves, they were minimal and most likely outliers.

Determining which exercises most likely presented more debugging moves for most participants was crucial in selecting coding exercises. Although some quick and simple exercises helped find patterns with some interactions, more meaningful patterns would be discovered in high debugging content problems.

5.2 Intrinsic and Extraneous Load

The study has shown a light on the issue of intrinsic load versus extraneous load. Intrinsic load is a measure of difficulty in learning a subject. The ineffectiveness of the way that subject is taught is extraneous load. Although the exercises in Phanon are designed to be quick and uncomplicated, there are two exercises from the study that were particularly difficult for most of the students. An evaluation of the two exercises determined that one was difficult because it represented a concept with a high intrinsic load. The other exercise was challenging for the students because of how the problem was presented to the user. This represents the extraneous cognitive load. A determination of which exercises cause extraneous load can help improve teaching methods by exposing confounding directions.

As described below, lesson 3.2.1 exercise six is the exercise that represented a high intrinsic cognitive load. The exercise can be tricky for students who do not know Python or have little experience in programming. High intrinsic load is represented in the exercise because solving the problem leads to a meaningful discovery about a syntax rule. If the individual is familiar with syntax in Python or general syntax rules, the solution is much more apparent. In contrast, Lesson 3.5.4 exercise five was written so that the students put in an unnecessary amount of cognitive effort for a solution that did not provide a meaningful discovery. This exercise was not a problem that the researcher coded but one that they evaluated for the exercise's effectiveness.

5.2.1 Lesson 3.2.1 Exercise 6

The solution to exercise six, see figure 3.3, is a simple one. The student is given a conditional with a Boolean in the conditional statement rather than the expression that the instructions describe. The simplicity is represented in the duration and number of codes in table 4.2. Some of the participants took very little time and moves to complete the exercise, and others took a long time and used many moves. Kyle, Sam, Hailey, and Matt took little time and effort to complete this exercise compared to the rest of the participants. Table 3.1 shows that out of the participants that took little effort and time to complete the exercise, Hailey is the only one that had not had any programming experience before the course. Kyle, Sam, and Matt have all had over ten hours of programming experience, and Kyle and Matt have also used Python. This indicates that the user is generally more proficient with this exercise if they have had prior experience with syntax.

In contrast, the less proficient participants with duration and number of moves in exercise six indicate a learning process for this problem. Sullivan et al. explain that a critical component between intrinsic and extraneous load is removing or limiting cognitive problem solving [18]. Jack 4.1.1 and Sarah's 4.1.2 temporal decomposition reveals their learning process as they discover an essential syntax rule. Within their decomposition, healthy patterns of intrinsic load are revealed instead of patterns of an extraneous load.

In step two of Jack's temporal decomposition, he begins coding and confidently announces what he believes he is supposed to do to answer the question. In his first attempt to solve the problem, he switches the Boolean keyword from 'False' to 'True' and adds 'valA == 7' after the colon. Even though this solution is not the correct syntax, Jack's actions demonstrate his understanding of the instructions and his understanding of the source of incorrect code.

For most exercise six, Jack tries varieties of his first method in different orders and syntax. For example in step three he tries 'if True valA == 7' and then in step eight he tries 'if True: == 7'. By trying different variations of his original answer, he demonstrates that

he understands the instructions' expected outcome. Jack's cognitive effort to understand the problem was minimal, but the proper syntax when a Boolean keyword is present is unknown. Because Jack is unfamiliar with the syntax represented in the code editor, he keeps the Boolean and adds his expression to the syntax.

At the end of the question, the researcher asked Jack if the false keyword made the question difficult. Jack responds with an explanation that he expected exercise six to be a part of a typical pattern in syntax by saying, "I just figured that it was like that's another way you can command it to do that". Jack is referring to the way Phanon introduces variations of the syntax that have a similar outcome. Learning syntax often involves learning different variations of writing code. For example, 'for i in range(4):' and 'for i in range(0,4):' returns the same values but are written slightly differently. Jack believed that the Boolean in the expression was an optional addition to the expression similar to the zero in 'for i in range(0,4):'.

Sarah's approach to this exercise was very different. Her first three steps were sense-making and drawing on knowledge resources steps. She starts by following the instructions with her mouse and confidently says, "So now they want me to write it myself." She looks at the code and then quickly navigates to the previous problem and says, "see back here it said value..". This is a clue that might indicate that she is looking at the code in the previous question where the expression 'valA $<$ 7' is presented. When reading the instructions, Sarah's confidence demonstrates that she understood the problem and supports a minimal extraneous load. Her quick reference to a previous question after reading the code indicates that she saw syntax that she was not familiar with and then used her resources for reference.

Sarah's first action for modifying the code is to remove the False as she says, "I don't think the false is supposed to be there". Her guess was very close to the correct solution, but she made other syntax mistakes on the first attempt, which caused the test to fail. She quickly puts the False back, and her next moves display a pattern similar to Jack's.

She attempts a combination of the Boolean and the desired expression before engaging in sense-making and drawing on knowledge moves. Sarah displays confidence when she returns to question six and explains that she was initially correct and that the false should not be there. Cognitively this exercise was not strenuous for Sarah. Sarah knows bits and pieces of the correct syntax, and her understanding of relationships between those bits of knowledge grew once she completed the exercise.

Matt's temporal decomposition was used to show the perspective of someone experienced with Python syntax and completed the question on the first attempt. Matt's ability to solve the exercise quickly suggests that even though the extraneous load is minimal on this exercise, there may be a way to reduce it even more for individuals like Jack and Sarah. Experience with syntax and programming seems to be a common characteristic for those that solve this problem quickly. Increasing the user's familiarity with how Boolean keywords interact with conditional expressions before exercise six may be a valuable way to improve the conditionals lesson.

Even though the students took more time than expected to complete exercise six, several indicators demonstrate a low extraneous load. A sign that students are having difficulty understanding the material represented is their frequency or time spent looking at instructions. Sarah and Jack spent most of their time solving this exercise, referencing previous questions. This shows that they understand the inherent concept, but they learn the correct syntax through trial and error.

5.2.2 Lesson 3.5.4 Exercise 5

The purpose of exercise five is to give students experience with how the two for-loops in a nested for-loop interact. Asking the student to change parts of the print statement can expose the fundamentals of nested for-loops. The instructions for this problem are listed below.

```
Change the code to output
i = 0
  j: 0
  j: 1
  j: 2
i = 1
  j: 0
  j: 1
  j: 2
done!
```

The code given to the student in the code editor shows the following.

```
1 for i in range(2):
2     print("i = " + str(i))
3     for j in range(3):
4         print(" j = " + str(j))
5 print("done!")
```

The solution to exercise five is to change the equal sign in the second print statement to a colon and remove one space. The steps to solve this problem are very simple and meant to be quick, but the average duration for the participants in this exercise was 1:10, with the maximum duration at 2:46. The participants struggled with this simply because they could not tell the difference between the code to output and the text editor's code. Switching the equal sign in the second print statement to the colon is a small change that the students had a hard time spotting. This indicates that this exercise's efficiency is based on the students' attention to detail and observation skills. This exercise gets presented to the student in a way that causes unnecessary struggle and demonstrates an extraneous cognitive load.

The extraneous cognitive load demonstrated in exercise five is a deviation from the routine exercise. The rest of the exercises that were observed had minimal if any extraneous cognitive load. This confirms that Phanon is an excellent resource for removing the extraneous load from learning syntax.

5.3 Precision and Speed

The participant's ability to create syntactically correct code quickly got better as students progressed through the lessons. As the students were introduced to new variations and parameters when writing code, they got more accurate, faster, and better at spotting bugs. Comparing exercise eight and exercise ten is an excellent example of two creation problems that show significant improvements in some participants.

Exercise eight, see figure 3.11, asks the student to write the inner for-loop in a nested for-loop. The outer for-loop is already present in the code editor on lines one and two. The students have already completed two lessons about for-loops before the lesson on nested for-loops. The students should be familiar with writing a for-loop, but the complexity of nested for-loops can cause confusion and errors when creating the code.

Scott, Trevor, and Brooke were the participants that spent the most time and engaged in the most debugging moves on exercise eight. Figures 5.1, 5.2, and 5.3 show the moves in time-span form with the transition from exercise eight to exercise ten. Exercise ten, see figure 3.12, asks the students to write the inner and outer loop of a nested for-loop. This is double the amount of code to write, and possibly more than double the effort from exercise eight. Despite the increase in code and effort, Scott, Trevor, and Brooke improved significantly in duration and debugging moves. One exercise between exercise eight and exercise ten asks the student to write the outer for-loop of a nested for-loop.

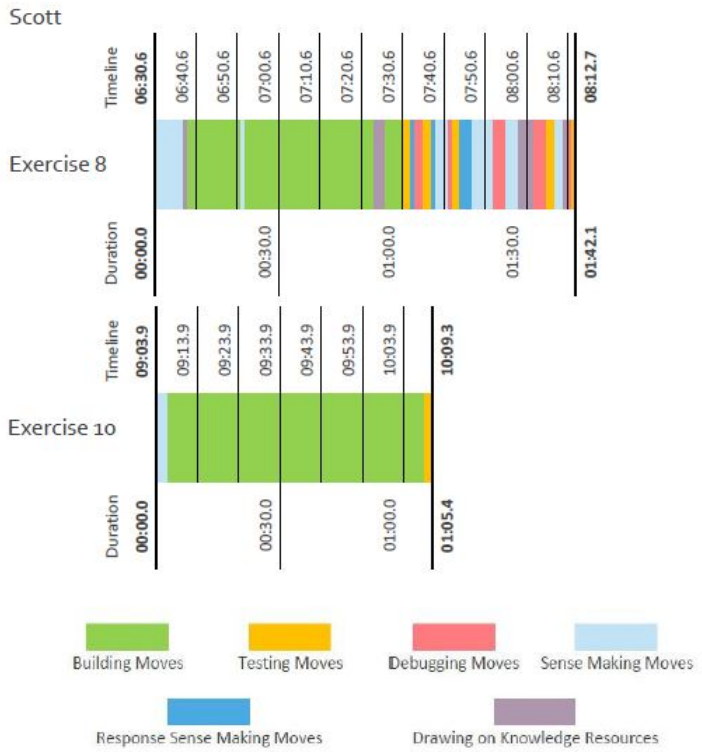

Fig. 5.1: Scott Exercise Eight Compared to Exercise Ten

Fig. 5.2: Trevor Exercise Eight Compared to Exercise Ten

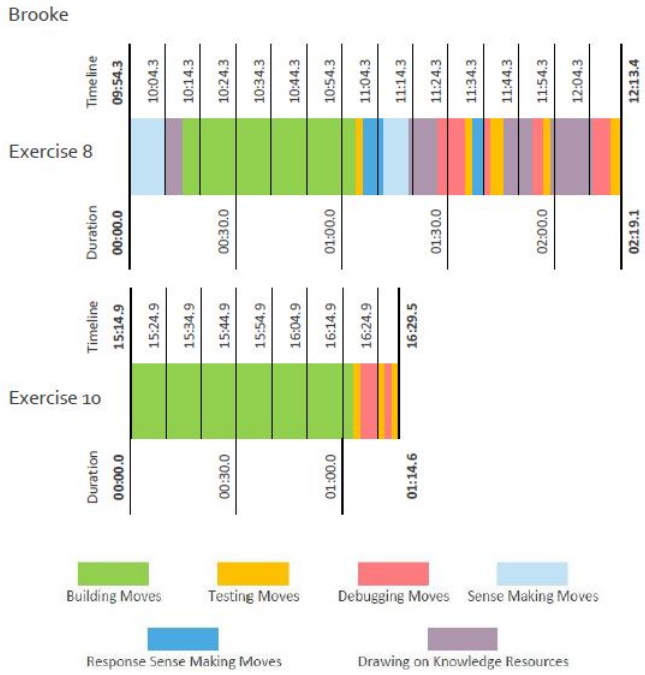

Fig. 5.3: Brooke Exercise Eight Compared to Exercise Ten

The improvements from exercise eight to exercise ten demonstrate the effectiveness of structured practice in developing procedural memory. As the student continues to practice writing code, some kinesthetic habits start to form, and they become faster and more accurate. As seen in figures 5.1 and 5.2 Scott and Trevor engaged in some sense making moves before creating the code and were successful on the first instance of testing moves. A pedagogical design of exercises 8, 9, and 10 are to write the same code several times and then generalize. Since the exercises are repeating the same code, we need to take care of concluding the effectiveness.

As seen in figure 5.3 Brooke engaged in some debugging moves in exercise ten, but she did not engage in any sense-making, response sense-making, or drawing on knowledge resource moves. This indicates that her exercise eight and nine experience helped her improve her ability to spot the flaws in her code quickly without using other resources.

5.4 Referencing Previous Questions

One common practice that the participants engaged in when debugging a syntax exercise is referencing the previous question. Referencing a previous question was categorized as a drawing on knowledge resources type of move. A common concern among introductory computer science courses, like CS1, is how often students use outside resources or previous code to complete their homework assignments. This pattern of utilizing previous questions could mean that students reference previous code in their homework assignments more than previously thought.

Table 5.1 shows the type of codes used with exercise six and how many times all participants referenced them.

3.2.1 Exercise 6 Code References	
Code	References
Create Code	13
Instructions	37
Modify Code	56
Mulligan	2
Output	5
Read Code	37
Reference Previous	77
Revert Exercise	6
Run Code	56
Test Failed	43
Test Results	1

Table 5.1: 3.2.1 Exercise 6 Code References

For exercise six, referencing a previous question was the most used code with 77 references. Each time a participant clicked on a previous question, the code was recorded, so if Jack clicked on question five and then question four, that would count as two codes. The

high number of referencing previous questions could also indicate that students spent time looking through the previous questions to find the syntax that could help them answer the question.

In figure 5.1 and 5.3 the purple sections indicate the time that Scott and Brooke spent looking at previous questions in exercise eight. In exercise ten, neither of them referenced previous questions, and Brooke only engaged in a few debugging moves. Considering the amount of improvement from exercises eight and ten for Scott and Brooke, a theory can be made that referencing previous questions is not detrimental to progress. The participant's ability to access other syntactic code examples most likely helps them get better at recognizing and using correct code.

5.5 Debugging and Grades

This study has provided evidence that the amount of time a student spends on syntax exercises does not relate to the grade they receive in the course. Jack, who is referenced in the temporal decomposition breakdown, was one of the students that struggled the most with exercise six. Matt is also referenced in the temporal decomposition breakdown for exercise six and performed the best in that exercise. When it comes to academic performance in the course, Jack outperformed Matt in each scoring category, reference table 4.5.

Students that engage in debugging moves often while completing Phanon exercises are not struggling in the course. This could indicate that students who engage in debugging are often getting more value from Phanon than those who are not. Phanon provides a safe and comfortable environment that allows the students to debug syntax problems in a modular way.

Phanon provides the ability for students to master the syntax, which gives them even footing to be judged based on their ability to solve problems. Once the effort in syntax is

removed, the student's assignments and exams are more of a reflection of their performance in the course [19].

5.6 Overall Affective State

Throughout the study, there were several verbal and physical indicators that the participants enjoyed Phanon syntax exercises. In Jack's temporal decomposition of exercise six, he figures out the solution, and he laughs and shakes his head. This behavior indicates that even if the student struggles while using Phanon, there is some comfort level that allows them to laugh at their mistakes. The controlled environment that Phanon provides allows the user to have fun because they are never truly stuck. The mulligan button, hints, and simplicity of the exercises give them a place to explore the fundamentals of programming without the stress and complexity present in assignments or projects. Below are some comments made during the think-aloud sessions that may represent the participant's state of mind or opinions of Phanon. Some comments were negative, but most of them seem to reflect a favorable opinion of the program or their circumstances.

Lesson 3.5.2 at 04:34

Hailey: "And I think that'll work. Yay!"

Lesson 3.2.1 at 05:59

Kristy: "I like these exercises a lot with the readings because they are so like, they're the little details. They get so frustrating when you're trying to do the program."

Lesson 3.2.1 at 09:14

Kristy: "I like this because it's teaching me the vocab. I think I have a little programming background. Well, I have a bigger programming background like 10 years ago when I was in school, and then I never used it. So this is helpful because I think the biggest thing I struggle with is the vocabulary."

Lesson 3.5.4 at 08:39

Brooke: "Great. I know, this is so simple. But now that I figured out I'm like, Oh, I'm the smartest person ever."

Lesson 3.2.1 at 02:06

Sam: "I really don't like these kinds of programming exercises. Because unless you do the code exactly the way they want you to do the code. It doesn't work, right? Even if you even if you get the desired output."

Lesson 3.5.4 at 11:17

Sam: "So that's okay, cool."

Lesson 3.2.1 at 04:26

Blake: "There we go. Cool. "

Lesson 3.5.4 at 06:28

Blake: "Nice "

Lesson 3.5.4 at 07:57

Kyle: "This is the kind of time where I wish I could just copy paste because I could just you know write the first for loop, copy paste it change two numbers"

Lesson 3.5.2 at 08:29

Kyle: "Sometimes it's just fun to see if the program will yell at me for you know being slightly different. It's half the fun in actual coding is making your variable silly names."

Lesson 3.5.4 at 09:19

Kyle: "This was fun."

These comments and the frequency of positive comments to negative comments are very similar to the responses received by Sullivan et al. [8] in their assessment of student's attitudes toward Phanon. In the survey results, 86% of students said that Phanon was helpful, 84% said they liked the exercises, and 8% found enjoyment throughout the exercises. Similar to Sam's comment about the exercises needing the exact format for acceptance, 5% of the students in Sullivan's survey found the exercises to be annoying, and 5% of the students said they did not like the exercises. The survey was a free response questionnaire so the results are based on the student's open-ended comments.

5.7 Threats to validity

There are several threats to validity that the reader should consider when discussing these results. One of those is the distribution of students. A variety of gender, programming experience, and academic abilities was desirable for this study but not a simple task. It was not easy to get women to participate in the study, even though many showed interest by completing the initial screen. Once invited to participate, several women told the researcher that they were no longer interested in participating or dropping the course. Before the study, all of the women from the course were invited by the researcher to participate. Only four women remained participants for all three sessions.

Another concern about the participants is that they were selected from the seven-week introductory computer science course. The results may not be able to be generalized to students in a regular fourteen-week course. Typically, the students who knowingly register for the shorter courses understand that there is a more significant amount of work in a shorter time. As a result, those individuals may be more determined and prepared for the workload of the course. Another perspective is that the students in the seven-week course may be disadvantaged since their work is expected to be complete in a shorter time frame. The lower grades of the participants we evaluated may be due to the inability to keep up with a demanding shortened course.

The semester that the researcher conducted the study may also have been a threat to our data's validity. The study was conducted in the 2020 fall semester in the middle of the COVID-19 pandemic. In think-aloud studies, the researcher generally conducts sessions in person [7]. However, that was not an option for this study. The study conducted through Zoom may be a less personal and more uncomfortable situation for the participants or the researcher. The other side to that is that it may have made the participants and researcher more comfortable since they could complete the study in a location of their choosing without the stress of transportation.

Researcher bias toward the encoding of actions should also be considered when reading the results. Future work of this thesis consists of recruiting other researchers to code the data. Some of the actions and moves that were coded were based on a judgment call from the researcher. Determining the validity of those judgment calls is a necessary step in moving forward with this research.

CHAPTER 6

CONCLUSION

Phanon has been adopted into the introductory computer science course curriculum for the success observed in several studies. This thesis discusses a qualitative analysis of the student's interaction with Phanon and the behaviors observed. The researcher observed students in the introductory course while they completed three lessons of syntax exercises. This thesis also discusses the observed patterns of minimal extraneous cognitive load and discovery through repetition.

Due to limitations with the amount of time it takes to code the exercises, the researcher and PI selected eleven exercises for analysis. There is a potential for more discovery of patterns and insight in the exercises that were not analyzed. Analysis of the effect the syntax exercises have on the student's performance when completing homework assignments could validate many of the theories discussed in this thesis. The most significant impact of mastering syntax is most likely in the student's ability to apply it to their assignments.

Through a qualitative analysis of the study in this thesis the following research question is addressed below.

RQ1: What patterns in student interactions with syntax exercises are observable?

Patterns of intrinsic load compared to extraneous load were observed and examined throughout the study. The researcher determined that extraneous cognitive load is minimal throughout the exercises providing an ideal learning environment for syntax. In an exercise where students took more extended amounts of time, a temporal decomposition method was used to evaluate whether there was an issue with extraneous cognitive load. The exercise was determined to be high in intrinsic load and low in extraneous load. This comparison means that there is an acceptable amount of necessary struggle in some exercises.

Through a qualitative analysis of the study in this thesis the following research question is addressed below.

RQ2: How do these behaviors affect student performance in learning computer syntax?

The students reinforce speed and precision they progress through lessons. The observation of improvement was especially evident in iterations of creation exercises. Throughout the lesson, several students decreased the amount of time it took them to complete the exercise and increased their accuracy when creating syntactically correct code. There was also an increase in the students' ability to spot flaws in their code or debug exercises quickly.

Previous exercises are a frequently used resource for students when they are doing syntax exercises. Students looked at previous exercises throughout the study when they were debugging and as a quick reminder for syntax. This resource is valuable for the students because it gives them quick exposure to syntactically correct code. Referring to previous exercises did not appear to have any adverse effect on learning outcomes.

Mastering syntax gives students an even footing in the course to evaluate their ability to solve problems. Each student demonstrated different patterns of debugging, with some being more efficient than others. There is evidence to suggest that the amount of time debugging does not correlate with the student's performance in the class. Several students that spent much time debugging syntax exercises were high performing students and received A letter grades in the course. Their performance in assignments and exams were also high. From this, it can be determined that debugging in Phanon does not mean that the student is struggling. Learning syntax is a separate process from problem-solving, and mastering it is essential and easy to implement.

Further evidence was discovered in the study that suggests the students generally like Phanon syntax exercises. Confidence and enjoyment are tremendous contributing factors to students' success in computer science. Throughout the study, several students mentioned that they liked the exercises or expressed joy as they solved a problem.

Phanon demonstrates an environment ideal for students to debug in modular ways and master syntax. The students' practice as they complete the syntax exercises is an essential step in removing the extraneous load from a difficult subject. By mastering syntax, the student can use their cognitive resources for essential skills like problem-solving. The evaluations in this study can improve some of the exercises in Phanon while also recognizing the critical role it plays in the CS1 course. As shown in [5,6,8], Phanon is an effective resource in helping students learn syntax and, as a result, scaffolding their problem-solving.

REFERENCES

[1] A. Z. C. B. John Edwards, Arto Leinonen and A. Hellas, "Programming versus natural language: on the effect of context on typing in cs," 2020.

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

[3] T. Jenkins, "On the difficulty of learning to program," 08 2002, pp. 53–58.

[4] A. Luxton-Reilly, "Learning to program is easy," 07 2016, pp. 284–289.

[5] J. Edwards, J. Ditton, D. Trninic, H. Swanson, S. Sullivan, and C. Mano, "Syntax exercises in CS1," in *Proceedings of the 16th Annual Conference on International Computing Education Research*, ser. ICER '20, 2020.

[6] J. H. J. V. D. B. John Edwards, Erika Fulton and K. Parker, "Separation of syntax and problem solving in introductory computer programming." IEEE, 2018.

[7] K. A. Ericsson and H. A. Simon, "Protocol analysis," *A companion to cognitive science*, vol. 14, pp. 425–432, 1998.

[8] S. B. Sullivan, "An analysis of syntax exercises on the performance of cs1 students," 2020.

[9] D. Trninic, "Instruction, repetition, discovery: Restoring the historical educational role of practice," *Instructional Science*, vol. 46, no. 1, pp. 133–153, 2018.

[10] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognitive psychology*, vol. 19, no. 3, pp. 295–341, 1987.

[11] I. R. Katz and J. R. Anderson, "Debugging: An analysis of bug-location strategies," *Human-Computer Interaction*, vol. 3, no. 4, pp. 351–399, 1987.

[12] J. Prather, R. Pettit, K. McMurry, A. Peters, J. Homer, and M. Cohen, "Metacognitive difficulties faced by novice programmers in automated assessment tools," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 41–50.

[13] J. Prather, R. Pettit, B. A. Becker, P. Denny, D. Loksa, A. Peters, Z. Albrecht, and K. Masci, "First things first: Providing metacognitive scaffolding for interpreting problem prompts," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 531–537.

[14] 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*, ser. ACE'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 105–113. [Online]. Available: https://doi.org/10.1145/3373165.3373177

[15] 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*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 84–90. [Online]. Available: https://doi.org/10.1145/3287324.3287378

[16] K. Charmaz and L. L. Belgrave, "Grounded theory," *The Blackwell encyclopedia of sociology*, 2007.

[17] M. K. . S. B. W. U. Swanson, H., "Characterizing student theory building in the context of block-based agent-based modeling microworlds." ser. International Society of the Learning Sciences, 2020.

[18] J. Edwards, J. Ditton, D. Trninic, H. Swanson, S. Sullivan, and C. Mano, "Syntax exercises in cs1," in *Proceedings of the 2020 ACM Conference on International Computing Education Research*, ser. ICER '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 216–226. [Online]. Available: https://doi.org/10.1145/3372782.3406259

[19] R. Lister, "Computing education research programming, syntax and cognitive load," *ACM Inroads*, vol. 2, no. 2, pp. 21–22, 2011.