Winter 10-19-2020

INVESTIGATING FACTORS PREDICTING EFFECTIVE LEARNING IN A CS PROFESSIONAL DEVELOPMENT PROGRAM FOR K-12 TEACHERS

Patrick Morrow
University of Nebraska-Lincoln, pmorrow@huskers.unl.edu

INVESTIGATING FACTORS PREDICTING EFFECTIVE LEARNING IN A CS PROFESSIONAL DEVELOPMENT PROGRAM FOR K-12 TEACHERS

by

Patrick Morrow

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Leen-Kiat Soh

Lincoln, Nebraska

June 2020

INVESTIGATING FACTORS PREDICTING EFFECTIVE LEARNING IN A CS PROFESSIONAL DEVELOPMENT PROGRAM FOR K-8 TEACHERS

Patrick Morrow, MS

University of Nebraska, 2020

Advisor: Leen-Kiat Soh

The demand for K-12 Computer Science (CS) education is growing and there is not an adequate number of educators to match the demand. Comprehensive research was carried out to investigate and understand the influence of a summer two-week professional development (PD) program on teachers' CS content and pedagogical knowledge, their confidence in such knowledge, their interest in and perceived value of CS, and the factors influencing such impacts. Two courses designed to train K-12 teachers to teach CS, focusing on both concepts and pedagogy skills were taught over two separate summers to two separate cohorts of teachers. Statistical and SWOT analyses were then performed using measures such as attitudinal surveys and knowledge assessments. Findings showed the PD program had a significant impact on the teachers, there was a positive correlation between teachers' pre-program confidence and knowledge, and additional insights on how to deliver such PD programs more effectively. Results will help inform K-12 CS PD program design.

ACKNOWLEDGEMENTS

First, I would like to thank everyone involved in the AIR@NE project. I would like to thank Dr. Gwen Nugent and Keting Chen for their work designing the knowledge tests and surveys and collecting the data. I would also like to thank them, Dr. Wendy Smith, Dr. Guy Tranin, Susan Prabulos, and Alan Holdorf as they were key contributors to the design and facilitation of the professional development program. Their hard work and dedication helped progress my research to the point it is at today.

Second, I would like to thank Emma Hubka for working with me on much of the data analysis for my work. Emma was always enthusiastic to help and provided insightful reflections throughout the data analysis project.

Third, I would like to thank Dr. Gwen Nugent, Dr. Witiwas Srisa-an, and Dr. Leen-Kiat Soh for serving on my thesis defense committee. I would especially like to thank my advisor, Dr. Leen-Kiat Soh, for his expert support throughout the research process. His guidance and unwavering support helped me throughout the research process. Dr. Soh taught me many lessons about research design, academic writing, and many other personal and professional tips to lead a happy, successful life. I can not thank him enough for the time and effort he has expended to helped me and my research progress to the place it is today. Dr. Soh always made himself available for feedback and suggestions on my work.

Lastly, I would like to thank my friends and family for supporting me over the two years of graduate school. Much of my time has been spent working on school work

over the last two years and they have been loving and understanding throughout my research process.

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1 Problem

The need for K-12 computer science (CS) instruction has become of great importance throughout the world as more and more career paths rely heavily on digital competency. Because of this, a gap exists in the availability of quality K-12 CS in-service K-12 teachers. We can address this gap by providing pre-service and in-service teachers with quality CS training through CS professional development (PD) programs. This research focuses on preparing the in-service teachers by evaluating our PD programs held in two consecutive summers with two separate cohorts of K-12 CS teachers. Our study aims to improve K-12 CS instruction by identifying what makes our in-service K-12 teachers learn CS effectively in our two-week CS PD program. The findings presented in this paper will aid future PD program designers by understanding how PD program designers should teach and how to evaluate the program to gain a useful insight into the program's effectiveness. PD designers will make specific adjustments to any PD program given different participant characteristics, such as grade level of instruction, experience level with computer science, and resources available in time and technology. This paper's findings will also help designers make those adjustments to cater to any PD program around the participants' needs. Overall, this research strives to improve the quality of instruction and students' access to a CS education at the K-12 level.

1.2 Motivation

In recent years there has been a push for an increase in Computer Science (CS) education as the number of CS jobs rises. A study by the Bureau of Labor Statistics

shows that 58% of all new STEM jobs are in computing, and 10% of STEM graduates are majoring in CS. This study identifies a significant disconnect between the requirements of the workforce and the ability of the education system to prepare students to meet those requirements. The desire to produce more CS majors is a view that is not unique to just industry leaders. A 2016 Gallup survey showed that 90% of parents want their child to learn CS (Google & Gallup, 2016). A more recent study by Gallup showed that 45% of high schools teach CS across 39 states (2019 State of Computer Science Education, 2019). The demand for CS curriculum in K-12 has exposed a substantial deficiency in the number of trained K-12 CS teachers, and in many states, there is no required training for teaching computing courses (Lang et al., 2013). The lack of participation in CS and the lack of trained CS educators at the K-12 level desperately needs to be addressed.

1.3 Gaps in Literature

Numerous projects have attempted to address the low levels of CS participation by offering different K-12 teacher professional development (PD) institutes or workshops. The primary focus of these workshops is to teach CS pedagogical knowledge and CS content knowledge to teachers. Typically, PD programs are unable to specialize in both areas due to their short duration to accommodate teachers' busy summer schedules. The workshops that heavily emphasized CS content knowledge left teachers lacking the ability to integrate the new content into their classrooms (Ericson et al., 2005; Neutens and Wyffels, 2018). The workshops that focused on CS pedagogy knowledge and available technology excited teachers to teach CS but left them with sparse confidence to

teach their students and a limited content base (McGee et al., 2019). These trends were clear in Chai et al.'s study of the factor technological, pedagogical, and content knowledge (TPACK) plays in helping new K-12 CS teachers succeed in integrating CS curricula in their classrooms (Chai et al., 2010). While all three components are essential PD programs components, Chai et al.'s study noted that the focus of the PD program needs to change based on the skills of the teachers in the program. Chai et al. identified pedagogical knowledge as a good starting point for pre-service teachers, while content knowledge is essential for in-service teachers (Chai et al., 2010). With more K-12 schools teaching CS, these studies have set the stage for new and exciting research in the field of PD for new CS teachers.

1.4 Proposed Study

Much of the research in the CS PD area strives to find the most effective strategy for delivering PD workshops and how the workshops can be adapted to prepare K-8 CS teachers better. Designing a one-size-fits-all PD workshop is difficult. However, understanding the traits and behaviors of the teachers could benefit CS PD designers in tailoring PD workshops. This research aims to improve understanding of in-service CS teachers, their strengths, their weaknesses, and their aptitude for learning CS, and how such characteristics manifest in observable behaviors in PD courses.

This research aims to measure and identify traits, behaviors, and motivations of K-8 teachers participating in a two-week CS PD program. As a critical step towards improving K-8 CS education, we hope to find traits, behaviors, and motivations that help predict course success as measured by CS content understanding. Understanding these

predictors will allow facilitators to provide timely interventions in future CS PD programs. For a teacher to be successful in a PD program, they need to improve their CS and CT content knowledge to a point where they feel confident enough to teach it. To strengthen their CS and CT knowledge, they need to be motivated and engaged throughout the PD program. The program designers adapt the program design as necessary to cater to the strengths and weaknesses of the group. If the facilitators determined that a group of teachers are not likely to succeed in the program, then the designers can make changes to address the issues hindering the teachers on a failing path, which will lead to better prepared CS instructors.

This study focuses on the following three research questions:

1. What was the impact of the CS summer PD on the teachers?
    a. knowledge of CS concepts
    b. knowledge of computational thinking
    c. CS attitudes
    d. confidence in CS knowledge
    e. confidence in teaching CS
2. What were the differences between teachers from a model school district (an urban school district with extensive CS curricular development and teacher PD) and teachers from other school districts? How did the program impacts differ?
3. Which factors lead to teacher success (e.g., knowledge test scores) in terms of CS understanding in the summer PD program? Specifically, this study investigates *confidence in CS content*, *plans to teach CS in the following*

AY, and *grade level of instruction* as potential predictors of teacher performance.

1.5 Contributions

Some significant findings have come from the two CS PD programs covered in Chapters 3 and 4. First, the analysis showed that both programs were successful in significantly improving the participants' CS knowledge test scores, CT knowledge test scores, CS confidence, and CS teaching confidence. The findings from Cohort 1 also showed that teachers with more experience in teaching CS had more confidence in CS than teachers with less experience, even though the two groups had similar knowledge test scores. Our assessment also showed no significant correlation between the grade level of instruction or the participants' plans to teach CS in the next academic year and their knowledge test scores. Lastly, the program evaluation showed that for Cohort 1, confidence in CS concepts had a strong correlation with the post-program knowledge test scores, but in Cohort 2, this did not hold.

During the process of designing the two PD programs, we also developed several course materials that will be helpful for other PD program designers to use. In this paper, we share resources from each of our two-week PD programs including the schedules (first cohort schedule: Figure 3.1, Figure 3.2, second cohort schedule: Figure 4.1, Figure 4.2), syllabuses, quizzes (Appendix A), homework assignments (Appendix B) and the adjustments we made to each of those items as we needed during the program and between programs. Our cohort participants' CS experience guided the development of these materials. From Cohort 1 to Cohort 2, we made several changes since the

participants' background and skills were slightly different in the two cohorts. These adjustments are essential for optimizing the effectiveness of each PD program.

1.6 Overview

First, in Chapter 2, the Related Work section discusses several CS PD programs and their effect on the K-12 CS education community (Section 2.1). In this section, we also investigate two key questions that will help guide future CS PD design (Sections 2.2 and 2.3). In the next chapter, Chapter 3, we discuss the details of the first cohort, two-week summer PD program delivery. Specifically, this chapter includes information on the *Program Structure* (Section 3.1), the *Data Analysis* (Section 3.2), the *Results* (Section 3.3), and the *Program Evaluation* (Section 3.4). The *Program Structure* section discusses the logistics of the program. The *Data Analysis* section describes the process of collecting the data and how it was analyzed. The *Results* section looks at the impacts of the CS PD program, the outcomes of the different teacher groups, and factors driving performance. The *Program Evaluation* section further complements the findings in the *Results* section with details about the nuances of delivering a CS PD program and insights learned. The next chapter is about the two-week, CS PD program for our second cohort of teachers. Chapter 4 is set up identically to Chapter 3 -- *Program Structure* (Section 4.1), the *Data Analysis* section (Section 4.2), the *Results* section (Section 4.3), and the *Program Evaluation* section (Section 4.4). Chapter 5 discusses the key differences between Cohort 1 and Cohort 2, both in terms of setup (Section 5.1) and outcomes (Section 5.2). Finally, the Conclusion includes a *Summary of Findings* (Section 6.1) of the two PD programs,

Recommendations (Section 6.2), and *Future Work* (Section 6.3) to come from these programs

Chapter 2: Related Work

The related work section contains three parts. First, we will discuss some PD programs in general to find some common themes. Next, we will look at two central questions through reviews of several programs. The first question is, "*To what extent should CS content be part of CS PD programs?*". PD programs must link CS concepts with CS pedagogy concepts, so we want to understand how different PD programs balance the CS concepts and the CS pedagogy in their programs. The second question is, "*Is text-based programming imperative when teaching CS teachers how to teach CS?*". The motivation behind this question is that we saw many teachers struggle with the programming side but expressed confidence in the concepts themselves. We weigh the importance of using text-based programming languages in CS PD by comparing PD programs that use text-based programming with programs that use visual programming languages instead.

2.1 General Review

Through the CS for All (Fancsali et al., 2018; Salac et al., 2019; Vogel et al., 2017) and CS10K (Brown & Briggs, 2015; Yadav et al., 2013) initiatives, there has been an increased call for CS participation in K-12. Qualified CS teachers are vital to integrating CS into K-12. There have been many efforts to develop PD programs that effectively prepare current teachers to teach CS. Teachers are still going into their classrooms unprepared to teach CS. Ericson et al. found such deficiencies in two of their CS PD workshops (Ericson et al., 2005). The first workshop was for teachers with little or no CS teaching experience, and the second was for teachers of a CS-AP high school course.

After the first course, 70.37% of teachers felt more capable in programming, 96.03% had a better idea of what to teach, and 88.89% got a better idea of how to teach CS. However, only 44.44% of the teachers felt ready to teach CS. Of the 17 teachers from the CS-AP workshop, 94.12% reported feeling more capable in programming, 88.24% has a better idea of what to teach, and 94.12% had a better idea of how to teach CS. 76.47% of the teachers felt ready to teach CS in the next school year. Overall, in their summer PD workshop for CS teachers, they found, post-workshop, that 56.82% of the teachers felt ready to teach CS in the next semester (Ericson et al., 2005). Even with an increase in programming and pedagogy knowledge, many teachers are still preparing to teach students with little confidence (e.g., 44.44%) in their ability to do so. Ericson et al. also found that 29% of all teachers wanted the workshop to go at a slower pace. Going forward, they believe creating a program that caters to the new introductory CS teachers who show signs of needing a slower pace before the class would improve their PD program (Ericson et al., 2005).

Research has identified ways to increase self-efficacy and use of computers in classrooms. Hatlevik et al. found there was a strong positive correlation between the amount of home computer use and ICT self-efficacy, which is vital to learning CS and learning to teach CS (Hatlevik et al., 2018). Wozney et al. also saw teachers with personal computers and access to "play with" potential classroom tools were more likely to integrate technology in the classroom (Wozney et al., 2006). However, most PD programs (e.g., Ahamed et al., 2010; Morreale et al., 2012) do not explore the differences between teachers with experience teaching CS (or experience using CS tools to teach other subjects) and teachers without CS education backgrounds. The study detailed in this

paper makes such comparisons to provide insight into the relationship between teacher CS experience and their CS knowledge, attitudes, and skills.

Another valuable PD approach is the Exploring Computer Science (ECS) PD program used by McGee et al. The ECS curriculum was designed for teachers to teach students CS through equity, inquiry, and CS concepts. Their curriculum aims to teach CS through real-world examples, such as making games that encourage learning about healthy eating (McGee et al., 2018). The PD program's workshop had five key components. The first two components focus on active learning (Desimone & Garet, 2015), the third focuses on equity in CS education, and the last two concentrate on making the teachers successful in the long term. McGee et al. used an Expectancy-Value-Cost (EVC) survey to measure the attitudes of the ECS students. They compared the EVC survey results to the students' course experience and to a Teaching Quality Index (TQI) based on a combination of two teacher practice quality instruments to measure the teachers' ability to "foster equity, inquiry, and development of CS concepts" (McGee et al., 2018). The students took the survey to determine the teachers' TQI. The authors found the TQI had a direct effect on the students' post-EVC scores, which in turn influences student outcomes. This finding shows that better-equipped teachers are having a direct impact on students' attitudes and their engagement in CS. Additionally, the more experience the teachers had in teaching ECS, the more the students' ECS scores improved from the pre-test to the post-test (McGee et al., 2018). McGee et al.'s method of measuring teacher performance and student learning outcomes could help in creating a universal measure for K-12 CS educators.

2.2 Question 1: To what extent should CS content be part of CS PD programs?

The first question addresses the design of PD programs and how computer science (CS) content delivery can be balanced to avoid overwhelming inexperienced in-service CS teachers while providing them with quality training of CS concepts. The goal of CS PD programs is to prepare current and future CS teachers to teach CS concepts. Program designers use two general approaches to achieve this goal. The first approach is through programming language training, where the teachers learn CS concepts through programming in high-level CS languages. The second approach is through CS unplugged activities. These activities can include CS concepts but focus more on computational thinking (CT) to introduce teachers to CS as CT draws on skills and professional practices that are fundamental to computer science (Sengupta et al., 2013). The CS unplugged approach allows teachers from all CS backgrounds to understand CS concepts without needing to learn a programming language or use any devices (Bell et al., 2012). Both approaches of CS PD programs vary from 1-5 days and can even be more than one week. Each duration raises different challenges and comes with varying program outcomes. Below is a discussion about each program's duration. This review will detail the design of CS PD programs of varying lengths (short, medium, long).

2.2.1 Short PD Programs (1-3 days)

Short PD programs are typically less than one week to accommodate teachers' summer schedules. Some programs are as short as 1-3 days (Morreale et al., 2012; Bower et al., 2017). There is not enough time to cover all CS concepts or CT concepts in-depth

in these programs. The 1-3-day programs have been successful by shifting their focus to training teachers on proven classroom tools and resources to apply to their classrooms right away. This type of program makes sense to improve the preparedness of teachers already equipped with adequate CS backgrounds.

Morreale et al.'s two, one-day workshops helped introduce teachers to CT by providing them sessions on curriculum materials, current university projects, internships, post-grad opportunities, and the importance of CS locally and nationally (Morreale et al., 2012). While Morreale et al. did not discuss why the two workshops were each one day long, the duration makes sense given the goal of the workshop (further PD design details in Table 2.1). Their goal was to (1) introduce new curriculum materials, (2) provide examples of collegiate projects, internship opportunities, and to show what being a CS major in college means, and (3) provide a broader understanding of computer science topics and careers (Morreale et al., 2012). The attendees took a pre- and post-program survey to evaluate their understanding of CS and CT topics. The survey results showed that ~90% of the attendees understood CT (+15% from pre-survey), and 86% understood why CT was necessary (+22% from pre-survey). In the survey, the researchers also asked the teachers which of the eight sessions during the first workshop were most impactful. Of the eight sessions provided during the first workshop, four of the sessions were reported as "immediately useful" by the attendees. This form of PD has successfully introduced the teachers to CT and how different teaching tools can be used (Morreale et al., 2012).

Table 2.1 Details of the PD programs of varying duration in the Related Work section.

		Topics Covered				
Program Designer	**Duration**	**Pedagogy**	**CS Content**	**CT Content**	**Text-based Programming Language**	**Visual Programming Language**
Morreale et al. (2012)	Short	x		x		
Bower et al. (2017)	Short	x		x		
Liu et al. (2014)	Medium	x	x			x
Pollock et al. (2017)	Medium	x	x		x	x
Milliken et al. (2019)	Long	x	x		x*	x*
Goode et al. (2014)	Long	x	x			x

* Participants could choose their language for the course.

Bower et al. also held four separate one-day workshops for 69 teachers of grades K-2, 3-4, 5-6, and 7-8 (Bower et al., 2017). Table 2.1 provides an overview of the program details. A pre- and post-workshop open-ended survey assessed the impact of the PD program. The survey evaluated the change in the teachers' understanding of CT concepts, strategies used to teach CT, technologies used to teach CT, and understanding the teachers' confidence gain from attending the workshop. The survey was analyzed by evaluating the open-ended responses for computation thinking practice, concepts, and perspective keywords. These results showed that the teachers could identify the keywords more effectively (141 keyword references pre-workshop vs. 312 keyword references post-workshop (Bower et al., 2017)). This analysis strategy does *not*, however, give us a deep understanding of the teachers' level of understanding regarding CT concepts. To gain more insights into the comprehension levels of the teachers, the facilitators could have paired a knowledge test with the survey. The most used pedagogy strategy listed by

the teachers was a “student-centered” strategy, which was consistent from pre-workshop to post-workshop. The teachers gained the most insights about technologies used to teach CT. Pre-workshop, only 42% of the teachers listed specific software used in the classroom and post-workshop, 72% of teachers listed teaching software such as Scratch, Visual Basic, Python, Hopscotch, Tynker, and more. The teachers also listed several robotics resources to develop CT skills in the classroom. Bower et al.’s workshop was also successful in significantly improving the teachers’ confidence in teaching CS (Bower et al., 2017). Pre-workshop, the teachers’ most significant obstacle to teaching was their lack of self-efficacy, as found from the pre-workshop survey. That changed post-workshop where most teachers listed “lack of resources” as the most significant obstacle over self-efficacy as well as other reasons. The program was successful in improving the teachers’ self-efficacy in a short amount of time by introducing the teachers to CT and some different tools they can use in the classroom. However, further targeted professional development training workshops were desired by the teachers following the program as well as additional time, resources, and peer mentoring.

From these two short PD programs, we can see significant self-efficacy improvements made in a short amount of time. While this improvement is encouraging, given the growing need for CS teachers, we argue that merely introducing teachers to the CT concepts over a 1-3-day workshop is not enough to prepare teachers for quality CS instruction.

2.2.2 Medium PD Programs (4-5 days)

The programs in the previous section were successful in preparing teachers for CS instruction in a small amount of time by providing resources and understanding of CT concepts. Medium length PD programs should be able to expand on the successes of the short PD programs by going more in-depth. Here we review medium length PD programs held by Liu et al. and Pollock et al. (Liu et al., 2014; Pollock et al., 2017).

Liu et al. used a 5-day game-centered development approach and a drag-and-drop programming language called Stencyl to prepare their teachers (Liu et al., 2014). Table 2.1 contains details about the program. Each of the five days contained two sessions, and each session contained one or two CS concepts. The concepts covered were classes, variables, methods, conditionals, booleans, loops, and lists. In the mornings, the teachers worked on existing Stencyl projects that covered the concept of the day. In the afternoons, the teachers created their curriculum for the concept using Stencyl to take back to their classrooms. Liu et al.'s team saw a 61% increase in concept knowledge (Liu et al., 2014). While the increase in content knowledge was significant, we do not see any analysis of the teachers' preparedness to teach their classrooms using these tools. Liu et al. were successful in building the teachers' understanding of CS concepts, Stencyl, and how to use Stencyl in the classroom. To see whether or not the teachers' will be able to extend what they learned to their classrooms, further evaluation will be needed.

Pollock et al. designed their 4.5-day PD program with a focus on CS content, pedagogical strategies for teaching CS, and strategies for broadening participation in CS (Pollock et al., 2017). The author gave no reasoning for the 4.5-day duration, but given

the focus of the program, this seems to be the minimum amount of time it would take to cover all topics. Table 2.1 provides details of the program. 28 of the 84 program participants also participated in the post-program interviews, 13 were CS teachers, and 19 were STEM teachers (total does not equal 28 because some teachers teach CS AND STEM). Other participants included business teachers, administrators, and librarians. To measure the impact of their PD program, education professionals held interviews with the 28 teachers who had completed at least one week of the PD and had a chance to integrate what they learned into their teaching. All 28 teachers had integrated CS concepts into their classrooms, and 11/28 teachers stated their increased self-efficacy as their greatest success in teaching CS principles post-PD (Pollock et al., 2017). As a result of the program, the teachers who participated in this PD program are better prepared. However, those who had prior programming expertise desired more advanced programming practice, while those without previous experience stated a desire to learn programming to keep up with their students (Pollock et al., 2017).

We saw significant increases in knowledge in both programs, although the two programs had slightly different goals. Pollock et al. focused on connecting CS and CS pedagogy while Liu et al. focused on content knowledge and mastery of a programming language (namely, Stencyl). Pollock et al. identified the goal of their PD program as "improve CS teaching by providing educators with content knowledge of CS and CS principles and helping them develop their pedagogical content knowledge related to CS" (Pollock et al., 2017). Liu et al.'s goal was to introduce CS teachers to CS content knowledge through Stencyl. Since Liu et al. did not evaluate the teachers' preparedness, it is difficult to say which was more successful in preparing teachers to teach (Liu et al.,

2014). One interesting thing to note in the medium-length programs is that *the extended length of the program allows for more creativity in the program design*. The short programs were similar in design, but the medium-length programs used different tools and approaches to CS education preparation.

2.2.3 **Long PD Programs (More than one week)**

With more time and added program flexibility, long PD workshops allow for added depth and breadth of knowledge. There was an increase in variety in the design of PD programs as the programs went from short to medium, so the long PD programs are expected to introduce even more range in goals, instructional strategies, and workshop tools.

Milliken et al. found success with their reworked two-week PD program (details found in Table 2.1). From 2012-2015, they held a 6-week PD program each year. Milliken et al. reduced the program to a three-week program in 2016 and again to a two-week program in 2017 and 2018 (Milliken et al., 2019). Although the program scaled down from six-weeks to two-week, the program remained 50% CS content focus and 50% pedagogy focus. The program focused less on strictly CS content, and more on a *Lead Learner* model where one group of teachers acts as the teachers, and the other groups act as the learners. The *Lead Learner* model helps all teachers participate as both teachers and students throughout the program. To evaluate the effectiveness of the PD program, Milliken et al. used 14 five-point Likert-scale items as part of their post-PD survey (Milliken et al., 2019). Despite reducing the duration of the program, they saw an increase in scores on items that asked about how efficiently the facilitators used their time

and items, asking about the quality of teaching techniques and content of included in the program. This result shows that as PD designers become more experienced about the critical aspects of CS PD, they can transform a 6-week program into a two-week program without damaging the quality of the program. The effectiveness of the *Lead Learner* model shows that "how to teach" is equally valuable as "what to teach." Of the 67 participants of the two-week program who took the post-program survey, 73% planned on adopting the Beauty and Joy of Computing (BJC) curriculum introduced during the program.

Additionally, all responses to questions about teacher preparedness ranged between 3.64 and 4.00 on a five-point Likert scale, which is relatively high. No pre-workshop preparedness survey was discussed in the paper since the paper was ultimately comparing the results of the program over the last three years. The final, two-week program design yielded the highest post-program preparedness scores (Milliken et al., 2019).

Goode et al. found success using the ECS model for PD and curriculum design in their two-year PD program (details found in Table 2.1). In the first year, the authors held a one-week PD program with quarterly follow-up sessions post-program. In year two, the authors held a second one-week program (Goode et al., 2014). Scratch, Lego Mindstorms, and CS Unplugged activities are typically used in ECS classes to deliver concepts of CS without having to spend much time learning a programming language, although no programming language was documented (Goode & Margolis, 2011). The ECS model strives to form long-term relationships with teachers. Darling-Hammond & Richardson found that programs between 30 hours and 100 hours spread over 6-12

months had the most significant positive effect. Darling-Hammond & Richardson also found that teachers who attended 80 or more hours of inquiry-based PD were more likely to adopt inquiry-based teaching strategies in their classrooms than teachers who attend for less than 80 hours (Darling-Hammond & Richardson, 2009). Goode et al. administered an end-of-year survey to understand how much the teachers learned throughout the program. Of the 23 participants who filled out the survey, 91% of participants listed the program as "useful" or "very useful" and all but one teacher found that the ECS PD had "some impact" or a "large impact" on their teaching of CS content, inquiry, and equity (Goode et al., 2014). Written responses to the end-of-year survey also showed strong connections between the curriculum, pedagogy, and equitable teaching practices. While these findings do serve as evidence to show that the 2-year program had a significant impact on the teachers' understanding of CS and CS pedagogy, we could better understand how far the teacher had come with a CS knowledge test. A knowledge test would also allow researchers to compare the results of their PD programs with that of Goode et al.

Frequently, feedback from PD programs shows a need for "more time" to cover topics during the programs. The program designed by Milliken et al. shows that changes can be made to a PD program, aside for increasing the duration, to provide ample time for the teachers to learn the concepts efficiently (Milliken et al., 2019). A high percentage of Goode et al.'s participants found their program to be "useful" and impactful (Goode et al., 2014). These programs both achieved high-levels of teacher preparedness by not only teaching about CS concepts and linking them to the classroom but also teaching the teachers how to deliver a specific curriculum. The two programs discussed in this section

are different in length but provide many of the same opportunities for their participants. With the added length of the program, the designers can follow a specific curriculum that helps the teachers understand what they will need to teach in their classroom and how they will need to teach it.

2.2.4 Conclusion

This review has shown that as the program duration changes, so do the goals and design of the program. Shorter programs are limited to preparing teachers by providing resources and teaching materials to their participants and do not allow enough time for the program designers to cover all or any core CS and CT concepts. Medium length programs could expand on the content introduced in the short programs. The medium-length programs added some CS content knowledge and some links to CS pedagogy as well. Medium length programs can cover CS concepts and CS pedagogy in an expedited fashion (Pollock et al., 2017), or they can focus on mastery of either CS concepts or CS pedagogy (Liu et al., 2014). The long-duration programs reviewed included ample practice on CS concepts but also focused on pedagogy practice as well. The longer durations also allowed for programs to include more information on what and how the teachers can teach in their classrooms, including full curriculums. The programs reviewed here show that there are many different approaches to deliver a CS PD program with varying levels of CS content knowledge. The amount of CS content knowledge covered in each program entirely depends on the length of the program. None of the work reviewed explains why they chose the duration they did. That information would help others trying to replicate their studies.

Additionally, in measuring the participants' progress, each of the programs in this literature review administered attitudinal surveys. While it is beneficial to gather the attitudes of the teachers, the written or verbal responses fail to provide a concrete way to compare the knowledge gained by the teachers. Attitudinal surveys, paired with a CS knowledge test, would be a more effective way also to measure changes in CS content knowledge. A fully validated CS knowledge test, for example, would allow researchers to compare changes in CS content knowledge between different CS PD programs.

2.2.5 **Recommendations**

When designing a PD program, it is necessary first to identify the goals of the program and identify any limitations. Examples of limitations could be program duration, participant background knowledge before the PD program, and school system curriculum restrictions. After reviewing the limitations, the designers can decide on the program structure.

For programs limited to a short program duration (1-3 days), success has been found by merely providing the teachers with materials and tools they can take to their classrooms and use immediately. Neither of the reviewed programs of short duration got into CS concepts in-depth. It seems the teachers would not have enough time to grasp the CS concepts in such a short duration. For that reason, it may be best to refrain from including CS concepts in any depth other than solely introducing the concepts. This duration of the program is better fitted for expanding the knowledge of K-12 CS teachers with solid backgrounds already. If the participants are new to CS and new teaching CS,

the short, 1-3-day workshop will not provide adequate depth of knowledge for the teachers to be appropriately prepared to teach.

For programs of medium length (4-5 days), a focused program goal becomes more critical. Depending on limitations aside from duration, the program can focus on teaching materials, CS concepts, CS pedagogy, or a mix of any two or three of those. For a program focusing on CS concepts, success was found by mixing text-based and visual languages or by avoiding text-based programming languages all-together. Instead, these programs can use drag-and-drop or visual programming languages. The best instructional strategy will likely depend on the goal of the program since time is limited, and only so much can be covered in 4-5 days.

For programs of longer durations, the most appropriate approach seems to be a 50/50 split of CS concepts and CS pedagogy coupled with a specific CS curriculum. The longer the program is, the more opportunities the program designers will have to follow-up the teachers participating in the program and steer them towards better CS instruction. However, Milliken et al. proved that their 6-week program was improved by shortening it to two-weeks, so merely making a program longer will not necessarily make the program more impactful (Milliken et al., 2019).

Finally, for programs of all lengths, it is necessary to provide some sort of support for the teachers throughout their journey of implementing CS in their classrooms. The in-program preparation can only take the teachers so far, and questions will inevitably arise as the teachers begin implementing the learned materials into their classrooms. Bower et al. found their participants indicated the need for “peer mentoring networks,” and Pollock

et al.'s participants expressed a need for collaboration and communication amongst peers (Bower et al., 2017; Pollock et al., 2017). The long-term projects by Milliken et al. and Goode et al. have this long-term facilitator/participant relationship embedded as part of their program (Milliken et al., 2019; Goode et al., 2014). A support-network post-program is a theme throughout successful professional development programs. It is often noted as a strongly recommended piece to add for any PD programs which does not have one set up. Another recommendation would be for each researcher to identify the reason behind the duration of their program, whether that be logistical or financial limitations, or if the duration was set because the designers were comfortable covering all concepts in the given time. With this information provided, other researchers can better reproduce the findings in these papers and better extend their programs from these successful programs. Lastly, the evaluation of each of these programs could be improved by adding a pre- and post-workshop knowledge test. With the knowledge test, it is easier to compare the results of the programs from year-to-year and compare with programs hosted by other research groups, and such comparison could complement attitudinal surveys well and provide additional insights. Each of these recommendations will help researchers to revise their PD program and to prepare quality CS teachers in the future better.

2.3 Question 2: Is programming imperative when teaching CS teachers how to teach CS?

The second question also addresses the design of PD programs, but this focuses on programs that incorporate CS concepts in different ways. Several programs incorporate programming languages such as Python, JavaScript, Java, or other high-level languages

to introduce CS concepts. In contrast, others use more CS-unplugged (no technology needed) approaches paired with visual programming languages such as Blockly, Scratch, or other visual programming languages. The programs reviewed in this section will help us understand the strengths and weaknesses of using text-based programming languages vs. visual programming languages to teach CS concepts to K-12 teachers.

2.3.1 Visual Programming Language Programs

This section discusses programming tools used by programs utilizing visual programming languages, the concepts they cover, and the successes found in the program.

The first program discussed in this section was developed at the University of California, LA, and the University of Oregon and was held by McGee et al. (McGee et al., 2019). The goal of the program is to increase equity in the field of computer science. To achieve this goal, the designers use the Exploring Computer Science (ECS) curriculum. The ECS curriculum uses activities that are designed to make the content "relevant, engaging, and stimulating for a diverse population of students" (McGee et al., 2019). Margolis points out, in her 2010 book, that CS taught as an abstract academic subject privileges access to mostly Caucasian, male students (Margolis, 2010). The ECS curriculum is designed to include a deep engagement of crucial CS concepts and uses the visual programming language, Scratch. This deep engagement is provided through meaningful problem-solving experiences, collaborative learning, and paired programming. The professional development program was designed to embody the same inquiry-based learning activities while also guiding the teachers to build inclusive

classroom culture. The program was a week-long and included five vital components, (1) collaborative inquiry in small groups, (2) inquiry specifically in the teacher-learner-observer model, (3) discussion and reflection about equitable practices, (4) ongoing PD throughout the school year and a second weeklong workshop the following summer, and (5) the formation of a learning community.

To evaluate the participants' ability to teach, McGee et al. distributed pre- and post-tests to the teachers' students. They used The Graide Network teaching assistants[1] to score the pre- and post-tests of the students. The Graide Network recruited and trained 26 undergraduate pre-service teachers to score the performance tasks, and they used the Facts software to conduct Many-Facet Rasch Measurement (MFRM) analysis (McGee et al., 2019). They saw more than 2 points of growth in the students' CT knowledge (11.7 on the pre-test and 13.8 on the post-test). Their second evaluation compared the students' course performance and its correlation with the development of CT after controlling for student characteristics. McGee et al. considered the student characteristics as pre-test scores, grade level, gender, race, special education, free or reduced lunch program status (low-income status), English language learner (ELL), attendance rate, cumulative GPA (only the year which the student completed the ECS curriculum), and the grade received in the ECS course (McGee et al., 2019). After controlling for those characteristics, they analyzed the correlation with these characteristics and the students' post-test scores. There was no statistical difference in post-test performance by gender, race/ethnicity, or level of family income. There was a negative difference in post-test performance by ELL

[1] The Graide Network finds trained teaching assistants and matches them with the needs of your program to evaluate students work.

and special education students. Their overall GPA, school attendance rate, and performance in the ECS course did show a significant correlation to post-test performance, and they saw a higher number of students achieve competency at post-test than pre-test. *While this does not tell us a lot about the preparedness or knowledge levels of the participants in the PD program, it does tell us the success their students (non-ELL and non-special education) found using the ECS curriculum.* Evaluating the teachers' students is a different way of analyzing the impact of a PD program that is typically paired with teacher-centered pre- and post-tests to gain a better understanding of the PD program impact. A valuable comparison that could then be made is teacher post-program test vs. student post-class test to identify the value of teacher performance in the PD program. More information on the PD program and the participants would have also been beneficial to understand how successful the program was in preparing the CS teachers.

Kong and Lao designed the next program. Kong and Lao focused on enhancing K-12 students' problem-solving ability through CT education. They believed the first step to achieving that is to prepare the K-12 teachers to teach about CT (Kong & Lao, 2019). Their program was implemented in the 2017/2018 academic year to 80 teachers. Of the 80 teachers, 46 were male, and 34 were female. The participants' average years teaching was 11.7 years, and 64 of the teachers had taught computer science or information technology courses. 20 of the 80 teachers held computer science degrees. The program contained two courses, the Teacher Development Course 1 (TDC 1) and Teacher Development Course 2 (TDC 2). Each TDC lasted 39 hours (13 3-hour sessions), and the first TDC must be completed to attend the second TDC. TDC 1 focused on building the teachers' knowledge of CT concepts, practices, and perspectives. At the end of TDC 1,

the teachers developed a mobile app to solve problems like those seen in the classroom. TDC 2 emphasized CT pedagogy and included paired programming, programming activities, and ways to evaluate student work (Kong & Lao, 2019).

Many of the CT concepts, practices, and perspectives from the first course were reviewed in the second course as well. The program used visual languages and pseudocode to deliver their TDC 1 and TDC 2 courses. This decision was made at the recommendation of Brennan and Resnick (Brennan & Resnick, 2012), which they consider to be an effective way to teach CT to beginners. To evaluate the effectiveness of the program, the designers constructed their own, five-question, paper-and-pencil test that provided the teachers with real-life problems and allowed space for pseudocode answers. They provided two test question examples. The first question focused on the teacher's ability to debug and can be found in Figure 2.1 (Kong & Lao, 2019). The second example was question four on the test and evaluated the participants' ability to abstract and algorithmically think (Kong & Lao, 2019). This question can be found in Figure 2.2.

1. There are two factories (#1 and #2) in a wheat company. #1 is used to process hard wheat, while #2 is used to process soft wheat. Both factories operate with a standard procedure. To control the quality of wheat, the temperature in both factories need to keep under 40 degree Celsius. However, the operation is problematic after someone has modified 2 steps of the procedure. Please help correct the following procedure.

If (kind of wheat = hard) Then
Move to #3 factory
Else
Move to #2 factory
	* Factory #1 (hard wheat)
	If (temperature < 40 degree Celsius) Then
	Dry the wheat
	Else
	Switch on fans to lower the temperature
	
	* Factory #2 (soft wheat)
	If (temperature < 40 degree Celsius) Then
	Switch on fans to lower the temperature
	Else
	Dry the wheat

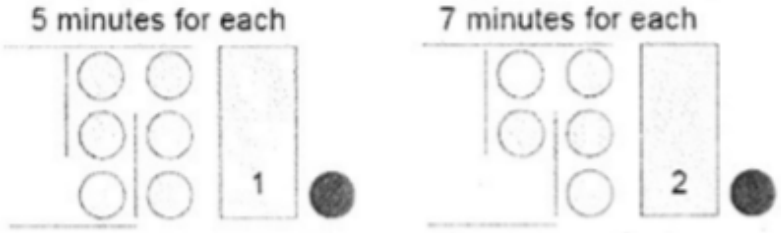
Remark: ">" is larger than, and "<" is smaller than

Figure 2.1 Question 1 of Kong and Lao's Summer PD program test of CT skills.

4. How to solve the following questions?

A lot of customers are queueing up at all checkout counters in a supermarket. The first counter is an express counter, in which each customer takes 5 minutes to checkout. The second one is a regular counter, in which each customer takes 7 minutes to check out.

(a) Now, there are 6 customers queued up at the first counter, while there are 5 customers at the second. You are going to checkout. Which counter will you choose to queue up?



(b) Assume that there are a lot of customers now. Please design a mobile application to help customers choose the fastest line. (Hint: use repeat, if … then … else …)

Please fill in the following steps

If

Then

Else

Figure 2.2 Question 4 of Kong and Lao's Summer PD program test of CT skills.

The test was administered on three separate occasions. The first test was administered before TDC 1, the second was the last part of TDC 1, and the third was after the TDC 2. The test inter-rater reliability was 0.98 for the first test, 0.97 for the second, and 0.99 for the third and had a Cronbach alpha score of 0.79, showing it had acceptable internal reliability (Kong & Lao, 2019). The teachers' test scores improved 2.54 points from pre-TDC 1 to post-TDC 1 and improved by another 2.62 points from post-TDC 1 to post-TDC 2 for a total gain of 4.32 points (Kong & Lao, 2019). Notice we did not see any CS concepts explicitly covered in this program, the testing of teachers' understanding of CT concepts involved pseudocode and CS concepts. Excluding CS concepts might be a strategic design decision in this situation since ¼ of the teachers held CS degrees, and many teachers had been teaching CS for many years. The goal of Kong and Lao's program was to introduce the teachers to CT concepts, and they were successful in doing so. Still, it could have been tied together with CS concepts to give the teachers a more well-rounded understanding of the relationship between CT and CS. Also, the evaluation method used, although statistically sound, makes it difficult to compare the program to other similar programs since Kong and Lao used an independently created evaluation tool (Kong & Lao, 2019).

In the visual programming language-centered programs, we saw a heavier emphasis on CT concepts over CS concepts. Noone and Mooney (2018) noted in their research on visual programming languages that researchers tend to agree that visual programming languages tend to fall short when facing complex CS. While this may be true, visual programming languages have been a successful tool when introducing teachers to CT concepts, as verified by Brennan and Resnick (Brennan & Resnick, 2012).

An opportunity for studying the success of visual programming languages on CS content knowledge would be to compare the content knowledge scores of two samples, one using visual programming languages and the other using text-based programming languages. That way, we can identify if visual languages can be successful in teacher CS as well as CT.

2.3.2 Text-Based Programming Language Programs

This section will highlight the advantages and shortcomings of text-based programming languages. In general, text-based programming languages encourage a deeper understanding of CS concepts to solve many problems compared to visual-based programming languages.

Lee et al. held a year-long PD program for 66 in-service high school STEM teachers (Lee et al., 2017). The goal of the program was to teach content and scientific practices in the spring and pedagogy and recruitment techniques during the summer. The PD had seven components: a kick-off conference, an online university course, fall and spring online debriefings, a summer workshop, facilitator support, an online community, project staff support, and a wrap-up workshop. The first weeks of the curriculum focused on fundamental CS concepts through CS Unplugged activities. Later, teachers had the opportunity to write programs using NetLogo, a text-based programming environment used for agent-based modeling. Lee et al. noted that the teachers came away from the user-based modeling exercises with "…a broader understanding of the use of CS and computational tools in scientific research across many fields" (Lee et al., 2017). The user-

based modeling language allows for connections between CS and real-world phenomena, which is why this language was chosen.

To gauge the teachers' CS concept understanding and attitudes toward CS, Lee et al. used a pre- and post-program survey. On the survey, 100% of the teachers from the PD program rated the PD "Very Good" or "Excellent" (Lee et al., 2017). The CS understanding also significantly improved from 68% pre-program to 73% post-program. Note, the 68% pre-program score is already high, so these participants were high-performing teachers coming into the program. The small increase was still statistically significant.

Additionally, all but one (65/66) teachers indicated feeling at least somewhat comfortable using computer models to conduct scientific inquiries. The outcomes from this program show that the program did an excellent job of engaging the teachers in CS practice and opening the teachers' minds to new ways CS can be used. It would be constructive for Lee et al. to share the CS questions from the survey so other researchers can see which topics were tested and improved by using text-based programming languages in the PD program. Another possible improvement to be made is to link the CS concepts and the CS pedagogy much sooner rather than in different workshops. Desmoine and Garet have found that explicitly linking CS teaching to the teachers' classroom lessons will lead to more success in preparing teachers to teach CS (Desimone & Garet, 2015). This link can be challenging to make when facilitating a PD program using a text-based programming language. A text-based programming language may not be an instructional tool used by the teachers in their classrooms; however, this link remains essential and needs to be heavily emphasized in the program. Finally, the year-

long length of this program is beneficial for the teachers' sustained learning, but this is logistically difficult to replicate in other programs. Overall, this program design is successful. The impact of this program could be made more transparent by providing more details on the measurement tools used.

Another program that was heavily content-focused using text-based programming languages was designed by Leyzberg and Moretti (Leyzberg & Moretti, 2017). Their goal was to offer a content-focused PD opportunity for teachers that lack strong CS backgrounds. The program was adapted from a college CS course to cover a week worth of content each day. The program was one week long, and the days went from 9 a.m. to 9 p.m. Each day consisted of a morning video lecture followed by a content break and then a second video lecture. The content breaks varied from day-to-day and included pedagogical tool discussion, discussions with the facilitators, and simply breaks between highly cognitive lectures. The participants lived on the campus during the program. During lunches, the teachers were encouraged to eat together and discuss each other's classrooms and how the different approaches they might use to incorporate the content from the program into their classrooms. The lectures provided hands-on experience with CS concepts, practice applying the concepts, and first steps towards creating assignments. The concepts taught during the PD were more advanced than most: input/output, recursion, algorithm, and data structure analysis, key-value data structures, Boolean logic, decimal/hexadecimal/binary conversions, machine learning, intractability (P vs. NP and NP-completeness), and circuit design (Leyzberg & Moretti, 2017). The average self-assessment on programming skills was 3.8/5, and on Java programming language was 3.5/5, where five means they are a "seasoned veteran." It was not clear whether these

self-assessment scores were pre-PD or post-PD. Regardless, these scores are exceptional, especially so if they are pre-PD scores. Daily surveys were administered to gauge the engagement and pace of the participants. The minimum daily average for engagement was 3.8/5, and the maximum was 4.5/5. The participants were also asked about the pace using a Likert scale where 1 meant "too slow," and 5 meant "too fast," meaning 3 is an ideal score. The maximum daily average was 3.5/5, and the minimum was 3.3/5 (Leyzberg & Moretti, 2017). This finding means the teachers felt the program was going only slightly "too fast," and the feedback was overwhelmingly positive. This program was fast-paced and covered some advanced CS concepts. It is encouraging that the participants could handle both the pace and the content presented. The teachers' ability to keep up with the advanced, fast-paced program suggests that the teachers did not lack strong CS backgrounds before the program, as Leyzberg and Moretti stated (Leyzberg & Moretti, 2017). If the purpose of this program were to offer strong CS content to teachers who lacked that, then it would seem logical to cover the basic CS concepts in-depth. Since the designers did not do this, it seems the participants may have had a better understanding of the basic CS concepts than was led on when this program was introduced. This research could be strengthened by expanding on the designer's definition of "strong CS background" since it seems to vary from this workshop to others. Again, it is encouraging to see the participants were able to handle the advanced CS concepts, but the program needs to be more explicit about the targeted participants.

Overall, *in the text-based programming language programs, we see more difficult concepts being covered during the programs*. Additionally, these programs are typically longer (one week or longer). Any shorter than one week, and the teachers likely will not

have time to learn the concepts and the programming language. Both programs were found to be beneficial to the participants and well-received.

2.3.3 Text-Based vs. Visual Programming Languages

In both text-based and visual programming language programs, we saw a significant increase in content knowledge scores. Although both program types saw increases in content knowledge, we suspect that the content knowledge tests focused on many different concepts. We also suspect that high performing participants from one program would not necessarily score highly on another program's content knowledge test due to the difference in the content covered. Both program types also saw similar positive feedback about the program design. In terms of a content knowledge advantage, it is difficult to find one between the two program types because each program uses a different measure. There seem to be two determinants for using one design over the other. The first is the allotted program length; any program under one weeklong will have a harder time introducing a text-based programming language. The other determinant is the goal of the participants and the program designers. Grades 6-8 teachers may require text-based programming experience to effectively teach their classrooms, whereas grade K-5 teachers may only need visual programming experience. Other factors go into this decision, but these two are the main factors influencing the program design.

2.3.4 Conclusions

This review has shown that there are differences between PD programs using text-based vs. visual programming language. While both types of programs showed an increase in the teachers' content knowledge, they do so in different ways and result in

different levels of CS content understanding. The visual programming languages allow the teachers to see the CS concepts abstractly. In contrast, the text-based programming languages request the teacher's attention to the intricacies of the respective programming language and how they are used to solve problems. While the program measures were not included in any of the four program reports, there were likely differences in the measures between the visual programming language programs and the text-based programming language programs. The most substantial difference between the two types of programs were the goals of the designers and the participants. For programs where the goal is to introduce new topics to teachers without much of a learning curve, a visual programming language would make sense to facilitate the PD program. For programs that seek to prepare the teachers by giving them in-depth knowledge of CS concepts, it would be more appropriate to use a text-based programming language. The trade-off in choosing a programming language is that visual programming languages are easy to learn but do not allow for in-depth CS content to be learned, and text-based programming languages are more challenging to learn. Still, they can provide a deeper understanding of CS concepts. We believe that these programs were rated highly by the participants because the design of the PD fit the background, experience, and goals of the participants well. This belief seems to be the case because the programs that covered much more difficult CS concepts had reviews similar to the programs that covered introductory CS and CT concepts and did not go as far in-depth.

2.3.5 **Recommendations**

During the program design period, it is best to evaluate the background of the teachers in the program and the learning outcomes associated with the program being

designed to ensure a beneficial program for the participants. An entry exam could be used to make sure that the teachers in the program will be ready to handle and benefit from the content covered during the program. Once the background of the teachers and the concepts they are missing is known, the next step is to decide if the program needs to be like a college CS 1 course or if the goal is to introduce CS and CT and not go in-depth on any of the CS concepts. If the goal is the teach CS in-depth and ample time is available to explore the complex CS concepts, it would be suitable to incorporate some text-based programming language. On the other side, if the goal is to introduce CS and CT to the teachers, then a visual programming language may be more appropriate. In general, it is also recommended that teachers are informed of the PD's goals and expectations accordingly before participating in the PD program to facilitate motivation. Likewise, it is also recommended that a PD program collects daily feedback and adjusts its design accordingly to tailor it better to teachers' ability and background. Another critical factor to pay attention to is the amount of time available for holding the PD program. If the program is less than a week in length (or 40 hours), it would be recommended not to try and introduce a new text-programming language since the learning curve of text-based programming languages could hinder the actual CS content learned. These are the main recommendations that can be made based on this review.

Chapter 3: Cohort 1 Summer PD Program

3.1 Program Structure

The PD program was held on two consecutive weeks in June 2019 and ran daily from 8:00 a.m. to 5:00 p.m. The PD program served 44 K-12 teachers. Of the 44 teachers, 29 teachers are elementary teachers (K–5), 17 are middle school teachers (6-8), and two are middle school teachers who also teach some high school classes (9-12). Some teachers belong to two groups (teach elementary and middle school students or teach middle school and high school students). The study contained 34 female teachers and ten male teachers.

3.1.1 Week 1 CS Content Course

The first-week course covered CS and CT topics. The schedule can be found below in Figure 3.1. The course was taught by a professor from a midwestern university and a team of four teaching assistants (TAs): one graduate and three undergraduates. All activities, assignments, and announcements were available for the teachers via the online learning tool, Canvas.

The teachers had homework assignments related to the content taught each day. The homework was assigned at the end of each day and was due at midnight on the same day. There was no assignment on the last day to allow time to finish the final project before the start of the second course. The first three homework assignments included an additional extra credit assignment, which extended the original assignment. The Cohort 1 assignments can be found in Appendix B.1. There was a cumulative exam on the last day

consisting of CS and CT knowledge tests. This exam was taken by 29 teachers pre-program and by all 44 teachers on the last day of the first course. The pre- and post-test made it possible to measure the 29 teachers' change in CS and CT content knowledge.

There were also three group activities based on Computational Creativity Exercises (CCE), designed to develop the teachers' CT skills through collaboration (Peteranetz et al., 2018). These exercises are akin to "CS Unplugged" exercises for open-ended problem solving using computational thinking and creative thinking skills (Miller et al., 2019). The CCEs can be found in Appendix C. Additionally, a final group project was assigned that allowed teachers to pick one CS topic and one CT topic and create a lesson for their respective grade levels. This group project can be found in Appendix B.1.5. The lessons were then presented in small groups, which included at least one member of the instruction team and one other teacher group. As part of the final project and after the presentations were delivered, the teachers individually created assignments to go along with their lesson plans. The final project can be found in Appendix B.1.6.

<table>
<tr><th></th><th>Monday</th><th>Tuesday</th><th>Wednesday</th><th>Thursday</th><th>Friday</th></tr>
<tr><td rowspan="5">Morning</td><td>Introduction</td><td>Homework 1 Discussion</td><td>Homework 2 Discussion</td><td>Homework 3 Discussion</td><td>Homework 4 Discussion</td></tr>
<tr><td>Team Building</td><td rowspan="3">Arrays (1D, 2D)/Loops</td><td rowspan="3">Functions</td><td rowspan="3">Search/Sort</td><td rowspan="3">Recap</td></tr>
<tr><td>Computional Thinking</td></tr>
<tr><td>Python Instruction/Install</td></tr>
<tr><td>Variables, Simple I/O, Data Structures</td><td>Arrays (1D, 2D)/Loops</td><td>Teaching and Learning Assignment Assigned</td><td>Search/Sort</td><td>Tests (1.5 hours allowed)</td></tr>
<tr><td>Lunch</td><td>Lunch</td><td>Lunch</td><td>Lunch</td><td>Lunch</td><td>Lunch</td></tr>
<tr><td rowspan="4">Afternoon</td><td rowspan="3">Selection</td><td>Search</td><td>Functions/Recursion</td><td>Search/Sort</td><td>Teaching and Learning Assignment Worktime</td></tr>
<tr><td>Everyday Object CCE</td><td>Storytelling CCE</td><td>Pathfinding CCE</td><td rowspan="2">Teaching and Learning Assignment Presentations</td></tr>
<tr><td>Search</td><td>Functions/Recursion</td><td>Teaching and Learning Assignment Worktime</td></tr>
<tr><td>Homework 1 Assigned</td><td>Homework 2 Assigned</td><td>Homework 3 Assigned</td><td>Homework 4 Assigned</td><td>Final Project Assigned</td></tr>
</table>

Figure 3.1 Cohort 1 Summer PD program's first-week CS/CT content course schedule.

3.1.2 Week 2 CS Pedagogy Course

The second-week course was held at a local school district conference center. The course was taught by four different CS teachers -- a college professor, a high-school teacher, a middle-school teacher, and an elementary school teacher. Presentations were arranged, so each instructor had a chance to talk about teaching the concepts of loops, variables, conditionals, and functions at their grade level, allowing teachers to understand curricular progressions across the K-8 grade span.

An outline of the course schedule can be found below in Figure 3.2. Daily reflections were completed online at the end of each day and were graded for completion.

Teachers were also divided into grade-level groups and were tasked with presenting a lesson they would deliver to their respective grade-level. The final assignment was an individual implementation plan that required the teachers to explain how they would be integrating CS into their curriculum in the following academic year.

	Monday	Tuesday	Wednesday	Thursday	Friday
Morning	Goals	Standards	Pedagogy	Classroom Management	Differentiation/CS4ALL
	Pedagogy	Teaching Robotics	Teaching Robotics	Group Lessons	Group Lessons
Lunch	Lunch	Lunch	Lunch	Lunch	Lunch
Afternoon	Concept: Loops	Concept: Variables	Concept: Conditionals	Concept: Functions	Implementation Planning
	Standards & Extracurriculars	TED Talk	Assessment	Lib Guides	Closing Survey

Figure 3.2 Cohort 1 Summer PD program second-week CS pedagogy course schedule.

3.2 Data Analysis

3.2.1 Description of Data

There are three sets of data:

1. The first data set is from a project-developed, pre- and post-program survey that assesses teacher self-confidence in (a) teaching CS (16 items, e.g., “I can adapt existing CS lesson plans to meet the needs of my students.”) and (b) their CS skills (6 items, e.g., “I can design and iteratively develop/refine CS programs.”). The confidence items were measured using a slider scale. The teachers indicated how confident they

were they could achieve each scenario by indicating a probability of success from 0 (0% confident) to 100 (100% confident)).

2. The second data set comes from a pre-post survey that assesses teacher attitudes towards CS. The nine attitudinal items used a Likert scale (1: strongly disagree, 2: disagree, 3: neutral, 4: agree, 5: strongly agree) to measure personal interest in CS (e.g., "I find the challenge of solving CS problems motivating.") and the perceived value of CS (e.g., "Reasoning skills used to understand CS can be helpful to me in my everyday life."). This instrument was developed by adapting the Computing Attitudes Survey (Dorn & Tew, 2015), which was validated with undergraduate CS students.

3. The third data set comes from a pre- and post-assessment measured teacher knowledge of CS concepts (Shell et al., 2017) and computational thinking (Peteranetz et al., 2020). The post-assessment measured CS and CT knowledge and was used as the final exam. The test separates the high performers from the low performers. Instead of the C average being around 70%-80% as a typical grade scale, the average test scores were around 50%, which indicates average performance and is not a failing grade (Shell et al., 2017).

3.2.2 **Participant Breakdown**

In this two-week summer PD program, there were three groups of teachers. The first group was the model-district CS teacher group, which consisted of 19 teachers from

a local model school district. This group of teachers were recognized nationally for their CS program. The second group was ten non-model-district CS teachers. Most of the teachers from these first two groups completed the pre- and post-program surveys participated in the pre-program knowledge test, and participated in the second-week course on CS pedagogy. The third group consisted of 15 non-CS teachers from rural districts around the state (not including the model district) who were involved in a program focusing on the development of educational leadership of rural teachers in STEM. These teachers were not planning to teach CS in the next academic year, did not participate in the pre- or post-program survey, the pre-program knowledge test, nor the second-week course on CS pedagogy. All 44 of the teachers who participated in the first-week CS content course took the post-program knowledge test as it was part of the grade for the course. A breakdown of the different groups and their participation can be found in Table 3.1.

Table 3.1 Breakdown of the participating groups in Cohort 1.

Group	Number of Participants in Group
Model-District CS Teachers	19
Non-Model-District CS Teachers	10
Non-CS Teachers	15
Non-Model-District Teachers	25 (Non-Model-District CS Teachers + Non-CS Teachers)
Research Cohort	29 (Model-District + Non-Model-District CS Teachers)
Pre-Survey	28 (Research Cohort - 1)
Post-Survey	25 (Research Cohort - 4)
Took Both Surveys	24 (Research Cohort - 5)
Pre-Test	29 (Research Cohort)
Post-Test	44 (Research Cohort + Non-CS Teachers)
First-week Course	44 (Research Cohort + Non-CS Teachers)
Second-week Course	27 (Research Cohort - 2 teachers who could not participate)

3.3 Results

3.3.1 Impact of PD Program on Cohort 1

The first research question was, "What was the impact of the CS summer PD on teacher's (a) knowledge of CS concepts, (b) knowledge of computational thinking, (c) CS attitudes, (d) confidence in CS knowledge and (e) confidence in teaching CS?". To address these questions, the pre- and post-survey data (31 total items each) collected from 29 participants who participated in both the pre- and post-program knowledge test were used. T-tests were used to compare each of the specified target groups. A breakdown of the results can be found in Table 3.2.

Table 3.2 Evaluation of the impact of the Cohort 1 CS PD program by comparing pre-program and post-program knowledge, attitude, and confidence scores (mean, standard deviation, t-value, degrees of freedom, significance value).

Test	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
Knowledge of CS	100	28	30.49	17.58	44	49.5	19.30	5.27	27	<.001
Knowledge of CT	100	28	54.76	17.68	44	65.45	14.73	3.38	27	<.005
CS Attitudes	5	28	4.54	0.43	25	4.60	0.32	1.22	23	0.24
Confidence in CS	100	28	61.42	27.41	25	71.53	23.17	2.96	23	<.01
Confidence in Teaching CS	100	28	73.51	21.70	25	83.40	11.26	4.49	23	<.001

3.3.1.1 Knowledge of CS Concepts

A paired t-test was used to find the teachers' knowledge of CS concepts improved significantly: $t(27) = 5.27$, $p < .001$. This result shows that the summer CS PD program had a significant positive impact on the teachers' CS concept knowledge.

3.3.1.2 Knowledge of CT Concepts

A paired t-test was also used to find the teachers' knowledge of computational thinking improved significantly: $t(27) = 3.38$, $p < 0.01$.

3.3.1.3 CS Attitudes

Only 24 of the 29 research cohort teachers completed both the pre- and post-program survey. Although teachers' attitudes improved from pre to post, a paired t-test showed no significant pre-post difference in teachers' attitudes: $t(23) = 1.22$, $p = 0.24$. The teachers possessed great attitudes pre-program ($M = 4.53$ on a five-point scale). This result indicates that the PD program had been able to recruit motivated teachers into the program, where increases in CS attitudes would be hard to achieve.

3.3.1.4 Confidence in CS Knowledge

The teachers' confidence in CS concepts was measured using a 6-item subset of the CS teaching confidence survey discussed above in Section 3.2.1. Again, only 24 of 29 teachers from the research cohort completed this survey both pre- and post-program. A paired t-test showed the teachers' confidence in CS concepts improved significantly from pre- to post-program: $t(23) = 2.96$, $p < 0.01$. However, of the 29 teachers that took the post-program CS confidence survey, 56% (14/25) of the teachers reported being over

70% confident with the CS concepts. This result is likely attributed to the short nature of the PD program. Some of the CS concepts were new to the teachers and could not be covered to the necessary extent. Additionally, the concepts were taught alongside programming in Python, and most teachers were new to programming in a high-level language. Many teachers struggled with syntax issues while learning new concepts, which may have kept the teachers from gaining confidence.

3.3.1.5 **Confidence in Teaching CS**

Only 24 teachers completed the survey, both pre- and post-program. Teachers' confidence in teaching CS improved significantly using a paired t-test: $t(23) = 4.49$, $p < .001$. Furthermore, of the 25 teachers who filled out the post-program survey, 80% (20/25) reported strong confidence (over 70%) in their ability to teach CS.

3.3.2 **Model-District vs. Non-Model-District Teachers**

The second research question addresses the difference in performance between the model-district CS teachers and the non-model-district teachers in the summer PD. Table 3.3 contains details about the data analysis performed in this section. Note, the non-model-district teachers include the ten non-model-district CS teachers and the 15 non-CS teachers. Before the summer PD program, the research cohort, 28 of the 44 participating teachers (19 from the model-district CS teachers and nine non-model-district CS teachers), completed the pre-program surveys on confidence and attitudes discussed earlier and knowledge tests described in the Section 3.2.1. The model-district CS teachers exhibited significantly more knowledge of CS concepts ($t(26) = 2.95$, $p < 0.01$), CT concepts ($t(26) = 2.28$, $p < 0.05$), CS concept confidence ($t(26) = 4.65$, $p <$

0.005), and CS teaching confidence ($t(26) = 4.54$, $p < 0.005$) than participating teachers from other districts. The model-district teachers have been involved in CS curricular development, training, support from teachers in CS education, learning progression and assessment, and meaningful use of resources to teach CS and CT (e.g., robots, programmable Altera boards, and other interfaces). Indeed, the model-district won a nation-wide award as a school district in K-12 CS education in 2018. Meanwhile, there was no significant difference between the two groups in terms of CS attitude: $t(26) = 1.55$, $p = 0.13$. This result again testifies to the high motivation of the teachers recruited into the PD program.

Recall, all 44 teachers, 19 model-district CS teachers, and 25 non-model-district teachers took a post knowledge test containing CS and CT concepts covered during the program as the week-one course's final test. There was no significant difference between the post-program knowledge test scores of model-district CS teachers and non-model-district teachers for both CS, $t(42) = 2.00$, $p = 0.06$, and CT concepts, $t(42) = 1.07$, $p = 0.29$. However, post-program, a significant difference between the model-district teachers and non-model-district teachers emerged when their CS concept confidences $t(23) = 3.11$, $p < 0.005$, CS teaching confidence ($t(23) = 4.54$, $p < 0.001$), and CS attitudes ($t(23) = 2.13$, $p < 0.05$) were measured. Note, only 16 of the 19 model-district teachers and 9 of the 25 non-model-district teachers completed the post-program CS concept confidences survey. This finding indicates that the teachers from the model-district were more confident than non-model-district teachers after the PD program, which has an insightful implication. These findings demonstrate that teachers with CS teaching experience (model-district teachers) have significantly more confidence post-program compared to

teachers with little-to-no CS teaching experience (non-model-district teachers) even though they have the same level of CS concept knowledge after experiencing the summer PD program. The lower confidence of non-model-district teachers could be due to their lack of familiarity with teaching CS or the lack of peer support and available CS-related resources.

Table 3.3 Cohort 1 Model-District (MD) vs. Non-Model-District (NMD) teacher mean, standard deviation, t-value, degrees of freedom, and significance values for each test.

Test	Scale	n_{MD}	$\bar{x}_{MD}$	σ_{MD}	n_{NMD}	$\bar{x}_{NMD}$	σ_{NMD}	t	df	p
Knowledge of CS (pre-program)	100	19	36	18	9	18	8	2.95	26	<.01
Knowledge of CT (pre-program)	100	19	60	15	9	44	18	2.29	26	<.05
Confidence in CS (pre-program)	100	19	73.91	18.34	9	35.06	25.05	4.65	26	<.005
Confidence in Teaching CS (pre-program)	100	19	83.25	11.69	9	52.92	24.05	4.54	26	<.001
CS Attitude (pre-program)	5	19	4.61	0.39	9	4.35	0.48	1.55	26	0.13
Knowledge of CS (post-program)	100	19	55.89	21.86	25	44.64	15.89	2.00	42	0.06
Knowledge of CT (post-program)	100	19	68.26	13.36	25	63.32	15.62	1.07	42	0.29
Confidence in CS (post-program)	100	16	80.78	16.70	9	55.07	24.72	3.11	23	<.005
Confidence in Teaching CS (post-program)	100	16	88.66	7.23	9	74.05	11.34	4.54	26	<.001
CS Attitude (post-program)	5	16	4.70	0.32	9	4.43	0.23	2.13	23	<.05

3.3.3 Factors Driving Teacher Performance

The third research question focused on factors that predicted success in the program. The factors evaluated were teacher confidence, plans to teach CS in the next year, and grade level of instruction.

3.3.3.1 Confidence in CS Content

A 6-item subset of the full 22-item pre-program survey was used to measure the teachers' confidence in the CS content (i.e., "*I can design and iteratively develop/refine CS program*."; "*I can document my programming solutions so that they are understandable to my peers*."; and "*I can decompose problems in ways that can be solved algorithmically*."). As described in Table 3.1, 28 teachers participated in the pre-program survey. Table 3.4 details the results of the data analysis in this section. A positive correlation was found between the 6-item subset and the post-program teachers' knowledge test scores ($r = 0.38$, $p < 0.05$). Based on this information, the test scores were divided into two groups based on the teachers' confidence levels, below-average confidence ($n_{below} = 11$), and above-average confidence ($n_{above,} = 17$), as indicated by the 6-item subset of the pre-program CS concept confidence survey. The average score on the confidence survey was 61.42 of 100, so that is the cut-off chosen for below- and above-average. A significant difference was discovered between the test scores of the teachers with above-average confidence and the teachers with below-average confidence, $t(26) = 2.17$, $p < 0.05$. These results suggest that pre-program CS content confidence levels can be used as an indicator of teachers' knowledge performance levels in a CS PD program.

Table 3.4 Measuring the impact of pre-program CS confidence by comparing Cohort 1 test scores of teachers with above (abv) average confidence coming into the program vs. teachers with below (blw) average confidence.

Test	**Scale**	n_{abv}	$\bar{x}_{abv}$	σ_{abv}	n_{blw}	$\bar{x}_{blw}$	σ_{blw}	***t***	***df***	***p***
Test scores (abv vs. blw)	100	17	63.76	17.05	11	51.03	11.52	2.17	26	<.05

3.3.3.2 Plan to Teach CS Following AY

29 of the 44 teachers participating in the PD had plans to teach CS at the K-12 level. There was no significant difference between the post knowledge test scores of the teachers who would be teaching CS in the following academic year (AY) to the teachers who would not, $t(42) = -0.29$, $p = 0.77$. Table 3.5 details the results of the data analysis in this section. A teacher's plan to teach CS in the following AY did not have an impact on their performance (in terms of their knowledge tests). A positive difference in performance from the teachers who would be teaching in the next school year was expected—with the premise that those teachers would be more motivated—but that was not the case.

Table 3.5 Evaluation of outcomes from Cohort 1 teachers planning of teaching (T) in the next AY vs. Cohort 1 teachers not teaching (NT) in the next AY on post-program test scores.

Test	Scale	n_T	$\bar{x}_T$	σ_T	n_{NT}	$\bar{x}_{NT}$	σ_{NT}	*t*	*df*	*p*
Test scores (T vs. NT)	100	30	58.28	15.91	14	59.68	11.30	-0.29	42	0.77

3.3.3.3 Grade Level of Instruction

No significant difference was found between the teachers' grade level of instruction (i.e., elementary (K-5) vs. middle-school (6-8) on the performance of the

teachers on the knowledge tests ($t(42) = 0.59$, $p = 0.55$). Table 3.6 details the results of the data analysis in this section. Better test scores were expected from the middle-school teachers since they need higher STEM capabilities to teach their grade-level. Instead, no significant difference was found between elementary teachers and middle school teachers in their knowledge test scores. The higher expectations of middle school teachers were not met, which could mean the necessary STEM capabilities of middle school teachers compared to elementary school teachers may not be significantly impacting their learning of CS content.

Table 3.6 Evaluation of Cohort 1 K-5 elementary (E) teachers vs. 6-8 middle school (M) teachers test scores.

Test	Scale	n_E	$\bar{x}_E$	σ_E	n_M	$\bar{x}_M$	σ_M	t	df	p
Test scores (E vs. M)	100	26	59.80	12.49	18	57.17	17.24	0.59	42	0.56

3.4 Program Evaluation

This section includes an evaluation of the program used in this study. SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis, a proven analysis tool (Hill & Westbrook, 1997), was used to identify what went well and what needed improvement. The strengths section (Section 3.4.1) of SWOT focuses on the successes. The weaknesses section (Section 3.4.2) pinpoints areas where that need to improve. The opportunities section (Section 3.4.3) focuses on how possible improvements based on feedback, insights, and experiences. The threats section (Section 3.4.4) highlights potential threats to the success of the program. SWOT analysis was used to help inform decisions made about the next PD program delivery.

3.4.1 Strengths

3.4.1.1 Instruction Team

There were enough members on the instruction team (one faculty instructor, one graduate TA, and three undergraduate TAs for the first-week course, and four master teachers as instructors for the week-2 course) to help all teachers promptly. The instruction team was adaptive to the teachers' needs throughout the two courses. They created new examples and altered course content on the fly to fit the teachers' needs.

3.4.1.2 Post-Course Knowledge of CT and CS Concepts

The 29 teachers from a local school district took the same pre- and post-program test over CT and CS concepts to measure their knowledge gained. The teachers who took the test had CS experience before the course. It was seen earlier that the teachers' CS and CT knowledge improved significantly. This improvement showed that the summer CS PD program had a positive impact on the teachers' CT and CS concept knowledge.

3.4.1.3 Sustained Duration

The program continues during the academic year and into the following summer, which gives the teachers more resources and time to learn the CT and CS concepts. A Virtual Community was set up through Listserv so the teachers can collaborate, share ideas, and ask each other for help after the course ends. During the academic year, the teachers will meet five times to go over the CT, and CS concepts learned over the summer, share class materials, and connect with the other teachers. The following

summer, the teachers will take a second two-week course on CT and CS concepts and CS and CT pedagogy.

3.4.1.4 Encouraged Collaboration

Through collaboration, the teachers were able to help each other better understand the difficult concepts. K-8 teachers are experts at breaking down difficult concepts into terms that are understood by their peers.

3.4.2 Weaknesses

3.4.2.1 Limited Active Learning in the First-Week Course

The first-week course used lecture-based learning mixed with hands-on group activities and programming tasks, but the lecture aspect did not engage the teachers. Teachers learned best when active learning activities followed short, brief lectures. Thus, more active learning activities were incorporated than initially planned.

3.4.2.2 Lack of Alignment Between Instructor vs. Teacher Goals in the First-Week Course

The goals of the instructor and the goals of the teachers did not align during the PD program. The instructor hoped the teachers would become capable programmers while learning CS and CT concepts while the teachers hoped to learn how to teach CS concepts to their students. The teachers were not prepared to learn the concepts through programming. The teachers had a difficult time with the programming language itself—especially its syntax and abstraction aspects—and therefore was not a practical approach for engaging teachers in learning about CS and CT concepts. We missed a significant

opportunity to link the concepts learned each day to their classroom instruction when we taught the CS concepts and the CS pedagogy separately.

3.4.2.3 **The Limited Virtual Community During Academic Year (AY)**

Slack, a Cloud-based instant messaging software, as a virtual community after the program, but the teachers did not make use of the site. The lack of engagement could be due to the teachers' unfamiliarity with Slack. Regardless, the virtual community moved to Listserv, a more accessible service that connects groups of people through their email. Both attempts to create a virtual learning community have fostered little to no communication. An active virtual learning community needs to be developed for future PD programs.

3.4.2.4 **Attempted to Cover Too Many CS Concepts**

It was planned for the first-week CS content course to cover basic concepts like strings, variables, conditions, and loops before progressing to more complicated concepts like functions, recursion, sorting, and searching. After covering the basic concepts, the teachers still had difficulty with loops and conditionals. Therefore, the teachers were not prepared for the transition to the more difficult concepts.

3.4.3 **Opportunities**

3.4.3.1 **Restructure Data Collection Tools for the Next Cohort**

Data collection tools need to be restructured for the next cohort for smoother data analysis. Services such as Google Forms can be used to collect teacher responses, store

them all in one place, and keep a consistent format so that the data analysis process will be efficient.

3.4.3.2 New Teacher Background Delivering a Fresh Approach

The next cohort of teachers taking the course will have no or little experience in teaching CS. The hope is that the new teachers will adopt a different approach to learning CS, allowing us to gain additional insights into what teachers' motivation, self-efficacy, perceived instrumentality, as well as approaches to learning, giving us a more comprehensive picture of teacher attitudes and learning performance.

3.4.3.3 Multiple Feedback Opportunities

Feedback collected from the teachers, and feedback still being collected will be used in designing upcoming PD programs. Feedback will be gathered during five meetings this academic year, from the in-class observations of the teachers teaching their students, and from the teacher leaders.

3.4.3.4 Funding for New Teaching Tools

The teachers in this study were funded to utilize new teaching tools in their classrooms. All elementary and middle school teachers receive funding to purchase CS instructional hardware and software as part of participating in the PD program. The first cohort used the available funds to purchase educational robots and tablets. The multiple feedback opportunities will show how new educational tools are utilized. The suggested tools can then be used in future programs to better familiarize the teachers with tools they could be using.

3.4.4 Threats

3.4.4.1 CS1 College Credit

Over the week, material covered needed to be reduced to accommodate the speed the teachers were learning. Thus, the material may have been altered to the point that not all the CS concepts specified in the course requirement were taught in-depth or at the intended level of rigor, though all basic CS concepts were covered. For example, at the beginning of the course, basic concepts (variables, Boolean logic, conditionals, loops, functions) and some advanced concepts (recursion, file I/O) were planned to be covered, but after altering the material only the advanced concepts, recursion, and file I/O, were briefly covered.

3.4.4.2 Individual Work is Challenging to Facilitate

The teachers were accustomed to collaborating on most assignments, and perhaps also because of their prior PD experiences, they prefer to continue to work together on their assignments. The teachers' collaboration made it challenging to design and facilitate individual work and comprehensive individual measures of CS and CT knowledge (e.g., assignments on reflection, analysis, and programming) in addition to the individual end-of-course knowledge tests.

3.4.4.3 Range of Instructors' Grade Levels

The teachers had varying levels of experience with CS and taught different grade levels. Catering materials to each grade level and experience level was a challenge. The

course was designed so teachers without a CS background could be successful, but teachers with CS background may have felt unchallenged.

Chapter 4: Cohort 2 Summer PD Program

4.1 Program Structure

The PD program was held on two consecutive weeks in June 2020 and ran daily from 8:00 a.m. to roughly 5:00 p.m. Due to the COVID-19 virus and social distancing guidelines, the program was taught online via Zoom video conferencing technology. The instructor used one camera to show his face and one camera to share slides, code, examples, document cameras, and teaching aids. Zoom breakout rooms were used heavily to facilitate group activities.

4.1.1 Morning CS Content Course

The program structure covered CS concepts using JavaScript in the morning session and CS pedagogy in the afternoon session. This section will focus on the morning, CS content session. The schedule for the morning can be found below in Figure 4.1 and Figure 4.2. The morning session was taught by a local high school teacher, a team of three teaching assistants (TAs): one graduate and two undergraduates, and two top-performing teachers from the previous cohort. All activities, assignments, and announcements were available for the teachers via the online learning tool, Canvas.

Morning Course Topic	Basic Syntax, Variables	Functions	Conditionals	Loops	Flow Charts
Afternoon "Plan"		Hour of addressing morning content - Elementary & Middle Functions	Hour of addressing morning content - Elementary & Middle Conditionals	Hour of addressing morning content - Elementary & Middle Loops	CSTA Standards
	6/8	6/9	6/10	6/11	6/12
	MONDAY	**TUESDAY**	**WEDNESDAY**	**THURSDAY**	**FRIDAY**
8:00 AM	Introductions. Get-to-know-you activity.	Morning welcome activity.	Morning welcome activity & Quiz #1.	Morning welcome activity.	Morning welcome activity & Quiz #2.
8:30 AM	Introducing JSFiddle. Basics of HTML, CSS, JS.	Condensing code with functions. Function inputs.	Conditionals discussion: fortune teller Google Form. if, else, else if distinctions	While	Basics of flow charts
9:00 AM	Printing ("HelloWorld"). Creating Variables.	Generating output with functions.	If, elseif, else Practice	Practice	"Formal" flow charts
9:30 AM	Getting input and storing it.	Variable scope.	Logical Operators	For	Making a program from a flow chart.
10:00 AM	Operators. Arithmetic, comparison, boolean, increment. (Printing outputs of operations)	Built-in functions: String functions	Practice in groups	Break	Making a program from a flow chart.
10:30 AM	Resource: w3Schools https://www.w3schools.com/js/	More String functions, explore W3Schools	Logical pathways within a function (ensuring return). Ternary Operator	Continue	Making a flow chart.
11:00 AM	Built in Math constants and functions. Random.	Parsing	Practice in groups	Practice activities	Making a flow chart and programming.
11:30 AM	Practice in groups	Practice in groups	Practice in groups	Practice in groups	Practice in groups
Assignment:	Math calculations	Functions Tasks	Leap Year/Fortune Teller	Palindrome/Primes	SecretMessage

Figure 4.1 Cohort 2 Summer PD program's CS/CT content morning course schedule – Week 1.

	Lists, Objects	Recursion	Sorting	Basics in a second language	Project/Test
	Pair programming in the classroom	Differentiation and assessment - CS4All	Project based learning		
	6/15	6/16	6/17	6/18	6/19
	MONDAY	**TUESDAY**	**WEDNESDAY**	**THURSDAY**	**FRIDAY**
8:00 AM	Morning welcome activity & Quiz #3	Morning welcome activity.	Morning welcome activity & Quiz #4	Morning welcome activity.	Morning welcome activity & Quiz #5
8:30 AM	Simple Arrays. Indexing.	Basics of recursive algorithms, stop condition	Sorting algorithms introduction: overview - https://www.toptal.com/developers/sorting-	Discuss other languages. https://www.tutorialspoint.com/codingground.ht	Research Survey
9:00 AM	String Split, Array functions	Fibonacci, Factorials	Writing an insertion sort.	Programming language exploration in pairs - pick a language and learn about it. Write a loop to	Research Survey
9:30 AM	Iterative Algorithms	Memory load of inefficient recursive algorithms.	Writing a selection sort.	Work time.	Break / Recap of two weeks' content.
10:00 AM	Enhanced for loops	Tower of Hanoi	Discuss bubble sort.	Present to the rest of the group.	Recap of two weeks' content.
10:30 AM	JavaScript Objects	Discuss merge sort.	Discuss quick sort.	Finish presentations.	Post-exam
11:00 AM	More enhanced for loops.	Interpret a recursive program to identify its function (Euclidean algorithm for GCD.)	Algorithm efficiency.	Regular Expressions, if time.	Post-exam
11:30 AM	Practice in groups	Practice in groups	Practice in groups	Regular Expressions, if time.	Post-exam
Assignment:	Register	Palindrome2	Alphabetize	Text Analysis	Final Project

Figure 4.2 Cohort 2 Summer PD program's CS/CT content morning course schedule – Week 2.

The teachers had homework assignments related to the content taught each day. The homework was assigned at the end of each morning and was due at midnight on the same day. Each homework assignment contained an extension that was optional but was put in place for the advanced teachers to challenge themselves. The Cohort 2 assingments can be found in Appendix B.2 There was a cumulative exam on the last day consisting of

CS and CT knowledge tests. This exam was taken by all 24 teachers pre-program and on the last day of the first course. The pre- and post-test made it possible to measure all 24 teachers' change in CS and CT content knowledge.

The morning session typically consisted of 15-30-minute lectures followed by 10-15-minute group activities. An example of one group activity (breakout session) from our Day 6 lecture on arrays can be found in Figure 4.3. Four CS content quizzes were administered throughout the program to help the instructors understand the teachers' understanding of past concepts as the program progressed. The Cohort 2 quizzes can be found in Appendix A.2. The quizzes gave the instructors an idea of which concepts to review before moving on. All the quizzes from Cohort 2 CS content course can be found in the Appendix. Additionally, a final group project was assigned that required teachers to create a hangman game. The project was put in place to allow the teachers to take something away from the class that they can show family members, friends, and their classrooms and inspire them to explore computer science further by adding components to their game. The project description can be found in Appendix B.2.10.

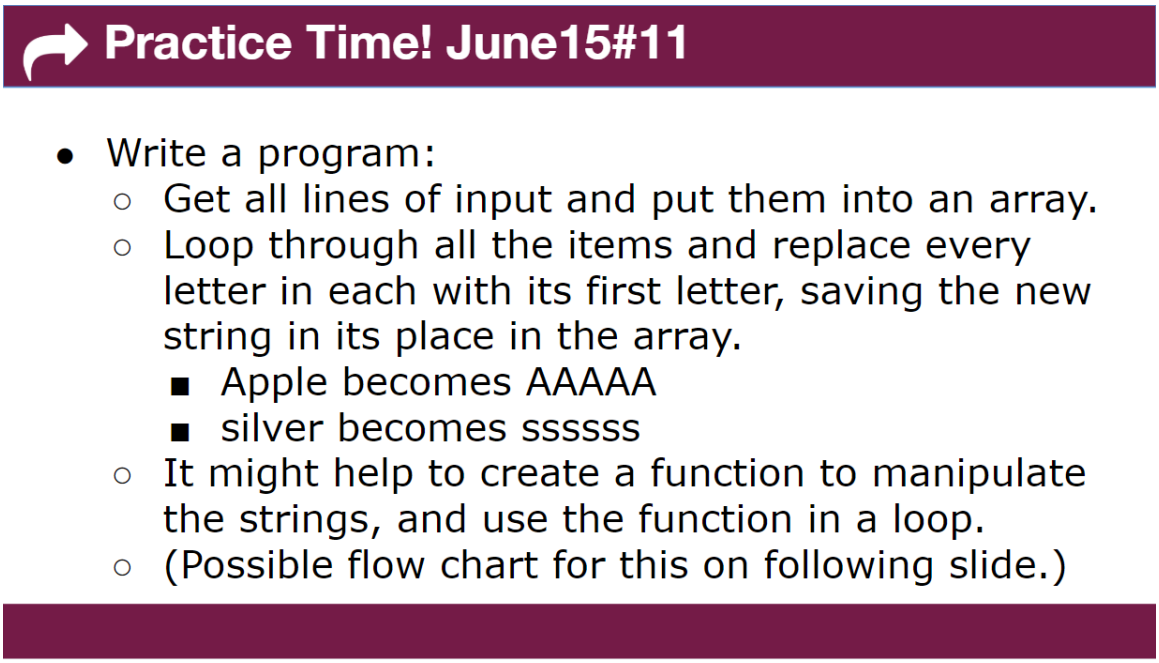


Figure 4.3 Cohort 2 Summer PD program Day 6 breakout session example.

4.1.2 Afternoon CS Pedagogy Course

This section will focus on the afternoon pedagogy session. The course was co-taught by six different CS teachers – three high school teachers, two middle-school teachers, and an elementary school teacher. The class met daily June 8-12 and June 15-19 from 1:00 pm to 5:00 pm, via Zoom, online. During the first week of the program, the lecture concentrated on a single CS concept and the CS concept aligned with the content taught during the morning CS concepts session. The purpose of the lectures was to show the teachers how to teach the concept to their respective grade levels. Therefore, the elementary, middle, and high school level instructors each discussed the concept and how it can be presented in their grade levels classrooms. During the second week, the focus shifts more towards robotics and tools the teachers will be able to use in their classrooms. Teachers were also divided into grade-level specific groups and were tasked with creating

and presenting a lesson plan for their respective grade-level to the rest of the class. The final assignment was an extension of the lesson plan they presented. The final assignment asked the teachers to write-up an implementation plan with lesson samples, demographics of their schools, and some reflections.

An outline of the course schedule can be found below in Figure 4.4 and Figure 4.5. Daily reflections were completed online at the end of each day and were graded for completion.

Morning Course Topic	Basic Syntax, Variables	Functions	Conditionals	Loops	Flow Charts
Susan	**Alan**	**Valerie**	**Kyleigh**	**Dan**	**Patrick**
	6/8	6/9	6/10	6/11	6/12
	MONDAY	**TUESDAY**	**WEDNESDAY**	**THURSDAY**	**FRIDAY**
1:00 PM	Computational Thinking Activity: Overview - Digital Breakout	Computational Thinking Activity: What's The Rule?	Computational Thinking Activity: Picture This!	Cohort 1 Panel - Patti, Olivia, Bethany	Computational Thinking Culminating Activity
1:30 PM	Debrief on Digital Breakout. Overview of afternoon course.	Small group discussion - CT Activities/Strategies	Small group discussion - CT Activities/Strategies	Cohort 1 Panel - Valerie, Patrick, (Matt, Lisa?)	Small group discussion - CT Activities/Strategies
2:00 PM	Content in the Classroom: Variables (Elementary) **Break at the end**	**Break**	**Break**	**Break**	**Break**
2:30 PM	Content in the Classroom: Variables (Middle & High)	Content in the Classroom: Functions (Elementary)	Content in the Classroom: Conditionals (Elementary)	Content in the Classroom: Loops (Elementary)	Content in the Classroom: Flow Charts (all levels)
3:00 PM	Grant details (Gwen/Wendy)	Content in the Classroom: Functions (Middle & High)	Content in the Classroom: Conditionals (Middle & High)	Content in the Classroom: Loops (Middle & High)	Building a successful CS program from the ground up - Multiple examples
3:30 PM					
4:00 PM	Independent Learning Time	Independent Learning Time	Independent Learning Time	Independent Learning Time	Independent Learning Time
4:30 PM					
Assignment:	Personal Learning Goal, Talk about experience level, Reflection on Variables	Reflection on Functions	Reflection on Conditionals	Reflection on Loops	Reflection on Flow Charts & the week overall

Figure 4.4 Summer PD program second-week CS pedagogy afternoon course schedule – Week 1.

Morning Course Topic	Lists	Recursion	Sorting	Basics in a second language	Project/Test

Susan	Alan	Valerie	Kyleigh	Dan	Patrick
	6/15	6/16	6/17	6/18	6/19
	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
1:00 PM	Ozobot Introduction	Dash or Cue Introduction (split elem and middle)	Other Robot Options- Patrick and Susan and ??	High School Robotics Class	What to do going forward?
1:30 PM	Ozobot Lessons for the Classroom	Dash or Cue Lessons for the Classroom	CS4All & CSTA discussion	Group 1 Lesson	Group 5 Lesson
2:00 PM	Break	Break	Break	Break	Break
2:30 PM	Content in the Classroom: Lists	Content in the Classroom: Recursion	Content in the Classroom: Sorting	Group 2 Lesson	Group 6 Lesson
3:00 PM	CS Teaching Strategies (Pair Programming, etc.)	Differentiation & Assessment	Project based learning	Group 3 Lesson	Group 7 Lesson
3:30 PM				Group 4 Lesson	Group 8 Lesson
4:00 PM	Independent Learning Time and Group Planning Time	Independent Learning Time and Group Planning Time	Independent Learning Time and Group Planning Time	Independent Learning Time	Independent Learning Time
4:30 PM					
Assignment:	Reflection on Lists & Pair Programming	Reflection on Recursion & Differentiation/Assessment	Reflection on Sorting/Project based learning		

Figure 4.5 Summer PD program second-week CS pedagogy afternoon course schedule – Week 2.

4.2 Data Analysis

4.2.1 Description of Data

There are three sets of data:

1. The first data set is from a project-developed, pre- and post-program survey that assesses teacher self-confidence in (a) teaching CS (16 items, e.g., “I can

adapt existing CS lesson plans to meet the needs of my students.") and (b) their CS skills (6 items, e.g., "I can design and iteratively develop/refine CS programs."). The confidence items were measured using a slider scale. The teachers indicated how confident they were they could achieve each scenario by indicating a probability of success from 0 (0% confident) to 100 (100% confident)). The survey was the same as that used in Cohort 1.

2. The second data set is from a pre-post survey that assesses teacher attitudes towards CS. The nine attitudinal items used a Likert scale (1: strongly disagree, 2: disagree, 3: neutral, 4: agree, 5: strongly agree) to measure personal interest in CS (e.g., "I find the challenge of solving CS problems motivating.") and the perceived value of CS (e.g., "Reasoning skills used to understand CS can be helpful to me in my everyday life."). This instrument was developed by adapting the Computing Attitudes Survey (Dorn & Tew, 2015), which was validated with undergraduate CS students. The survey was also the same as that used in Cohort 1.

3. The first data set comes from a pre- and post-assessment measured teacher knowledge of CS concepts (Shell et al., 2017) and computational thinking (Peteranetz et al., 2020). The post-assessment measured CS and CT knowledge and was used as the final exam. The test separates the high performers from the low performers. Instead of the C average being around 70%-80% as a typical grade scale, the average test scores were around 50%, which indicates average performance and is not a failing grade (Shell et al., 2017). The assessment was also the same as that used in Cohort 1.

4.2.2 Participant Breakdown

The PD program served 24 K-12 teachers. Of the 24 teachers, 18 teachers are elementary teachers (K–5), 6 are middle school teachers (6-8), and 2 teach high school classes (9-12). Some teachers belong to two groups (teach elementary and middle school students or teach middle school and high school students). The study contained 20 female teachers and four male teachers.

4.3 Results

4.3.1 Impact of PD Program on Cohort 2

The same research questions proposed and answered in Cohort 1 (Section 3.3) are re-evaluated for Cohort 2. The first research question was, "What was the impact of the CS summer PD on teacher's (a) knowledge of CS concepts, (b) knowledge of computational thinking, (c) CS attitudes, (d) confidence in CS knowledge and (e) confidence in teaching CS?". To address these questions, the pre- and post-survey data (31 total items each) collected from 24 participants who participated in both the pre- and post-program knowledge test were used. Again, t-tests were used to compare each of the specified target groups.

4.3.1.1 Knowledge of CS Concepts

A paired t-test was used to find the teachers' knowledge of CS concepts improved significantly: $t(23) = 3.39$, $p < .001$. The improved CS concept scores show that the Cohort 2 summer CS PD program had a significant positive impact on the teachers' CS concept knowledge.

4.3.1.2 Knowledge of CT Concepts

A paired t-test was also used to find the teachers' knowledge of computational thinking improved significantly: $t(23) = 7.52$, $p < 0.0001$.

4.3.1.3 CS Attitudes

All 24 teachers completed both the pre- and post-program surveys. The teachers' attitudes showed no significant change from pre- to post-program, $t(23) = -0.18$, $p = 0.86$. The mean attitudes scores regressed slightly from pre- to post, although the post-program attitude scores were still high ($M = 4.34$ out of 5).

4.3.1.4 Confidence in CS Knowledge

The teachers' confidence in CS concepts was measured using a 6-item subset of the CS teaching confidence survey discussed in Section 4.2.1 above. All 24 teachers from Cohort 2 completed this survey both pre- and post-program. A paired t-test showed the teachers' confidence in CS concepts improved significantly from pre- to post-program: $t(23) = 5.51$, $p < 0.0001$.

4.3.1.5 Confidence in Teaching CS

Again, 24 of the 24 teachers completed both the pre- and post-confidence survey. A paired t-test showed that the teachers' confidence in teaching CS improved significantly, $t(23) = 6.31$, $p < 0.0001$. Table 4.1 details the results of the data analysis in this section.

Table 4.1 Evaluation of the impact of the CS PD program by comparing Cohort 2 pre-program and post-program knowledge, attitude, and confidence scores (mean, standard deviation, t-value, degrees of freedom, significance value).

Test	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
Knowledge of CS	100	24	24.36	15.66	24	41.67	17.99	3.39	23	<.005
Knowledge of CT	100	24	46.30	15.86	24	68.06	10.52	7.52	23	<.001
CS Attitudes	5	24	4.36	0.31	24	4.34	0.49	0.18	23	0.86
Confidence in CS	100	24	50.22	27.13	24	72.78	17.69	5.51	23	<.001
Confidence in Teaching CS	100	24	64.35	19.31	24	85.31	7.86	6.31	23	<.001

4.3.2 Factors Driving Teacher Performance

The third research question focused on factors that predicted success in the program. The factors evaluated were teacher confidence, plans to teach CS in the next year, and grade level of instruction.

4.3.2.1 Confidence in CS Content

A 6-item subset of the full 22-item pre-program survey was used to measure the teachers' confidence in the CS content (i.e., "*I can design and iteratively develop/refine CS program*."; "*I can document my programming solutions so that they are understandable to my peers*."; and "*I can decompose problems in ways that can be solved algorithmically*."). No significant correlation was found between the 6-item subset measuring confidence in CS concepts and the post-program teachers' knowledge test scores ($r = 0.19$, $p = 0.85$). This result suggests that pre-program CS content confidence levels may not be a reliable indicator of teachers' knowledge gains in a CS PD program.

4.3.2.2 **Grade Level of Instruction**

For Cohort 2, we grouped each teacher into two separate groups based on the highest level of education they must deliver. Group 1 teachers are elementary (K-5) teachers, and group 2 (6^{th} grade and above) are middle school teachers and high school teachers. If a teacher is responsible for all grades, K-12, we grouped that teacher in group 2 since the highest level of instruction is above the sixth-grade level. We found no significant difference between the teachers' grade level of instruction (i.e., elementary (K-5) vs. middle-school (6-8)) on the performance of the teachers on the CS knowledge test ($t(22) = 1.42$, $p = 0.17$) or the CT knowledge test ($t(22) = 0.54$, $p = 0.60$). Table 4.2 details the results of the data analysis in this section. Better test scores were expected from the middle-school and above teachers since we believed they would need higher STEM capabilities to teach their respective grade-level. We believed this boost in STEM capabilities would aid them in learning CS. Instead, no significant difference was found between elementary teachers and middle school teachers in their knowledge test scores. The higher expectations of middle school teachers were not met, which could mean the necessary STEM capabilities of middle school teachers compared to elementary school teachers may not be significantly impacting their learning of CS content.

Table 4.2 Evaluation of Cohort 2 K-5 elementary (E) teachers vs. 6-8 middle school (M) teachers CS knowledge test scores.

Test	Scale	n_E	$\bar{x}_E$	σ_E	n_M	$\bar{x}_M$	σ_M	t	df	p
CS Test scores (E vs. M)	100	14	37.36	19.05	10	47.69	15.30	1.42	22	0.17
CT Test scores (E vs. M)	100	14	67.06	0.10	10	69.44	0.12	0.54	22	0.60

4.4 Program Evaluation

This section includes an evaluation of the program used in this study. This evaluation method is the same that was used after Cohort 1 to identify strengths, weaknesses, opportunities, and threats. (SWOT). Again, SWOT analysis is a proven analysis tool (Hill & Westbrook, 1997) that was used to identify what went well and what needed improvement. The strengths section (Section 4.4.1) of SWOT focuses on the successes of the program. The weaknesses section (Section 4.4.2) pinpoints areas where that need to improve. The opportunities section (Section 4.4.3) focuses on how possible improvements based on feedback, insights, and experiences. The threats section (Section 4.4.4) highlights potential threats to the success of the program. SWOT analysis was used to help inform decisions made about the next PD program delivery.

4.4.1 Strengths

4.4.1.1 Easily Accessible Programming Language

JavaScript and JSFiddle.com made programming more approachable as opposed to Python and the IDE used for the first cohort. There was minimal setup to begin coding. Using JavaScript allowed many of the Cohort 2 participants to feel comfortable programming in just two weeks.

4.4.1.2 Zoom Video Conferencing Breakout Rooms

The facilitators of the program used breakout rooms through Zoom to allow the teachers to work in groups on daily activities. The breakout rooms always had at least one facilitator and no more than five teachers to a room. These breakout rooms helped

alleviate the awkwardness of video instruction and yielded valuable discussions and collaboration throughout the course. These breakouts also broke up the lectures where teachers could practice hands-on learning and reinforce each lecture topic promptly.

4.4.1.3 **Zoom Video Conferencing Screen Share Technology**

Another unforeseen benefit of online instruction was the ease of collaboration through screen sharing. Problem-solving through observation of other's code helped each teacher to understand better where their issues. In a traditional classroom, the facilitators would go to each teacher's desk and look at their code with them. With the online instructional format, all discussion participants can view the screen at the same time without having to move seats or leave their work.

4.4.1.4 **Program Duration**

The program length was adequate for the facilitators to cover all CS concepts without rushing through any of the concepts too quickly. The program duration also allowed for the concepts to be linked with the pedagogy side in the afternoon, which allowed the teachers to think about how they might apply the concepts they just learned into their classrooms. The duration also allowed for more robust programming assignments to be administered since the teachers were well-acquainted with each concept during the day.

4.4.1.5 **Linking CS Content and CS Pedagogy**

In cohort 2, we designed the program intentionally to couple the two courses each day. Programming was learned in the morning and could be reinforced in the afternoon of each day as a practice in computational thinking: algorithmic (being methodical, creating a flowchart), problem decomposition (functions, creating a flowchart), evaluation

(debugging, analysis of correctness), pattern recognition (connecting the dots, leveraging what has been learned syntax-wise, assimilating similar bugs), generalization (seeing similar problems in syntax errors, learning useful debugging approaches), and abstraction (the use of variables, the use of arrays to store values, the use of functions, the representation of mathematical equations using variables). Coupling the courses together helped motivate teachers to appreciate and recognize the need to learn how to program to teach with more confidence and readiness, even when they are only teaching CS to grades K-5 and especially for teachers teaching CS to grades 6-8.

4.4.2 Weaknesses

4.4.2.1 Traditional Learning Tools Were Unavailable

Explaining more intricate concepts was made increasingly difficult, with the inability to draw on a whiteboard. Many times, a visual representation of a concept is easier to understand, and providing that was made more difficult through online instruction. The facilitators were forced to find new ways to explain concepts in detail. Though Zoom provided annotations on-screen, it was not easy to draw using a touchpad.

4.4.2.2 Breakout Rooms Limited Facilitator-To-Facilitator Interactions

During the breakout rooms, there would be times when one of the facilitators would be unable to answer a student's question. In a traditional classroom, the facilitator might call over another facilitator to try to explain the answer in a different way to assist the student. With the breakout rooms, that facilitator-to-facilitator interaction did not occur. Note that the facilitators, instead, used a separate platform (i.e., Slack) to interact.

4.4.2.3 **The Limited Virtual Community During Academic Year (AY)**

No virtual community was established for the participants to share ideas post-program and collaborate as they start creating lesson plans for the upcoming school years. We expect that some of the teachers exchanged emails or phone numbers, but we also expect that some teachers did not and will, therefore, need to communicate with the facilitators for help throughout the year.

4.4.2.4 **Course Expectations Not Clear Upon Signing Up**

Many of the teachers expressed confusion as to the goal of the PD program. The initial confusion was the expectation that the teachers would learn to program in addition to learning about CS concepts, despite that the course syllabus, shared days before the course, was clear on the expectations. The elementary teachers especially were surprised by this since they would not likely be teaching their students to program. The expectations must be made clear right away, so the teachers come into the program with the right mindset to approach the challenge of learning CS and programming concepts.

4.4.3 **Opportunities**

4.4.3.1 **Monitor the Exploration of New Ways to Teach CS to K-12 Students**

Many of the participants in this cohort did not have solid lesson plans before attending this program. It will be intriguing to see how they adapt what they learned in the program to their classrooms. Throughout the year, there will be opportunities for the teachers to share their successes and failures in their classrooms. This opportunity will give insight into the teachers' process of creating curriculum material from the PD

program instruction and the validity of teaching CS through online tools, like Zoom and Canvas.

4.4.4 Threats

4.4.4.1 No Monitoring of Teachers During Evaluations

Since the program was delivered online, there is no way to know if the teachers used outside sources to aid them during the individual assessments at the end of the program. Measures were taken to combat collaboration between students during the assessments (muting all teachers and disabling chat features), but there was no way to stop all forms of outside collaboration.

4.4.4.2 Significant Program Changes from Cohort 1 to Cohort 2

Due to COVID-19, the Cohort 2 summer PD program was moved to online. The online instruction was a significant change to the format of the program and made it difficult to compare the outcomes of the two programs since they are vastly different.

4.4.4.3 Distractions of Learning from Home

Again, due to COVID-19, the Cohort 2 summer PD program was held online. The online format meant that many of the teachers participated in the program from their own homes. With the ability to turn off the video, teachers may have been stepping away during lectures. We have no way of knowing the amount of time the teachers were away from the screen during the lecture. So, while we feel like we delivered all the content necessary, because of the distractions from learning at home and the ability to leave the

lecture undetected, teachers may have missed content if they stepped away from the computer.

4.4.4.4 **Difficult to Measure Teacher Participation in Small Group Discussions**

Again, due to COVID-19, the course was taught online, via Zoom. A challenge of using Zoom is that only one person in a small group can talk at any time. Teachers who are more willing to let others talk stay silent for long periods. The ability to mute the camera and microphone in Zoom makes it challenging to know their level of engagement. While the groups were sharing code, the facilitators assume that all teachers are following along. To check each teacher's code during the small group session would have taken too much time, so the introverted teachers may not have followed along with the code. Therefore, it would have been easy for a teacher to skip practice sessions, which would yield lower confidence and knowledge scores.

Chapter 5: Cohort 1 vs. Cohort 2

When designing the Cohort 1 PD program, many of the design decisions were experimental. Cohort 1 taught us many things about how to teach a CS PD program. We planned to make small changes from Cohort 1 to Cohort 2, so comparisons could be drawn about the changes made. Due to the COVID-19 pandemic, our design was forced to change drastically. In this chapter, we will detail the changes that were made from Cohort 1 to Cohort 2 and compare the program outcomes of Cohort 1 and Cohort 2.

5.1 Cohort 1 to Cohort 2 Changes

In this section, we will discuss the program design changes from Cohort 1 to Cohort 2. As mentioned above, some design decisions were forced upon the program by the local guidelines due to the COVID-19 pandemic. Table 5.1 summarizes the similarities and differences between the two cohorts.

Table 5.1 Details of Cohort 1 and Cohort 2 CS PD designs.

Cohort	Delivery	CS Content Course Schedule	CT Content Course Schedule	Lead Instructor	Instruction Team	Programming Language/IDE
Cohort 1 (Summer 2019)	In-person	Week 1: (8AM-5PM)	Week 2: (8AM-5PM)	College CS Professor	Lead Instructor, 1 graduate TA, 3 undergraduate TAs	Python/ PyCharm
Cohort 2 (Summer 2020)	Online	Week 1 & 2: (8AM-12 noon)	Week 1 & 2: (1PM-5PM)	High School CS Teacher	Lead Instructor, 2 Cohort 1 top-performers 1 graduate TA 2 undergraduate TA	JavaScript/ JSFiddle

5.1.1 In-Person to Online

Arguably the most significant change from Cohort 1 to Cohort 2 was the change from in-person instruction to online instruction. Our design team was forced to deliver the PD program online due to COVID-19. The classroom set up in Cohort 1 was a disadvantage because it was difficult to hear the facilitator and a challenge to see the whiteboard. These issues were solved by switching to the online format, but other issues arose as a result. A common challenge expressed by the teachers was balancing all the different windows necessary to participate in the course. This challenge was an unforeseen disadvantage that we were unable to mitigate throughout the program. Teachers who had access to multiple monitors found it easier to manage because they could leave the Zoom window open while coding or viewing the slides on the other screen. The online format proved to be challenging for the instruction team as well. Teaching over Zoom made it challenging to read the teachers' body language and identify where the teachers started feeling lost or remained engaged or stayed in the room, especially if the teachers' video was turned off. Small breakout rooms and constant communication with all participants was crucial to overcoming this obstacle.

5.1.2 Schedule

In Cohort 1, the design team decided to hold the CS content course during the first week from 8 am - 5 pm with an hour break for lunch. This week was overwhelming for many teachers. Then, we facilitated the second-week CS pedagogy course. This course was much more laid back and well-received by the teachers. While designing Cohort 2, we saw the opportunity to improve the program by holding both courses for half-days

over two weeks. The CS content course was held in the morning, and the CS pedagogy course was held in the afternoon. The schedules can be found in Figure 4.1, Figure 4.2, Figure 4.4, and Figure 4.5. By changing the structure in this, we not only broke up the challenging CS content course into small, digestible pieces, but we also created opportunities for the teachers to immediately link the CS content from the morning to their classrooms in the afternoon CS pedagogy class. This structure change helped make the CS content course more approachable, as it gave teachers more days to absorb the new CS topics and practice programming.

5.1.3 **Lead Instructor**

In Cohort 1, the lead instruction was a CS professor from UNL. The professor was accustomed to teaching in a college lecture, whereas the teachers participating in the program were used to elementary, middle school, and high school classrooms. These are two drastically different learning environments, and we saw a disconnect between the participants and the lead instruction throughout the course. In Cohort 2, we chose to replace the lead instructor with a local high school who was on the instruction team in Cohort 1 but taught only the CS pedagogy course for the first cohort. The high school instructor was able to draw on his experience with first-time CS learners to help connect with the teachers. We may be able to go a step further and choose a lead instructor from an elementary or middle school classroom. The relationship between the lead instructor and the teachers is vital for building an environment where the teachers are comfortable asking questions and interjecting during the fast-paced lecture to ask for clarification.

5.1.4 Instruction Team

In Cohort 1, the CS content course instruction team was made up of the lead instructor who was, a college CS professor, and four teaching assistants (one graduate teaching assistant and three undergraduate teaching assistants). The instruction team size was adequate for the large cohort size (44 teachers). However, no one on the instruction team had experience linking the CS concepts to a K-12 classroom. In Cohort 2, we filled this void by recruiting two top-performing teachers from Cohort 1 to join the instruction team along with the lead instructor who was, a high school CS teacher, and three teaching assistants (one graduate teaching assistant and two undergraduate teaching assistants). The Cohort 1 teachers with recent experience in learning and integrating the CS content into their classrooms was an invaluable addition to our instruction team.

5.1.5 Programming Language and Integrated Development Environment (IDE)

In Cohort 1, we chose to teach Python using the PyCharm IDE. We chose Python because the syntax is simple and is widely discussed as a first programming language for beginners to learn. However, teachers had issues with PyCharm, and Python versions throughout the course, and the instruction team was fixing issues related to Python and PyCharm throughout the course. In Cohort 2, the new lead instruction chose to change the language to JavaScript and use the internet tool, JSFiddle, as an IDE. The new language and IDE worked great for several reasons. First, JSFiddle is widely available, and once a free account is created, all the work done during the course will be saved on the site. JSFiddle did not require any set-up instructions, which made the introduction to

code near-seamless. The teachers need to get comfortable with the IDE and programming language quickly in a two-week PD program. Quickly onboarding the teachers with JSFiddle was a crucial step to delivering a successful PD program. Lastly, JavaScript, like Python, is regarded as another excellent programming language for beginners. The simple syntax and ease of execution made learning a new programming language, a difficult task for beginners, much more straightforward.

5.2 Program Outcomes (Cohort 1 vs. Cohort 2)

In this section, we will discuss the program outcome similarities and differences between Cohort 1 and Cohort 2 and some further evaluation we can do while comparing the two cohorts. First, we will compare the impacts of each program. Then, we will compare the participants' outcomes based on their backgrounds. Finally, we will look at the factors that drove teachers to perform better in each program.

5.2.1 Impact of PD Programs

5.2.1.1 Knowledge of CS Concepts

The change in teachers' knowledge of CS concepts was found using paired t-tests in both Cohort 1 and Cohort 2. In both Cohorts, the teachers' knowledge of CS concepts improved significantly from pre- to post-program, as seen in Table 5.2. The improvements from pre-program to post-program were more impressive in Cohort 1 than in Cohort 2, although the difference in post-program scores was not significant, as indicated by Table 5.3 and Figure 5.1. Since our Cohort 1 and Cohort 2 programs were vastly different, it is difficult to say precisely why the Cohort 1 teachers performed better.

It could be attributed to the Cohort 1 participants' CS background or the in-person instruction style over the online instruction style used in Cohort 2.

Table 5.2 Evaluation of the impact of the CS PD program from pre-program to post-program for Cohort 1 and Cohort 2.

Test	Cohort	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
Knowledge of CS	1	100	29	29	19.67	44	49.5	19.30	5.27	27	<.001
Knowledge of CS	2	100	24	24.36	15.66	24	41.67	17.99	3.39	23	<.005

Table 5.3 Two-sample t-test between Cohort 1 post-program CS knowledge test scores and Cohort 2 post-program CS knowledge test scores.

Test	Scale	n_{c1}	$\bar{x}_{c1}$	σ_{c1}	n_{c2}	$\bar{x}_{c2}$	σ_{c2}	t	df	p
Knowledge of CS	100	44	49.5	19.30	24	41.67	17.99	1.64	66	0.11

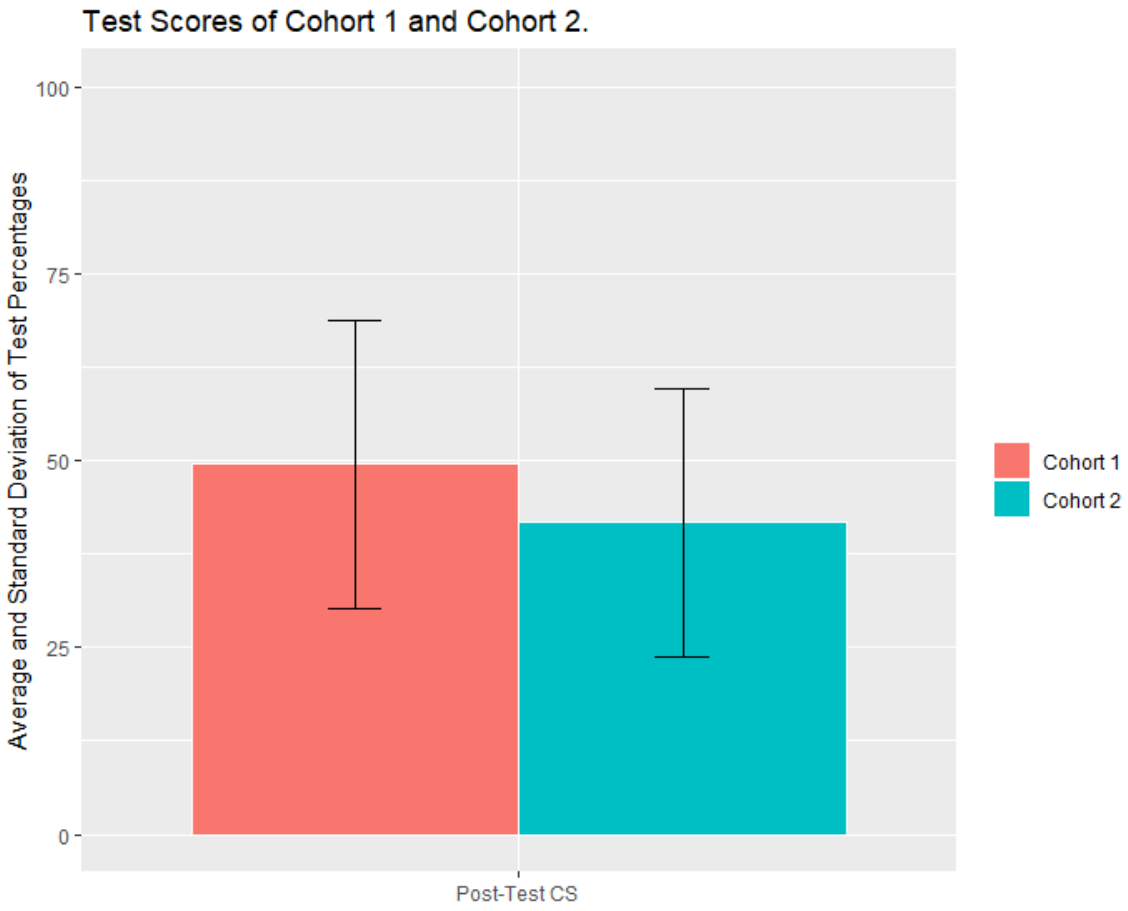


Figure 5.1 Post-program CS test scores in Cohort 1 and Cohort 2.

5.2.1.2 Knowledge of CT Concepts

We used paired t-tests to evaluate the change in the teachers' knowledge of computational thinking from pre- to post-program in Cohort 1 and Cohort 2. In both programs, the teachers' CT scores significantly improved, as shown in Table 5.4. There was no significant difference in the post-program CT exam scores between the two cohorts. We evaluated this using a two-sample t-test: $t(66) = 0.78$, $p = 0.44$. In both cohorts, the teachers performed much better on the CT exam compared to the CS exam, as seen in Figure 5.2.

Table 5.4 Evaluation of the impact of the CS PD program on the CT knowledge of the Cohort 1 and Cohort 2 participants from pre- to post-program.

Test	Cohort	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
Knowledge of CT	1	100	28	54.76	17.68	44	65.45	14.73	3.38	27	<.005
Knowledge of CT	2	100	24	46.30	15.86	24	68.06	10.52	7.52.	23	<.005

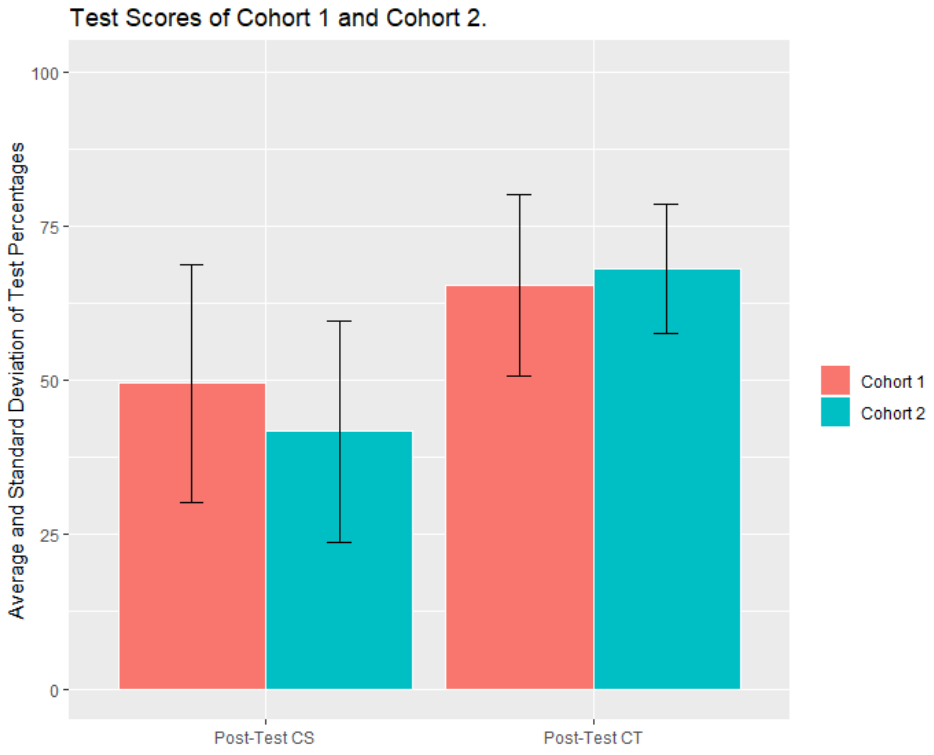


Figure 5.2 Cohort 1 vs. Cohort 2 post-program CS test scores and Cohort 1 vs. Cohort 2 post-program CT test scores.

5.2.1.3 CS Attitudes

In both Cohort 1 and Cohort 2, we saw no significant change in the teachers' attitudes towards CS from pre-program to post-program, as shown in Table 5.5.

However, we do see a significant difference in the post-program CS attitude scores from Cohort 1 to Cohort 2. Cohort 1's participants had significantly better attitudes towards CS than the Cohort 2 participants post-program, ($t(47) = 2.22$, $p < 0.05$) . This finding is surprising because the instructors felt that Cohort 2 went smoother than Cohort 1. Also, Cohort 1 and Cohort 2 did not have significantly different attitudes pre-program, ($t(50) = 1.59$, $p < 0.11$), and neither changed significantly from pre-program to post-program as seen in Table 5.5. Again, with so many changes from program to program, it is hard to identify contributing factors towards the difference in CS attitudes. One speculation is that our CS attitude survey is not accurately measuring the teachers' CS attitudes. We were also surprised to see the Cohort 2 teachers' attitudes regress from pre-program to post-program, and we were also surprised to see such small change from pre- to post-program in both Cohorts. This finding also hints potentially inadequancy of the survey used to measure of the teachers' CS attitudes.

Table 5.5 Evaluation of the impact of the CS PD program on the CS attitudes of the Cohort 1 and Cohort 2 participants from pre- to post-program.

Test	Cohort	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
CS Attitude	1	5	28	4.52	0.43	25	4.60	0.32	1.22	23	0.24
CS Attitude	2	5	24	4.36	0.31	24	4.34	0.49	0.18	23	0.86

5.2.1.4 **Confidence in CS Knowledge**

As discussed earlier, the teachers' confidence in CS concepts was measured using a 6-item subset of the CS teaching confidence survey. Both Cohort 1 and Cohort 2 showed the teachers' confidence in CS concepts improved significantly from pre- to post-

program. Coming into the program, both Cohort 1 and Cohort 2 teachers had a wide range of confidence levels, as noted in Table 5.6 and Figure 5.3. Post-program, the CS confidence levels became even between the two cohorts. This finding means the Cohort 2 teachers' confidence levels increased much more than the Cohort 1 teachers. It is encouraging to see high confidence scores from both cohorts, given the challenging nature of the CS PD program. It would be beneficial to identify precisely which parts of the PD program helped boost the teachers' confidence in CS. A strong case could be made that merely providing the teachers with the CS pedagogy course would be enough to boost their confidence in CS. The pedagogy course does an excellent job of familiarizing the teachers with difficult concepts in enjoyable and approachable ways. Further investigation could be done to find the exact pieces of the program that contributed most to the teachers' boost in CS confidence.

Table 5.6 Evaluation of the impact of the CS PD program on the CS confidence of the Cohort 1 and Cohort 2 participants from pre- to post-program.

Test	Cohort	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
CS Confidence	1	100	28	61.42	27.41	25	71.53	23.17	2.96	23	<.01
CS Confidence	2	100	24	50.22	27.13	24	72.78	17.69	5.51	23	<.001

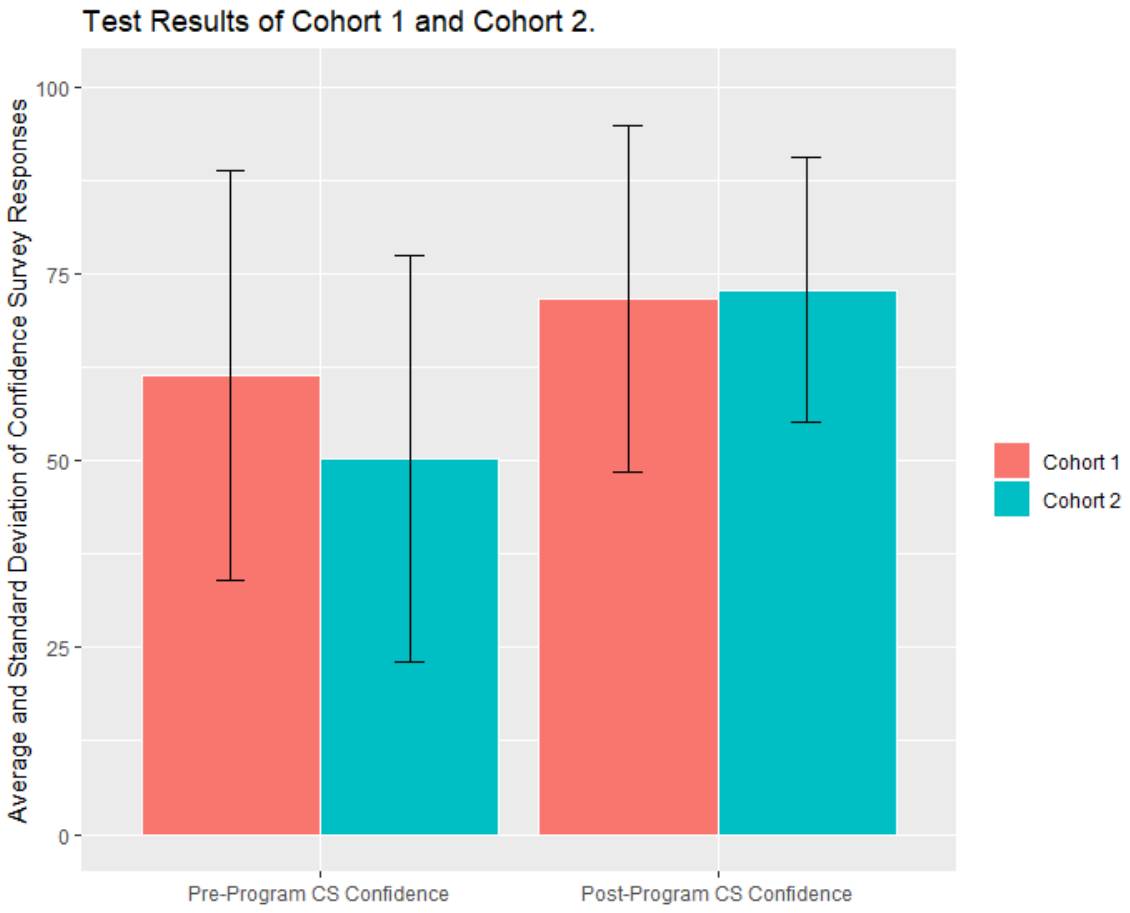


Figure 5.3 Cohort 1 participants' post-program CS confidence levels vs. Cohort 2 participants' post-program CS confidence levels.

5.2.1.5 Confidence in Teaching CS

In this section, our evaluation is similar to the last section, but instead, we evaluated the full pre- and post-confidence survey to measure the teachers' confidence in teaching CS. The confidence measure asks the teachers how comfortable they would be in handling several different scenarios. Again, a paired t-test showed that in both Cohorts, the teachers' confidence in teaching CS improved significantly. Table 5.7 and Figure 5.4 detail the change in CS teaching confidence from Cohort 1 to Cohort 2. There was no

significant difference between the CS teaching confidence between Cohort 1 and Cohort 2 teachers. Instead, both Cohorts improved and post-program, their confidences were much alike. As we discussed in the last section, it would be helpful to identify precisely where the confidence in CS teaching came from in the program. The CS teaching specifically is more likely to have come from the CS pedagogy course since the goal of that course is to provide information about how CS is currently being taught in other schools and how the teachers can integrate the same ideas in their classrooms.

Table 5.7 Evaluation of the impact of the CS PD program on the CS teaching confidence of the Cohort 1 and Cohort 2 participants from pre- to post-program.

Test	Cohort	Scale	n_{pre}	$\bar{x}_{pre}$	σ_{pre}	n_{post}	$\bar{x}_{post}$	σ_{post}	t	df	p
CS Teaching Confidence	1	100	28	73.51	21.70	25	83.40	11.26	4.49	23	<.001
CS Teaching Confidence	2	100	24	64.35	19.31	24	85.31	7.86	6.31	23	<.001

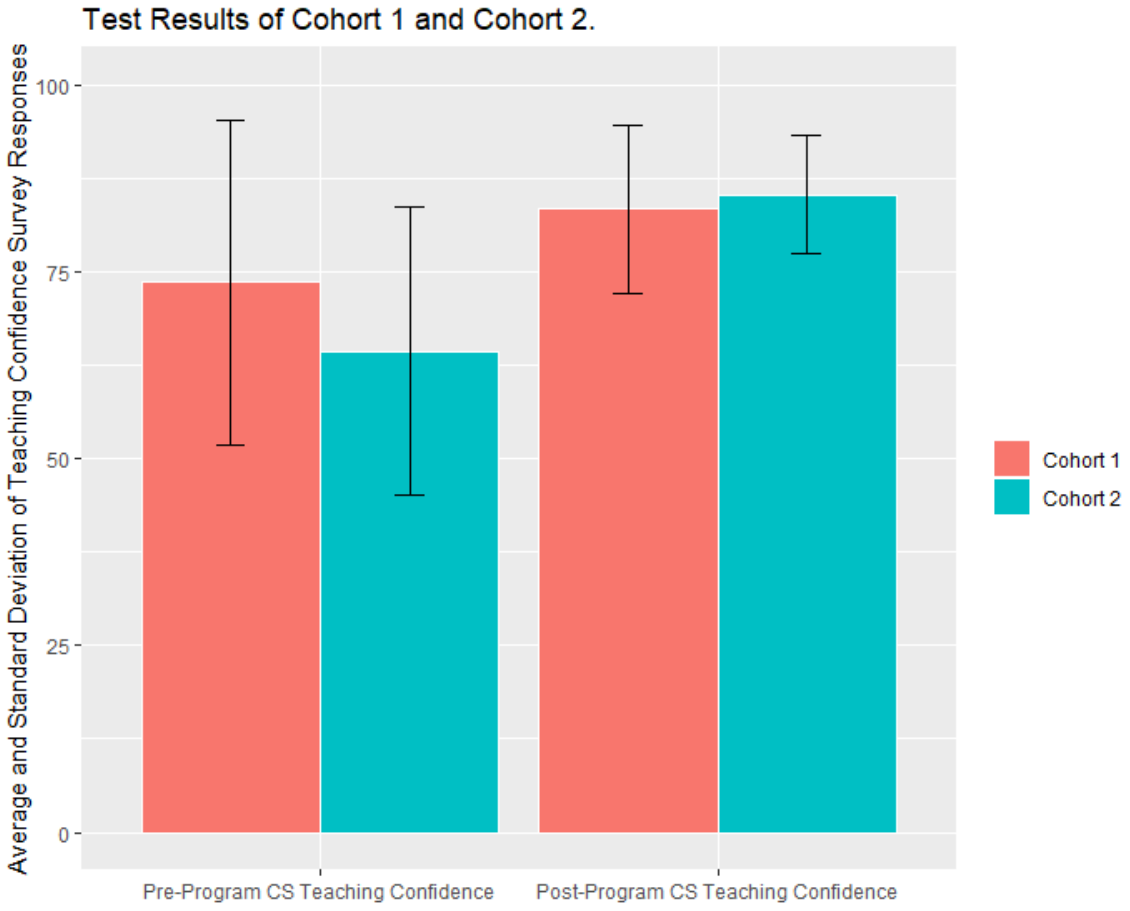


Figure 5.4 Post-program average and standard deviation of Cohort 1 participants' CS teaching confidence vs. Cohort 2 participants' CS teaching confidence.

5.2.2 Model-District vs. Non-Model-District

Our second research question focused on the learning outcomes of two different groups, model-district teachers, and non-model-district teachers. As noted in Section 3.3.2, the model-district teachers are teachers who are part of an award-winning school district in K-12 CS education. These teachers have had access to CS tools and resources for years, so their experience and knowledge of CS should have been higher than the non-model-district teachers coming into the program. More information about the model-

district teachers can be found in Section 3.3.2. Of the 44 teachers who participated in Cohort 1, 28 teachers completed the pre- and post-program surveys on confidence and attitudes and the pre- and post-program knowledge tests. Of those 28, 19 were model district teachers, and 9 were non-model district teachers. As we saw in Section 3.3.2, the model-district CS teachers, pre-program, exhibited significantly more knowledge of CS concepts, CT concepts, and CS concept confidence than participating teachers from other districts. Of our 24 Cohort 2 teachers, none of them were from the model district. In this section, we want to, again, compare the learning outcomes of model-district teachers (19 from Cohort 1) and the non-model-district teachers (9 from Cohort 1 and 24 from Cohort 2). We also want to compare the learning outcomes of the Cohort 1 non-model-district teachers to the Cohort 2 non-model district teachers to see if the model-district teachers helped enhance the learning ability of the Cohort 1, non-model district teachers.

5.2.2.1 **Model-District vs. All Non-Model-District**

As stated before, there were 19 model-district teachers, all from Cohort 1 and 33 non-model-district teachers, 9 from Cohort 1 and 24 from Cohort 2. When comparing these two groups of teachers' pre-program results, again we see that the model-district-teachers performed significantly better, in all five categories: knowledge of CS concepts ($t(50) = 3.08$, $p < 0.005$), knowledge of CT concepts ($t(50) = 3.00$, $p < 0.005$), CS concept confidence ($t(50) = 3.98$, $p < 0.005$), CS teaching confidence ($t(50) = 4.21$, $p < 0.005$), and CS attitudes ($t(50) = 2.41$, $p < 0.05$). The difference in each category is illustrated in Figure 5.5 and Figure 5.6.

The differences between the two groups were less significant post-program than they were pre-program. Recall, all 44 teachers from Cohort 1, 19 model-district CS teachers, and 25 non-model-district teachers took a post knowledge test containing CS and CT concept and only 16 of the 19 model-district teachers and 9 of the 25 non-model-district teachers completed the post-program CS concept confidences survey. All 24 of the Cohort 2 non-model-district teachers completed all three measures. Therefore, in total, we have 19 model-district teachers and 49 non-model district teachers who completed the post-program knowledge tests. For the confidence survey, 16 model district teachers, and 33 non-model district teachers. One difference from pre-program to post-program is that there was no significant difference between the *post-program CT knowledge test scores* of model-district teachers and non-model-district teachers, $t(66) = 0.68$, $p = 0.50$. However, post-program, there was still a significant difference between the model-district teachers and non-model-district teachers when evaluating their *CS concept knowledge* ($t(66) = 2.58$, $p < 0.05$), *CS concept confidences*, ($t(47) = 2.13$, $p < 0.05$), *CS teaching confidences* ($t(47) = 2.27$, $p < 0.05$), and *CS attitudes* ($t(47) = 2.70$, $p < 0.01$). Again, these differences are illustrated in Figure 5.7 and Figure 5.8.

It is not surprising that the model-district teachers who had strong backgrounds in CS education were more prepared and performed better in the CS PD program. *It is encouraging that the CS PD program boosted the non-model-district teachers' CT knowledge to be similar to the model-district teachers and nearly closed the gap between the non-model-district and model-district teachers' CS knowledge, CS confidence, CS teaching confidence, and CS attitudes*. *The fact that both groups saw significant gains in four of the five categories (CS attitudes saw no significant improvements) encourages us*

that the program will work for teachers of varying backgrounds.

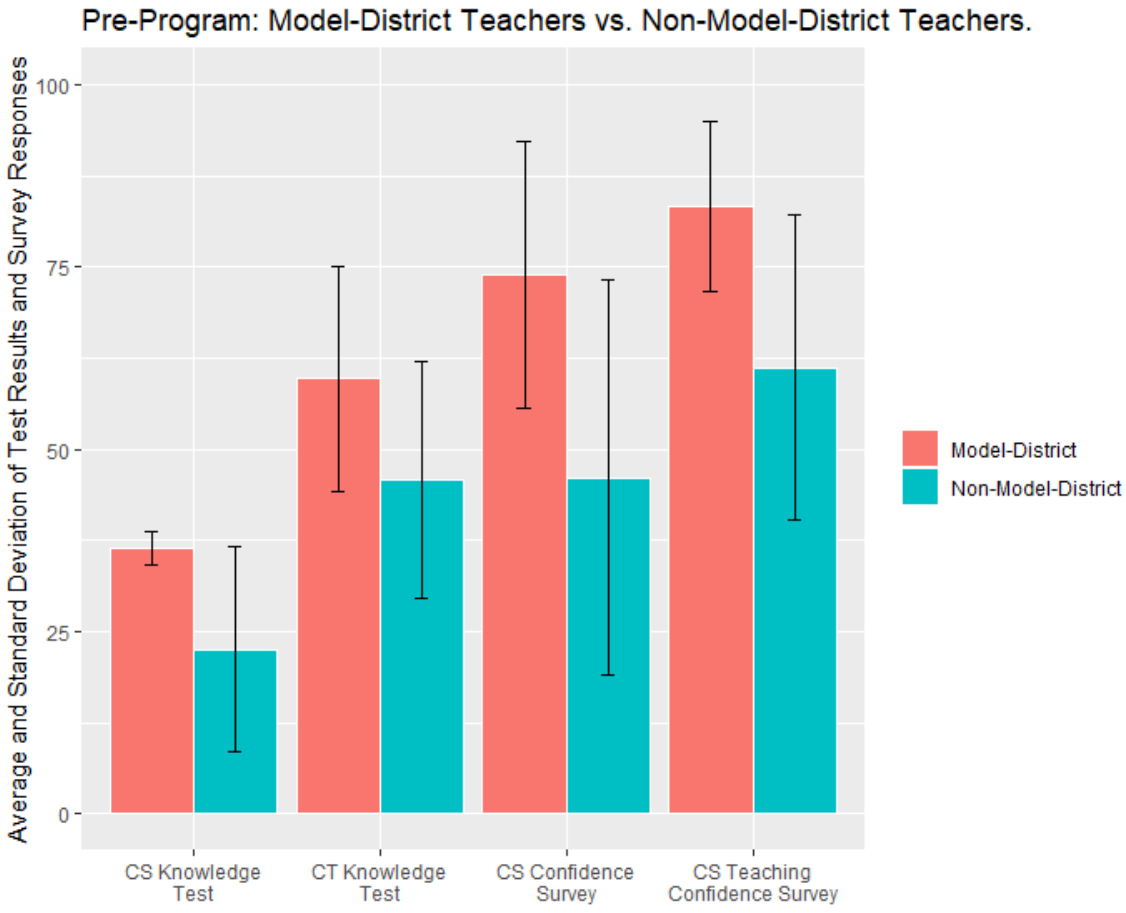


Figure 5.5 Pre-program averages and standard deviations of model-district vs. non-model-district teachers' CS knowledge test scores, CT knowledge test scores, CS confidence survey responses, and CS teaching confidence survey responses.

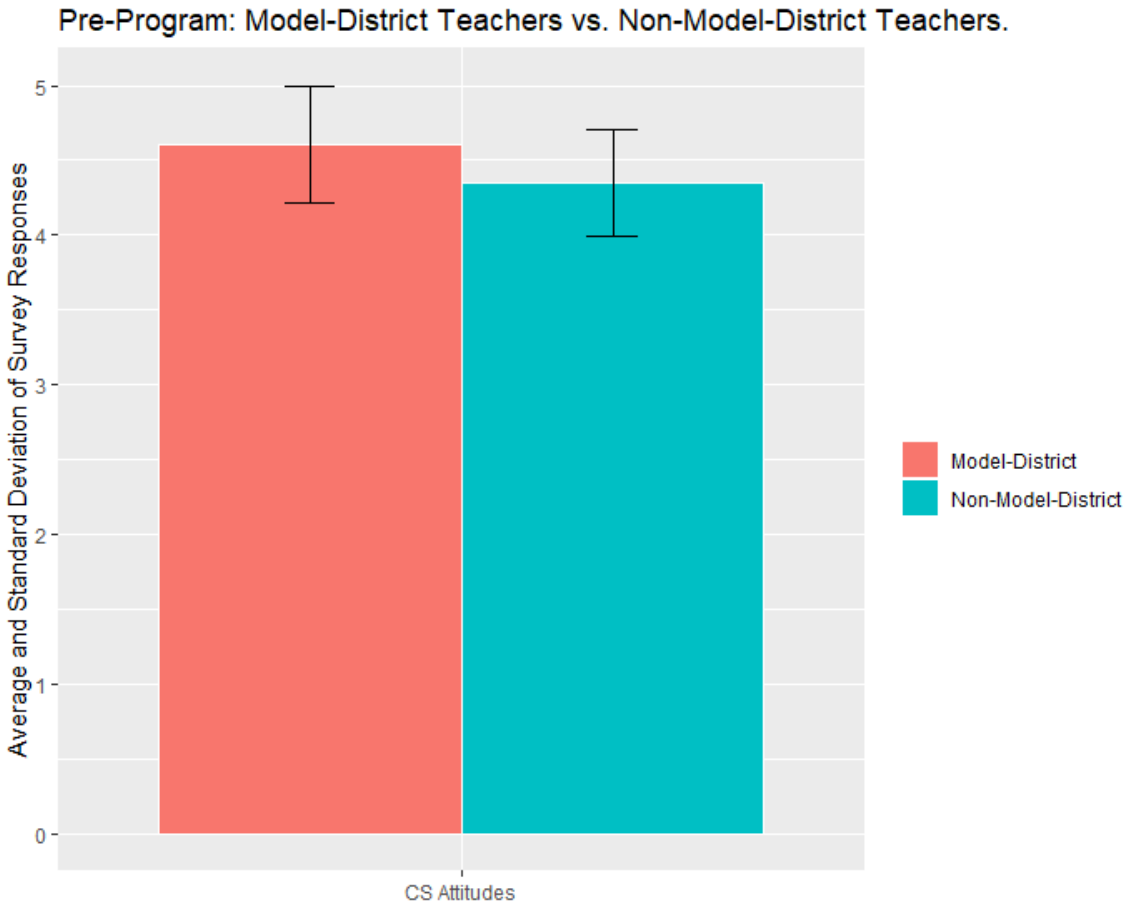


Figure 5.6 Pre-program averages and standard deviations of model-district vs. non-model-district teachers' CS attitudes.

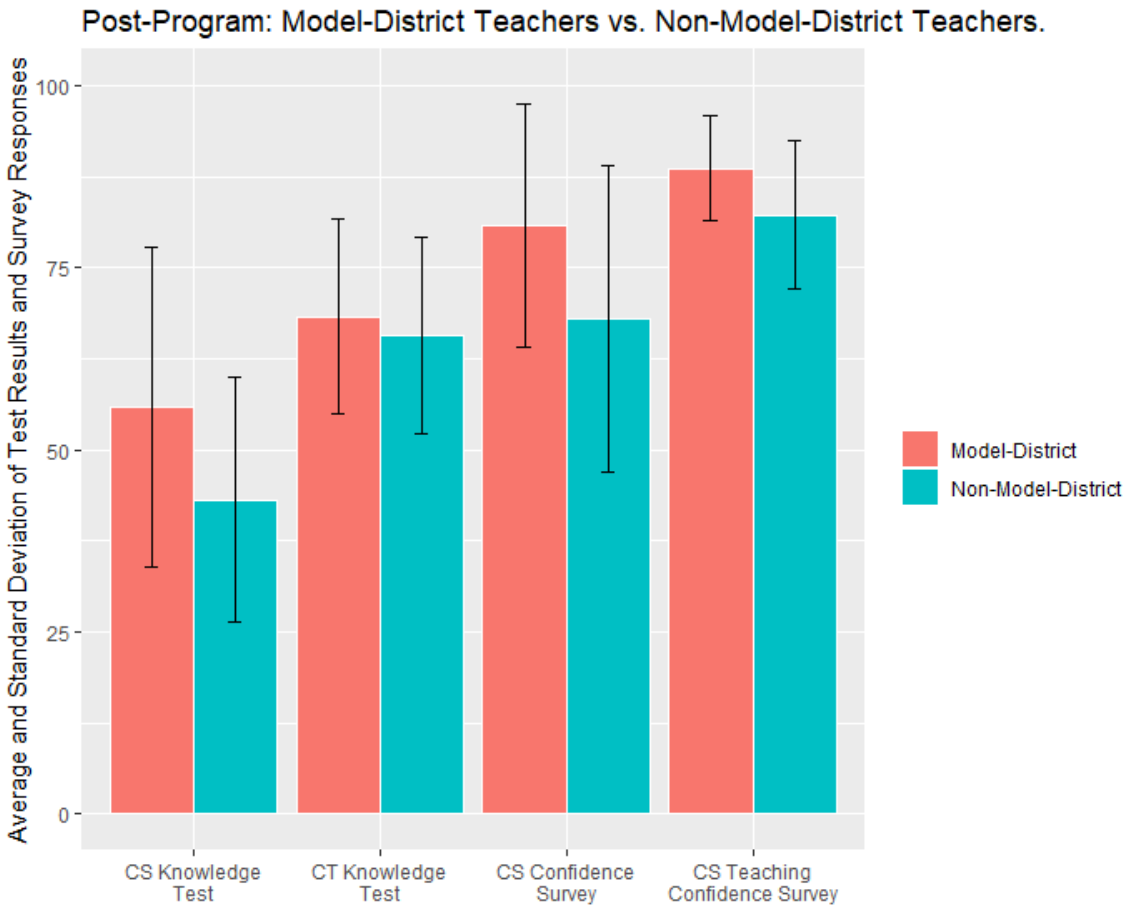


Figure 5.7 Post-program averages and standard deviations of model-district vs. non-model-district teachers' CS knowledge test scores, CT knowledge test scores, and CS confidence survey responses.

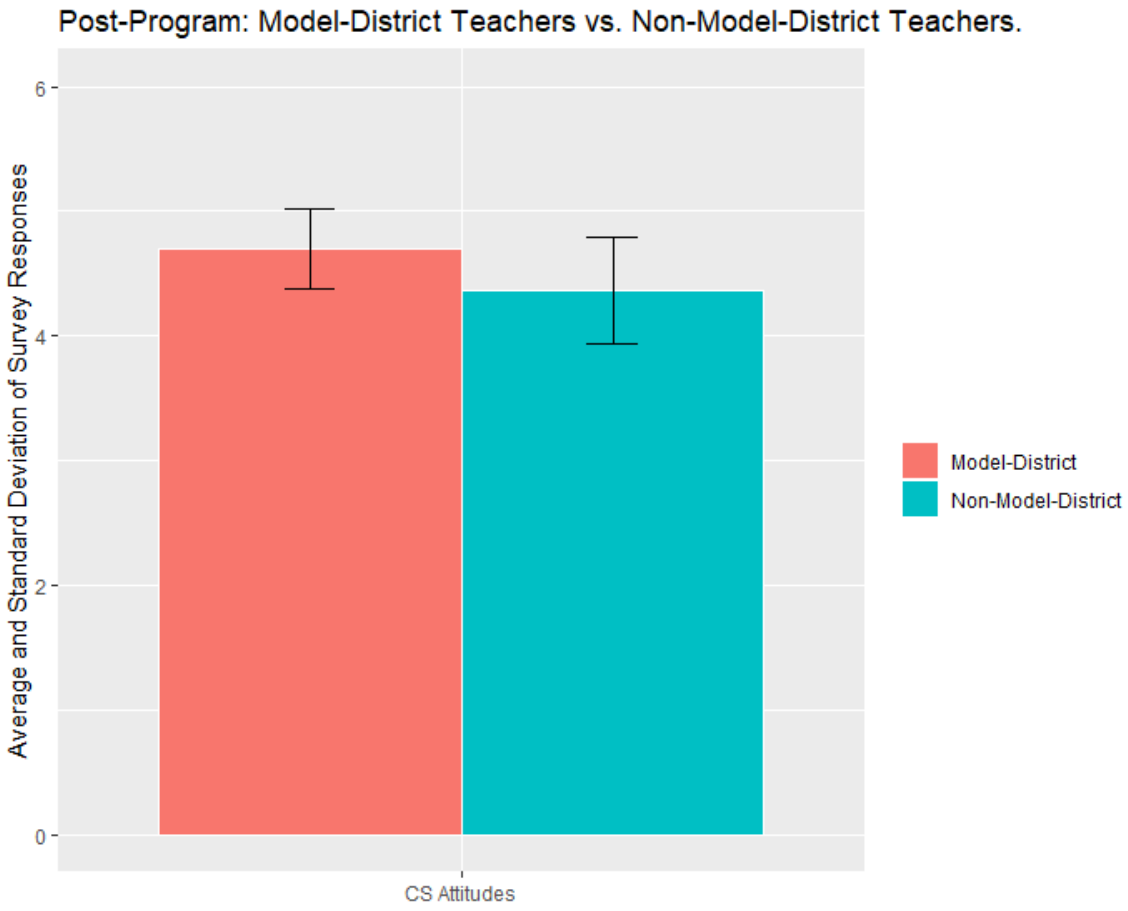


Figure 5.8 Post-program averages and standard deviations of model-district vs. non-model-district teachers' CS attitudes.

5.2.2.2 Cohort 1 Non-Model-District vs. Cohort 2 Non-Model-District

We want to evaluate the difference in knowledge and confidence from pre-program to post-program for these two groups to find out if the Cohort 1 non-model-district teachers had an advantage by working closely with the model-district teachers.

In Cohort 1, 9 non-model district teachers completed the pre-program tests and survey. Cohort 2 had 24 teachers who completed both the pre-program tests and the survey. We can see that coming into the program, there was no significant different

between the two groups CS knowledge ($t(31) = 1.17$, $p = 0.25$), CT knowledge ($t(31) = 0.29$, $p = 0.78$), CS confidence ($t(31) = 1.46$, $p = 0.15$), CS teaching confidence ($t(31) = 1.42$, $p = 0.17$), or CS attitudes ($t(31) = 0.08$, $p = 0.94$). This finding is to be expected since neither group had strong CS backgrounds. Our interest lies in the post-program results. Remember, all 25 Cohort 1 non-model-district teachers took the post-program knowledge tests, but only 9 of the 25 completed the post-program confidence survey. All 24 of Cohort 2 teachers took both the post-program knowledge test and the confidence survey. From conducting two-sample *t*-tests, we see that the *Cohort 1 teachers had significantly more CS concept confidence and CS teaching confidence post-program compared to the Cohort 2 teachers* ($t(31) = 2.29$, $p < 0.05$ and $t(31) = 3.24$, $p < 0.005$, respectively) but there was *no* significant difference in the two groups CS knowledge ($t(47) = 0.61$, $p = 0.54$), CT knowledge ($t(47) = 1.24$, $p = 0.22$), or CS attitudes ($t(31) = 0.53$, $p = 0.60$) post-program. Meanwhile, recall that the model-district teachers' pre- and post-program knowledge of CS, while significantly higher than the non-model teachers, was still relatively low (M=55.89/100). Whereas, the model-district teachers' confidence was relatively high (M=80.78/100). We speculated that if the model- district teachers were going to be able to assist the non-model district teachers in any way, it likely would have been in boosting their confidence in CS. We discussed earlier how we believe most of the confidence gain is coming from the CS pedagogy course since this is where they learn to apply the CS concepts in their classrooms. For the Cohort 1 non-model district teachers, they had an advantage because not only did they have the instructors telling them how CS can be taught in the classroom, but their peers in the PD program could give advice and recommendations on how CS can be taught in the classroom. In the CS

concepts course, all the model-district and non-model district teachers are at a similar level of understanding, and so the model-district teachers cannot provide as much assistance to the non-model teachers. Further testing will be needed to validate this hypothesis and to identify the amount of confidence gained from each class.

5.3 Conclusion

We found success in Cohort 2 like that of Cohort 1 despite (or perhaps because of) changes to the program's method of instruction (in-person to online), schedule design (week 1 CS content course, week 2 CS pedagogy course to morning CS course, afternoon CS pedagogy course), lead instructor (university professor to high school teacher), and programming language and IDE (Python and PyCharm to JavaScript and JSFiddle). Several smaller items, such as homework assignments, office hours, group structure, and more, changed because of these more significant changes. Both Cohort 1 and Cohort 2 teachers saw significant improvements to their CS and CT knowledge, confidence in CS concepts, and confidence in teaching CS. Our findings encourage us to believe that, although our program design changes significantly, the program remained effective in preparing teachers to teach CS. We did see that some results were significantly higher in Cohort 1 than Cohort 2, but because of the vast number of differences between the two cohorts, it is difficult to determine what factors led to the variance in outcomes.

Additionally, we saw that the Cohort 2 non-model-district teachers performed similarly to the non-model-district teachers of Cohort 1 on the knowledge tests. The main difference between the Cohort 1 non-model-district teachers and the Cohort 2 non-model-district teachers was that the Cohort 1 teachers gained significantly more confidence in

their CS capabilities on the confidence survey. Again, it is difficult to determine the exact reason for the difference in outcomes between cohorts. However, we believe the collaboration between the Cohort 1 model-district teachers and the Cohort 1 non-model-district teachers during the CS pedagogy course was beneficial for the non-model-district teachers to understand how CS is being taught in the classroom. This advantage could be a crucial confidence amplifier which led to the non-model-district teachers' superior confidence.

5.4 Recommendations

As mentioned before, we had to change our program to be online due to COVID-19. At first, we considered canceling the course because we did not know the logistics behind facilitating an online CS PD program. What we found is that a CS PD program can be effective through online facilitation. Therefore, we encourage those who are in similar situations to carry out the PD program even if the logistic challenges of online facilitation are uncertain.

We also found that the change in lead instructor for the CS professor to the high school teachers was beneficial to increase the participants' comfort level. The high school teacher was able to use more familiar terms and connect with the participant much easier. We believe the closer the lead instructor is to the average grade level of the participants, the better the lead instructor will be in connecting the CS content with the participant's target grade-level. However, keep in mind that it is also essential that the lead instructor has a strong understanding of all the concepts taught during the program and has significant experience in teaching CS topics.

Another recommendation is to use a programming language that is widely available and simple to install and use. We found many issues using Python and PyCharm in our first cohort due to package versions being different among students and installation setting getting change that should not have been changed. In Cohort 2, we used JavaScript and JSFiddle. JavaScript and JSFiddle were much more comfortable for the participants to use because all they needed was a link to the JSFiddle site, and they could begin programming. It is also nice that all the teachers' work is saved on JSFiddle and can be easily shared and accessed at any time.

Lastly, we found our schedule design in Cohort 2 to be much lower stress for the participants. In Cohort 1, we held the CS content course for the first week and the CS pedagogy course during the second week. Many of the Cohort 1 participants were feeling overwhelmed and mentally fatigued in the middle of the first-week CS content course. The second week CS pedagogy course was light on CS concepts and focused more on how the teachers will teach in their classrooms. This arrangement made the second-week course a much more comfortable course for the teachers and resulted in a low-stress environment. In Cohort 2, we decided to break the high-stress, CS content course up and teach CS content in the mornings, and CS pedagogy in the afternoons. The observed attitudes of the Cohort 2 teachers were significantly better than that of the Cohort 1 teachers during the CS content course. By having the CS pedagogy course in the afternoon, the teachers were given a mental break and were also able to immediately connect the content taught in the morning with materials they can use in their classrooms. We found this to be an essential design change that should facilitate better CS understanding and instruction.

Chapter 6: Conclusion

6.1 Summary of Findings

This Thesis discussed the work we have done over the past two years to develop, facilitate, and analyze two separate two-week, CS PD programs for K-8 teachers. In this study, we sought to answer three distinct research questions:

1. What was the impact of the CS summer PD on the teachers?
    a. knowledge of CS concepts
    b. knowledge of computational thinking
    c. CS attitudes
    d. confidence in CS knowledge
    e. confidence in teaching CS
2. What were the differences between teachers from a model school district (an urban school district with extensive CS curricular development and teacher PD) and teachers from other school districts? How did the program impacts differ?
3. Which factors lead to teacher success (e.g., knowledge test scores) in terms of CS understanding in the summer PD program? Specifically, this study investigates *confidence in CS content*, *plans to teach CS in the following AY*, and *grade level of instruction* as potential predictors of teacher performance.

To answer the first two questions, pre- and post-program surveys and pre- and post-knowledge tests were used to measure each of the summer PD program's impacts on

the teachers. In both programs, we saw significant improvements in CS knowledge and CT knowledge test scores, and in the teachers' confidence in CS and in teaching CS. We saw no significant effect on the teachers' attitudes towards CS, which was a surprising result. We further investigated these impacts by comparing the model-district teachers' outcomes against the outcomes of non-model district teachers in both Cohort 1 and Cohort 2. Overall, the model-district teachers performed better on the post-program, CS knowledge test and showed higher levels of CS confidence. However, the Cohort 2, non-model-district teachers did outperform the Cohort 1 model-district teachers on the post-program, CT knowledge test. Overall, our finding shows that teachers gain additional confidence and knowledge from experiences from within their districts, their K-8 instruction, and other factors external to the PD program.

In Cohort 1, we made an insightful observation. The experienced, model-school district teachers showed higher confidence levels while having similar test scores. We saw this as interesting because we felt the model-school district teachers were confident enough to teach CS without needing to have a deep CS background. We speculated that this was because the teachers knew they could teach, and had been teaching, successfully without being able to perform well on the CS and CT knowledge tests. Furthermore, this shows us that during the PD, we need to focus on providing hands-on pedagogical experiences to help boost the teachers' CS confidence rather than only focusing on the CS concepts. This support for this speculation was strengthened in Cohort 2, where we made it a focus to link the CS content with the pedagogy by holding the CS content course and the pedagogy course on the same day. As a result, the Cohort 2 teachers gained more confidence than Cohort 1 teachers despite having less CS background. Since there were

many changes to the design of the PD in Cohort 2, it will take further investigation to confirm this finding, but it is encouraging, nonetheless.

In Cohort 1, we found that pre-program CS confidence was a reliable predictor of success in the program. In Cohort 2, we found that pre-program confidence was not mandatory to have success in the program. We wanted to investigate many more factors that could be indicators of success in Cohort 2, but due to the variety of different program changes, it was challenging to control and identify any variables as predictors of teacher success. Future work will need to be done on this third research question to find an answer. In addition to our original variables of interest, *confidence in CS content*, *plans to teach CS in the following AY*, and *grade level of instruction,* we would like to investigate the teachers' *comfortability with technology, the number of years teaching CS,* and the teachers *problem solving ability and mathematical thinking skills.*

6.2 Future Work

Based on these findings, the next step is to plan, implement, and facilitate more PD programs. Facilitating more PD programs would allow us to understand the implications of our design changes and tease out the nuances behind each of our findings from the first two cohorts. Additionally, we need to set up a structured PD program with little-to-no changes from year-to-year so we can begin testing and identifying our variables of interest to find valid predictors of success in our CS PD program. Lastly, a line of future work that would benefit the entire CS PD community would be to create a validated CS and CT knowledge test so PD programs can be all be compared. This sort of measure would help guide PD program designers, so they know which concepts need to

be covered in their CS PD program to best prepare the teachers' for CS instruction. Another direction would be to compare and contrast the similarities and differences in teaching CS1 (i.e., introductory CS) to K-12 teachers and post-secondary students, to obtain insghts that could inform CS educators on how to more effectively teach K-12 teachers.

References

Ahamed, Sheikh Iqbal, Dennis Brylow, Rong Ge, Praveen Madiraju, Stephen J. Merrill, Craig A. Struble, and James P. Early. 2010. "Computational Thinking for the Sciences: A Three-Day Workshop for High School Science Teachers." Pp. 42–46 in. ACM.

Bell, T., Rosamond, F., Casey, N.: Computer science unplugged and related projects in math and computer science popularization. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 398–456. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30891-8 18

Bower, M., Wood, L. N., Lai, J. W. M., Howe, C., Lister, R., Mason, R., Highfield, K., & Veal, J. (2017). Improving the Computational Thinking Pedagogical Capabilities of School Teachers. Australian Journal of Teacher Education, 42(3), 53–72.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada, 1, 25.

Brown, Quincy, and Amy Briggs. 2015. "The CS10K Initiative: Progress in K-12 through 'Exploring Computer Science' Part 1." Inroads 6:52–53.

Chai, Ching Sing, Joyce Hwee Ling Koh, and Chin-Chung Tsai. 2010. "Facilitating Preservice Teachers' Development of Technological, Pedagogical, and Content Knowledge (TPACK)." Journal of Educational Technology & Society 13(4):63–73.

Darling-Hammond, L., & Richardson, N. (2009). Research review/teacher learning: What matters. Educational leadership, 66(5), 46-53.

Desimone, Laura M., and Michael S. Garet. 2015. "Best Practices in Teachers' Professional Development in the United States." Psychology, Society, & Education 7(3):252.

Dorn, Brian, and Allison Elliott Tew. 2015. "Empirical Validation and Application of the Computing Attitudes Survey." Computer Science Education 25(1):1–36.

Ericson, Barbara, Mark Guzdial, and Maureen Biggers. 2005. "A Model for Improving Secondary CS Education." Pp. 332–336 in ACM SIGCSE Bulletin. Vol. 37. ACM.

Fancsali, Cheri, Linda Tigani, Paulina Toro Isaza, and Rachel Cole. 2018. "A Landscape Study of Computer Science Education in NYC: Early Findings and Implications for Policy and Practice." Pp. 44–49 in Proceedings of the 49th ACM Technical Symposium on Computer Science Education, SIGCSE '18. New York, NY, USA: ACM.

Goode, J., & Margolis, J. (2011). Exploring computer science: A case study of school reform. ACM Transactions on Computing Education (TOCE), 11(2), 1-16.

Goode, J., Margolis, J., & Chapman, G. (2014). Curriculum is not enough: The educational theory and research foundation of the exploring computer science professional development model. Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE '14, 493–498. https://doi.org/10.1145/2538862.2538948

Google and Gallup. 2016. "Trends-in-the-State-of-Computer-Science-Report.pdf." Retrieved January 8, 2020 (http://services.google.com/fh/files/misc/trends-in-the-state-of-computer-science-report.pdf).

Hatlevik, Ove Edvard, Inger Throndsen, Massimo Loi, and Greta B. Gudmundsdottir. 2018. "Students' ICT Self-Efficacy and Computer and Information Literacy: Determinants and Relationships." Computers & Education 118:107–19.

Hill, Terry and Roy Westbrook. 1997. "SWOT Analysis: It's Time for a Product Recall." Long Range Planning 30(1):46–52.

Kong, S.-C., & Lao, A. C.-C. (2019). Assessing In-service Teachers' Development of Computational Thinking Practices in Teacher Development Courses. 976–982. https://doi.org/10.1145/3287324.3287470

Lang, Karen, Ria Galanos, Joanna Goode, Deborah Seehorn, & Fran Trees. 2013. Bugs in the system: Computer science teacher certification in the US. New York, NY: ACM.

Lee, I. A., Psaila Dombrowski, M., & Angel, E. (2017). Preparing STEM Teachers to Offer New Mexico Computer Science for All. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 363–368. https://doi.org/10.1145/3017680.3017719

Leyzberg, D., & Moretti, C. (2017). Teaching CS to CS Teachers: Addressing the Need for Advanced Content in K-12 Professional Development. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 369–374. https://doi.org/10.1145/3017680.3017798

Liu, J., Lin, C.-H., Wilson, J., Hemmenway, D., Hasson, E., Barnett, Z., & Xu, Y. (2014). Making games a "snap" with Stencyl: A summer computing workshop for K-12 teachers. Proceedings of the 45th ACM Technical Symposium on Computer Science Education - SIGCSE '14, 169–174. https://doi.org/10.1145/2538862.2538978

Margolis, J. (2010). Stuck in the shallow end: Education, race, and computing. MIT press.

McGee, Steven, Randi McGee-Tekula, Jennifer Duck, Catherine McGee, Lucia Dettori, Ronald Greenberg, Eric Snow, Daisy Rutstein, Dale Reed, Brenda Wilkerson, Don Yanek, Andrew Rasmussen, and Dennis Brylow. 2018. "Equal Outcomes 4 All: A Study of Student Learning in ECS." SIGCSE '18 Proceedings of the 49th ACM Technical Symposium on Computer Science Education.

McGee, Steven, Ronald I. Greenberg, Randi McGee-Tekula, Jennifer Duck, Andrew M. Rasmussen, Lucia Dettori, and Dale F. Reed. 2019. "An Examination of the Correlation of Exploring Computer Science Course Performance and the Development of Programming Expertise." Pp. 1067–1073 in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19. New York, NY, USA: ACM.

Miller, L. D., Leen-Kiat Soh, and Markeya Peteranetz. 2019. "Investigating the Impact of Group Size on Non-Programming Exercises in CS Education Courses." Pp. 22–28 in.

Milliken, A., Cody, C., Catete, V., & Barnes, T. (2019). Effective Computer Science Teacher Professional Development: Beauty and Joy of Computing 2018. Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, 271–277. https://doi.org/10.1145/3304221.3319779

Morreale, Patricia, Goski, Catherine, Jimenez, Luis, and Stewart-Gardiner, Carolee. 2012. "Measuring the Impact of Computational Thinking Workshops on High School Teachers." Journal of Computing Sciences in Colleges.

Neutens, Tom and Francis Wyffels. 2018. "Bringing Computer Science Education to Secondary School: A Teacher First Approach." Pp. 840–45 in.

Peteranetz, Markeya, Shiyuan Wang, Duane Shell, Abraham E. Flanigan, and Leen-Kiat Soh. 2018. "Examining the Impact of Computational Creativity Exercises on College Computer Science Students' Learning, Achievement, Self-Efficacy, and Creativity | Request PDF from the Authors." Retrieved January 8, 2020 (https://www.researchgate.net/publication/323328484_Examining_the_Impact_of_Computational_Creativity_Exercises_on_College_Computer_Science_Students'_Learning_Achievement_Self-Efficacy_and_Creativity).

Peteranetz, Markeya, Patrick M. Morrow, and Leen-Kiat Soh. 2020. Development and validation of the computational thinking concepts and skills test. In Proceedings of ACM SIGCSE conference (SIGCSE'20), March 11-14, 2020, Portland, OR, USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366813.

Pollock, L., Mouza, C., Czik, A., Little, A., Coffey, D., & Buttram, J. (2017). From Professional Development to the Classroom: Findings from CS K-12 Teachers. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 477–482. https://doi.org/10.1145/3017680.3017739

Salac, Jean, Max White, Ashley Wang, and Diana Franklin. 2019. "An Analysis Through an Equity Lens of the Implementation of Computer Science in K-8 Classrooms in a Large, Urban School District." Pp. 1150–1156 in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19. New York, NY, USA: ACM.

Sengupta, P., Kinnebrew, J., Basu, S., Biswas, G., Clark, D., Sengupta, P., Kinnebrew, J., Basu, S., & Biswas, G. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. Education and Information Technologies, 18, 351–380. https://doi.org/10.1007/s10639-012-9240-x

Shell, D.F., Soh, L.K., Flanigan, A.E., Peteranetz, M.S. and Ingraham, E., (2017), Improving Students' Learning and Achievement in CS Classrooms through Computational Creativity Exercises that Integrate Computational and Creative Thinking. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 543-548). ACM.

Vogel, Sara, Rafi Santo, and Dixie Ching. 2017. "Visions of Computer Science Education: Unpacking Arguments for and Projected Impacts of CS4all Initiatives." Pp. 609–614 in Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, SIGCSE '17. New York, NY, USA: ACM.

Wozney, Lori, Vivek Venkatesh, and Philip Abrami. 2006. "Implementing Computer Technologies: Teachers' Perceptions and Practices." Journal of Technology and Teacher Education 14(1):173–207.

Yadav, Aman, Susanne Hambrusch, Tim Korb, and Sarah Gretter. 2013. "Professional Development for CS Teachers: A Framework and Its Implementation." Future Directions in Computing Education Summit.

2019 State of Computer Science Education. (2019). Retrieved from https://advocacy.code.org/

Appendix

Appendix A Quizzes

A.1 Cohort 1 Quizzes

A.1.1 Cohort 1 – Quiz 1: Conditionals

Question 1 1 pts

What is the expected output of the following code block when the user inputs **10.67**?

```
x = int(float(input()))

if x < 10:
    output = 1
elif x == 10:
    output = 2
else:
    output = 3

print(output)
```

- ○ 3
- ○ 2
- ○ 1

Question 2 1 pts

What is the expected output of the following code block?

```
x = 5
y = 8
z = 1

if x >= 6:
    output = 1
elif y < 10 and z < 5:
    output = 2
elif z == 1:
    if x >= 10:
        output = 3
    else:
        output = 4
else:
    output = 5

print(output)
```

- ○ 1
- ○ 4
- ○ 3
- ○ 5
- ○ 2

A.1.2 Cohort 1 – Quiz 2: Arrays & Loops

Question 1 1 pts

What is the expected output of the following code block?

```
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(len(arr))
```

- ○ 8
- ○ 10
- ○ 9
- ○ 7

Question 2 1 pts

What is the expected output of the following code block?

```
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
output = []

array_length = len(arr)
i = array_length - 1
while i >= 0:
    output.append(arr[i])
    i -= 1

print(output)
```

- ○ [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
- ○ [8, 7, 6, 5, 4, 3, 2, 1, 0]
- ○ [9, 8, 7, 6, 5, 4, 3, 2, 1]
- ○ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Question 3 1 pts

What is the expected output of the following code block?

```
arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
output = []

array_length = len(arr)
i = array_length - 1
while i > 0:
    if i < 5:
        output.append(arr[array_length - i])
    else:
        output.append(arr[i])
    i -= 1

print(output)
```

- ○ [0, 1, 2, 3, 4, 4, 3, 2, 1, 0]
- ○ [0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0]
- ○ [9, 8, 7, 6, 6, 7, 8, 9]
- ○ [9, 8, 7, 6, 5, 6, 7, 8, 9]

A.1.3 Cohort 1 – Quiz 3: Functions

Question 1 1 pts

What is the expected output?

```
def should_i_kayak(rain, wind):
    if rain or wind > 20:
        print("Heck no, I ain't yakin'!")
    else:
        print("Get the keys, we're going yakin'!")


should_i_kayak(False, 10)
```

- ○ Heck no, I ain't yakin'!
- ○ Get the keys, we're going yakin'!

Question 2 1 pts

What is the expected output?

```
def should_i_kayak(rain, wind):
    if rain or wind > 20:
        print("Heck no, I ain't yakin'!")
    else:
        print("Get the keys, we're going yakin'!")


should_i_kayak(False, 40)
```

- ○ Get the keys, we're going yakin'!
- ○ Heck no, I ain't yakin'!

Question 3 1 pts

What would you rename this function instead of "my_func" to make it more descriptive based on what the function does?

```
def my_func(x):
    x = x / 2
    return x
```

- ○ multiple_by_2
- ○ add_2
- ○ divide_by_2
- ○ subtract_2

Question 4 1 pts

What is the expected output of the following code block?

```
def my_func(x):
    x = x / 2
    return x


num = 64
count = 0

while num > 1:
    count += 1
    num = my_func(num)


print(count)
```

- ○ 7
- ○ 6
- ○ 2
- ○ 5

A.1.4 Cohort 1 – Quiz 4: Sort & Search

Question 1 1 pts

After two passes of the Exchange Sort, what should the following list be? (in an ascending order sort)

[3, 9, 8, 6, 4, 1, 10]

- ○ [3, 6, 4, 1, 8, 9, 10]
- ○ [3, 8, 6, 4, 1, 9, 10]
- ○ [3, 8, 9, 6, 4, 1, 10]
- ○ [3, 1, 8, 6, 4, 9, 10]

Question 2 1 pts

After two insertion operations of the Insertion Sort, what should the following list be? (in an ascending order sort)

[3, 9, 8, 6, 4, 1, 10]

- ○ [3, 8, 9, 6, 4, 1, 10]
- ○ [3, 6, 4, 1, 8, 9, 10]
- ○ [3, 1, 8, 6, 4, 9, 10]
- ○ [3, 8, 6, 4, 1, 9, 10]

Question 3 1 pts

After two passes of the Selection Sort, what should the following list be? (in an ascending order sort)

[3, 9, 8, 6, 4, 1, 10]

- ○ [3, 1, 8, 6, 4, 9, 10]
- ○ [3, 8, 9, 6, 4, 1, 10]
- ○ [3, 6, 4, 1, 8, 9, 10]
- ○ [3, 8, 6, 4, 1, 9, 10]

A.2 Cohort 2 Quizzes

A.2.1 Cohort 2 – Quiz 1: Functions

Question 1 1 pts

What is the expected output of this program?

```
function main(){
    wearCoat();
    goOutside();
}
function goOutside(){
    print("I'm outside now");
}
function wearCoat(){
    print("Putting on my coat");
}
```

- ○ I'm outside now!
- ○ Putting on my coat!
- ○ Putting on my coat!
  I'm outside now!
- ○ I'm outside now!
  Putting on my coat!

Question 2 — 1 pts

What is the expected output of this program?

```
function main(){
    print(myFunction("hello"));
}
function myFunction(input){
    var output = input + input;
    return output;
}
```

- ○ hellohello
- ○ hello
  hello
- ○ inputinput
- ○ output

Question 3 — 1 pts

What would you rename this function instead of "my_func" to make it more descriptive based on what the function does?

```
function my_func(x) {
    x = x / 2;
    return x;
}
```

- ○ add_2
- ○ subtract_2
- ○ multiple_by_2
- ○ divide_by_2

Question 4 — 1 pts

What is the expected output of this program?

```
function main() {
    var result = mystery(4, 2);
    print(result);
}
function mystery(a, b) {
    a = mystery2(a);
    b = mystery2(b);
    return a + " " + b;
}
function mystery2(x){
   x = x / 2;
   return x;
}
```

- ○ 4 2
- ○ 2 1
- ○ 2 2
- ○ 1 2

A.2.2 Cohort 2 – Quiz 2: Loops & If-Statements

Question 1 1 pts

Which column in **Table A** contains the correct output of this program?

```
function main() {
  for(var counter = 1 ; counter <= 7 ; counter++ ){
    print(counter);
  }
}
```

Table A

A	B	C	D
1 2 3 4 5 6	0 1 2 3 4 5 6	1 2 3 4 5 6 7	counter counter counter counter counter counter counter

- ○ A
- ○ B
- ○ C
- ○ D

Question 2 1 pts

What values of a, b, and c, will give you an output of "C"?

```
function main() {
  var a = ?;
  var b = ?;
  var c = ?;
  if( a > b ){
      if (a < c){
        print("A");
      } else {
        print("B");
      }
  } else {
      if(b > c){
          print("C");
      } else {
          print("D");
      }
  }
}
```

- ○ a = 10
  b = 20
  c = 15
- ○ a = 10
  b = 15
  c = 20
- ○ a = 20
  b = 10
  c = 15
- ○ a = 20
  b = 15
  c = 10

Question 3 1 pts

Which condition could fill in the blank to make this program print "YAY!"?

```
function main() {
  var text = "This sentence.";

  if(  _______  ){
      print("YAY!");
  } else {
      print("Aww");
  }
}
```

- ○ text == "This"
- ○ text.includes("word")
- ○ text.length > 3
- ○ text != "This sentence."

Question 4 1 pts

Which code snippet could replace the questions marks (???) to make this program print "ABABABAB"?

```
function main() {
  var text = "";

  while(text.length < 7){
      ???
  }
  print(text);
}
```

- ○ text = text + "A";
- ○ text = text + "AB";
- ○ text = text + "ABA";
- ○ text.includes("ABABABAB")

Question 5 1 pts

Which column in **Table B** contains the output of the following code:

```
function main(){
    for(var i=0 ; i < 10 ; i++){
        if( i % 2 == 0){
            print(i);
        }
    }
}
```

Table B

A	B	C	D
0 2 4 6 8 10	1 3 5 7 9	0 1 2 3 4 5 6 7 8 9	0 2 4 6 8

- ○ A
- ○ B
- ○ C
- ○ D

A.2.3 Cohort 2 – Quiz 3: Loops & Lists

Question 1 1 pts

What is the expected output of this program?

```
function main() {
    var array = [0,1,2,3,4,5,6,7,8,9];
    print(array.length);
}
```

- ○ 8
- ○ 9
- ○ 10
- ○ 11

Question 2 1 pts

What is the expected output of this program?

```
function main() {
    var array = ["a", "b", "c", "d", "e", "f", "g", "h"];
    print(array[3]);
}
```

- ○ c
- ○ d
- ○ 3
- ○ 4

Question 3 1 pts

Which code can be placed in the blank such that **array** is filled with each word separately in their own array element?

```
function main() {
    var text = "the quick brown fox jumps over the lazy dog";
    var array = _____;
}
```

- ○ split(text)
- ○ text.split()
- ○ text.split(" ")
- ○ text.slice(" ")

Question 4 1 pts

Which of the following loop declarations is correct to loop through all of the index values for an array and print them each on their own line?

- for(var i=0; i<length; i++){
  print(array[i]);
  }

- for(var i=0; i<array; i++){
  print(array[i]);
  }

- for(var i=0; i<array.length; i++){
  print(array);
  }

- for(var i=0; i<array.length; i = i + 2){
  print(array[i]);
  }

- for(var i=0; i<array.length; i++){
  print(array[i]);
  }

Question 5 1 pts

What is the expected output from this code:

```
function main() {
    var array = [];
    for(var i=0; i < 7; i++){
        array.push(i);
    }
    for(var i=0; i < 7; i++){
        array[i] = array[i] * 3;
    }
    print(array[4]);
}
```

- ○ "array[4]"
- ○ 4
- ○ 12
- ○ 15

Question 6 1 pts

What is the value of the variable count at the end of running this code:

```
function main() {
    var array = [3, 1, 2, 5, 4, 8];
    var count = 0;
    for(var i=0; i < array.length; i++){
        if(i == array[i]){
            count = count + 1;
        }
    }
}
```

- ○ 2
- ○ 3
- ○ 6
- ○ 8

A.2.4 Cohort 2 – Quiz 4: Functions, Loops, & Recursion

Question 1 1 pts

What is the expected output of this program?

```
function main(){
    print(func(machine("hello")));
}

function func(text){
      return "00" + text;
}

function machine(text){
    return text.substring(1);
}
```

- ○ 0hello
- ○ 00hell
- ○ 00ello
- ○ 0ello

Question 2 1 pts

What is the expected output of this program?

```
function main(){
    print(recur("strawberry", 3));
}

function recur(text, num){
    if(text.length <= num){
        return text;
    } else {
        var newText = text.substring(1) + "a";
        return recur(newText, num+1);
    }
}
```

- ○ rryaaaaaaa
- ○ straaaaaaa
- ○ str
- ○ aaa
- ○ erryaaaaaa

Question 3 1 pts

What is the expected output of this program?

```
function main(){
    var tracker = 0;
    for( var outer = 0 ; outer < 4 ; outer++ ){
        for( var inner = 0 ; inner < outer ; inner++ ){
            tracker++;
        }
    }
    print(tracker);
}
```

- ○ 0
- ○ 3
- ○ 6
- ○ 10

Question 4 1 pts

Which two function calls should go in the blanks so that the program prints output of 20?

```
function main(){
    print(___?___(___?___(funcA(4))));
}
function funcA(num){
    return num*2
}
function funcB(num){
    return num+2;
}
function funcC(num){
    return num/2;
}
```

- ○ funcA(funcB(
- ○ funcB(funcA(
- ○ funcC(funcA(
- ○ funcB(funcC(

Appendix B Assignments

B.1 Cohort 1 Assignments

B.1.1 Cohort 1 – Assignment 1

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Programming Assignment 1: Hello World

Points: 100 points. Assignment Date: June 3, 2019 Due Date: Midnight June 3, 2019

Objectives

1. To familiarize with writing and running Python programs and the Python environment
2. To familiarize with the use of conditionals and branching
3. To familiarize with data structures
4. To familiarize with standard input/output in Python
5. To be exposed to the use of built-in functions

Part 1: csce805thomework01part01.py

Problem

Write a program that will prompt the user for a choice between 1 and 5, with each choice being a different language. When the user enters a choice, the program displays "Hello, World" in the chosen language to the screen. After that, the program exits.

Here are some additional requirements:

- The program is required to display an explanation of the choices in the beginning before prompting the user for a choice.
- The program is required to display the user's choice – this is known as "echoing user input".
- If the user enters a choice that is invalid, the program is required to display an error message (such as "Sorry, your input is invalid.") and then exit the program.
- The program should always display a message ("Thank you for using the Multilingual HelloWorld program. Bye!") before exiting.
- You are required to use five languages (English, Spanish, Italian, Malay, and Dutch) and the correct translations for "Hello, World" (Hint: Try using Google Translate).
    - Make sure you include the comma between "Hello" and "World" for each of your translations
    - Make sure both "Hello" and "World" begin with capital letters
- You must document your program (see https://devguide.python.org/documenting/).
    - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
    - Inline comments in the program
- IMPORTANT TIP: For the webgrader to work, your prompts must be the same, word for word and line for line, as the example input/output.
    - Some of the prompts from the example below have been provided for you.

Part 1: Example Input/Output

```
Welcome to the Multilingual HelloWorld program!
The language choices are:
1. English
2. Spanish
3. Italian
4. Malay
5. Dutch
Please enter your choice of language:
4

You have chosen language #4: Malay.
Here is "Hello, World" in Malay: "Hai, Dunia"
Thank you for using the Multilingual HelloWorld program.  Bye!
```

Part 2 (BONUS): csce805thomework01part02.py

Problem

Modify your code from part 1 to make a program that will prompt the user for a choice between 1 and 5, with each choice being a different language. After the language is chosen, the program must display the user's choice of language.

After the program displays the chosen language, the program should then ask for 'H' or 'G', with 'H' being the translation for "Hello, World" and 'G' being the translation for "Goodbye, World". Once the salutation is selected, the program should display the translation for the salutation. After that, the program exits.

Here are some additional requirements:

- If the user enters a choice that is invalid on either part 1 or part 2, the program is required to display an error message (such as "*Sorry, your input is invalid.*") and then exit the program.
- The program should always display a new message ("*Thank you for using the Multilingual HelloWorld/GoodbyeWorld program. Bye!*") before exiting.
- You are required to use five languages (1. English, 2. Spanish, 3. Italian, 4. Malay, and 5. Dutch) and the correct translations for "Hello, World" and "Goodbye, World" (Hint: Try using Google Translate).
- You must document your program (see https://devguide.python.org/documenting/).
    - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
    - Inline comments in the program
- IMPORTANT TIP: For the webgrader to work, your prompts must be the same, word for word and line for line, as the example input/output.
    - Some of the prompts from the example below have been provided for you.

Part 2: Example Input/Output

```
Welcome to the Multilingual HelloWorld/GoodbyeWorld program!
The language choices are:
1. English
2. Spanish
3. Italian
4. Malay
5. Dutch
Please enter your choice of language:
2
You have chosen language #2: Spanish
The salutation choices are:
H. Hello
G. Goodbye
Please enter your choice of salutation:
H
Here is "Hello, World" in Spanish: "Hola, Mundo"
Thank you for using the Multilingual HelloWorld/GoodbyeWorld program. Bye!
```

Handin

1. The submission deadline for all handins is **June 3rd, 2019, 11:59:59 pm**.
2. You are required to handin all program files.
3. You are required to handin online the above using http://cse.unl.edu/handin/
4. You can check your submissions at https://cse.unl.edu/~cse805t/grade/

Grading

- Part 1 (100 points)
  - Full points will be rewarded for completing part 1 of the assignment with zero diffs on the webgrader
    - Part 1 has 8 test cases
    - Solving 7/8 cases with zero diffs will get you 87.5% of the 100 possible points (100 * .875 = 87.5 points)
    - Solving 6/8 cases with zero diffs will get you 75.0% of the 100 possible points (100 * .75 = 75.0 points) and so on.
- Part 2 (10 bonus points)
  - Completion of part 2 will be rewarded with bonus points. The max amount of bonus points possible is 10 points.
    - Part 2 has 24 test cases.
    - Solving 23/24 cases with zero diffs will get you 95.8% of the bonus points (10 * .958 = 9.58 points).
- If you are not pleased with the grade you receive for this assignment, we will reopen the assignment submission after the last day of class, Friday, June 7th, for resubmissions.
- All resubmissions are due the same day as the final project, **July 7th, 2019, 11:59:59 pm**

Think About

Now, think about what if we want to build a system that automatically translates an English word, phrase, or sentence into any of the five chosen languages or *any* language? How should we break down this problem? First, do we break the problem down into translating a word, translating a phrase, and

translating a sentence? Why? Or why not? And then what? Language by language? Could we somehow make use of Google Translate API? (What is an API?)

B.1.2 Cohort 1 – Assignment 2

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Programming Assignment 2: Hello Data

Points: 100 points. Assignment Date: June 4, 2019 Due Date: Midnight June 4, 2019

Objectives

1. To familiarize with writing and running Python programs and the Python environment
2. To familiarize with the use of loops (e.g., the for and while loops)
3. To familiarize with data structures, particularly arrays/lists/2-D arrays
4. To familiarize with file input/output in Python
5. To be exposed to the use of built-in functions
6. To be exposed to the use of built-in modules or packages (e.g., import math)
7. To familiarize with the use of online documentations on Python

Part 1: csce805thomework02part01.py

Problem

Write a program that will prompt the user to enter a series of numbers between -999 and 999 (inclusive). When the user enters a number outside the specified range, the program will stop prompting. After that, the program will display several statistics: (1) the number of numbers entered, (2) the average value of the series of numbers, (3) the minimum value of the numbers, (4) the maximum value of the numbers. Here are some additional requirements:

- The program is required to display an explanation of the program (e.g., its expected range of input values) in the beginning before prompting the user for a number.
- The program is required to use at least one loop structure.
- Your program is ***not*** allowed to use Python's built-in functions that compute the minimum, maximum, and average values of a series of numbers.
- If the user input is something other than an integer, print the error message: "Invalid Input", then stop prompting and print the statistics.
- If there are no numbers to perform statistics on, print "No statistics since no numbers entered." and exit the program.
- You must document your program (see https://devguide.python.org/documenting/).
  - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
  - Inline comments in the program
- IMPORTANT TIP: For the webgrader to work, your prompts must be the same, word for word and line for line, as the example input/output.
  - Some of the prompts from the example below have been provided for you.

Part 1: Example Input/Output

```
Welcome to the Loopy Statistical Analysis program!
This program generates five statistics based on the series of numbers you enter.
```

```
Valid numbers are between -999 and 999.  When you are done entering your numbers,
please simply enter a number outside the above specified range.
1. Please enter a number:
10
2. Please enter a number:
20
3. Please enter a number:
30
4. Please enter a number:
40
5. Please enter a number:
50
6. Please enter a number:
999999
Exiting entries...

Here are the statistics:
-------------------------
Number of numbers: 5
Minimum value: 10
Maximum value: 50
Average value: 30.0
```

Part 2: csce805thomework02part02.py (BONUS)

Problem

This part is bonus and is not required. Write a program that works similarly to part **1**. The program will prompt the user to enter a series of numbers between -999 and 999 (inclusive). In this part of the assignment, you must use a 2D array. All even numbers from the series of numbers must be stored in the **1st** dimension of the array. All odd numbers from the series of numbers must be stored in the 2nd dimension of the array. When the user enters a number outside the specified range, the program will not add that number to the array and the program will stop prompting. After that, the program will display several statistics for each of the two dimensions of the array. First for the even numbers (or the **1st** dimension). Then print the same statistics for the odd numbers (the 2nd dimension). Here are the statistics to print out: (1) the number of numbers entered, (2) the average value of the series of numbers, (3) the minimum value of the numbers, and (4) the maximum value of the numbers. Here are some additional requirements:

- The program is required to display an explanation of the program (e.g., its expected range of input values) in the beginning before prompting the user for a number.
- The program is required to use at least one loop structure.
- Your program is ***not*** allowed to use Python's built-in functions that compute the minimum, maximum, and average.
- If the user input is something other than an integer, print the error message: "Invalid input…", then stop prompting and print the statistics.
- If there are no numbers to perform statistics on, print "No statistics since no numbers entered." and exit the program.
- You must document your program (see https://devguide.python.org/documenting/).
    - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
    - Inline comments in the program

- IMPORTANT TIP: For the webgrader to work, your prompts must be the same, word for word and line for line, as the example input/output.
    - Some of the prompts from the example below have been provided for you.

Part 2: Example Input/Output

```
Welcome to the Loopy Statistical Analysis program!
This program generates five statistics based on the series of numbers you
enter.
Valid numbers are between -999 and 999.  When you are done entering your
numbers, please simply enter a number outside the above specified range.
1. Please enter a number:
2
2. Please enter a number:
3
3. Please enter a number:
4
4. Please enter a number:
5
5. Please enter a number:
6
6. Please enter a number:
999999
Exiting entries...

Here are the statistics:
--------------------------
Evens
--------------------------
Number of numbers: 3
Minimum value: 2
Maximum value: 6
Average value: 4.00
--------------------------
Odds
--------------------------
Number of numbers: 3
Minimum value: 1
Maximum value: 5
Average value: 3.00
```

Handin

1. The submission deadline for all handins is Midnight June 4, 2019. **Late handins will *not* be accepted or graded.**
2. You are required to handin a screen capture of your "testing session" using your program.
3. You are required to handin a python file named: csce805thomework02.py.
4. You are required to handin online the above using http://cse.unl.edu/handin/

5. You can check your submissions at https://cse.unl.edu/~cse805t/grade/

Grading

- Part 1 (100 points)
  - Full points will be rewarded for completing part 1 of the assignment with zero diffs on the webgrader
    - Part 1 has 50 test cases
    - Solving 45/50 cases with zero diffs will get you 90.0% of the 100 possible points (100 * .900 = 90.0 points)
    - Solving 40/50 cases with zero diffs will get you 80.0% of the 100 possible points (100 * .800 = 80.0 points) and so on.
- Part 2 (10 bonus points)
  - Completion of part 2 will be rewarded with bonus points. The max amount of bonus points possible is 10 points.
    - Part 2 has 39 test cases.
    - Solving 45/50 cases with zero diffs will get you 90.0% of the 10 possible points (10 * .900 = 9.0 points)
    - Solving 40/50 cases with zero diffs will get you 80.0% of the 10 possible points (10 * .800 = 8.0 points) and so on.
- If you are not pleased with the grade you receive for this assignment, we will reopen the assignment submission after the last day of class, Friday, June 7th, for resubmissions.
- All resubmissions are due the same day as the final project, **July 7th, 2019, 11:59:59 pm**

Think About

Now, think about what if we want to build a system that computes statistics for thousands of numbers, or even, millions of numbers. Do we want to input the numbers one by one manually? What would be some common challenges or issues with that approach? Are there other ways to input the data? By the same token, what if we want to generate different types of statistics, for different subsets of numbers, and thus will produce many different tables? How should we store the tables of numbers? Do we want to copy the numbers down one by one and re-enter them, say, into an Excel spreadsheet? (Hint: Think about file I/O.) (Hint: Think about Big Data, Scalability, and Reliability, and how they relate to Informatics.)

B.1.3 Cohort 1 – Assignment 3

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Programming Assignment 3: Hello Data Files

Points: 100 points. Assignment Date: June 5, 2019 Due Date: Midnight June 5, 2019

Objectives

1. To familiarize with writing and running Python programs and the Python environment
2. To familiarize with the use of loops (e.g., the for and while loops)
3. To familiarize with data structures, particularly arrays/lists
4. To familiarize with using functions in Python
5. To be exposed to the use of built-in functions
6. To familiarize with the use of online documentations on Python

Problem Part 1: csce805thomework03part01.py

Take the code you wrote for assignment 2 and modify it to use a function to calculate the statistics. Just like in assignment 2, you will need to calculate several statistics: (1) the number of numbers entered, (2) the average value of the series of numbers, (3) the minimum value of the numbers, (4) the maximum value of the numbers. Here are some additional requirements:

- Your program is required to use a custom-built function to calculate and report the statistics.
  - The function must take the list of numbers to analyze as an input argument.
- The program is required to display an explanation of the program (e.g., its expected range of input values) in the beginning before prompting the user for a number.
- The program is required to use at least one loop structure.
- Your program is not allowed to use Python's built-in functions that compute the minimum, maximum, and average values of a series of numbers.
- If the user input is something other than an integer, print the error message: "Invalid Input", then stop prompting and print the statistics.
- If there are no numbers to perform statistics on, print "No statistics since no numbers entered. "and exit the program.
- In the end, you output should look the same or similar to that of your assignment 2.
- You must document your program (see https://devguide.python.org/documenting/).
  - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
  - Inline comments in the program

Example Input/Output: Part 1:

```
Welcome to the Loopy Statistical Analysis program!
This program generates five statistics based on the series of numbers you enter.
Valid numbers are between -999 and 999.  When you are done entering your numbers, please
simply enter a number outside the above specified range.
1. Please enter a number:10
2. Please enter a number:20
3. Please enter a number:30
4. Please enter a number:40
5. Please enter a number:50
6. Please enter a number:999999
Exiting entries ...

Here are the statistics:
Number of numbers: 5
Minimum value: 10
Maximum value: 50
Average value: 30.00
```

Problem Part 2: csce805thomework03part02.py

This part is bonus and is not required. Write a program that works similarly to part 1. The program will prompt the user to enter a series of numbers between -999 and 999 (inclusive). Then, in the function you will calculate the same statistics as part 1. The difference in part 2 is, after the user enters a number outside of the range, then you will prompt the user to input whether they would like stats for the positive numbers in the list or the negative numbers in the list. You will add another parameter to the function to take in the new input the user gives. The users input will determine if you calculate the statistics for the positive numbers or the negative numbers. Here are some additional requirements:

- You must add a second parameter to the function that calculates the statistics
    - The first parameter will still hold the list of numbers that the user inputted.
    - The second parameter should be a *string* value of either 'positive or 'negative'.
        - If you pass 'positive' to the statistics function, then the program will calculate the statistics for only the positive values.
        - If you pass 'negative' to the statistics function, then the program will calculate the statistics for only the negative values.
        - If something other than 'positive' or 'negative' is passed into the function, then the program should print out "Invalid Input" and quit without printing any statistics

- The program is required to display an explanation of the program (e.g., its expected range of input values) in the beginning before prompting the user for a number.
- The program is required to use at least one loop structure.
- Your program is not allowed to use Python's built-in functions that compute the minimum, maximum, and average.
- If the user input is something other than an integer, print the error message: "Invalid input", then stop prompting and print the statistics.
- If there are no numbers to perform statistics on, print "No statistics since no numbers entered. "and exit the program.
- You must document your program (see https://devguide.python.org/documenting/).
    - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
    - Inline comments in the program

Example Input/Output: Part 2 (BONUS):

```
Welcome to the Loopy Statistical Analysis program!
This program generates five statistics based on the series of numbers you enter.
Valid numbers are between -999 and 999.  When you are done entering your numbers, please
simply enter a number outside the above specified range.
1. Please enter a number:10
2. Please enter a number: -10
3. Please enter a number:20
4. Please enter a number: -20
5. Please enter a number: 999999
Exiting entries ...
Would you like stats for positive or negative numbers?positive
Here are the statistics for the positive numbers:
Number of numbers:  2
Minimum value: 10
Maximum value: 20
Average value: 7.50
```

Handin

1. The submission deadline for all handins is Midnight June 5, 2019. **Late handins will *not* be accepted or graded.**
2. You are required to handin all program files.
3. You are required to handin online the above using **http://cse.unl.edu/handin/**

Grading

- Part 1 (100 points)
  - Full points will be rewarded for completing part 1 of the assignment for accurate solutions
    - We will be testing each solution by hand. We will be looking for
      - Accurate calculations
      - Good use of a custom function
      - Correct parameters used
- Part 2 (10 bonus points)
  - Full points will be rewarded for completing part 1 of the assignment for accurate solutions
    - We will be testing each solution by hand. We will be looking for
      - Accurate calculations
      - Good use of a custom function
      - Correct parameters used
- If you are not pleased with the grade you receive for this assignment, we will reopen the assignment submission after the last day of class, Friday, June 7th, for resubmissions.
- All resubmissions are due the same day as the final project, **July 7th, 2019, 11:59:59 pm**

Think About

Now, think about what if we want to modify how the columns are arranged (e.g., adding new attributes, removing some attributes) or how the rows are to be filtered so that only certain subsets are being considered for analyses. Do we need then to create many versions of the csv input file? What are some potential pitfalls with maintaining many versions of the csv input file? Furthermore, think about the output file format. What if our users or customers want something different, e.g., in a different format, or to derive or visualize different types of results from your analyses? How would you then create an output file that is more universally useful? (Hint: Think about databases.)

B.1.4 Cohort 1 – Assignment 4

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Programming Assignment 4: Hello Sort

Points: 100 points. Assignment Date: June 6, 2019 Due Date: Midnight June 6, 2019

Objectives

1. To familiarize with writing and running Python programs and the Python environment
2. To familiarize with the use of loops (e.g., the for and while loops)
3. To familiarize with data structures, particularly arrays/lists
4. To familiarize with sorting algorithms
5. To be exposed to the use of built-in functions
6. To be exposed to the use of built-in modules or packages (e.g., import matplotlib)
7. To familiarize with the use of online documentations on Python

Part 1: exchangeSortPart01.py, selectionSortPart01.py, insertionSortPart01.py - Problem

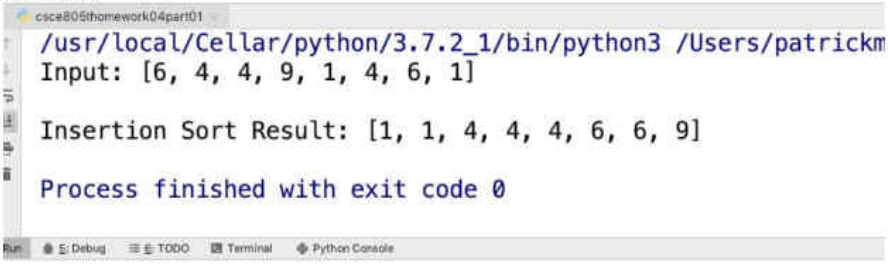
Modify the provided programs to sort the lists in descending order rather than ascending order. Here are some additional requirements:

- Each function will take in an array of integers (unsorted) and return an array of integers (sorted in descending order)
- You must print out what kind of sort you used and the sorted array.
- You must document your program (see https://devguide.python.org/documenting/).
    - Name, Date, Affiliation, a description of the program, what inputs does it need, what outputs does it generate
    - Inline comments in the program

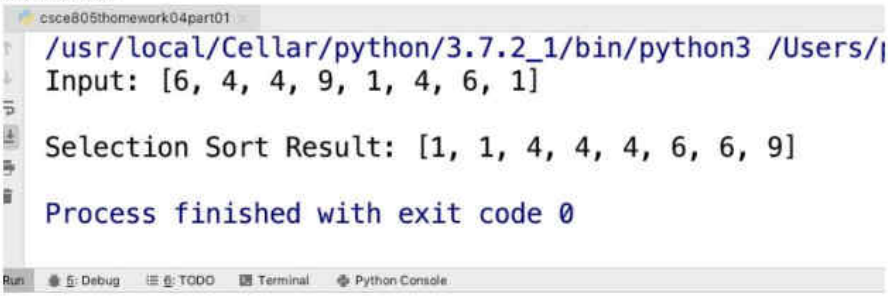
Example Input/Output: Part 1: exchangeSortPart01.py, selectionSortPart01.py, insertionSortPart01.py

Pick One

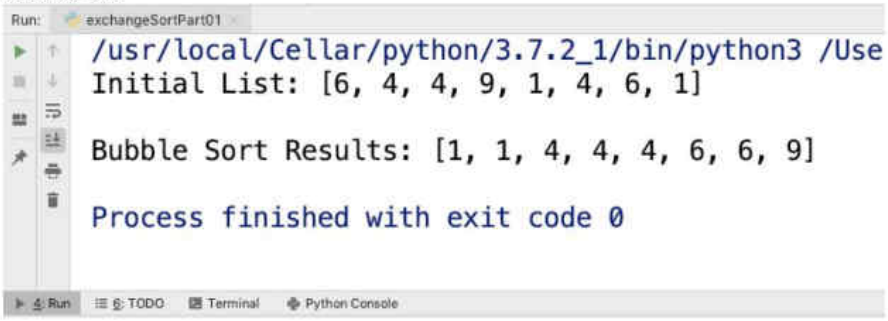
- Insertion Sort



- Selection Sort



- Exchange Sort



Handin

1. The submission deadline for all handins is Midnight June 6, 2019. **Late handins will *not* be accepted or graded.**
2. You are required to handin all program files.
3. You are required to handin online the above using **http://cse.unl.edu/handin/**
4. You can check your submissions at **https://cse.unl.edu/~cse805t/grade/**

Grading

- Part **1** (100 points)
  - Full points will be rewarded for completing part **1** of the assignment with an accurate sorted print out and a written sorting function.

- If you are not pleased with the grade you receive for this assignment, we will reopen the assignment submission after the last day of class, Friday, June 7th, for resubmissions.
- All resubmissions are due the same day as the final project, **July 7th, 2019, 11:59:59 pm**

Think About

Think about why some sorting algorithms may be better for large data sets but worse for smaller datasets. Why does Python use quick sort as the default sorting algorithm for the language?

B.1.5 Cohort 1 – Teaching and Learning Assignment

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Teaching and Learning Assignment: Lecture Design

Points: 100 points. Assignment Date: June 3, 2019 Due Date: Midnight June 7, 2019

Objectives

1. To familiarize with creating lectures to deliver Computational Thinking and Computer Science content
2. To familiarize with developing activities for such a lecture

Description

In this assignment, you will be paired with 4-5 team members teaching the same grade as yourself. Together, you will design a lecture plan for a single day in your CS class. On Friday, June 7th, all groups will be paired with another group to share their lecture plan. Your lectures may be activities, small programming projects, PowerPoint slides, etc.. It's your classroom, so be creative!

Presentation: You will share your lecture with another group. Each group will describe their lecture plan for about 15-20 minutes. Each one of your group members needs to take part in the presentation. In your presentations, you will need to include:

- The grade level your content is geared towards
- At least one Computational Thinking skill:
  - Problem decomposition, Pattern recognition, Abstraction, Generalization, Algorithmic design, Evaluation
- At least one CS topic:
  - Variables, Arrays, Conditionals, Loops, Functions, Algorithms, Sorting, Searching, Recursion, Debugging, Complexity

If your lecture includes some online resource or activity, give a demo of the activity and explain how you will use it.

Lecture Materials: Lecture materials include anything you plan to use during the class (slides, handouts, online resources, etc..). You need to hand in everything you plan to use in the lecture. Along with the materials, you will need to include an itinerary detailing how the lecture will be delivered. This should contain enough detail for someone to recreate your lecture with this document. The itinerary will need to include:

- The length of the lecture
- Step by step detailed guide
  - Include length of each activity
  - What is done during each activity
- Links to online materials used

Handin

1. The submission deadline for handin is Midnight June 7, 2019. **Late handins will *not* be accepted or graded.**
2. You are required to handin all lecture files (including pdfs, slides, and handouts) and the itinerary.
3. You are required to handin online the above using http://cse.unl.edu/handin/

Grading

- Presentation (20 points)
- Lecture Materials (80 points)

Think About

How does your lecture engage your students? Since you have had the chance to switch roles and be the student rather than the teacher during this course, what tactics have you noticed that make learning CS more approachable? Can you implement some of these tactics to make the lecture more appealing to students? How can you improve this lecture going forward?

B.1.6 Cohort 1 – Final Project

CSCE805T Beginning Computer Science for Teachers
Summer 2019

Final Project: Assignment Design

Points: 100 points. Assignment Date: June 7th, 2019 Due Date: Midnight July 7th, 2019

Objectives

1. To familiarize with creating assignments based on your lectures to reinforce Computer Science concepts and Computational Thinking skills
2. To familiarize with anticipating the significance and expected difficulty of a particular Computer Science concept or a Computational Thinking skill in understanding a problem and developing a solution

Description

This assignment extends the Teaching and Learning Assignment. This time, you will work alone (not in groups) to design an assignment to follow your lesson plan. Based on the Computer Science concepts and Computational Thinking skills delivered in your Teaching and Learning Assignment, you will design a homework assignment that you can use in your classrooms. You are required to complete the following:

Assignment Specification This document should be similar to all the ones you have received for assignments in this course (including the one you are reading now). It should include your assignment's objectives and clear descriptions of each problem you would like the students to complete. In your assignment, you are also required to provide a bonus or extra credit section.

Solution For each problem in the assignment, you should complete a correct, working solution. If your assignment includes a programming problem, you need to write a correct solution program. If your assignment includes some other activities, you will need to detail what success looks like for each of the activities. This solution requirement also applies to the bonus or extra credit section of the assignment.

Sample Input/Output You will need to detail some sample inputs and what the expected output should be. This way, the students can check their work. That is, if they give the same input, they should get the same output. If possible, complete a few sample inputs/outputs more is better, but you are only required to complete at least one input/output. You will need to also provide the actual test inputs and outputs that will allow you to test your students' solutions and grade them accordingly.

Rubric You will need to provide a document describing how students' solutions will be evaluated. You need to detail which areas of the assignment will be graded and what you will be taking points off for.

Reflection You are required to write a report responding to the following prompts:

1. Describe how you think this assignment will improve the student's understanding of the Computer Science (CS) concepts and Computational Thinking (CT) skills you have chosen.
2. Discuss which CS concepts that you are *most* confident in in terms of knowledge content and the reasons behind your response.
3. Discuss which CT skills that you are *most* confident in in terms of knowledge content and the reasons behind your response.
4. Discuss which CS concepts that you are *least* confident in in terms of knowledge content and the reasons behind your response.
5. Discuss which CT skills that you are *least* confident in in terms of knowledge content and the reasons behind your response.
6. Discuss which CS concepts and/or CT skills that you feel the least confident in *teaching*, and the reasons behind your response.

Handin

1. The submission deadline for handin is Midnight July 7th, 2019. **Late handins will *not* be accepted or graded.**
2. You are required to handin all assignment files (Handouts, Rubric, Solution(s), Sample Input/Output, etc.).
3. You are required to handin online the above using **http://cse.unl.edu/handin/**

Grading

We will be looking to make sure that you have submitted the Assignment Specification, Solution, Sample Input/Output, and the Reflection. If it doesn't make sense for your assignment to have one or more of the documents, please include a text file describing why it doesn't need the document. If all documents are present, we will then grade each of the document based on quality. Your documents should be clear, thorough, and effectively cover one of the Computer Science and Computational Thinking topics.

Think About

Think about how you would approach the assignment you created. What skills are needed for you to complete the assignment? After completing the assignment, it might be best to go back to your lecture and modify it to make sure the students have the skills needed to complete the assignment.

B.2 Cohort 2 Assignments

B.2.1 Cohort 2 – Assignment 1

CSCE805T – Assignment 1: Formulas

Project Summary

In this project you will write a program to evaluate some math calculations.

Background Information

Area of a rectangle = bh
Perimeter of a rectangle = $2b + 2h$
Area of a circle = πr^2
Circumference of a circle = $2\pi r$

Project Details

Your program should get a list of numbers from the user as input, each on a different line.

- A rectangle's base length.
- A rectangle's height.
- A circle's radius.
- A number of seconds.

Once you have gotten the input, calculate the rectangle's area and perimeter, and the circle's area and circumference. Then convert the number of seconds into the correct number of hours, minutes, and remaining seconds. Each of your calculations should be printed with complete information about it.

Sample Input

4
7
3
8000

Sample Output (For the sample input given)

A 4 by 7 rectangle has area 28 and perimeter 22.
A circle with radius 3 has area 28.274333882308138 and circumference 18.84955592153876.
8000 seconds is 2 hours, 13 minutes, and 20 seconds.

Project Extensions

Once you get the basics done, work on this challenge if you have time. (It is not a required component, but is good for additional practice!)

- Add 4 more input spots to represent two coordinate points (x, y) Then calculate the distance between the two points and print it out.
  The distance between points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

B.2.2 Cohort 2 – Assignment 2

CSCE805T – Assignment 2: Functions

Project Summary

In this project you will write several functions that each achieve a specific goal.

Project Details

Your program will need to contain each of the following functions:

- time(s) - Given a single input representing a number of seconds, return the same text from your first program indicating the number of hours, minutes, and seconds.
- switcheroo(t) - Given a single input string, return a string with the first half switched with the second half. For example, "hello!" would return "lo!hel"
- inject(a,b) - Given two input strings, return a string with the second string put in the middle of the first. For example, inject("hello!","bye") would return "helbyelo!"
- fourthRoot(n) – given a single input number, return its fourth root (which is also the square root of the square root).

Sample Input/Output

Your program won't need to have any specific input given in the input box, just having the functions in the code will be sufficient. Though, you can use the input to test your functions.

Project Extensions

Once you get the basics done, work on this challenge if you have time. (It is not a required component, but is good for additional practice!)

- mask(t,n) – given an input string and an input number representing a percentage (a decimal from 0 to 1) return just that percentage of the string. For example, mask("tenletters",0.7) would return "tenlett". And mask("four",0.75) would return "fou". You can round up or down as you wish if the percentage doesn't work evenly for the number of letters.

B.2.3 Cohort 2 – Assignment 3

CSCE805T – Assignment 3: Conditionals

Project Summary

In this project you will write a function to determine whether or not a given year is a leap year, along with a couple other specific tasks involving conditional checks.

Background Information

A leap year is any year that is a multiple of 4, excluding multiples of 100 except for multiples of 400 (which ARE, in fact, leap years).

Project Details

Your program will need to contain each of the following functions:

- Given a single input representing a year, return whether or not the year is a leap year.
- A "fortune teller" with at least 4 different outcomes from at least 2 different decisions. (They don't have to be sensical...have some fun with it!) You can use numerical choices or words.

The program should determine which task to demonstrate based on the first line of the input box.

Sample Input/Output

Input	Output
leap year 2400	The year 2400 is a leap year.
leap year 2100	The year 2100 is not a leap year.
fortune teller 1 2	There are good things in your future.
fortune teller 1 1	Be careful around pools of magma.
fortune teller 2 1	You will find happiness in what you already have.

Project Extensions

Once you get the basics done, work on these challenges if you have time. (They are not required components, but are good for additional practice!)

- isFactor(a,b) - Write a function with two inputs and determine if the first number is a factor of the second number. (A number is a factor of another number if it divides it with a remainder of 0.)
- breakUp(t,n) - Write a function with two inputs: the first a string and the second a number, and determine if the string can be broken evenly into that number of groups. For example, breakUp("hello",3") would return false because 5 letters cannot be broken up evenly into 3 groups. breakUp("tenLetters",2) returns true but breakUp("tenLetters",3) returns false.

B.2.4 Cohort 2 – Assignment 4

CSCE805T – Assignment 4: Loops

Project Summary

In this project you will create functions to practice loops involving strings.

Background Information

A palindrome is a string or number that is the same backwards as it is forwards.

Project Details

Your program will need to contain each of the following functions, solving them with **loops** (the first one works well with a **for** loop, the second works well with a **while** loop):

- isPalindrome(t) – Given a single input string, return true if it is a palindrome and false if it is not.
- trimZeroes(t) – Given a single input string, return a string that has all of its beginning zeroes removed.
    - Hint: **substring(1)** is a quick way to remove the first character of a string

Additionally, your program should use each function based on the input provided (similar to how it did with the leap years and fortune teller yesterday). This time, however, your input can have several different commands with each line being both the command AND the string. You should use the **startsWith** command to determine the first word and the **substring** command to get the string you're working on.

Sample Input

trim 0002356
trim 0ag02
palindrome hello
palindrome step on no pets

Sample Output (For the sample input given)

2356
ag02
"hello" is not a palindrome
"step on no pets" is a palindrome

Project Extensions

Once you get the basics done, work on this challenge if you have time. (It is not a required component, but is good for additional practice!)

- Create a function to determine if a number is prime or not.
    - A number is prime if it is only divisible by 1 and itself. You will need a loop to go through a bunch of number to see if the input number is divisible by each of them.

B.2.5 Cohort 2 – Assignment 5

CSCE805T – Assignment 5: Secret Message

Project Summary

In this project you will write functions to encode and decode a secret message.

Background Information

A Caesar Cipher is an encoding in which each letter is shifted by a constant amount. For example, by shifting each letter one to the right, "hello" becomes "ifmmp". Shifting by two to the right instead we get "jgnnq".

Project Details

You will come up with an encoding algorithm (you can use a Caesar Cipher, or one of your own choosing). You can assume the input string is in all lowercase letters. Punctuation should be ignored and left where it is.

Your program will need to contain each of the following functions:

- encode(n) - Given a single input string, return an encoded string.
- decode(t) – Given a single input string, return a decoded string. (Just reverse of the encode.)

Additionally, your program should use each function based on the input provided.

Sample Input (Using a right 4 Caesar shift)

encode the quick brown fox jumps over the lazy dog
decode xli uymgo fvsar jsb nyqtw sziv xli pedc hsk
decode lipps!

Sample Output (For the sample input given)

xli uymgo fvsar jsb nyqtw sziv xli pedc hsk
the quick brown fox jumps over the lazy dog
hello!

Project Extensions

Once you get the basics done, work on this challenge if you have time. (It is not a required component, but is good for additional practice!)

- Try to preserve capitalization, so under the right 4 shift "Hello" would be "Lipps" instead.

B.2.6 Cohort 2 – Assignment 6

CSCE805T – Assignment 6: Register

Project Summary

In this project you will write a program to represent a store's menu and ordering system.

Project Details

You will need to have at least 3 different items to be purchased at your store. Each item will have a name and a price. By interpreting the input provided in the input box, you should add and remove items from the "cart" and when the input shows "pay" you will print the complete order and total cost.

Your cart should be an array with the items in it. You might want a function to determine the appropriate price given the item as input.

The input you're checking for should be "add <item>", "remove <item>", and "pay".
The output should start by printing out the menu, then a blank line, then the customer's order and total cost. (What that looks like, exactly, is up to you!)

A selection of flow charts are included on the next page to get you thinking about the steps you need to accomplish. Creating functions for "add", "remove", and "pay" might be helpful for decomposing the problem into those smaller parts. Decomposition is your friend!

The **toFixed**(2) function can print a number with 2 decimals quite nicely.

Sample Input

add pineapple
add banana
add banana
add banana
remove banana
pay

Sample Output (For the sample input given)

pineapple - $2.30
banana - $0.49
apple - $0.79

Your order is: pineapple,banana,banana
Your total is $3.28

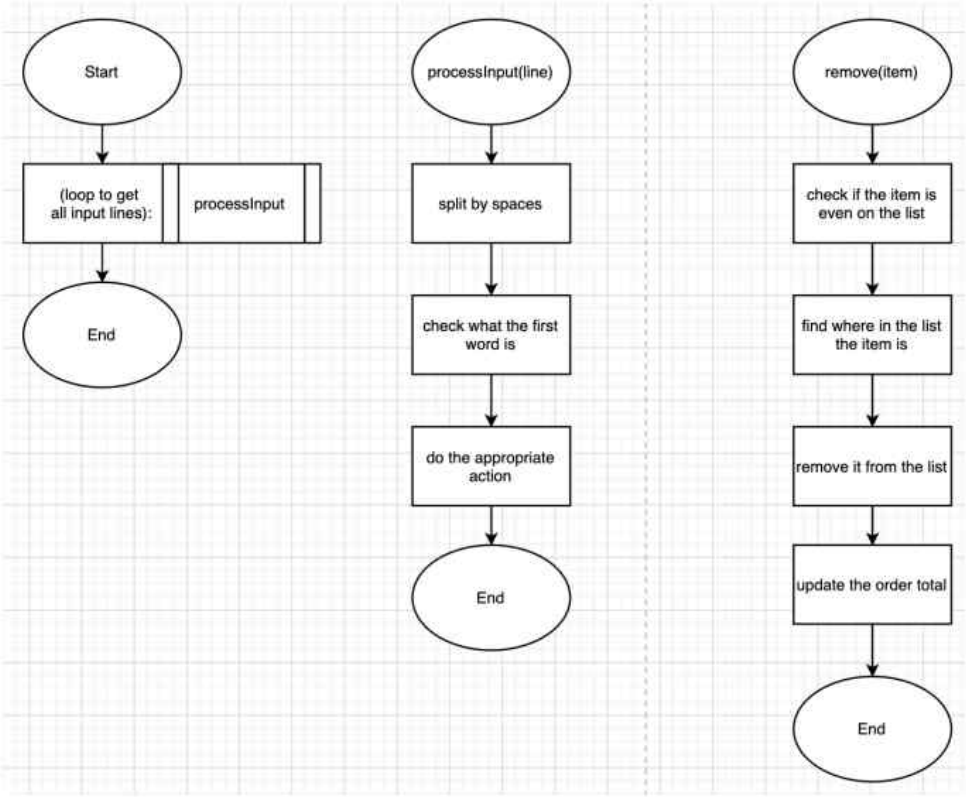
Project Extensions

Once you get the basics done, work on these challenges if you have time. (They are not required components, but are good for additional practice!)

- Rather than just printing out each item in the end, print each item with a count of how many of it there are. For example, the output above would be "Your order is: pineapple – 1, banana – 2"
- Let the input add a specific number of an item instead of just one at a time. For example, a possible input could be "add 3 banana".

CSCE805T – Assignment 6: Register

The charts below might help you to organize your thoughts for this project



B.2.7 Cohort 2 – Assignment 7

CSCE805T – Assignment 7: Recursion

Project Summary

In this project you will interpret some recursive algorithms to describe what they do. Problem 3 (and the extension) requires a coded function, but it will not be that lengthy. You can put all your responses directly in the submission box for the assignment on canvas. (Just copy and paste the code for the function(s).)

1. Identify what this recursive function calculates for a and b. Hint: start with some small values for **a** and **b** like 2 and 6. This is equivalent to a common simple mathematical formula.

```
function recursiveFunction(a, b){
  if(a == b){
    return a;
  }else if(a == b-1){
    return a+0.5;
  }else if(b < a){
    return recursiveFunction(b, a);
  }else{
    return recursiveFunction(a+1,b-1);
  }
}
```

2. Describe the pattern of numbers created by this recursive function when you use successive input values of **n** starting at 1. (Start by plugging in 1. Then 2. Then 3. Etc.)

```
function recursionTime(n){
  if(n == 1){
    return 1;
  }else{
    return recursionTime(n-1) + n;
  }
}
```

CSCE805T – Assignment 7: Recursion

3. The function shown below produces the same result as the code we made last week to add dashes (-) to the end of a string until it is 10 characters long. Write a similar *recursive* function that mimics the "trim" function from last week that gets rid of the leading zeroes at the beginning of a string.

```
function recursiveParty(text){
  if(text.length < 10){
    return recursiveParty(text) + "-";
  }else{
    return text;
  }
}
```

Project Extension

(As per usual, this portion is not required but it can be a good challenge!)

Create another recursive function to mimic the palindrome function which takes in an input and checks if it's a palindrome or not.

B.2.8 Cohort 2 – Assignment 8

CSCE805T – Assignment 8: Sorting

Project Summary

In this project you will write a program that will sort a list of words.

Project Details

Your program will read all of the words from the input, store them in an array, sort them by length (with alphabetical order as the "tiebreaker"), and print out the sorted list (each word on a new line). For example, all of the 3-letter words would come before all the 4-letter words, etc. And all of the 3-letter words should be sorted alphabetically with each other, etc. You can either come up with a creative approach, or utilize the **stability** of some of the sorting algorithms (meaning once you sort it alphabetically, if you then sort by length they will maintain their alphabetical order within each length). You can assume that each word is all lowercase, and you may use whichever algorithms you would like.

Sample Input

snorkel
apple
pie
peach
ape
chocolate

Sample Output (For the sample input given)

ape
pie
apple
peach
snorkel
chocolate

Helpful Hints

The functions you should create/use are:

- Sort alphabetically (remember arrays already have an algorithm for this!)
- Sort by length – remember to make a copy of the array with **slice**(0) first!
- Apply your alphabetical sort first, **THEN** your length sort.
- Think about breaking your program up step-by-step rather than thinking about the program as a whole all at once!
- Draw a flowchart or write out steps.

Project Extensions

Once you get the basics done, work on these challenges if you have time. (They are not required components, but are good for additional practice!)

- Create a new sorting function that sorts the words based on their *last* letters instead. (So if there is a "tie" in the last letter, their second-to-last letters would be compared, etc.)

B.2.9 Cohort 2 – Assignment 9

CSCE805T – Assignment 9: Text Analysis

Project Summary

In this project you will write a program that will analyze a block of text for various characteristics.

Project Details

Your program will read all of the text from the input and determine each of these characteristics. Create a different function (hints of how to define them shown) for each characteristic.

- The number of words. (Each word being space-separated.)
  -
```
function countWords(text){
```
- The number of words that contain a particular letter. (That letter should be one input to the function.)
  -
```
function countWordsWith(text,letter){
```
- The letter frequency of each letter. The output of this function should be an array with the values.
  -
```
function letterCount(text, letter){
  var count = 0;
  //loop through the length of text
    //if each letter matches the input letter, increase count
  return count;
}
function letterFrequency(text){
  var output = []; //this will end up with 26 different numbers in it
  var alphabet = "abcdefghijklmnopqrstuvwxyz";
  //loop through all 26 letters (in the alphabet variable perhaps)
    //use letterCount(text,<whichever letter you're on>)
    //store that number into the output array
  return output;
}
```

This program will not make you choose between one function or another, but rather apply all three functions to the same text and print output showing the results. (Pick any letter you want in your code for the **countWordsWith()** function.)

Sample Input

This is a sentence.
And this is another one!

Sample Output (For the sample input given)

There are 9 words in the text.
There are 5 words with s.
a: 3
b: 0

CSCE805T – Assignment 9: Text Analysis

c: 1
d: 1
e: 5
f: 0
g: 0
h: 3
i: 4
j: 0
k: 0
l: 0
m: 0
n: 5
o: 2
p: 0
q: 0
r: 1
s: 5
t: 4
u: 0
v: 0
w: 0
x: 0
y: 0
z: 0

Project Extensions

Once you get the basics done, work on this challenge if you have time. (It is not a required component, but is good for additional practice!)

- Try to add another function to count the number of sentences. This is more challenging since sentences can end in different ways!

B.2.10 Cohort 2 – Final Assignment

CSCE805T – Assignment 10: Final Project

Project Summary

For this project you will work with a group to plan out and code a Hangman game. To get you organized here are several specific steps to complete. Each step will need to be submitted for your project. You can submit just one assignment for your entire group (make sure each group member's names are on it though!)

1. Play a game of Hangman with your group and as you go through the steps, try to determine what coding concepts you can identify within the game. Write down your thoughts to be submitted with the assignment.
    a. Variables
    b. Functions
    c. Conditionals
    d. Loops
    e. Lists
2. Make a flow chart of the gameplay logic. You do not have to have all the right shapes, but it might help you if you do try to use the correct ones. The more detailed you get with your diagram, the more helpful it will be for your coding. The functions below might be helpful to think about. (Keep the functions in part 3 below in mind for your charts!)
3. Write your program!
    a. Start with this Fiddle – https://jsfiddle.net/aholdorf/1xbykvaz/latest
    b. You will want to create several functions to address the individual components of the game.
        i. Pick word – Create an array of possible words to pick from and then pick one of them randomly. (If you generate a random integer between 0 and the list length, you can use it to get a value from your array.)
        ii. Guess letter (already defined for you in the base) – The **letter** parameter will have the typed letter in it, and this function will run automatically when the Guess button is clicked. This function should:
            1. Determine if the guessed letter is correct or not
            2. Then use the appropriate "wrong" or "correct" function you create. (See iii and iv below.)
        iii. Mark wrong guess – Use the already-created array for wrong guesses and add any wrong letters to the array (making sure that you only count them if they aren't already in the array). Additionally this should increase the count variable for wrong guesses.
        iv. Mark correct guess – Update your word to keep track of which letters have been guessed.
        v. Format display word – Determine how to display the appropriate blanks for your word (a common algorithm is to have your **word** variable store hidden letters as uppercase and visible letters as lowercase, or vice-versa, then loop through it to add either blanks or letters to the display word).
        vi. Check win – Determine if the game is over from a win! (How do you know you have won Hangman?)
        vii. Check loss – Determine if the game is over from a loss. I have it set up so that the 7th wrong guess shows the game is over in the image. (What else should happen to the display word when the game is over?)

CSCE805T – Assignment 10: Final Project

Reminders of where to find information for the things we've covered:

Monday June 8 – Variables
Tuesday June 9 – Functions
Wednesday June 10 – Conditionals
Thursday June 11 – Loops
Friday June 12 – Flow charts

Monday June 15 – Arrays

(The rest—recursion and sorting—aren't necessary for this project.)

Appendix C Computational Creativity Exercises (CCEs)

C.1 CCE 1 – Everday Object

CSCE 805T: Introduction to Computer Science for Teachers

CSCE 805T Exercise 1: Everyday Object

Objectives

The objectives of this exercise:

- Computational:
  - Decomposition: Breaking down a comprehensive description of an object into detailed descriptions of (1) its function(s), (2) the need(s) it fulfills and (3) its physical attributes.
  - Abstraction: Describing a generic example of an everyday object by focusing on its essential or typical functions and physical attributes without regard to trivial variations (such as color other variations).
  - Evaluation: Logically, methodically and completely describing an everyday object in sufficient detail and in clear, non-technical language such even if the name of the object is omitted that any reader could recognize the object and understand how it works.
  - Learning about the description and design process for modular programming by describing an everyday object in detail including why the object is needed and how the object functions
  - Learning about abstraction and function characterization by identifying properties of an everyday object
  - Learning about specifying input, output, and function of a module or object clearly
  - Learning about hiding details of the inner workings of an object without sacrificing the functionality of the object
- Creative:
  - Surrounding: Looking at an everyday object in new ways, using all of your senses to understand how it's made and how it functions.
  - Capturing: Using written language to describe all the different details and characteristics of this everyday object so you can work with it in new ways.
  - Challenging: Describing the operations of an everyday object with words and also as a computer program.
  - Broadening: Imagining that this everyday object doesn't exist and acting like its inventor and trying to fulfill a need by creating something new and useful.
- Collaborative:
  - Being open to all points of view and resolving group conflicts in a constructive way.
  - Giving and receiving thoughtful and constructive feedback in order to develop your group project.
  - Meeting group deadlines, including completing your individual work in a timely manner.
  - Contributing substantially to the group process, using your skills, knowledge and experience.
  - Working together as a team to achieve a common goal; being able to both compete against and cooperate with other teams.

PROBLEM DESCRIPTION

You will be using language to try to clearly and thoroughly describe the functions of an ordinary object that you might use every day. You will be acting like the inventor of that object, imagining that it does *not* yet exist and trying to describe what need would be fulfilled by your (new) object and how (specifically) it will function.

Each group will have a Group on Canvas. Have one member of your team create a file and add it is the "Files" tab on your group page. The file should be named "Everyday Object by CSCE 805T Group <Name>" where <Name> is your group name (e.g., Everyday Object by CSCE 805T Group Awesome).

Any member may create the group file. Note that there should be only one file created per group. Before you create a new file, make sure that one doesn't already exist.

Your group will **choose** a common, functional, everyday object from the list in Appendix A. Your challenge is to imagine that this object does not exist and to describe in written language (1) the mechanical function of your object, (2) what need is fulfilled by this object, and (3) the physical attributes of your object.

You must describe the object's function, the need it will fulfill and its physical attributes in clear, non-technical language that any user could understand. Your description must be specific enough so that someone who had never seen the object could recognize it and understand how it works and understand what benefits it provides.

This description process is very important for developing algorithms in computer science. An algorithm consists of the series of steps necessary to solve a given problem. By using algorithms, we can solve problems without having to constantly "reinvent the wheel" and spend the time, money, etc. to figure out each step ourselves. However, if any of these steps are unclear, we can have difficulty following the algorithm, which can lead to serious repercussions. For example, if the formulation algorithm used to mix the concrete for a road or bridge is unclear, workers may make a mistake during pouring leading to reduced service life. Or, if the business plan algorithm for a new company is confusing, venture capitalists may be reluctant to invest leading to failure of the business. To avoid these repercussions, the developer should make every effort to make the algorithm's description as clear as possible for all steps. In other words, characterization of processes is key; it allows us to abstract a process and then convert it into a formal problem or solution.

For example, if your object is a "colander" you might *begin* to describe it as *"a circular object, approximately 12" in diameter and 9" in height, made of metal or heat-resistant plastic, which is used in cooking to drain pasta after cooking or to hold food for washing or steaming. Its holes are large enough for water and other liquids to drain but small enough so that food will not leak through. A base or foot enables it to sit on a counter or in a sink and handles allow easy carrying and a means to suspend it over a cooking pot for steaming ..."*

1. Part One

1.1. Written Description

Generate your written description of your object. Your description must include the following:

1. The mechanical **function(s)**/use(s) of the object (E.g., "This object, which I call a "hammer" is used to drive nails into wood or other materials . . .)
2. What **need(s)** the object fulfills (E.g., Instead of using a brick to drive nails, the hammer . . .)
3. The **physical attributes** of the object. These include:
   - components or parts (E.g., "The hammer has a handle and a head. The head may have a curved claw like end so that nails can be removed . . . ")
   - shape or materials (E.g., "The head is metal. The handle may be wood or metal and may have rubber padding . . .")
   - general dimensions (E.g., "The hammer may range in length from . . .")
   - connections between parts (E.g., Positions of parts such as inside, outside, top, bottom or relationships between parts, such as fixed, fitted, detaches, swivels, etc.)

Your description *should start with the name of the object* and must have a **minimum** of 150 words. You must have at least one function, at least one need, and a minimum of 6 physical attributes for your object to receive full credit. Keep in mind that physical attributes may involve all of your senses. Remember, to receive any credit you must have contributed to the description of the object by writing or editing the description in the body of on the document.

This written description is very important for writing functions in computer science. Functions are blocks of code written to perform a discrete task. With functions we do not need to repeat the same blocks of code multiple times in the same program. This makes the source code more organized and also makes future changes to the code easier (and less error prone) since we only have to change a function once rather than updating each block separately. When you first start programming, you can get away with writing functions in an ad hoc manner while coding. However, for larger programs, and when working with a group, a written description is critical to making sure all the functions are written correctly. For example, imagine writing all the functions necessary for the F-22 Raptor jet fighter which consists of about 1.7 million lines without starting from a detailed description.

2. Part Two

2.1. Analysis and Reflection

Post your Analysis and Reflection responses in the "Discussion" tab of your Canvas page, NOT in the file that has the written description.

You are expected to discuss these analysis and reflection questions among your group. One member must start a new topic for EACH Analysis or Reflection by clicking on the "+ Discussion" button In the Topic Title area, type "Analysis" or "Reflection," in the Comment area, paste in the Analysis or Reflection questions, and under "Option" make sure "Allow threaded replies" is checked. Using the Analysis or Reflection questions as prompts, each member will post his or her responses as a reply to the original comment. This process will keep the group's Analysis and Reflection in separate threads and make it easier to follow the development of your answers.

You will be graded individually based upon your contributions to the group Analysis or Reflection. In order to receive individual credit for Part 2, each group member must contribute to the answers to these questions. **Group members who do not contribute to the Analysis or Reflection Discussion will not receive credit.**

2.2. Analysis

Respond to these questions: (1) Considering your object as a computer program, draw a diagram that shows all its functions as boxes (name them), and for each function, its inputs and outputs. Are there shared inputs and outputs among the functions? (2) Looking at the list of physical attributes, organize these such that each is declared as a variable with its proper type. Can some of these attributes/characteristics be arranged into a hierarchy of related attributes/characteristics?

This diagraming process is important for problem analysis in computer science particularly and in all problem solving in general. Just as we have organized similar blocks of code using functions, we can organize functions with similar inputs and outputs together. This process provides a "big picture" view of the program which is vitally important for initial development of the code and future changes. For example, software for large insurance companies may contain many similar functions used for different insurance plans all of which need to be updated after a law is changed.

2.3. Reflection

Respond to these questions: (1) Considering your response to Analysis 1, are there functions that can be combined so that the object can be represented with a more concise program? Are there new functions that should be introduced to better describe your object such that the functions are more modular? (2) Have you heard of abstraction? How does abstraction in computer science relate to the process of identifying the functions and

characteristics as you have done in this exercise?

This abstraction process is used in many programming languages to allow similar functions to be written more concisely, and to be more easily understood in a conceptual way. The basic idea is to write the source code completely for only one function in such a group. The rest of the functions use this function as a baseline adding only the source code necessary for their specific tasks. In this way, source code common to multiple functions needs to be written only once. Again, the main advantage is in terms of organization—including defining the relationships between functions—and making updates to the functions. For example, in a simulation game, you could have hundreds of functions for customizing character appearance. By using abstraction, you can avoid having to update all hundreds of functions when you change the common source code on character appearance.

Deadlines and Hand-In

Part 1 Deadline – [6/4/19, 3:00 p.m.]: You should have completed the description of the object by the Part 1 deadline above. This description should be posted to the "Files" tab of your group's Canvas page.

Part 2 Deadline – [6/4/19, 9:00 p.m.]: Your analysis and reflection responses are due by the Part 2 deadline above. Your individual Analysis and Reflection comments must be posted in the "Discussion" tab of your group's Canvas page.

Grading

Part 1. Object description posted in the body of the group Canvas page. Each group member must contribute to the description by writing or editing to receive credit. The description must include at least one function, one need and six physical attributes for full credit.

Part 2. Analysis and Reflection: graded individually. Each member must post in the Discussion with a minimum of 3-5 coherent, relevant sentences for full credit.

Late work will not be graded.

Appendix A. List of Objects

List of objects:

zipper
mechanical pencil
binder clip
ziploc bag
scissors
tape measure
stapler
nail clippers
umbrella
flashlight
can opener
clothespin
sticky notes (Post-Its)
toilet paper holder
revolving door
computer mouse
pliers
ballpoint pen
mousetrap
screwdriver
pocket calculator
sundial
belt
solid air freshener

Appendix B. Example of Patent Description for Scotch Tape

The US Patent Database, uspto.gov, has examples of how common objects were described for patent purposes.

You can view the original patent application for masking tape and its extension, Scotch tape (Patent 1,760,820; May 27, 1930) at http://patimg1.uspto.gov/.piw?Docid=1760820&idkey=NONE

Your described object should be able to meet the requirements of a "utility patent." That is it is new, useful, functions as described, is non-obvious, and is not simply a combination of other existing inventions or a remaking of an existing object.

END OF EXERCISE

C.2 CCE 2 – Path Finding

CSCE 805T: Introduction to Computer Science for Teachers

CSCE 805T Exercise 3: Pathfinding I

Objectives

The objectives of this exercise:

- Computational:
  - Algorithmic thinking: (1) As a designer, developing a set of step by step instructions to generate a visual pattern on a base grid from straight line segments, and (2) As a responder to another group's instructions, carry out their instructions methodically
  - Abstraction: Using relative rather than absolute coordinates to generate the pattern instructions
  - Decomposition: Breaking down a 9x9 grid into simpler 3x3 components
  - Pattern recognition: Pattern Recognition: As a designer, identifying similar (perhaps repeated) steps that create similar visual patterns on the grid and then using loops to simplify those steps, and (2) As a responder to another group's instructions, identifying similar (perhaps repeated) steps that create similar visual patterns on the grid, and using shortcuts to carry out those steps faster
  - Evaluation: Testing your instructions to make sure that another user can accurately recreate your pattern on a new base grid
  - Learning about the benefits of looping by using looping to simplify the instructions and allow patterns to be repeated
  - Learning how to design modular functions by modifying the instructions to use *relative* coordinates
  - Further developing testing skills by using the modular function to generate a more complex pattern
- Creative:
  - Surrounding: using your senses of touch and sight to follow a set of instructions and perceive that they produce the intended geometric pattern
  - Capturing: creating new outputs and using new ways to represent and save data by writing a description of a path which will generate a drawing of a geometric pattern
  - Challenging: looking at written descriptions in new ways as you both generate and follow them to recreate a drawing of a geometric design
  - Broadening: acquiring new information and skills by understanding how simple rules can generate complex patterns
- Collaborative:
  - Being open to all points of view and resolving conflicts in a constructive way.
  - Giving and receiving thoughtful and constructive feedback in order to develop your group project.

- Meeting group deadlines, including completing your individual work in a timely manner.
- Contributing substantially to the group process, using your skills, knowledge and experience.
- Working together as a team to achieve a common goal; being able to both compete against and cooperate with other teams.

Problem Description

Using a grid, you will be designing a geometric visual pattern. You will then create a set of written instructions (an algorithm) for another group to follow which will accurately generate your geometric pattern on another grid.

Think of your pattern as a module that can be repeated, reflected and rotated to generate complex patterns. See Appendix A for visual examples of complex traditional quilt patterns using a simple module.

Any member may create the group file. Note that there should be only one file created per group. Before you create a new file, make sure that one doesn't already exist.

1. Part One

Your group will generate a grid that is 9 x 9 with at least ¼ inch squares. You can generate this grid at http://incompetech.com/graphpaper/lite/ and download a PDF for printing if you do not have ¼ inch graph paper. Label your grid axes such that the origin (0,0) is in the upper left corner.

Design your base pattern by drawing line segments on the grid. To simplify the set of instructions, please ***only*** use straight line segments (no curves) and restrict these line segments to horizontal, vertical or diagonal (45° angle) lines.

Build a complex 9 x 9 pattern using three operations: (1) shift, (2) reflect, and (3) rotate. The *shift* operation allows a base pattern to be replicated at a different location. The *reflect* operation allows a base pattern to be reflected horizontally or vertically. The *rotate* operation allows a base pattern to be rotated 90 degrees clockwise or counterclockwise. Figure 1 shows examples of these operations performed on a base pattern.

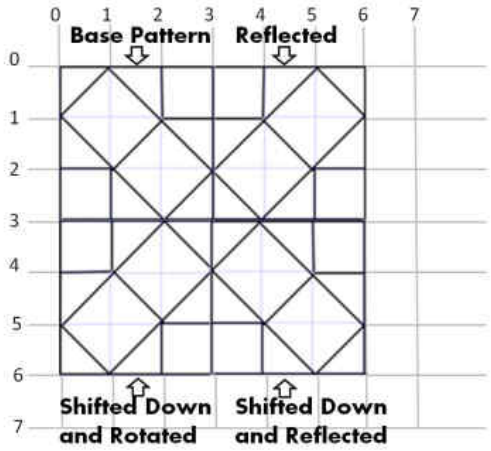


Figure 1: Examples of Base Pattern along with Reflect, Rotate and Shift Down.

Note that depending on your pattern, rotating or reflecting may produce the same visual result.

You are also allowed to use a loop structure to repeat to generate multiple shifts, reflections, or rotations. For example,

```
Loop 3 times
  Rotate-counter-clockwise()
End Loop
```

In the above example, a base pattern will be rotated 3 times counter-clockwise.

The use of **looping** to allow for repetition in functions is extremely important in computer science. Loop structures allow similar operations (or instructions) to be repeated inside a function using a compact representation (in the source code). Obviously, this is very convenient for programmers who do not want to cut and paste the same source code multiple times. This compact representation also reduces the size of the final program, which can be useful when hardware resources are limited, for example, on an interplanetary probe or nano machine. However, the real benefits of using loop structures come into play when these similar operations need to be modified. Suppose the application you are developing is a Facebook game with 999 different ways to configure character appearance. The testers working for your company find a bug in the rendering engine for character appearance. With a looping structure, you may only need to change the rendering engine in a single place, potentially saving you 998 extra changes. Of course, designing the operations to be similar enough to take advantage of looping structures, while still satisfying the design features is not always easy. Creative thinking skills during the design process can help the programmer leverage looping structures.

IMPORTANT: Note that in order to do shifting, reflection, or rotation operations, the base pattern's instructions cannot use **absolute references** for the coordinates. Instead, **relative coordinates** are needed.

To illustrate the need for relative coordinates, suppose we want to move three segments horizontally. The instruction to move from 0,0 to 3,0 (with the first coordinate as horizontal and the second as vertical) works fine for base pattern, but will not work after we shift and start drawing at 4,0 because it uses absolute references (0,0 and 3,0). Instead, we need to modify the instruction to go from the current coordinates x,y to x+3,y. This instruction uses relative coordinates (x,y to x+3,y). Using these relative coordinates will give the desired end coordinates for the base pattern when x=0, y=0 and also for the shifted pattern when x=4, y=0. Now, suppose that in addition to shifting, we want to rotate the base pattern 90 degrees. After the rotation, we need to move vertically rather than horizontally. Again, modifying the instruction requires only a small change when using relative coordinates: x,y to x,y+3.

Thus, before you start performing the above operations on the base pattern, please revise the instruction set for the base pattern to use only relative coordinates. (***Hint***: Think of using variables!)

IMPORTANT: **The number of line segments in your complex pattern must be at least 30% of all possible lines in a 9 x 9 grid.** Note that a 9 x 9 grid can have at most 90 horizontal segments, 90 vertical segments, and 162 diagonal segments for a total of 342 line segments. After rounding up, this means that your complex pattern must have at least 103 segments.

The use of **relative coordinates**, and, in extension, variables, is important for practicing computer science. Functions using relative coordinates provide several benefits compared to using those absolute coordinates. First, a function using relative coordinates is more flexible than a function using absolute coordinates. The following examples show how relative coordinate flexibility is useful for both the user and the machine: (1) the user can run a function when he/she is at a similar, but not exactly the same, starting point (e.g., at 0,1 rather than 0,0) and (2) the machine can assign the function to available addresses in the memory rather than waiting until absolute addresses are available. Second, a function using relative coordinates is much easier to modify than one using absolute coordinates. In functions using absolute coordinates, since the exact coordinates are passed from one line to the next, a change to the coordinates on one line must be propagated throughout the rest of the function. This can be extremely time-consuming and error-prone on functions with hundreds (or thousands) of lines. On the other hand, such a change to functions using relative coordinates generally only affects a single line—making them easier to modify. The benefits of using relative rather than absolute coordinates also extend to using **variables** over hard-coded values. In general, functions using variables are more flexible and easier to modify than hard-coded values. However, these functions can also be more difficult to design since they require a programmer who can think creatively about how to write each line using variables rather than hard-coded values.

NOTE: Teams submitting patterns with less than 103 segments will only be rewarded partial credit.

NOTE: Teams without rotating, reflecting AND shifting elements will only be rewarded partial credit.

IMPORTANT: Your written instructions for generating the complex 9 x 9 pattern must be sufficiently clear to enable another group to accurately draw your pattern on a similar grid. You will only be rewarded partial credit if the grader or another group cannot follow your instructions.

Please post your set of instructions on the main Canvas page. Label your instructions "9 x 9 Grid Instructions <Group name>." **Again, do NOT upload your drawn pattern to your Canvas page.**

2. Part Two

2.1. Generate Complex Patterns from Instructions by Other Groups

Each group should navigate to a ***different*** group's Canvas page and "claim" their pattern instructions by posting in the "Discussion" area of their Canvas page. (Each group's pattern instructions may have only ONE group attempting to follow it.) Each group will then try to follow the written instructions of the other group by drawing on a blank grid labeled following the same (0,0) numbering. When you think you have followed the other group's instructions correctly, post an image of the other group's base pattern (a jpg of your grid drawing) on that group's page. If you think that the other group's instructions are invalid and you cannot complete the pattern, post your reasoning on that group's page along with an image of your incomplete pattern.

NOTE: **Groups will lose credit in part two for not correctly following the base pattern instructions of another group or identifying why another group's instructions are incomplete or invalid.**

2.2. Analysis and Reflection

Post your Analysis and Reflection responses in the "Discussion" area of your Canvas page.

You are expected to discuss these analysis and reflection questions among your group. One member must start a new topic for EACH Analysis or Reflection by selecting "New Comment." In the Topic area, type "Analysis" or "Reflection" and in the Comment area, paste in the Analysis or Reflection questions. Using the Analysis or Reflection questions as prompts, each member will post his or her responses as a reply to the original comment. This process will keep the group's Analysis and Reflection in separate threads and make it easier to follow the development of your answers.

You will be graded individually based upon your contributions to the group Analysis or Reflection. In order to receive individual credit for part 2, each group member must contribute to the answers to these questions. **Group members who do not contribute to the Analysis or Reflection Discussion will not receive credit.**

Analysis. Respond to these questions: (1) Which operations created more interesting patterns or complex patterns that are more different from the base patterns? Explain. **(2)** Compare the number of lines of your instruction set for the complex pattern and the number of lines of your instruction set for the base pattern. Is it possible to reduce the number of lines in either set by using the three operations? If yes, please describe and show. If no, please explain.

Reflection. Respond to these questions: (1) Reflect on the impact of having a loop structure in your instruction set. What are the benefits or advantages for you as a designer of an instruction set, as well as for you as a "drawer" following an instruction set. **(2)** Reflect on the impact of having relative references in your instruction set on replicating or duplicating processes or solutions on different items. Illustrate your reflection using a recipe designed for 4 people being adapted for 8 people, for example.

DEADLINES AND HAND-IN

Part 1 Deadline – [6/6/19, 3:00 p.m.]: You should have finished modifying your base pattern to use relative coordinates. You should also have posted the written instructions for your complex pattern using shift, reflect, and rotate on your Canvas page.

Part 2 Deadline – [6/6/19, 9:00 p.m.]: You should have attempted to follow the instructions of another group and posted your drawn solution to the other group's Canvas page. Your individual Analysis and Reflection comments must be posted in the Discussion area of your group's page.

GRADING

Part 1. Group credit for generating complex pattern instructions. Your instructions must meet the stated requirements and credit is deducted for if the patterns do not use relative coordinates, do not contain rotating, reflecting and shifting elements or if your instructions are incomplete or invalid.

Part 2. Pattern Generation from another group's instructions: Group credit for generating and posting another group's pattern from their instructions OR for identifying the flaws in the other group's instructions and posting the resulting incomplete pattern. Analysis and Reflection: graded individually. Each member must post in the Discussion with a minimum of 3-5 coherent, relevant sentences for full credit.

Late work will not be graded.

END OF EXERCISE

C.3 CCE 1 – Modular Storytelling

CSCE 805T: Introduction to Computer Science for Teachers

CSCE 805T Exercise 2: Modular Storytelling

Objectives

The objectives of this exercise:

- Computational:
  - Decomposition: Dividing a story into separate chapters and independently developing each chapter, using fixed "story points" as chapter separators (the last line of each chapter becomes the first line of the next).
  - Abstraction: Using the story points to get an overall idea of what the story might be and writing an individual chapter with enough detail to advance the story without getting too distracted by the details.
  - Evaluation: Identifying logical inconsistencies in the individually written chapters and determining the most efficient way to revise the chapters in order to form a consistent and coherent story.
  - Pattern Recognition: Reviewing the individual chapters, identifying consistent story lines to keep, and identifying one or two inconsistent parts that need revisions in order to minimize the amount of changes.
  - Learning about how large programs are developed using modular programming by separately writing chapters for a story connected by fixed Story Points at the beginning and end of each chapter.
  - Learning about the need for the debugging and testing process by revising the chapters to make the flow of the story more logical and cohesive.
  - Learning about the need for a methodical and logical debugging process to make sure that a program's observed output matches the expected output.
  - Learning about identifying logical inconsistencies in a product or solution and using methodical approaches to resolve them.
- Creative:
  - Surrounding: using your senses of sight, sound, smell, touch and your imagination to connect two seemingly unrelated things (two Story Points) in a logical and coherent way.
  - Capturing: learning to take novel and spontaneous outputs (your two Story Points) and use language to fill in the blanks by writing your way from one point to another.
  - Challenging: applying computational debugging to a story by taking independently generated chapters and editing them so they form a consistent narrative.
  - Broadening: increasing your ability to problem-solve and to collaborate by taking the inputs of others (your group's individual chapters) and making them into an effective and functional whole—a logical and cohesive story.
- Collaborative:
  - Being open to all points of view and resolving group conflicts in a constructive way.

- Giving and receiving thoughtful and constructive feedback in order to develop your group project.
- Meeting group deadlines, including completing your individual work in a timely manner.
- Contributing substantially to the group process, using your skills, knowledge and experience.
- Working together as a team to achieve a common goal; being able to both compete against and cooperate with other teams.

PROBLEM DESCRIPTION

Your group will be telling a story with several **short** chapters. However, you won't write the story as a linear line from A to B. Instead, you'll be working in a non-linear way, developing chapters independently and then working together to shape them into a coherent story.

To begin, your group will choose a series of **Story Points** from the five series in Appendix A . For each series, these Story Points will be the pivotal moments in the story. They will act as inputs and outputs to your individual **chapters**. Each **chapter** will begin and end with one of these Story Points. For example, Chapter 1 will begin with Story Point 1 and end with Story Point 2. Chapter 2 will begin with Story Point 2 (repeating it) and end with Story Point 3 and so on.

Each person in your group will write a chapter of the story that **connects** two Story Points. You will each write your chapter independently of each other. All you know is how the story (and each chapter) starts and how it ends: the input and the output.

Each group will have a Group on Canvas. Have one member of your team create a file and add it is the "Files" tab on your group page. The file should be named "Storytelling by CSCE 805T Group <Name>" where <Name> is your group name (e.g., Storytelling by CSCE 805T Group Awesome).

Any member may create the group file. Note that there should be only one file created per group. Before you create a new file, make sure that one doesn't already exist.

What does this exercise have to do with creativity? When you tell a story you get to imagine. You get to ask yourself "what if," which is one of your most powerful thinking tools. And you're only limited by the internal logic of your story, and not by external conditions or lack of resources. Constraints force creativity. Telling a story where you are given a beginning and ending point and are forced to work within those constraints has additional benefits as well. First, it prepares you for real-life collaboration, where you need to enter into someone else's vision or to incorporate their way of thinking or doing things into your own vision. Second, it connects you with the rich artistic tradition of "the Exquisite Corpse" where unplanned elements and chance connections can produce rich and unexpected results. Writing this story will give you experience with "giving chance a chance" and seeing how accidents and even mistakes can

suggest fruitful new directions. It will also develop your ability to take ideas (or components) that you have generated in a non-linear way and put them together in a linear/logical way in order to develop a finished product.

1. Part One

1.1. Picking a Story Series and Assigning the Chapters

Your group should first pick a story series from one of the five possibilities in Appendix A. (It's OK if more than one group picks the same series of Story Points.) Then create a new Discussion in the "Discussion" tab of your groups Canvas page and post the series of Story Points you've chosen and who is responsible for writing each chapter. Your group should list each chapter and the author:

Chapter 1: <Author>
Chapter 2: <Author>
…
…

This list is the scaffolding for your individual chapters. Remember, a chapter is Story Point 1 to 2, Story Point 2 to 3, and so forth. You MUST start Chapter 1 with Story Point 1. You MUST end Chapter 1 with Story Point 2. You MUST begin Chapter 2 by REPEATING Story Point 2 and so on.

Note: If your group has N students, then you are required to use the first N+1 Story Points for these chapters. For example, if N = 4, then you will have four chapters using Story Points 1-5. **This means that if your group has fewer than 7 members, you won't use all of the Story Points.**

1.2. Writing the Chapters

Group members will then **separately** write the chapters. You will ***not consult*** with your group members and you are ***not allowed to share your chapter*** with your group members. You will shape your chapter solely on the Story Points.

When you upload your individual chapter, do so in the list of Chapters and Authors you've posted previously. Remember to begin and end with your assigned Story Points.

The requirements of each chapter are as follows:

1. Each chapter must advance the story by ***starting from the previous Story Point and ending at the next Story Point.*** As an example, the first chapter should **start** with Story Point 1 and **end** with Story Point 2. Chapter 2 should start with Story Point 2 and end

with Story Point 3, and so on. Effectively, you must transform the input to your chapter into the output of the next chapter through storytelling.

2. Each chapter must be between 100 and 200 words. This will allow you to effectively illustrate the story, but without leaving too much room to embellish.

See Appendix B for an example story with four chapters using five Story Points. Then look at Appendix C to see how the individually written chapters were revised into a coherent story.

> Writing a chapter based on Story Points is similar to how we write functions in computer science. To make the design of large programs feasible, the inputs and outputs for a function are generally known before that function is written. In a sense, the way the code for the function is written is less important than whether it produces the correct output value for a given input value. Of course, the function must mesh seamlessly with all the other functions in the program. Additionally, the function must work for all possible input values not just one or two. For example, a must work for all manner of collisions not just one at 30 mph. Meanwhile, a program solution is often made up of a sequence of functions, where the output of the first function is fed into the second function as input, and the second function's output is fed into the third function, and so forth.

1.3. Uploading the Chapters

The chapters should be uploaded to the group's Canvas page when instructed. This is because we want you to work on your chapters on your own and keep them "secret" until you discuss them with your group during part two.

Note: Each chapter's author is **responsible** for uploading his or her chapter. If the author does not uploaded his/her chapter, then the author for that chapter will not receive credit for part one.

All the individual chapters should be uploaded to your group's file on your Canvas page. Each chapter should be pasted into to the body of the page just below the list entry for that chapter. As an example, the content for the first chapter should be uploaded just below the entry for Chapter 1: <Author>.

2. Part Two

2.1. Debugging

Review the chapters that have been uploaded to your group's Canvas page. Most likely not all of your chapters will make sense in the context of each other (and this is normal and expected).

After you review all of your group's chapters, you need to "debug" your story by resolving the inconsistencies.

To debug your story, pick one chapter as the **anchor** and make minimal changes to it. Then revise the other chapters so that they logically fit with this anchor chapter. Make sure you are keeping your chapter structure and designated author structure intact. Each chapter author should revise his or her chapter.

Conduct this process in a way that allows the individual nature of each chapter to remain, but also makes the entire story more cohesive.

This debugging and testing process is important for writing programs in computer science and is also useful in many other fields. Programmers are only human and they make mistakes when writing programs due to fatigue and other factors. Additionally, when working for a large company like Microsoft, programmers rarely write original and stand-alone code. Instead, programmers modify existing code to add new "features" to the program. Mistakes are common due to a lack of familiarity with that code particularly when dealing with code written by someone else. Without extensive debugging and testing to fix these mistakes, a company could lose millions of dollars when customers do not buy the program. Furthermore, testing is also common in many other fields. For example, engineers at NASA extensively test a new probe before it travels 127 million miles to Mars and fix bugs or components that do not work well. And, business executives test a new commercial on a test group of audience and revise it before they spend millions of dollars for a Super Bowl timeslot.

This debugging process is also similar to a group brainstorming process where the group members bring together their own individual ideas and points of view. These possible solutions may be quite diverse, and each will have its own strengths and weaknesses. How can the group combine the best aspects of these various approaches so that the selected solution solves the problem in the most effective way?

Important: If you have a missing chapter (i.e., if a group member failed to participate), you do not need to write the entire missing chapter. Instead, fill in the gap with a shortened version with the starting/ending points (if not already included) and 1-2 new sentences. For example, if you have Chapter 1 which uses Story Points 1-2 and Chapter 3 with Story Points 3-4, you only need to write the 1-2 sentences for the shortened version. Please indicate which chapters are missing to allow the graders to take this into consideration. Simply add (MISSING CHAPTER X) at the start of the new sentences.

2.2. Analysis and Reflection

Post your Analysis and Reflection responses in the "Discussion" tab of your Canvas page.

You are expected to discuss these analysis and reflection questions among your group. One member must start a new discussion for EACH Analysis or Reflection by selecting "New Discussion." In the Topic area, type "Analysis" or "Reflection" and in the Comment area, paste in the Analysis or Reflection questions. Using the Analysis or Reflection questions as prompts, each member will post his or her responses as a reply to the original comment. This process will keep the group's Analysis and Reflection in separate threads and make it easier to follow the development of your answers.

You will be graded individually based upon your contributions to the group Analysis or Reflection. In order to receive individual credit for part 2, each group member must contribute to the answers to these questions. **Group members who do not contribute to the Analysis or Reflection Discussion will not receive credit.**

Analysis. Respond to these questions: (1) What was the most difficult part of "debugging" your story? Did entire chapters need to be rewritten? Or could you manage to reconcile between each chapter with each other by making only minor changes? **(2)** What would you have changed about your initial storytelling process to make the debugging process easier? Would you have made your story more straightforward and logical, or more ridiculous and expansive? Essentially, how would you go about writing the story so that it includes the fewest number of "bugs"?

Reflection. Respond to these questions: (1) Compare this process to working in a large team on a software project or working in a team on any complex problem. In what ways are the two processes similar? In what ways are they different? **(2)** How would you change the rules of the assignment to guarantee that a minimal number of "bugs" are created? (Assume that all chapters must still be written simultaneously.)

The debugging process for writing programs often involves finding logical errors where the program runs but the observed output value does not match the expected output. These errors can be difficult to find and remove without providing the formal logic for what needs to be true at each step to achieve the expected output in the program. In a way, this process is similar to writing a script for a movie. The writer needs to make sure that every scene flows into the next to avoid inconsistencies that will distract the viewer. Indeed, the debugging process is prevalent in all disciplines, and particularly critical in engineering such that results from testing are fed back into the design process to refine the product or solution.

Deadlines and Hand-In

Part 1 Deadline – [6/5/19, 2:30 PM]: You should have all written your initial chapters. Post your individual chapter on your group's Canvas page **Do not upload your chapter before 2:30 on 6/5/19.**

Part 2 Deadline – [6/5/19, 9:00 PM]: Your group's revised story and your group's Analysis and Reflection are due. Your revised story must be posted on your group's Canvas page and your individual Analysis and Reflection comments must be posted in the Discussion tab of that page.

Grading

Part 1. Chapters graded individually; only partial credit rewarded if the minimum word count isn't met.

Part 2. Debugging: graded as a group, but each member must edit his/her chapter to receive points. Analysis and Reflection: graded individually. Each member must post in the Discussion with a minimum of 3-5 coherent, relevant sentences for full credit.

Late work will not be graded.

Appendix A. Five series of Story Points; choose one series for your group (groups may choose the same series of Story Points)

Series 1

Story Point 1: "OPEN UP! THIS IS THE POLICE!"

Story Point 2: Panting, Kevin ran down the unfamiliar alley way, hoping there would be an exit ahead.

Story Point 3: The pendant wasn't particularly remarkable, but something drew his eye to it... something he couldn't quite put into words.

Story Point 4: "It doesn't respond like that for everybody," the green-eyed girl said. "You're the first person it has activated for in over 25 years."

Story Point 5: "This is illegal... and insane!" he exclaimed.

Story Point 6: Kevin was holding his breath to hide his position... but the pendant was flashing in a rainbow of colors, and nothing he could do could hide it. Surely it would give away his position.

Story Point 7: "You know your journey has only just started, right?"

Story Point 8: As he boarded the helicopter, he reflected on how this crazy journey began and smiled. He knew he had a long ways to go before he was ready.

Series 2:

Story Point 1: "Go! And take the ring!" exclaimed Andrew.

Story Point 2: The volcano started to erupt again.

Story Point 3: "There is a high chance of rain in the next few hours..." Melissa said.

Story Point 4: "We are so fortunate to have you in our team!", Roger said to Melissa.

Story Point 5: After five full days, they managed to see the sunlight again.

Story Point 6: Looking around this deserted place he saw enormous foot prints—like Big Foot.

Story Point 7: "He is the one that truly deserves the ring!", Melissa told Andrew.

Story Point 8: After a very long journey, Melissa finally arrived in London and soon realized that her bank account was empty.

Series 3:

Story Point 1: "You should park the car over in that area!" Dylan told Kyler, pointing out a place near the gas station.

Story Point 2: Alice managed to get her armor suit and immediately called Kyler.

Story Point 3: They quickly realized how cold it is in the new realm and started to pay attention to the enemy.

Story Point 4: Alice realized the emotional turmoil that was haunting Dylan and shouted: "Dylan, let's get out of here! We do not belong to this place!"

Story Point 5: Kyler managed to obtain the secret code from the wealthy man.

Story Point 6: The elevator started to shake and all of them were in serious danger.

Story Point 7: When they finally woke up, the train was speeding along the Trans-Siberian railway.

Story Point 8: Although Dylan was not sure if he did his job right, he felt he didn't let his friends down.

Series 4:

Story Point 1: "Life is good," Judy said, as she looked at her fiancée Daniel, and her best friend Oliver.

Story Point 2: Suddenly, everything started to tremble and Oliver started to scream.

Story Point 3: The trees started to bend heavily due to the sheer power of the northern winds.

Story Point 4: Judy grabbed Daniel and consoled him.

Story Point 5: They found a small boat in a deserted backyard.

Story Point 6: The road was quite steep and they had to carry Oliver since he was badly hurt.

Story Point 7: On the other side of the ravine, they found an old woman.

Story Point 8: "Oh no, not again …", Judy looked at the sky, as her new husband Daniel clutched her arm.

Series 5:

Story Point 1: At 8 AM Mars time, the spaceship safely took off from the Martian soil.

Story Point 2: The crew was running out of ideas but Taylor realized that they could reach warp 10 if they fixed the superconducting circuits in compartment 2A.

Story Point 3: "Look, it's Gliese 642 D! Hooray!" captain Michelle exclaimed.

Story Point 4: "Get out of there now! You will get burned!" Michelle shouted to Anthony and Laura.

Story Point 5: "This is quite significant for our scientific purposes! We should take a sample from this material!" Dr. Koch said.

Story Point 6: "Sir, you do not have the required rights to access the system!", a robotic voice told Taylor.

Story Point 7: They took the pills and in a few minutes they entered a deep sleep state.

Story Point 8: "On behalf of the entire humanity, we congratulate you and your team for what you have achieved on this mission!" the President bestowed the Galactic Freedom awards on captain Michelle and her team.

Appendix B. Sample story: four chapters independently developed for a different set of Story Points. (See Appendix C, an attachment, for how these individual chapters were revised into a coherent story.)

Chapter 1.

Matthew is eating a peanut butter sandwich, while Kimberly is preparing this year's taxes in the living room.

The TV is on but nobody watches it. Matthew stares at Kimberly, while Kimberly is focused on the stack of receipts on the table.

As she goes through the receipts and enters the numbers onto her laptop computer, tired and frustrated with the tediousness of the task, Kimberly realizes that she has been doing the taxes for the family ever since she and Matthew married five years ago.

She stops and looks over at Matthew. Matthew swallows a mouthful of his sandwich, and says, "What?"

Kimberly says nothing, eyeing Matthew's sandwich, potato chips, and the glass of Pepsi in front of him.

Sensing the tension in the room, Matthew averts his eyes and glances over at the TV. Hearing yet nothing from Kimberly, Matthew returns to Kimberly. He suddenly sees the receipts held in Kimberly's hands. "Oh no!" he thinks, "The receipts! The receipts!" Feeling a pang of guilt and panic, Matthew blurts out, "I am sorry!" and immediately moves towards Kimberly.

Puzzled and taken aback by the apologetic look on her husband, Kimberly has her mouth open but fails to make a sound, as Matthew sits down beside her on the couch.

Matthew sobs as he tells Kimberly of the terrible event that had just occurred.

Chapter 2.

Matthew sobs as he tells Kimberly of the terrible event that had just occurred.

Kimberly tries to calm down Matthew and offers him a cup of tea. While Matthew is starting to feel better, she suddenly receives a call from her workplace. An emergency situation occurred and she has to go there immediately. She explains to Matthew the issue and then goes straight to her car. After a few minutes of driving on the highway, she observes in the mirror a black Cadillac that continues to stay behind her. After a while, the black Cadillac approaches her more and she can see quite well the driver. He is a black bearded man, with a patch on his left eye and a rabbit upper lip. She observes how he is continuously looking at her and smiling in a threatening, quite evil manner. She remembers from what Matthew told her, that the man chasing her quite resembles Matthew's description. After about 30 minutes of chasing her along the road, she decides to do a risky maneuver so that she can escape the chase.

In a split second, Kimberly manages to swerve around the menacing looking man, standing in the middle of the highway, but not without throwing the car off the road.

Chapter 3.

In a split second, Kimberly manages to swerve around the menacing looking man, standing in the middle of the highway, but not without throwing the car off the road.

Their car sailed went airborne, sailing over a ditch, and landed in a mysterious bog. The car began to sink into a substance like quicksand. Matthew tried the door. It was locked! Kimberly tried the windows. They wouldn't open! Matthew dialed his cell phone. No service! Their car sank deeper and deeper into the muck. Kimberly and Matthew looked at each other with tenderness and horror. "I'm sorry," Kimberly said. "I love you," Matthew replied. Surrounded by blackness and breathing their last breaths, suddenly their car began to ascend. It rose higher and higher, back into the open air, and then was set down slowly and gently onto the deserted highway. Kimberly and Matthew gasped for breath and then gasped in surprise. The menacing man was standing on the hood. He turned, before their eyes, into three twinkling lights which flew off into the distance. The car was muddy but the engine started right away. As they drove off…

Matthew can't help but say a silent prayer of gratitude for the great kindness he had received.

Chapter 4.

Matthew can't help but say a silent prayer of gratitude for the great kindness he had received.

Matthew struggled to his feet, still a bit disoriented, then turned to help Kimberly to her feet. The "menacing" man was busy trying to rev the engine. "I think I have it, if you want to push!" he called to them, as he pulled the car out of the ditch on to the shoulder of the highway.

Kimberly and Matthew climb up out of the ditch and the man is getting out of the car.

"Sorry for the trouble folks! I'll just be on my way."

"Hey, it's no trouble! Thank you for how much you helped us," Matthew said.

With that, the man smiled and turned to walk away down the highway, as Matthew and Kimberly drove home. They reach their home, give each other a melancholy, knowing smile, and enter the house.

Matthew and Kimberly settle down at the kitchen table, grateful that they can finally finish their meal.

End of exercise
